ISSN 0280–5316
ISRN LUTFD2/TFRT--5512--SE

Fuzzy Control of a Domestic Hot Water System

Jerker Sjögren

Department of Automatic Control
Lund Institute of Technology
September 1994

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September 1994

Document Number
ISRN LUTFD2/TFRT--5512--SE

Author(s)
Jerker Sjögren

Supervisor
Lars Halling, TA, Karl-Erik Årzén, LTH

Sponsoring organisation
Tour & Andersson, Malmö

Title and subtitle
Fuzzy Control of a Domestic Hot Water System (Fuzzy-reglering av ett tappvarmvattensystem)

Abstract

The purpose of this thesis has been to examine how Fuzzy Control can be applied on a domestic hot water system. A regular PI-controller has been replaced by a Fuzzy Controller and tuned in order to improve the control action compared to the PI-controller. This controller has then been extended by giving the controller more information about the system. An investigation of using a Supervising Controller has also been made in order to coop with slowly varying parameters.

It is obvious that it is hard to optimize a Fuzzy Controller since there is an enormous amount of parameters to tune. The controllers derived in this thesis have improved control action compared to the original controller even though they are not optimized in any way. However, the results can be seen as a briefing of possibilities for further development.

Key words
Fuzzy Control, Heat Exchanger, Domestic Hot Water System

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
65

Recipient's notes

Security classification

Contents

1. Introduction 2

2. Domestic Hot Water Control 4
- 2.1 General 4
- 2.2 Control Problems 4
- 2.3 An Existing Controller 6

3. Mathematical Models of the System 9
- 3.1 The Heat Exchanger 9
- 3.2 The Valve 11
- 3.3 The Sensor 13
- 3.4 Recirculation loop 13
- 3.5 The Simulink Model 13

4. Fuzzy Control 15
- 4.1 Fuzzy Sets 15
- 4.2 Fuzzy Logic 16
- 4.3 Fuzzy Logic Systems 16
- 4.4 Fuzzy Logic Control 20
- 4.5 FuzzyCODE — A Design Toolbox 22

5. Direct Fuzzy Control 23
- 5.1 PI-Design of a DFC 23
- 5.2 A Controller State-Space Approach 26
- 5.3 The Extended PI-Controller 28
- 5.4 A Fuzzy Controller with 3 Input Signals 30
- 5.5 Summary 35

6. Supervisory Fuzzy Control 37
- 6.1 Designing an SFC 37
- 6.2 Non-linear Functions with Fuzzy Control 39
- 6.3 Summary 42

7. Implementation Issues and Coding 43
- 7.1 Real-time Systems 43
- 7.2 Modula-2 Coding 43
- 7.3 Real-Time Simulation 44

8. Conclusions 45

9. References 46

A. Simnon Code 47

B. Using Simulink 54

C. Modula-2 Code 59

1. Introduction

Domestic hot water systems are claimed to be hard to control with classical control theory. Even though the process is well known it is not very likely that one can derive a controller with satisfactory results. This is, of course, due to the non-linear characteristics of the process. Because of the many varying parameters it is a process that has many different working points and therefore it is hard to derive a linear controller which behaves well in the whole working area.

Instead of analyzing the process the traditional way, one can try to make an alternative approach. By analyzing the behavior of the system under different conditions, rather than studying the origin of the dynamics, one can achieve a reasonable understanding of the process characteristics.

This is probably one of the most common ways to analyze a more complicated problem. This is also the case when introducing controllers to processes which has been controlled manually for a long time. The operator knows how and when to push the buttons to obtain good control action even though the system could be very complex and highly non-linear.

Fuzzy logic and in particular *fuzzy control* is one of the new areas in control-theory that has been vividly discussed the last couple of years. Some claim that this is *the* way of controlling anything that moves while others are more critical. The word *fuzzy* has become a key word for many manufacturers and is frequently used as a sales argument for different kinds of home electronics.

The aim of this thesis is to investigate the possibilities of applying fuzzy control to domestic hot water control. The work has been performed in cooperation with Tour & Andersson (TA), a swedish manufacturer of control systems for HVAC (Heating, Ventilation and Air-Conditioning) processes.

In this work we will take a closer look at two different fuzzy applications. Firstly we will discuss the use of a fuzzy controller as a direct replacement of the ordinary controller, a so called *direct fuzzy controller*, DFC. Secondly we will take a look at the fuzzy controller as a so called *supervisory fuzzy controller*, SFC. Used as a SFC the aim of the controller is to collect data from the environment in order to provide the ordinary controller with correct parameter values. The two ways of implementation can be seen in Figure 1.1.

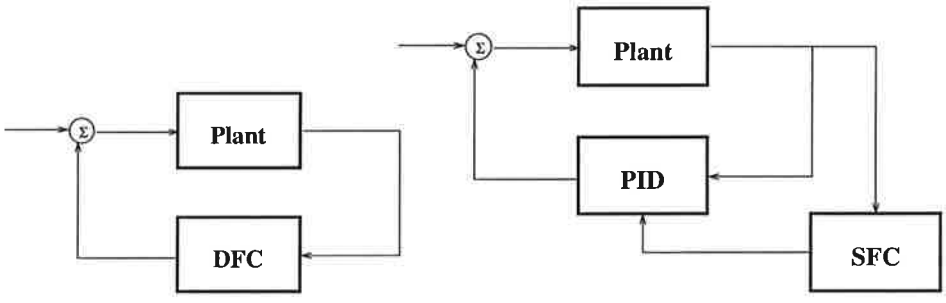


Figure 1.1 The structure of a DFC (Direct Fuzzy Controller) and a SFC Supervisory Fuzzy Controlller).

The purpose of this thesis is also to show some of the problems and constraints with this type of control and hopefully give the reader an insight and understanding of fuzzy techniques.

Chapter 2 gives an introduction to domestic hot water control in general and some of the problems. Here is also the controller presented that is currently used by TA. Chapter 3 will derive some of the mathematical models used for simulation. In Chapter 4 some fuzzy logic theory is presented. Here is also a description of the toolbox *FuzzyCODE* found. Further, in Chapter 5 and 6 some design problems and solutions are discussed for a DFC and a SFC respectively. Finally, in Chapter 7 the implementation of a controller and realtime systems in general are described. In the Appendix much of the code and the models used in the different programs and environments are found.

Jerker Sjögren, September 1994

2. Domestic Hot Water Control

In this chapter some aspects of domestic hot water control are discussed and what might be possible to improve and what is not.

2.1 General

The system to be examined in this thesis is illustrated in Figure 2.1.

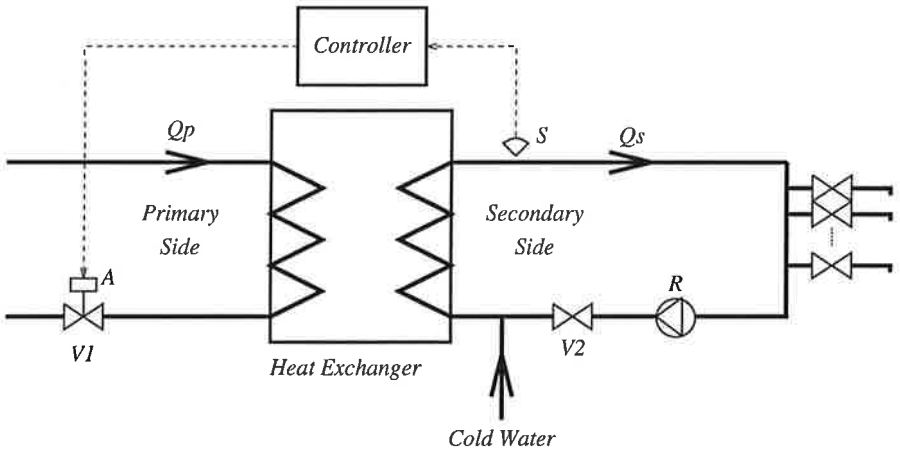


Figure 2.1 An overview of the control problem. Note that the valve, $V1$, is placed *after* the primary water has passed through the heat exchanger. That is because the water is cooler on this side which is an advantage.

Hot water on the primary side comes through the valve, (V_1). By measuring the temperature on the secondary side in (S), the aim is to keep the temperature constant on the secondary side by controlling the actuator, (A). The actuator is controlling the valve (V_1) which regulates the primary flow (Q_p). The system also has a recirculation pump (R) running in order to always keep the secondary flow (Q_s) above zero. Since the pipes in a hot water system can be very long, (up to 100 m) the hot water would cool down in the pipes during zero load without this pump.

The system is affected by different load disturbances in the form of changing secondary flow when water is tapped from the system. Other problems are changed boundary conditions such as different water-pressures and variating temperatures. Thus the problem is a *regulator-problem* and not a *servo-problem* since the reference value is kept constant all the time and there will only be the load disturbance to control.

2.2 Control Problems

Things that has to be taken into consideration when designing a controller are generally called "problems". In this Section we will discuss the non-linearities of the system and some other control problems that might occur.

Nonlinearities

Since one of the ideas of fuzzy control is the capability of handling unknown processes and specially non-linear ones, one of the more important analysis that has to be done is to find out how the non-linearities appear.

To get a rough idea of the behavior it would be desirable to make a step response for certain working points and measure the rise-time and the static gain for these points. Since the secondary flow is the most variating parameter it therefore makes sense to find out the non-linear behavior for different secondary flows. Assume that the reference temperature is constantly set to $50°C$, which is the normal domestic hot water temperature. Assume further that the valve is adjusted so that we achieve approximately this temperature for a certain secondary flow. It will then be possible to make a change of the valve position around the reference temperature, $50°C$. This has been done for secondary flows from 10–100% of the maximum secondary flow and for each flow a change of the valve position of 5% of the maximum valve opening. The result is shown in Figure 2.2.

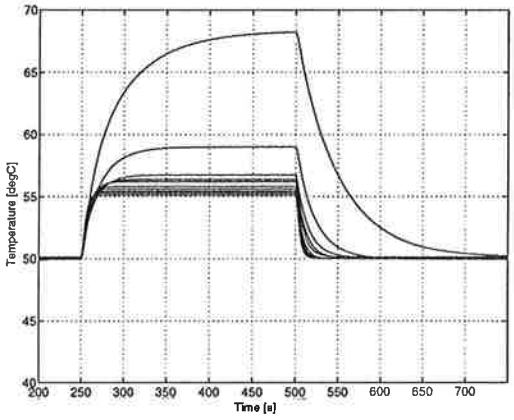


Figure 2.2 A change of the valve position of 5% of the maximum valve opening for the open loop heat exchanger system for different load disturbances. Note that the open loop system includes the valve as well. Starting with a load of 10%, *upper curve*, up to 100%, *lower curve*, in steps of 10%.

Measuring the static gain and the time constants for the different loads gives the diagrams in Figure 2.3.

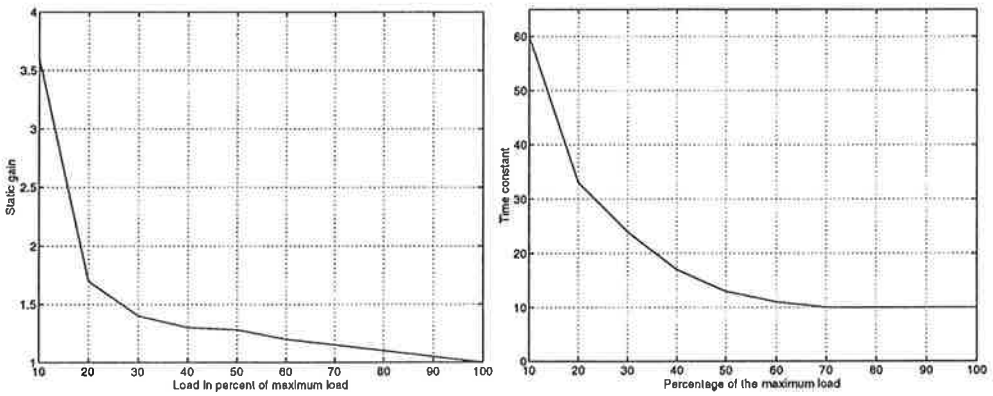


Figure 2.3 The left diagram shows the system's static gain verses the load. The diagram to the right shows the time constant verses load disturbance.

As we can see the open system is hardly linear even though the valve

compensates to some extent for the heat exchanger non-linearity. (Further discussion in Section 3.2.)

We will use these results later in Section 5.4 when taking consideration to the load level in the controller.

Naturally these are not the only non-linearities. Other non-linear behaviors have been noticed such as the valve oscillation for lower load disturbances.

Other Problems

The actuators in our system move from 0 to 100% in about 15 seconds. Since this frequency is well in the range of the heat exchanger's itself this will be a big constraint when designing the controller.

Domestic hot water systems are generally characterized not only by the heat-exchanger itself but are also strongly influenced by changing boundary conditions. This could be such things as the primary water temperature which could typically vary from $65°C$, in the summer, to $90–100°C$, in the winter. Even the cold water temperature may vary slightly, from about 5 to $10°C$. The primary water pressure can also vary due to season, typically between $150–500kPa$.

The valve itself can be a problem when the system has little or no load at all. We will take a closer look at this problem in section 3.2.

The load, i.e. the secondary flow, is also characterized by very big changes. From no load at all during the night to almost full load at certain hours. These changes can be very fast — a change of 50% of the maximum flow in a few seconds is not unlikely.

It is therefore of great importance that the controller to be derived is capable of handling large changes of the process parameters. This will raise the question:

What kind of controller is to be used?

With the knowledge of varying parameters it is natural to think of an adaptive control design to solve the problem. This has also been done in different works [Nesler, 1986], [Radke and Isermann, 1987]. One problem with adaptive control is that it is difficult to ensure that the parameter estimator functions correctly when large load disturbances cause dramatic changes of the operating point.

However, this work will focus on other solutions such as fuzzy control and gain-scheduling. These methods are claimed to take good care of this kind of difficulties.

2.3 An Existing Controller

The classical way of controlling this type of processes is to use some kind of PI-controller with preset parameters. The worst case would be that these parameters never would be changed during operation which will, most likely, result in a non-optimal controller.

The existing controller for the system is a PI-like controller developed at Tour & Andersson AB (TA) in Malmö. This controller will constitute the basis of the derived controllers in this thesis since it has been proved to behave well. A block scheme of this controller can be seen in Figure 2.4.

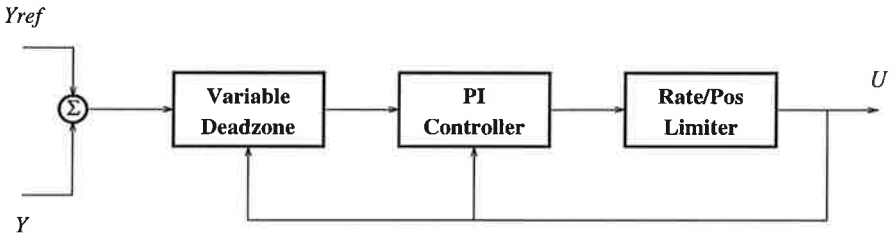


Figure 2.4 Block scheme of the controller developed at TA in Malmö.

As illustrated it is possible to divide the controller into three different blocks, namely:

- Variable deadzone
- PI-controller with anti-windup
- Rate- and output-limiters

Here will now follow a step by step description of the different blocks in the controller.

Variable Deadzone

The input block has been provided with a deadzone for the error. When the error is found in a certain range close to zero the deadzone block gives zero as output which will force the controller to give a constant output signal.

A general problem with control valves are that they cannot control a flow below a certain limit, known as the *smallest controllable flow*. When the valve position is decreased below the position corresponding to the smallest controllable flow, the flow will almost instantly drop to practically zero. (Only a certain *leakage flow* will remain.) This means further that if you would like to keep a steady, small primary flow it is likely that the valve will begin to oscillate.

An improvement of the design above is to let the deadzone be variable and dependent of the load level. Since this quantity can be hard to measure the second best to do is to measure the control signal which of course is highly dependent on the load level. With this design it is possible to save the valves during low load of the system by letting the deadzone increase when load decreases. This will of course affect the quality of the control but in return the actuator will run longer without failure. If the switching level between small and large deadzone is appropriately selected, the control performance will mainly be deteriorating when there is no hot water consumption.

A schematic illustration of the deadzone is shown in Figure 2.5.

Rate and Output Limiters

Since the actuator cannot change positions instantly, nor does the valve have unlimited control range, we will need some limitations for the output. A rate limiter to limit the control signal's change of rate and further a saturator that only allows values in a certain range. The block can be illustrated as in Figure 2.6.

PI-Controller with Anti-Windup

As a control algorithm an ordinary PI-controller on position form is used with an anti-windup function added to ensure that the controller does not give control signals which cannot be followed by the actuators. The anti-windup

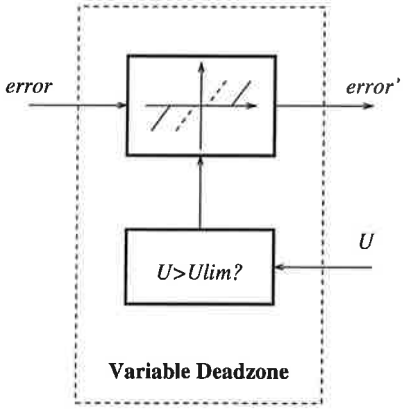


Figure 2.5 The deadzone implementation in the TA controller. There are two crisp limits of the deadzone which are changed when the limit, U_{lim}, is exceeded.

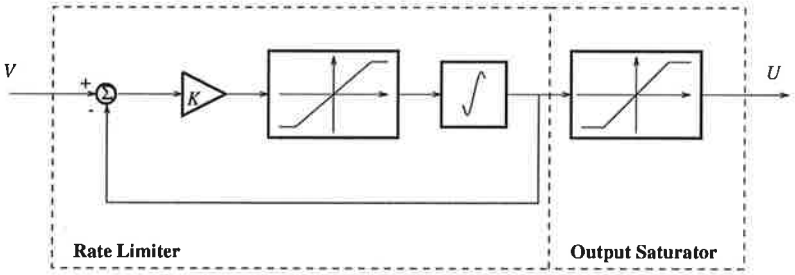


Figure 2.6 Rate- and output-limiters.

function is implemented so that a *tracking error* is calculated. The tracking error is the difference between the real output and the output that the controller gives. This signal is further amplified and added to the integral part of the controller. This is a standard implementation of a PI-controller.

3. Mathematical Models of the System

Simulating systems in some kind of computer environment is surely the most common way of developing new controllers or any other new equipment. To be able to simulate *any* system it is of great importance that there exists a proper model to work with. Apart from the accuracy of the model it is also essential that it is not too complex and hence requires too much power from a computer's point of view. If that would be the case, the calculations will take too long time and the model can be discarded.

3.1 The Heat Exchanger

A heat exchanger is a unit which transfer heat energy from one medium to another. In a hot water system there is one hot, primary side, from where the heat energy is taken and one less hot, secondary side, to where the heat is transferred to. A principal illustration of the heat exchanger is shown in Figure 3.1.

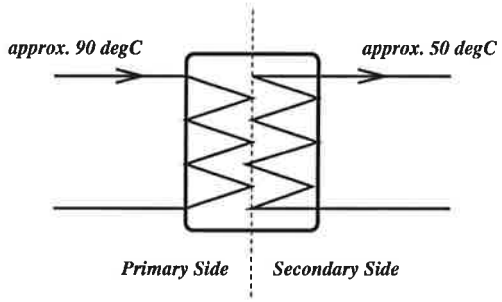


Figure 3.1 Principal illustration of a heat-exchanger.

Basic Relations

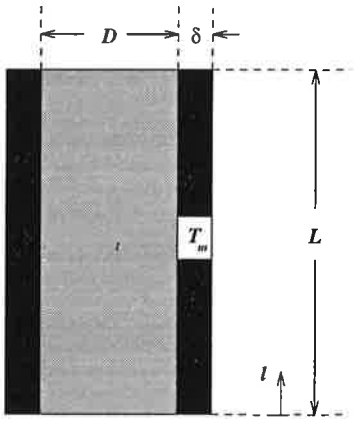


Figure 3.2 A pipe-section filled with water.

For a pipe section filled with flowing water (illustrated in Figure 3.2) the heat balance can be expressed as

$$\frac{\partial T_l}{\partial t} + w\frac{\partial T_l}{\partial l} + \frac{\pi D \alpha_l}{S\rho_l c_{pl}}(T_l - T_m) = 0 \tag{3.1}$$

where

T	—	Temperature $[°C]$
D	—	Inner diameter of the pipe $[m]$
L	—	The length of the pipe section $[m]$
l	—	Length variable $[m]$
S	—	The cross-section area of the pipe $[m^2]$
ρ	—	Density $[kg/m^3]$
α	—	Heat transfer coefficient $[W/m^2K]$
c_p	—	Heat capacitivity $[J/kgK]$
w	—	Mean velocity for the medium $[m/s]$
t	—	Time $[s]$

with the indices

l	—	Liquid (water)
m	—	Metal (The pipe)

Since the expression in (3.1) is a partial difference equation, which is difficult to handle, it is convenient to make the approximation

$$\frac{\partial T_l}{\partial l} = \frac{\Delta T_l}{L}$$

This means that if we consider the temperature constant in one, small compartment with the length, L, we can rewrite equation (3.1) to

$$\frac{dT_l}{dt} + w\frac{\Delta T_l}{L} + \frac{\pi D \alpha_l}{S\rho_l c_{pl}}(T_l - T_m) = 0 \tag{3.2}$$

This equation can be applied directly on heat exchangers with an identical balance equation on the secondary side.

If it is further assumed that

- the heat capacitivity of the metal is divided by two and each half of it is added to the total heat capacity for the primary and the secondary side respectively,
- the heat transfer to the surroundings can be discarded,
- the temperature regarded is the mean value of the incoming temperature to the compartment and the temperature inside it.

The heat balance equation for one compartment on the primary side can then be written as

$$C_p\frac{dT_p}{dt} = q_p c_p(T_{pi} - T_p) - \frac{UA}{2}(T_{pi} + T_p - T_{si} - T_s) \tag{3.3}$$

with

C_p — Heat capacity for the compartment $[J/K]$
q_p — Primary flow $[l/s]$
c_p — Heat capacitivity for water $[J/kgK]$
T — Temperature $[°C]$
U — Overall Heat transfer coefficient $[W/m^2K]$
A — Area between two compartments on contrary sides $[m^2]$

Note that

$$\frac{1}{U} = \frac{1}{\alpha_p} + \frac{\delta}{\lambda} + \frac{1}{\alpha_s}$$

where

α_p — Heat transfer coefficient, prim. side, water/metal $[W/m^2K]$
α_s — Heat transfer coefficient, sec. side, water/metal $[W/m^2K]$
λ — Thermal conductivity of metal $[W/mK]$
δ — Wall thickness $[m]$

It has been shown [Halling, 1991] that equation (3.2) is improved for stationary flows if the mean value of the compartment temperature and the incoming temperature to the compartment is used instead of only the compartment temperature.

This model uses 5 compartments which gives us a satisfactory resolution of the compartment temperatures.

3.2 The Valve

The valve used in this system is a so called *equal percentage valve*. It has its name from the characteristics which is exponential. As we shall see, the heat exchanger's characteristics is logarithmic, which means that a valve like this will roughly compensate for the static non-linearity in the heat exchanger. This is a common way of trying to linearize a heat exchanger system's transfer function. The problem is that although this model will work well for some operating conditions it is impossible to create a general compensation in order to make the system linear for the whole working space. An *ideal* coordination between a valve and a heat exchanger is shown in Figure 3.3. Note that this coordination only compensates for the *static* non-linearity and we will still have to deal with the *dynamic* non-linearities that might exist.

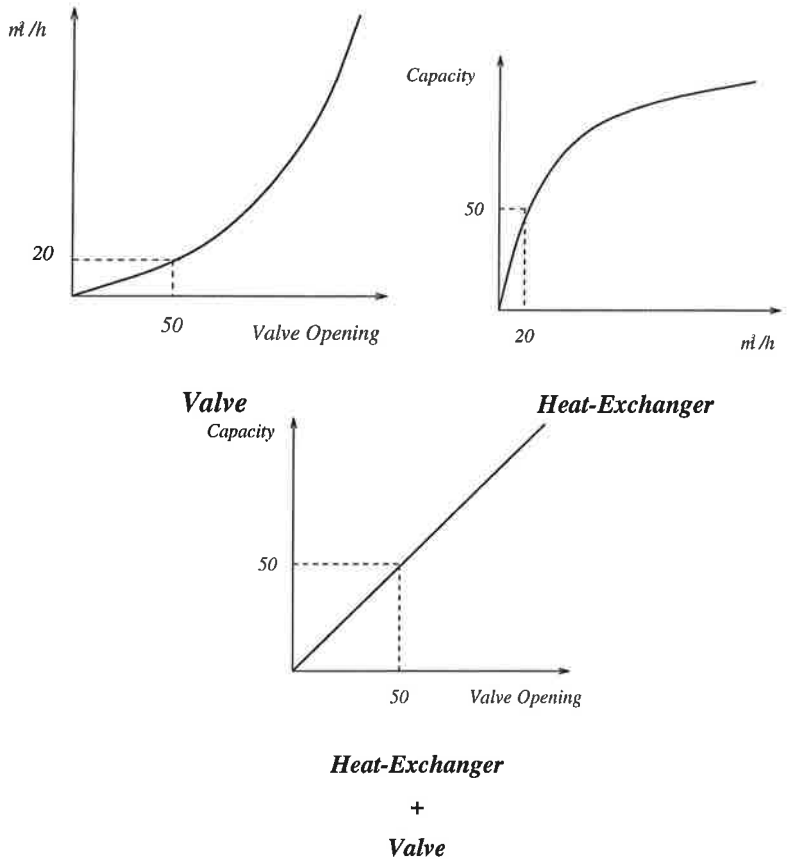


Figure 3.3 How the valve's and the heat exchanger's characteristics are coordinating.

The model characteristics of the *real* valve is shown in Figure 3.4.

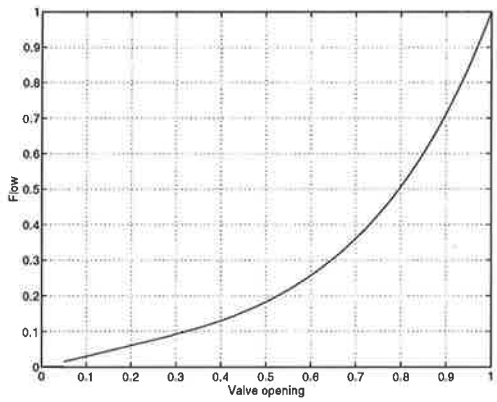


Figure 3.4 The real valve's characteristics. Note that in the first part, $h = [0, 0.05]$, the flow is *not* equal to zero but has a leakage flow of $q_{leak} > 0$.

In the first part of the characteristics, the valve is actually closed and the flow, q_v, ought to be zero. In reality there will however appear a leakage flow which can be considered constant in the interval $h = [0, 0.05]$, where h is the valve position. $h = 0$ corresponds to a closed valve and $h = 1$ corresponds to a fully open valve.

Next part can be considered linear as

$$q_v = ln(s)s^{1/ln(s)-1}h$$

where

$$h = [0.05, 1/ln(s)]$$

and s is equal to the *control range*, in our valve $s = 30$. $(1/ln(s) \approx 0.294$, i.e. $q_v = Ch$ where $C = 0.308$.) Note that the slope of the linear part crosses origin which means that there will occur a discontinuity between the leakage part and the linear. As we have discussed before this will cause problems and has to be considered when deriving a controller.

The third part of the valve is considered exponential with the form of

$$q_v = s^{h-1}$$

where

$$h = [1/ln(s), 1]$$

3.3 The Sensor

The temperature sensor can be modeled as a first order system with the time-constant $t_s = 1.5s$, i.e. as

$$G(s) = \frac{1}{1 + t_s s}$$

3.4 Recirculation loop

This loop is modeling the dynamics in the recirculation loop. The system is modeled as a first order system with a time delay. The time delay corresponds to the time that the water needs to flow through the pipes. With the model

$$G(s) = \frac{k}{1 + t_c s} e^{-t_d s}$$

where $k = 0.9$ and $t_c = t_d = 50s$, the temperature fall in the recirculation loop will be approximately $5°C$.

3.5 The Simulink Model

The model used by TA has then been translated into the simulation toolbox of SIMULINK. This toolbox is described in Appendix B and we will here only have a brief look at the block structure of the simulation model.

The model that is used for the simulations can be seen in Figure 3.5.

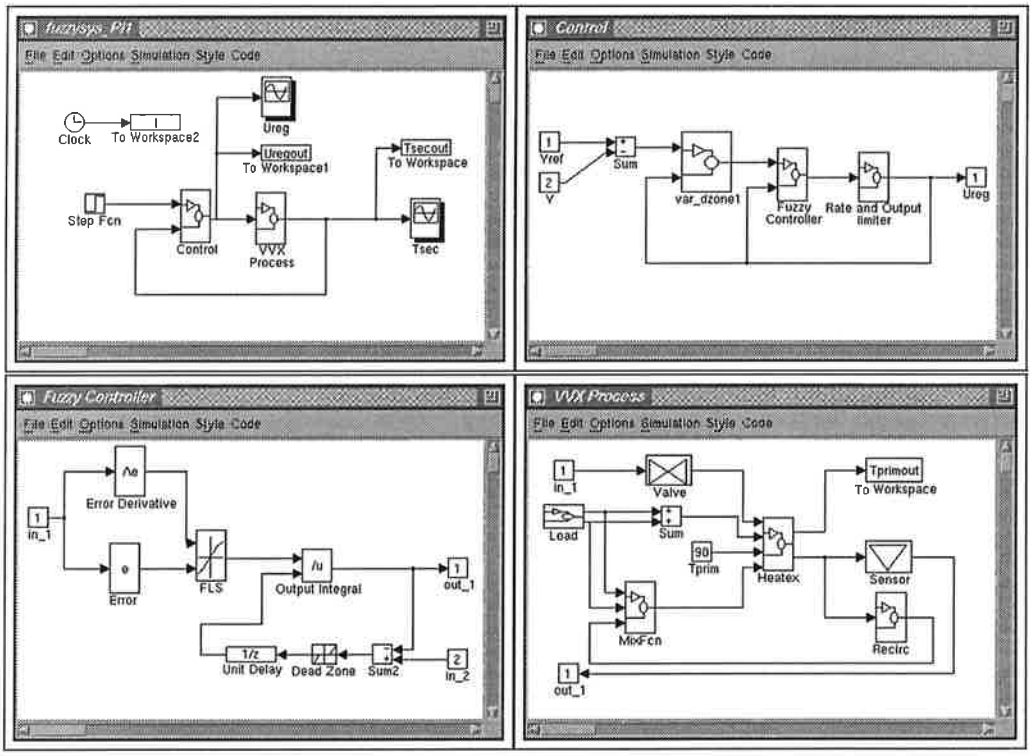


Figure 3.5 *Upper Left* Describes the whole system divided into the controller and the process. *Upper Right* Shows the controller part of the system with its three blocks: Variable deadzone, Controller and the Output limiter. *Lower Left* Illustrates the actual control algorithm, in this case an *incremental fuzzy controller* with anti-windup. *Lower Right* The process divided into subsystems. In the middle we can see the heat exchanger (marked with *Heatex*) with its four inputs (from top to bottom): Primary flow, Secondary flow, Input Primary temperature and Input Secondary temperature. The two outputs are the Output Primary temperature and the Output Secondary temperature.

4. Fuzzy Control

Classical control theory is characterized by the use of mathematical models of the process and the controller is based upon these models. A fuzzy controller, on the other hand, is made up from a set of rules and the control action is taken based on them, such as

```
IF <condition 1> AND <condition 2> AND ... AND <condition N>
   THEN <control action>
```

A fuzzy controller for an oven could for example have a rule like

```
IF temperature IS high AND temperature IS increasing
   THEN energy supply IS small
```

4.1 Fuzzy Sets

Fuzzy sets are made up by linguistic values as "HIGH","SMALL", "INCREASING", "OK" etc. The "fuzzy" thing in these expression is the vagueness of when these statements are fulfilled. For example, how small is "SMALL" and what is "OK"? The main idea in fuzzy set theory is that a value is not only "SMALL" or *not* "SMALL" but could be "SMALL" to *any* degree in the interval $[0, 1]$.The value 0, in the interval, represents *false* and 1 represents *true* as in ordinary boolean logic. A comparison of fuzzy logic representation and classical boolean representation is shown in Figure 4.1.

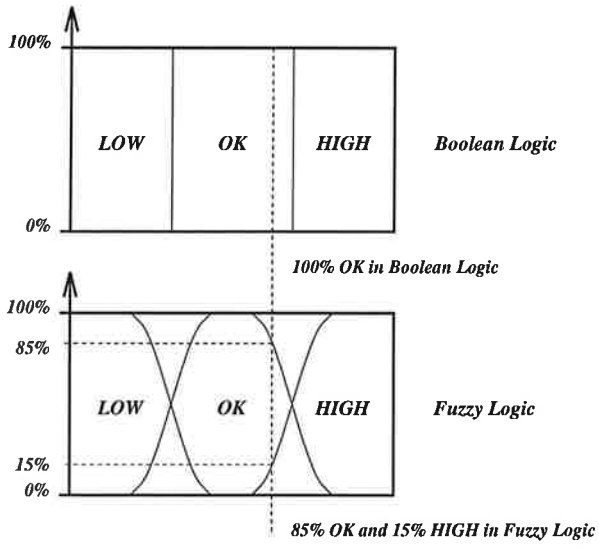


Figure 4.1 A comparison between a boolean and a fuzzy representation of the sets "LOW", "HIGH" and "OK". An example of an evaluation for a given input is illustrated by the dashed line.

The *membership function*, $\mu_A(x)$ is a function which will give a value in the interval $[0, 1]$ for every given value, x. There are three common shapes of the membership function: *trapezoidal, gaussian* and *singletons*. We will only focus

on a special case of the trapezoidal function, namely the *triangular* function with so called *full overlapping*. Full overlapping means that the membership functions overlap each other fully, and the sum of them, for every given input, x, will be constantly equal to 1, i.e.

$$\sum_{i=1}^{C} \mu_i(x) = 1; \quad \forall x$$

The points that characterizes the triangular sets are called *modal points*. The fully overlapped, triangular fuzzy sets are illustrated in Figure 4.2.

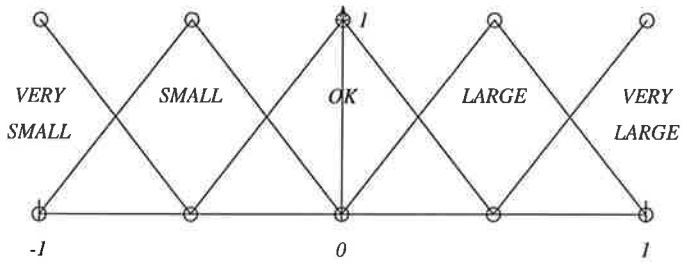


Figure 4.2 Fully overlapped, triangular shaped fuzzy sets with the modal points marked as small circles.

4.2 Fuzzy Logic

By combining the sets with logic such as "AND", "OR" and "NOT", it is possible to build up rules for the fuzzy controller. The operators have different interpretations by different theories although some are more common than others. One frequently used definition of the operators is

Operation	*Definition*
A and B	$min[\mu_A; \mu_B]$
A or B	$max[\mu_A; \mu_B]$
not A	$1 - \mu_A$

4.3 Fuzzy Logic Systems

A *fuzzy logic system* (FLS) can be divided into three different operations represented by

The Fuzzifier maps numerical observations, $x_0 \ldots x_n$, into linguistic (fuzzy) variables, such as 43% HIGH and 15% OK.

The Inference Engine performs the weighting of the fuzzy variables made from a knowledge base.

The Defuzzifier converts the linguistic variables into crisp values, $y_0 \ldots y_m$.

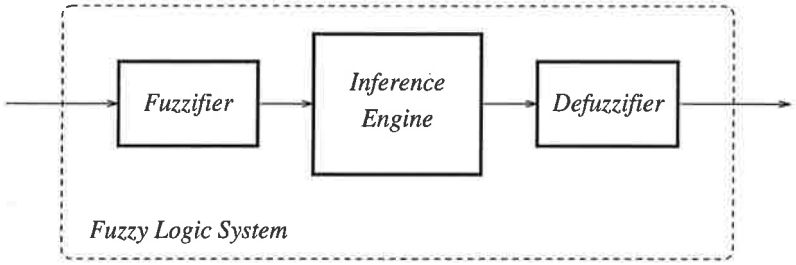


Figure 4.3 The architecture of a fuzzy logic system.

The architecture of an FLS can be seen in Figure 4.3. The methods for performing these three operations have different originators, and therefore several methods exist. We will here only discuss the most common ones.

The Fuzzifier

For fuzzification there only exists one way of converting the crisp values into linguistic and it has already been described in Section 4.1 and 4.3. The evaluation of a fuzzy condition is illustrated in Figure 4.4.

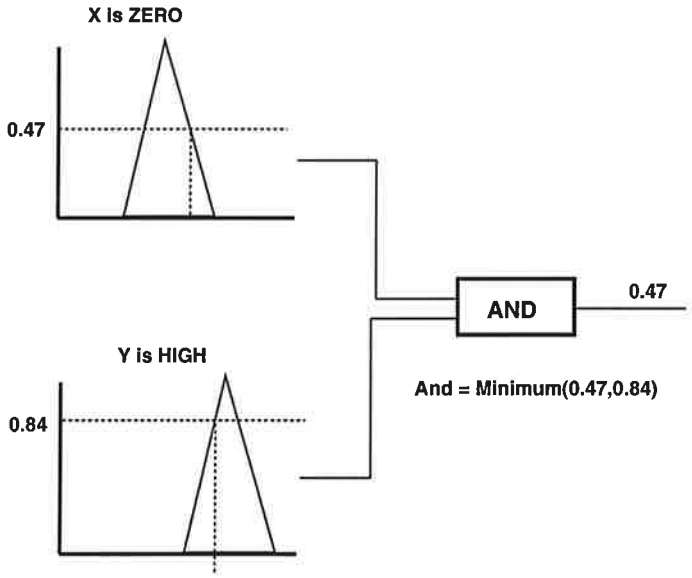


Figure 4.4 Evaluation of a fuzzy condition.

The output of the fuzzifier is in the form of *beliefs* in different rules. The belief of a rule is determined by the degree to which the condition part of the rule is fulfilled. Note that several rules can — and normally have — a belief greater than zero.

The Inference Engine

For the inference engine two main methods exist:

- The *max-min* method (also called the *Mamdani* method) or
- the *max-dot* method (also called the *Larsen* or *product* method.)

In the max-min method the minimum of the membership function of the control action, $\mu(x)$, and the belief, b, in the rule will give the output function as $\mu_w(x) = min[b, \mu(x)]$.

In the max-dot method the membership function, $\mu(x)$, is instead multiplied with the belief, b, which will give the output function as $\mu_w(x) = b\mu(x)$.

The two inference methods are illustrated in Figure 4.5.

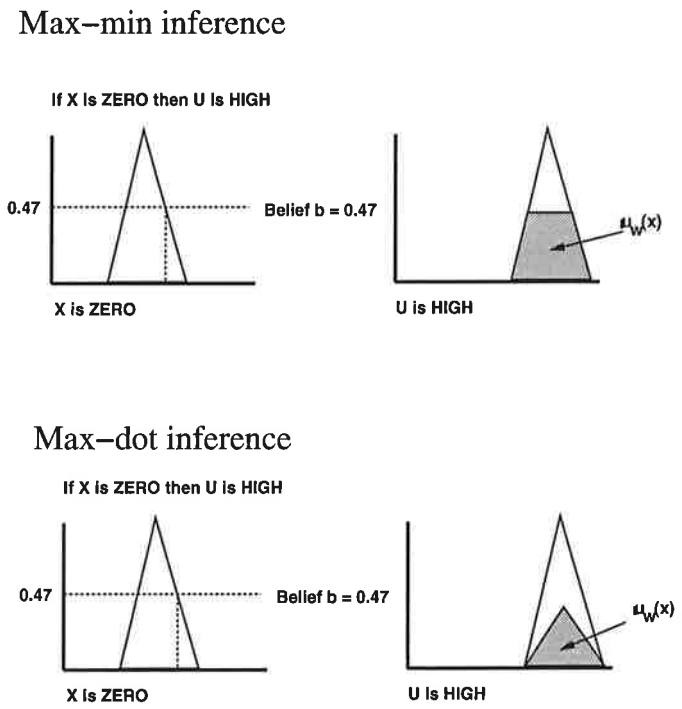


Figure 4.5 Comparison of the max-min and the max-dot methods of inference.

Whether the max-min method or the max-dot method has been used the combined control action, $\mu_u(x)$, has to be calculated. This could be done by pointwise take the maximum of all weighted membership functions as

$$\mu_u(x) = max[\mu_{w1}(x), \mu_{w2}(x) \ldots \mu_{wn}(x)]; \quad -\infty < x < \infty$$

This is illustrated in Figure 4.6.

Another method of calculating $\mu_u(x)$ is to take the sum of all weighted membership functions, i.e.

$$\mu_u(x) = \sum_{i=1}^{n} \mu_{wi}(x); \quad -\infty < x < \infty$$

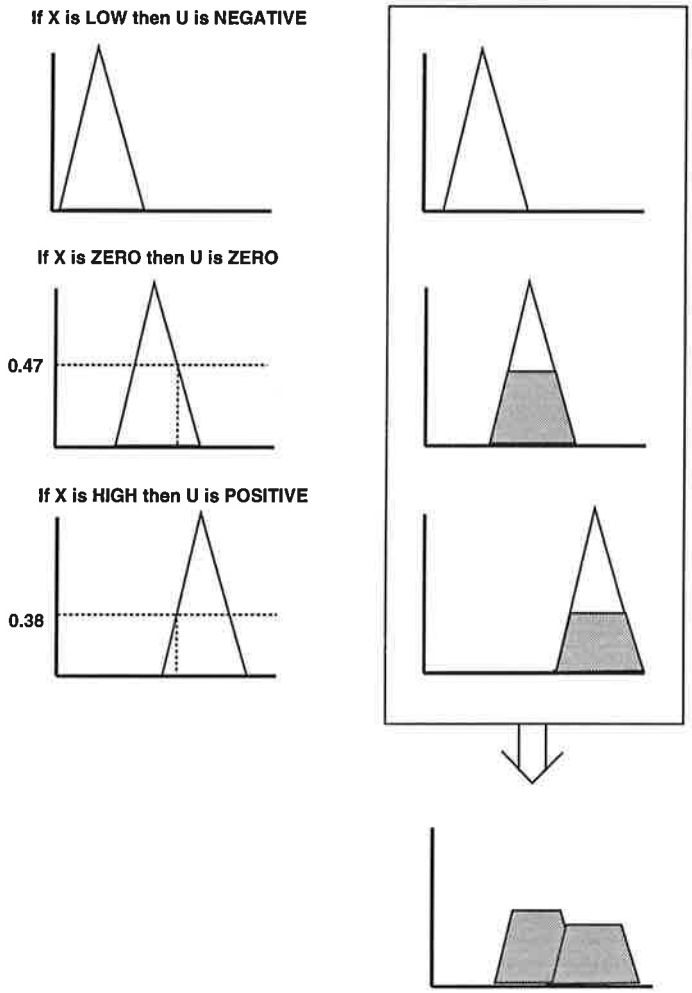


Figure 4.6 How the combined control action is calculated from all the weighted membership functions using the maximum method.

The Defuzzifier

In order to obtain a crisp value from the combined control action some kind of method for weighting this function has to be derived. For the defuzzifier there are normally three methods mentioned, namely

- The *mean-of-maxima* method
- The *center-of-area* method
- The *center-of-gravity* method

The first method takes the x that maximizes function $\mu_u(x)$. If there exists several maximas or the maxima is flat then it takes the mean of the maximas.

A better method is the center-of area method which takes the x that divides the area under the function into two equal parts.

An third, alternative method is the center-of-gravity method which calculates the center of gravity under the fuzzy sets and then chose the x according to this.

The three methods and how they work are illustrated in the figures 4.7 and 4.8.

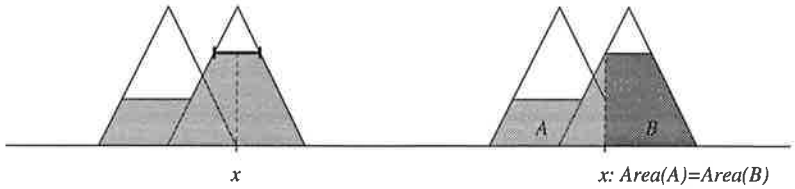


Figure 4.7 The mean-of-maxima method (*left*) and the center-of-area method (*right*).

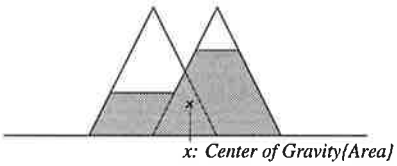


Figure 4.8 The center-of-gravity method.

4.4 Fuzzy Logic Control

A fuzzy controller is made up of some linear filters and the FLS. The schematic fuzzy controller is shown in Figure 4.9.

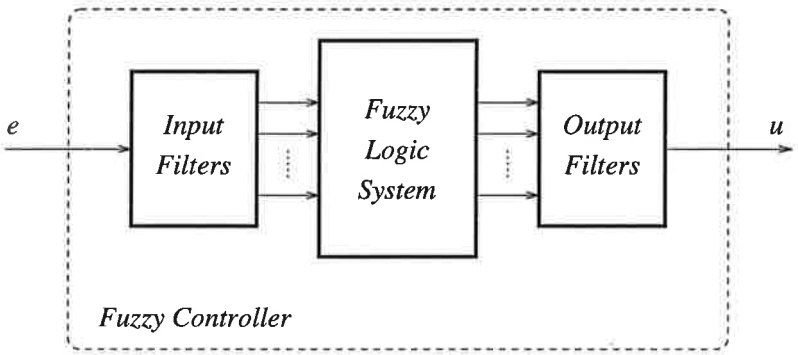


Figure 4.9 A block illustration of the fuzzy controller.

The input filters could for example be a derivative filter or an integrator which will introduce some kind of dynamics into the controller. Because the FLS is restricted to signals in the interval $[-1, 1]$ it is necessary to normalize the input signals. The gain is called *normalization gain*. This gain is normally chosen so that the maximum input never exceeds the interval $[-1, 1]$.

The FLS itself performs a static, often non-linear mapping from the input to the output and hence never introduces any dynamics to the controller. The mapping can however be designed to be linear as we will see in the next Chapter. An example of a FLS mapping area can be seen in Figure 4.10.

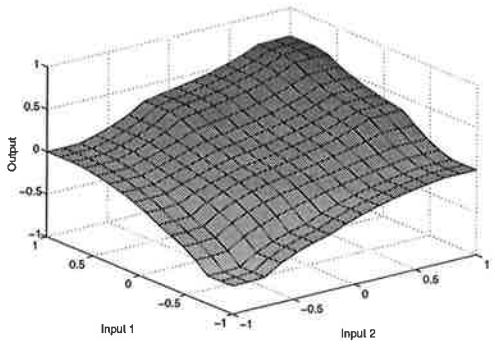
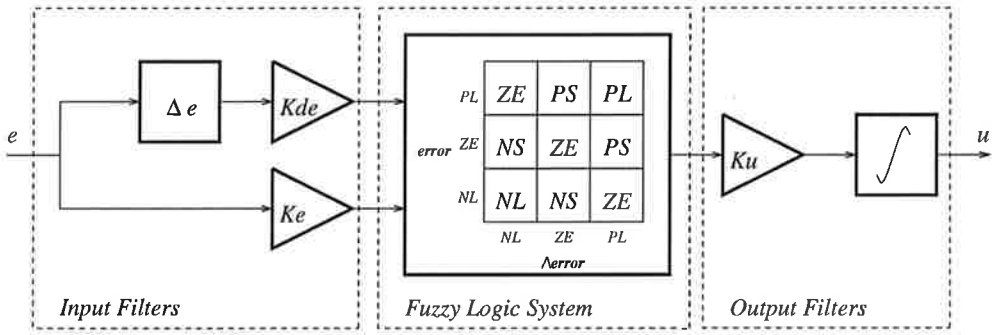


Figure 4.10 An example of a non-linear mapping area of an FLS. The FLS performs a mapping from two inputs to one output.

The output filtering is normally either an integrator or nothing at all. Since the output also is found to be in the interval $[-1, 1]$ it will be necessary to denormalize the output to its final value. This output gain is called *denormalization gain.*

An example of a fuzzy controller and some often used notations in fuzzy control can be seen in Figure 4.11. The structure in this example is equivalent to a PI-controller.



NL=Negative Large
NS=Negative Small
ZE=Zero
PS=Positive Small
PL=Positive Large

Figure 4.11 An example of a fuzzy controller with two inputs (Δe and e) for the FLS. The notation for the rule base in the FLS is a common way of representing the linguistic values.

4.5 FuzzyCODE — A Design Toolbox

In order to design a fuzzy controller a design toolbox has been developed at the Department of Automatic Control at Lunds University. The toolbox is part of the master thesis [Johansson, 1993]. It has a graphical interface and heavily relies on the MATLAB package. It allows the user to specify up to three inputs with seven sets for each input. A 3-D view of the FLS output area can be shown and gives an easy understanding of a fuzzy system when tuning the parameters. The controller can be simulated and evaluated in the SIMULINK environment. An illustration of the design environment is shown in Figure 4.12.

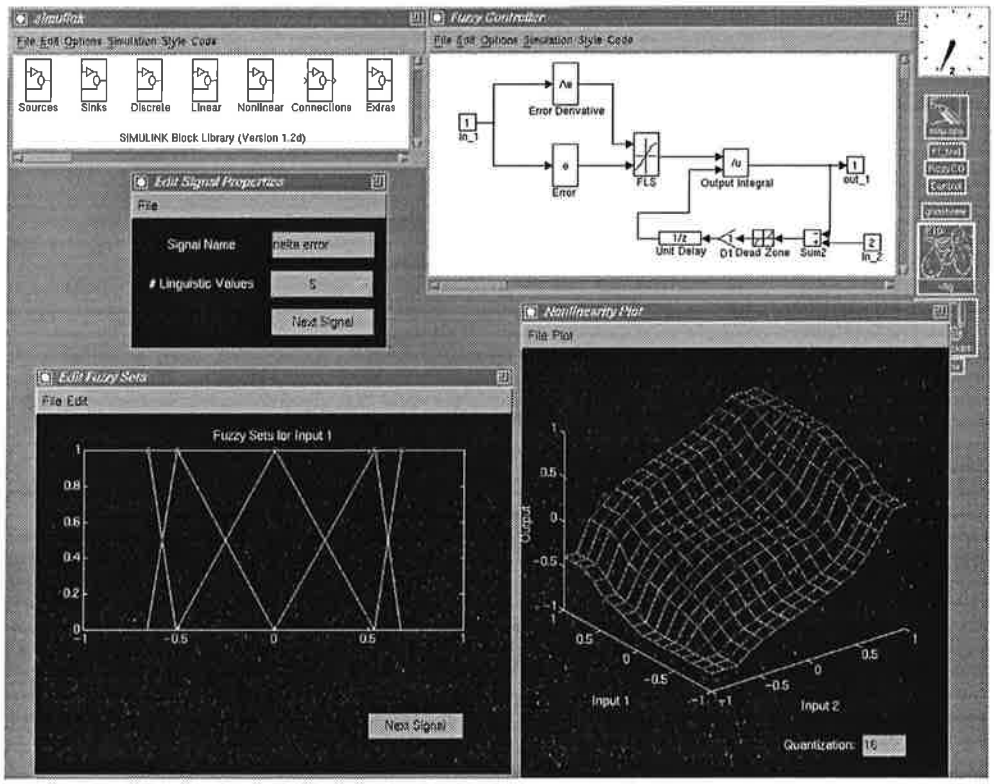


Figure 4.12 The graphical interface of the fuzzyCODE toolbox.

5. Direct Fuzzy Control

Designing fuzzy controllers from scratch is not easily done. The number of parameters to tune and hence the degrees of freedom become enormous. Although this is what makes the controller work well for non-linear systems it also makes it hard to derive an optimal fuzzy controller.

The aim of the following two chapters is to give an understanding of the reasoning process when designing a fuzzy controller. Although there are no step-by-step solutions of fuzzy design there are some basic rules to keep in mind.

5.1 PI-Design of a DFC

This section will describe one way to design a so called *direct fuzzy controller*, DFC for short. The DFC is used to replace an existing controller. A natural way of starting the design of such a system is to try to imitate the already existing controller. After a while when the knowledge about the system is increasing you might be able to tune the fuzzy controller's parameters in order to improve the behavior. Therefore we will now take a closer look at how it is possible to derive a fuzzy PI-controller, i.e. a "linear" fuzzy controller.

Creating a Linear FLS Area

Since fuzzy controllers normally are non-linear it is of great interest to know whether it is possible or not to derive a linear fuzzy controller as a special case of the non-linear ones.

In [Johansson, 1993] some different interpolation methods are discussed. It is there shown that if we use the *max-dot* inference-method and then *sum* the weighted membership functions one can obtain a linear fuzzy controller. Other forms of inference methods may also result in a linear controller if restrictions are put on the fuzzy sets.

The max-dot inference method is used in this thesis. With a restriction of only using rectangular, non-overlapping output sets there will be no difference between *summing* the weighted membership functions and taking the *maximum* of them. We will use the maximum method in the future.

In FuzzyCODE, earlier described in Section 4.5, we use triangular input sets and rectangular output sets. We also use the interpolation method described above, which gives us a possibility to create a linear controller.

A linear FLS area must of course consist of a linear *rule base*. With three input sets and five output sets for a two-input and one-output controller we choose the rule base shown in Figure 5.1.

error \ /error	NL	ZE	PL
PL	ZE	PS	PL
ZE	NS	ZE	PS
NL	NL	NS	ZE

Figure 5.1 The rule base for a PI-controller, with the same notation as used in Section 4.4. The FLS has three input sets and five output sets.

With triangular input and rectangular output sets as in Figure 5.2 we obtain an FLS area like in Figure 5.3.

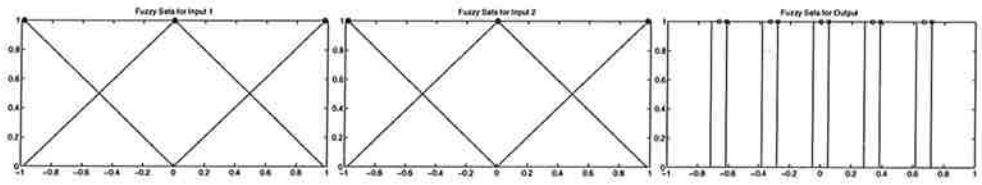


Figure 5.2 The input sets and the output sets for a PI-controller.

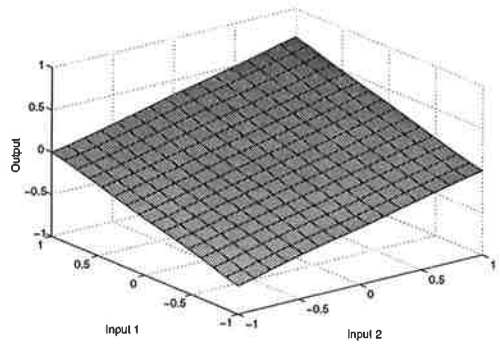


Figure 5.3 The FLS area for the linear fuzzy controller.

Controller Gain

The gain in a fuzzy controller is decided by the derivative of the FLS area. The slope for each input signal's direction will therefore decide how much the gain is for each input. An example of a PI-controller's FLS, i.e. a linear fuzzy controller, has been given in Figure 5.3.

For our imitation of the PI-controller we have two inputs, the error and the integral of the error, and one output, the control signal. The structure of such a controller can be seen in Figure 5.4.

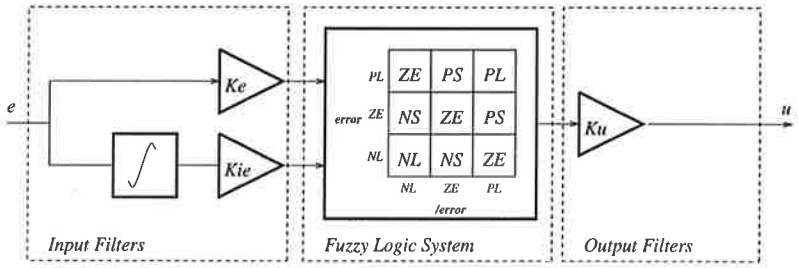


Figure 5.4 An example of the block structure of a PI-controller on position form.

Hence the design parameters to choose are:

- Normalization gain for the error, K_{Ne}.
- Normalization gain for the integral of the error, K_{Nie}.
- Denormalization gain for the output from the FLS, K_{Du}.

Since our aim is to imitate the existing controller from TA with proportional gain= 2 and integral time= $10s$, we would like the proportional gain, K_p, and the integral gain, K_i, to be

$$K_p = 2$$
$$K_i = 0.2$$

Further we would like to keep the signals in the FLS within the range of $[-1, 1]$.

Since the set point normally will be around $50°C$, the error will most unlikely exceed $50°C$. Thus a good choice of normalization gain for the error, K_{Ne}, could be

$$K_{Ne} = 0.02$$

By making the FLS area *symmetric* in the terms of equal derivatives for the two inputs, i.e. use an FLS like the one in Figure 5.3, we will also have equal gain for the inputs. In Figure 5.3 the FLS gain, K_{FLS}, is measured directly in the figure and found to be

$$K_{input1} = K_{input2} \approx 0.25$$

If we would like

$$\frac{K_p}{K_i} = 10$$

as in the original controller, we have to choose

$$K_{Nie} = 0.002.$$

Remaining to choose is only the denormalization gain, K_D. Regarding that we desire $K_p = 2$ and $K_i = 0.2$ the equation

$$K_{tot} = K_N K_{FLS} K_D$$

gives us $K_D = 400$.

Instead of varying the input gains it is of course also possible to make the FLS area non-symmetric and increase or decrease the slope for either of the two signals.

When adjusting the gains one have to make sure that the signals keep within the area of the FLS. An easy check will show that the error has to be far from zero for a rather long time before the integral of the error exceeds the FLS limits.

Position or Incremental Form?

Since it is possible to filter the inputs and the outputs in different ways we are able to create different kind of controller structures performing the same kind of action.

The controller *we* would like to derive is a PI controller. Hence we can choose either a design on *position form* , as discussed in the section above, or, we can choose to implement the controller on an *incremental form.* If we choose the latter design we have a derivative filter as one input filter. As an output filter we choose an integrator. The difference between the two designs is shown in Figure 5.5.

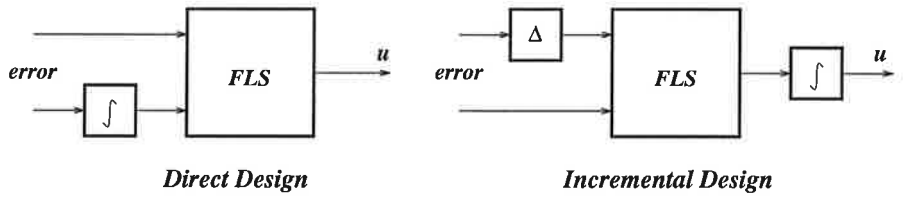


Figure 5.5 The difference between a direct *(left)* and an incremental design *(right).*

It has been reported [Ericsson and Östberg, 1993] easier to derive a fuzzy controller from an incremental design rather than a direct design. It is rather easy to understand that it is hard to grasp how to set the output when the *error integral* is high or low. It is probably a lot easier to understand how to increase or decrease the output when the *error derivative* or the error is high or low. This has also been verified by my own experiments and therefore I will only use the incremental design in the rest of the thesis.

5.2 A Controller State-Space Approach

By looking at the controller's state-space plane, i.e. the area of the FLS, while considering a step-response it is possible to get an idea of the system's behavior. See Figure 5.6.

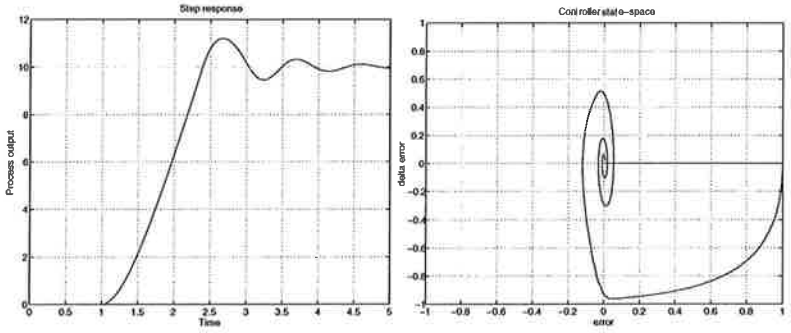


Figure 5.6 Step response *(left)* and the controller phase-plane *(right)*.

There has been suggested [Driankov and M.Reinfrank, 1993] a rule map (Figure 5.7) where different regions of the controller phase-plane corresponds to different modes of control.

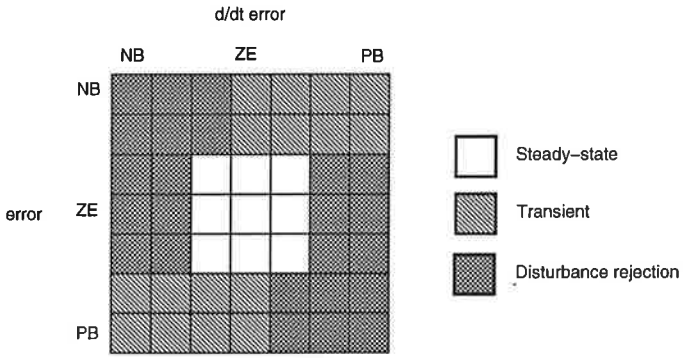


Figure 5.7 The figure is illustrating the FLS area and how it can be divided into different regions of modes.

Starting out with a linear PI-controller, as in the beginning of this section, it is then possible to tune the different areas of the controller phase-plane in order to affect different modes of the controller.

For instance, if we would like to increase the gain for disturbances we simply increase the output for this area (according to the rule map in Figure 5.7) in order to achieve a greater derivative (=gain) there. An example of a simulation with a negative and a positive load disturbance and a transient can be seen in Figure 5.8.

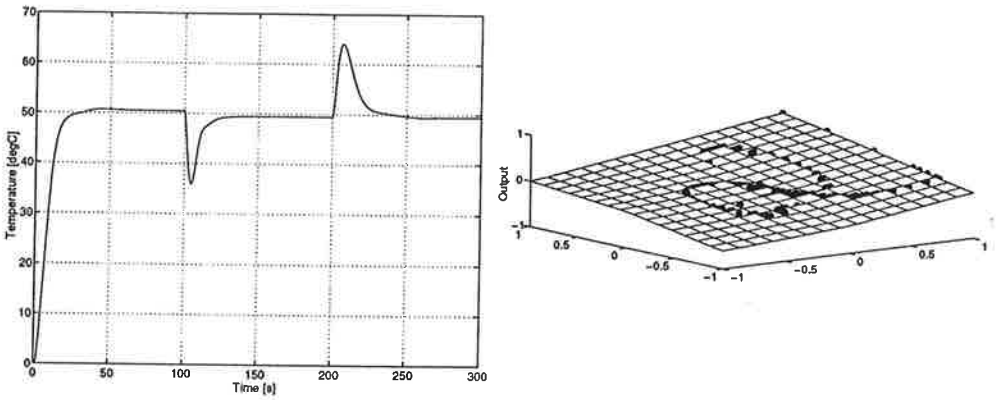


Figure 5.8 *Right:* An example of a simulation with a negative and a positive load disturbance and a transient put on the system. The dots mark the sample moments. The sample time is $0.5s$. The simulation starts in the corner $(1,1,1)$ where the initial transient can be followed to the stable state $(0,0,0)$. The two dotted ellipses are the negative and the positive load disturbances. When the system is in steady-state the error and the error derivative will be zero and hence the sample moments will be dotted in the middle of the FLS area. *Left:* The corresponding output-time diagram.

5.3 The Extended PI-Controller

In order to improve the controller we have to tune the parameters of the PI-controller. The rest of the original controller is kept as it is, i.e. we just replace the PI-controller block with a fuzzy controller. Both the variable deadzone block and the limiter block are left intact.

Let us say that we have a load disturbance of 25% of the maximum secondary flow. Further we make a load disturbance step at $t = 100s$ to 75%. This causes a drastic change of the working point but is still a realistic change of the secondary flow. A load disturbance step response for the existing controller as described above is shown in Figure 5.9.

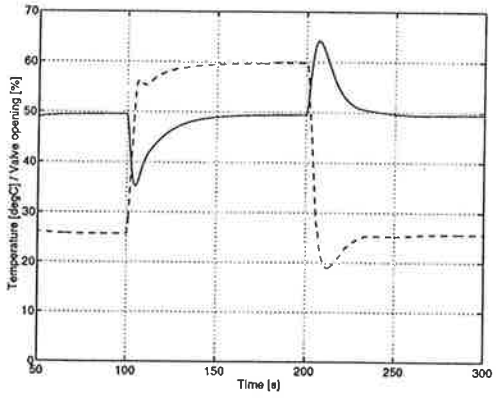


Figure 5.9 A load step response from 25% to 75% at time $t = 100s$ and back to 25% at time $t = 200s$ made with the existing controller. The set point is set to $50°C$. The solid line shows the secondary temperature and the dashed line shows the control signal. Note that this controller has a deadzone of $0.5°C$ which causes the static error that can be observed.

What is desirable to improve?

The question is highly relevant and requires some thought before it is answered.

Is the over-/undershoot too big? It is required in [Finska Värmeverksföreningen, 1983] that the secondary temperature must not deviate more than $15°C$ from the reference value when a change of the load is made with 50%. The existing controller manages just to fulfill this criteria. Hence we would like to decrease the over-/undershoots.

Further it is required that the temperature must not deviate more than $2°C$ from the reference value 120 seconds after the step is made. This criteria is well fulfilled but it might be interesting to see if it is possible to decrease this time.

Using a Fuzzy Controller to Improve Control Action

If we would like to speed up the load step response, we will soon notice that the constraining factor is the actuator which in the existing controller already moves as fast as it can when the error derivative is negative, i.e. in the first part of the load disturbance step. Thus we have to concentrate on improvement of the step when the error derivative is positive. By increasing K_p and K_i in the fuzzy controller we notice that it is possible to decrease the time when the error derivative is positive to a certain degree. The increment of these gains has been made by increasing the normalization gains. What happens is of course that we make the system less stable by increasing K_i, i.e. the phase margin is decreasing, and compensate a little when increasing K_p. (Think of a Bode diagram.) The step response for the system with increased gains is shown in Figure 5.10.

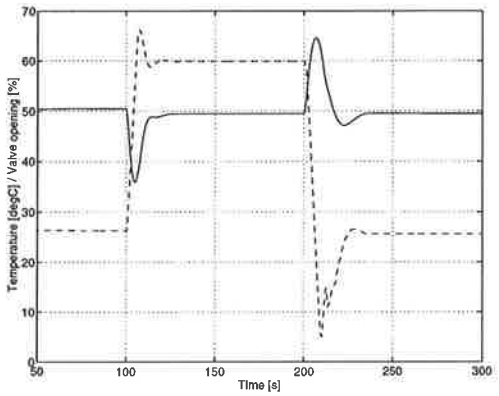


Figure 5.10 A load step response from 25% to 75% at time $t = 100s$ and back to 25% at time $t = 200s$ made with increased gains. The control signal's bad behavior at time $t \approx 215s$ is a result of the big deadzone when the control signal is low.

The problem with this is that the system does not behave in the same way when we set the load level back to 25% again. The secondary temperature tends to make a slight undershoot as shown in Figure 5.10. Thus it would be desirable to increase the gain for increasing load disturbances but keep the gain low for decreasing load disturbances. By looking in the phase plane of the controller as in the Figure 5.8 one can notice that the control signal for a load disturbance appears in different areas depending whether the load step is positive or negative. Thus it is easy to increase the gain of the controller for the area where the error is positive and keep the gain low for the area where the error is negative. See Figure 5.11.

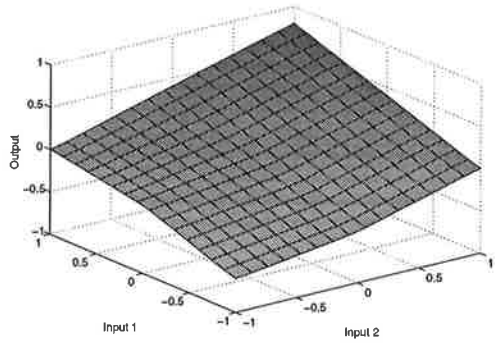


Figure 5.11 An example where the FLS area is slightly "bent" in order to increase the gain for different modes of the controller.

In this way we have derived a controller which behaves well for steps between 25% and 75% of the maximum load.

The result of such a controller's load step response is shown in Figure 5.12.

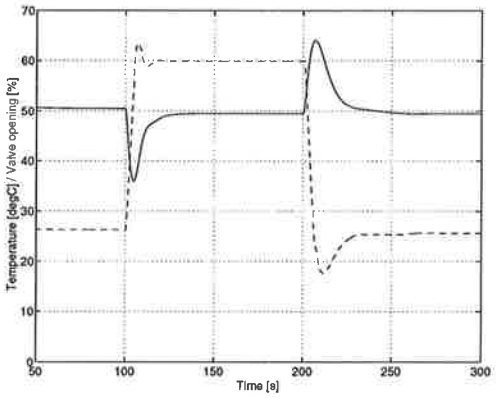


Figure 5.12 A load step response from 25% to 75% at time $t = 100s$ and back to 25% at time $t = 200s$ made with the nonlinear controller.

Since the aim is to derive a controller which work well for all kinds of load disturbances we have to investigate the behavior of our controller for different load levels. We will soon find out that it is impossible to optimize *one* controller for all kinds of working points.

As we have seen in section 2.2 the system is non-linear for different load disturbances and thus it might be a good idea to include either the control signal or the load level, i.e. derive a fuzzy controller with 3 input signals consisting of the error, the derivative error and either the control signal or the load level.

5.4 A Fuzzy Controller with 3 Input Signals

We would like to include the load level as a third input signal to the fuzzy controller. Since this signal is hard and expensive to measure the second best we can do is to measure the control signal which gives us an approximation of what the load level is. By including the control signal in the controller it is possible to create a controller with different gains for different load levels.

Assume that the existing controller is an optimal controller for a load of 50%. Let us approximate the system with a first order system as

$$G(s) = \frac{K}{1+Ts} \tag{5.1}$$

with the gain, K, and the time constant, T, taken from the measurements in Section 2.2. In other words, we approximate the process, including the valve, with the transfer function 5.1 with different gain and time constant for each load level. In this way we can derive a controller for each working point with the same characteristics as the existing controller. This model will only be valid for load disturbances exceeding the valve discontinuity, earlier discussed in section 3.2, and hence we can only design a controller in this way for loads exceeding this critical load. The valve discontinuity starts when the load disturbance is about 10%.

This will result in a controller which has the form of the controller in Figure 5.13.

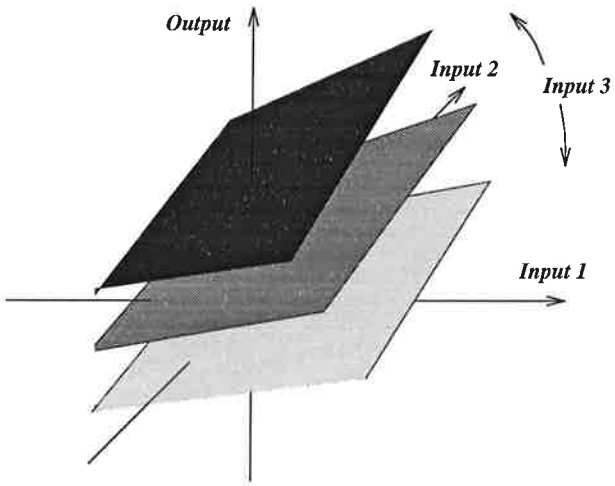


Figure 5.13 The phase plane of a fuzzy controller with 3 input signals.

When the simulations with this kind of controller was done, the Fuzzy-CODE did not support a three input fuzzy controller. In order to simulate such a system we therefore had to make a special solution.

Hence the three input controller that has been used in this thesis has been made so that the third input is put into two different look-up-tables and further connected to the normalization gains of the two other inputs, i.e. the third input is controlling the two other input's normalization gains. This will very much work like a fuzzy controller with three inputs.

Note that this kind of fuzzy controller is very similar to an ordinary controller with gain-scheduling.

After tuning, two different kinds of load disturbance step responses are compared with the original controller, shown in Figure 5.14 on the next page.

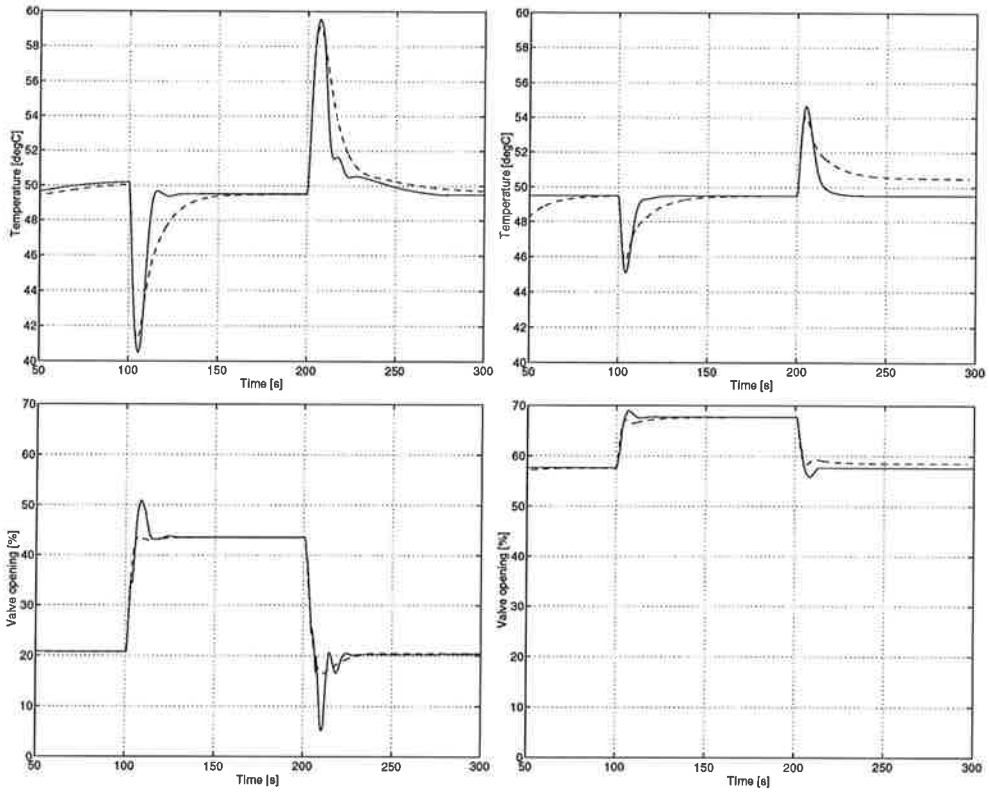


Figure 5.14 Comparison of the existing controller (dashed line) and the 3 input controller (solid line). ***Left upper:*** A load step response from 20% to 45% at time $t = 100s$ and back to 20% at time $t = 200s$. ***Right upper:*** A load step response from 70% to 95%. The lower diagrams shows the corresponding control signals. The bad behavior of the control signal for the three-input-controller for low load is probably due to the big transient of the control signal.

If it would be possible to measure the load level we could derive a controller similar to the one described above. The result would be slightly improved since the load level is more stable than the control signal and further it gives us correct information immediately after the step is made.

A similar comparison to the one in Figure 5.14 is done in Figure 5.15 between the controller with the control signal as the third input and the controller with the load level as an input.

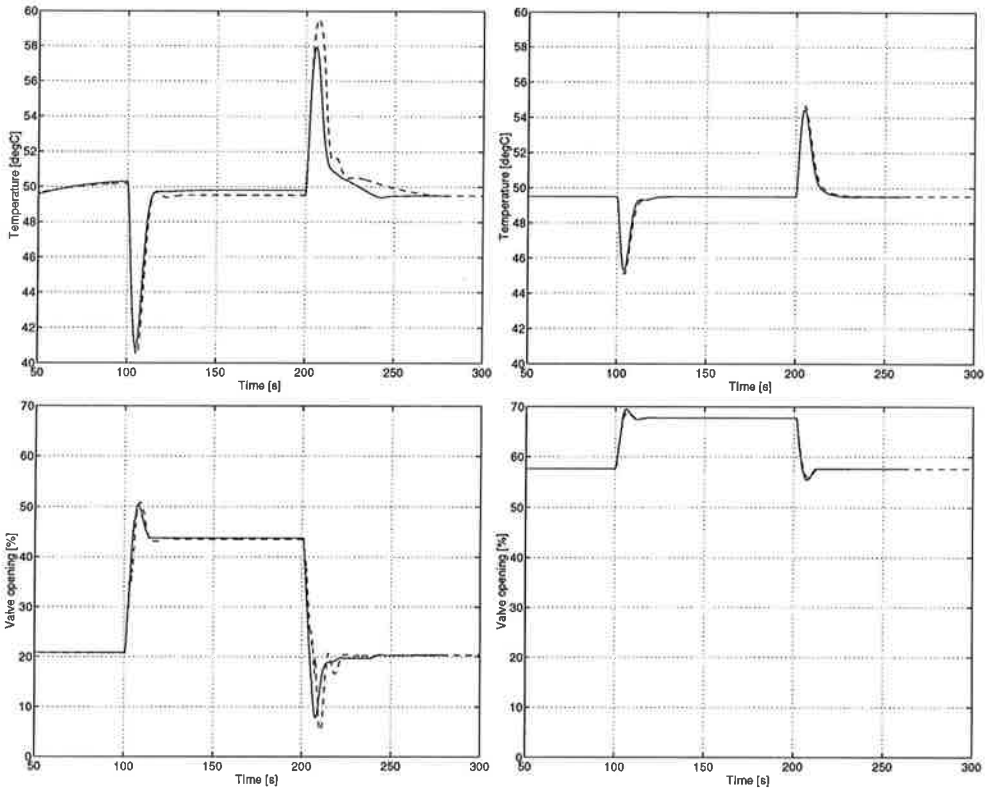


Figure 5.15 Comparison of the controller with the control signal as the third input (dashed) and the controller with the load level as the third input (solid). *Left upper:* A load step response from 20% to 45% at time $t = 100s$ and back to 20% at time $t = 200s$. *Right upper:* A load step response from 70% to 95%. The lower diagrams shows the corresponding control signals.

If we assume that it is possible to measure the load level we can also make a direct feed forward of the load level, i.e. add the required extra control signal to the output of the controller and only compensate for the transients in the actual controller. Still with the load level as a third input to the DFC.

Thus the required amount of control signal is measured for different load disturbances when the system is stable and the transients does not affect the system any more. This information is put into the "feed forward box". This box could either be implemented as an extra fuzzy controller with only one input or an ordinary look-up-table. The non-linear mapping for a look-up-table would look like in Figure 5.16.

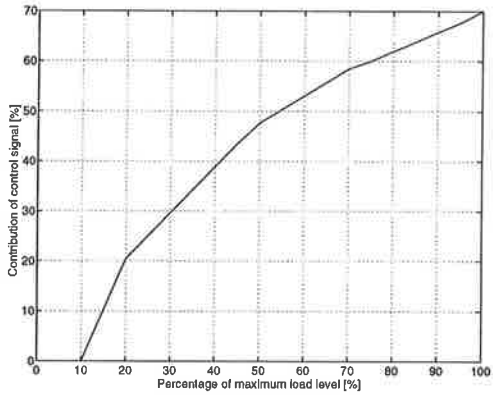


Figure 5.16 The non-linear mapping for a look-up-table.

More about this kind of fuzzy controllers is found in Section 6.2. The block scheme in Figure 5.17 shows how the controller is meant to work.

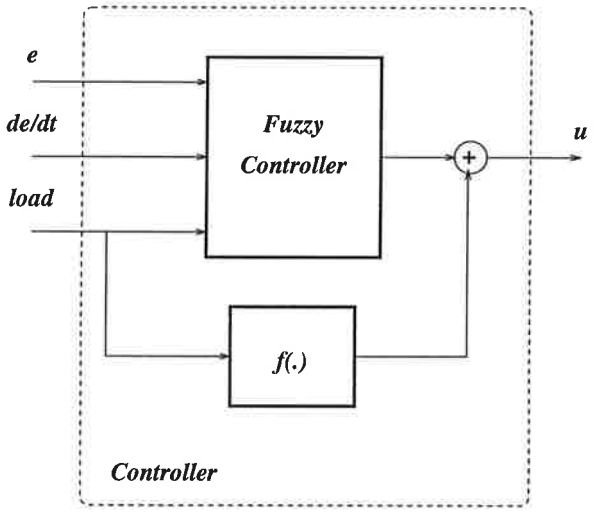


Figure 5.17 A fuzzy controller with 3 input signals where the load is used both in the fuzzy controller and also as a feed forward signal through either a single input fuzzy controller or a look-up-table.

This system is simulated and compared with the 3 input controller with the load level as the third input in Figure 5.18.

We can see that it is possible to reduce the over-/undershoot a great deal with this design. Although these steps are not very big, the same behavior can be observed when increasing the step size.

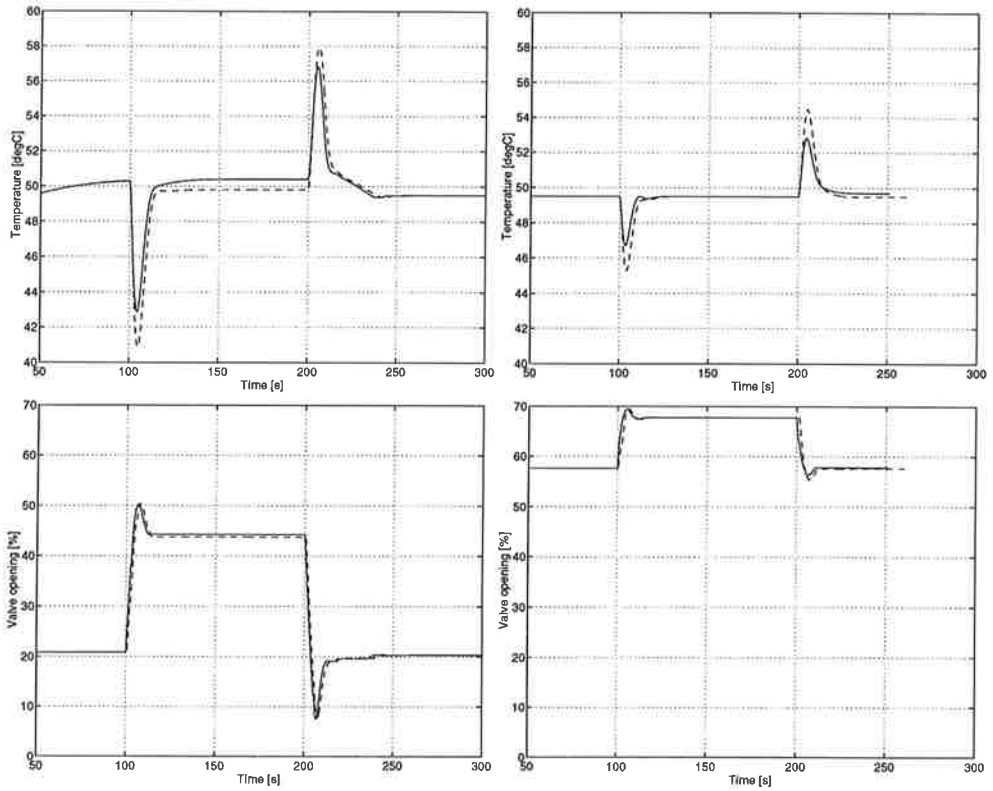


Figure 5.18 Comparison of the load level controller (dashed) and the controller with an extra feed forward of the control signal (solid). *Left upper:* A load step response from 20% to 45% at time $t = 100s$ and back to 20% at time $t = 200s$. *Right upper:* A load step response from 70% to 95%. The lower diagrams shows the corresponding control signals.

5.5 Summary

In this chapter we have first taken a look at the system with an existing controller developed by TA. In order to speed up the control action we tried to tune the proportional and the integral gain. We then noticed that we would like higher gain for positive load changes than for negative. By exchanging this controller with a Direct Fuzzy Controller, DFC, we were able to increase the gain differently for different types of load changes. This was however only possible to do for a certain load level. When changing the load level this is not a good solution.

In order to improve the control action for different load levels we would like to include the load level as a *third* input. Since the load level sometimes can be hard to measure we tried to include the controller output instead. This gave us slightly improved control action but unfortunately a bit unstable result in the transients.

Assuming that it would be possible to measure the load level we included this instead. This gave us a more stable response.

In our last controller design we added an extra feed forward box which performed a direct mapping of the load level to a pre-set output control level. The actual controller only compensated for the transient errors in this case and we achieved a rather good fuzzy controller.

A general advantage of the DFC is that it is easy to understand how the rules work. Further it is also possible to design a controller exactly as you would like it. The DFC is probably most useful for processes where an

experienced operator is available. Together with a control engineer they are likely to derive a well working rule base for a DFC. I find it hard to believe that it will be easy or even possible to derive a well performing controller from scratch for an unknown process. I would rather say that to succeed in deriving a good DFC one would have to spend a lot of time in order to improve action of an ordinary, linear controller. The advantage of fuzzy control is that it is always possible to derive a linear controller and as the knowledge grow with the use of the process it then might be possible to tune the linear controller to a better, non-linear one.

6. Supervisory Fuzzy Control

A supervisory fuzzy controller will, as previously discussed, collect data from the environment in order to provide the ordinary controller with correct parameter values. This means great freedom of design but also harder work to tune the controller for all different kinds of boundary conditions. Again, this type of controller has been developed from the fact that a manual operator knows pretty well how to tune the ordinary controller in order to obtain good control action.

In the end of this Chapter we will also take a look at a way of creating an arbitrary non-linear mapping and especially the use of building static inverses of a transfer function.

6.1 Designing an SFC

In our case we have several kinds of slowly variating parameters such as the ones discussed in section 2.2. By simulating the system with different (realistic) setups of parameters one will soon notice that there are parameters that affect the performance of the system more than others. These parameters are for instance the *primary water pressure* and the *primary water temperature*. These parameters are claimed to vary between 150–500kPa and 65–100°C respectively. By tuning the ordinary controller for extreme setups of these parameters and meanwhile let the load disturbance vary from 0–100% of the maximum load it is possible to derive rules for the SFC.

The block scheme for an SFC system is shown in Figure 6.1.

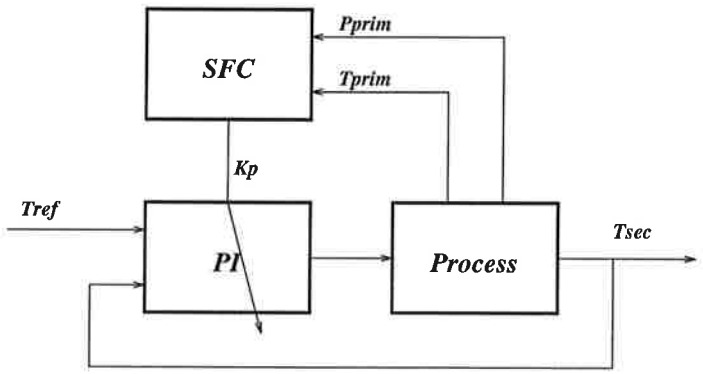


Figure 6.1 The structure of an SFC system.

One setup of rules which has been proved to work well is shown in the Figure 6.2.

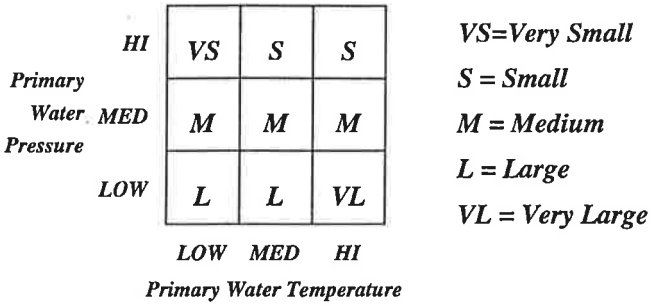


Figure 6.2 The rule base for the derived SFC with the notation as to the left. The two inputs are Primary water pressure and the Primary water temperature. The output from the FLS is the proportional gain which is varying between 1.5 to 2.5.

The inputs of the SFC in Figure 6.2 are the primary water temperature, T_p, and the primary water pressure, p_p, provided that these quantities can be measured. The SFC output will then determine the controller's proportional gain, K_p. The controller gain has to be decreased for high primary water pressures because the valve tends to oscillate if the gain is too high. Likewise, the gain has to be increased for low primary water pressures. The primary water temperature does not affect the system as much as the pressure and hence it does not affect the fuzzy rules as much either.

The performance of this controller is good compared to the ordinary controller. A good way of measuring the performance of a system is to see how a so called *loss function* is growing as time is increasing. The loss function is described by

$$J = \int_0^t \sqrt{(y_{ref} - y)^2}\, dt$$

i.e. the loss function describes how much the secondary temperature deviates from the reference value under a certain time, t. The loss function makes it easy to compare the performance of two systems. The original system from TA is compared with the SFC system in Figure 6.3 where the loss functions for both systems are shown.

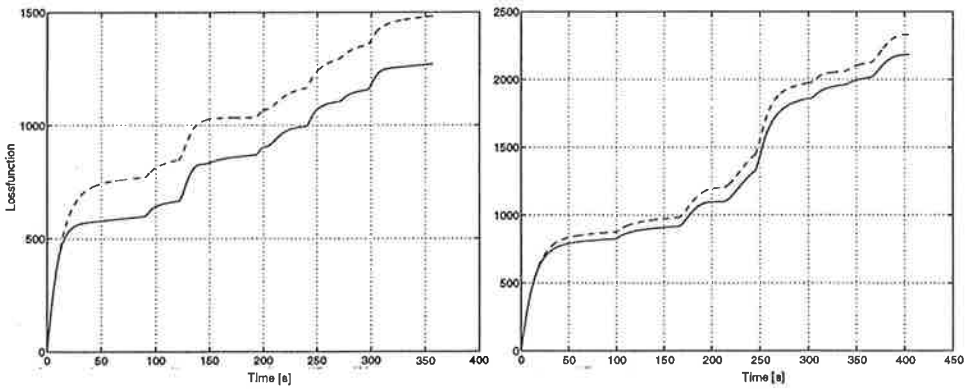


Figure 6.3 The loss function for a winter case (*left*) and a summer case (*right*). The original system's loss function is marked with a dashed line and the SFC system's with a solid. The load profile is a random function with "not to large" load changes. Note that the time scale is in *seconds* which rather should have been *minutes* or even *hours*. This is unfortunately due to the long simulation times which makes it a time consuming job to get a more correct picture of the SFC system.

Note that no consideration is taken to the load level in this system, i.e. it is a *two-input controller*. Since high gain causes problems when the secondary flow is low (actually when u <valve opening in h_0, see Section 3.2) this is a big limitation when tuning the SFC. This system could probably be improved by using a controller with the load level as a third input signal.

6.2 Non-linear Functions with Fuzzy Control

In Section 3.2 we discussed a little about compensation of non-linear transfer functions, such as the example with the valve and the heat exchanger. In this Section we will take a closer look at this problem and how it might be possible to create a non-linear function with fuzzy control.

In our case it could be interesting to see if we can create the non-linear function in our feed forward box with fuzzy control, earlier discussed in Section 5.4. However we will instead give an example of how to create an inverse function of the non-linear valve in order to linearize this. Please note that this is only an illustration of the fuzzy method and it is generally *not* a good idea to linearize the valve in a domestic hot water system.

Example: Linearization of a valve

The valve has only one input and one output and hence we will only need a fuzzy controller with the same number of inputs and outputs.

Our valve's characteristics look like in Figure 6.4 (*left*). Our goal is to find an inverse of this curve with fuzzy control. Since the inverse function should perform a direct mapping of the input to the output, we must not introduce any dynamics to the controller. Thus we only need to normalize the input signal, find a normalized inverse function of the valve's characteristics and finally denormalize the output.

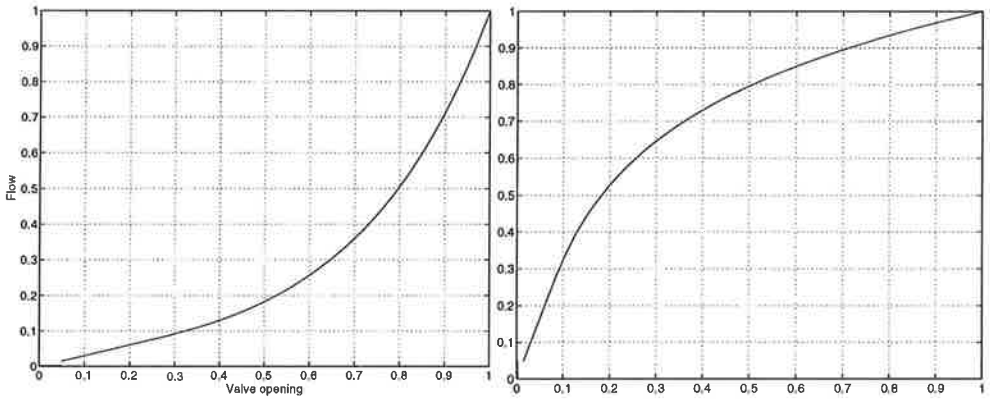


Figure 6.4 The valve's characteristics (*left*) and its inverse (*right*).

We start with finding the fuzzy inverse. By pointwise taking the inverse of the valve function we will find the inverse function as in Figure 6.4, right.

Next we have to decide the number of input and output sets. We will here start with seven input sets and equal number of output sets. By first making a linear fuzzy controller (Figure 6.5 left), and then "drag" the sets so that the modal points will cross the inverse function, we have created a first approximation. (Figure 6.5 right)

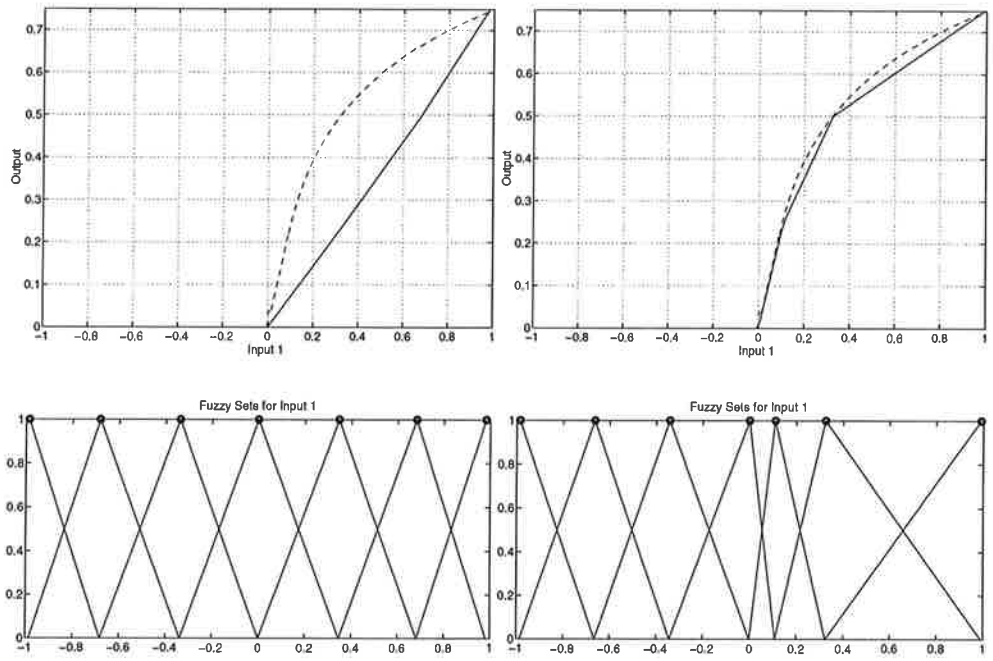


Figure 6.5 The first approximation (*upper left*) and the second approximation (*upper right*) of the inverse. The solid line shows the approximation and the dashed shows the real inverse. The lower figures shows the input fuzzy sets respectively.

The output sets we are using in the FuzzyCODE have a rectangular shape. In [Johansson, 1993] it is shown how the output sets effect the FLS output. If the *area* of the sets is changed, the FLS derivative will be changed in this point. More specifically the derivative of the FLS will change for a certain point/area if the difference between the areas of two neighboring sets is else than zero. An illustration of this phenomena is shown in Figure 6.6.

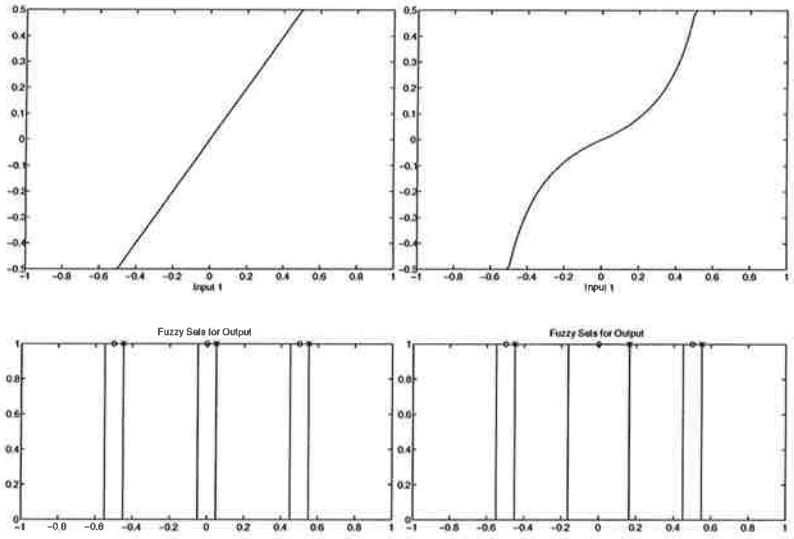


Figure 6.6 How the output fuzzy sets affects the FLS output in a single input-output case. To the left, the normal case for a linear P-controller. To the right a "soft" deadzone is included by changing the output fuzzy sets.

This can be used in our attempt to derive an inverse of the valve's characteristics. By changing the area of the output sets right we are able to "bend" the linear parts between the modal points in Figure 6.5 (right). In this way we can achieve a rather good approximation of the inverse, see Figure 6.7.

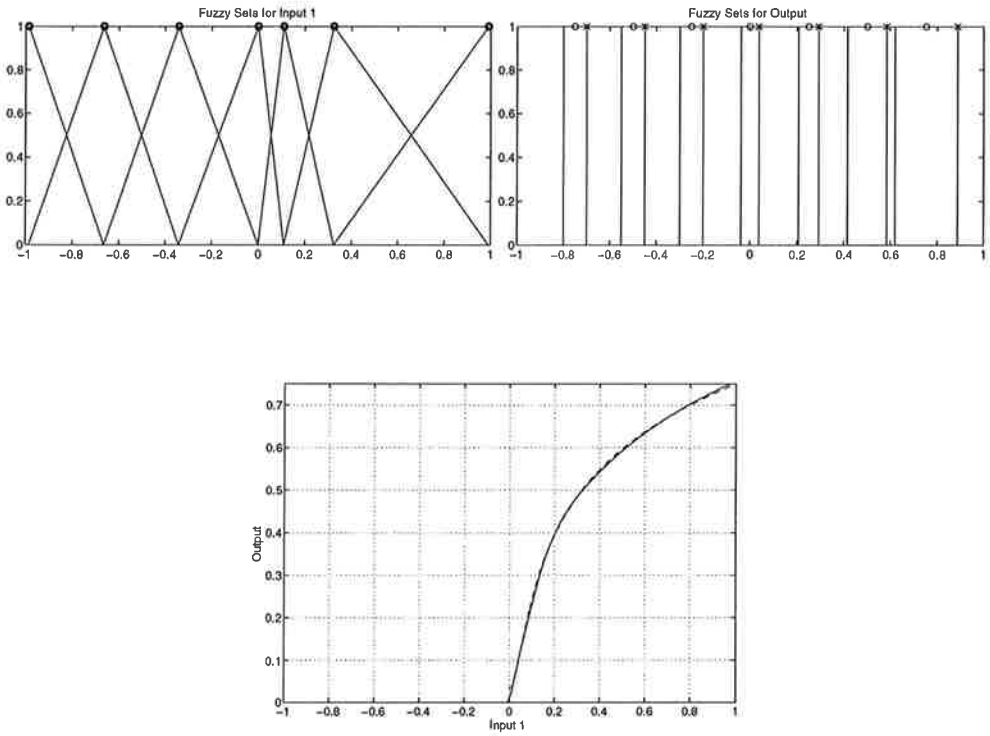


Figure 6.7 The final result of the fuzzy inverse and its input and output sets.

Although this works perfectly well when simulating the system, one of the benefits of having the equal percentage valve is to compensate for the heat exchanger's characteristics. The compensation we derived above will therefore result in an even more non-linear system and obviously it will not improve the design.

We could however replace the feed forward box, as discussed in the beginning of this Section, with this type of controller, i.e. used as a non-linear mapping of a desired input-output relationship.

The valve linearization can be seen as an example of how one can get rid of a static non-linearity in a process.

6.3 Summary

The use of SFC is probably much easier to grasp than the DFC. Since it is normally well known how you would like the controller to behave for different working conditions it is easy to just apply these rules on the SFC. Since the ordinary controller usually is a compromise between different working conditions it would be surprising if the control performance did not improve.

7. Implementation Issues and Coding

The aim in this work was to implement the fuzzy controller in a real-time language in order to run it against a real process. However, the time did not allow us to do so, we had to compromise and run the controller against a real-*time* simulated process. This chapter will only give a briefing of how to implement real-time processes and controllers. More to read is found in [Nielsen, 1991].

7.1 Real-time Systems

A real-time system is generally characterized by the introduction of *sampling* of the signals and in some cases discretization of control algorithms, filters etc. The fact of having a limited amount of time to deal with can cause some problems if not caution is taken.

In this thesis the process is rather slow ($\omega_{max} \approx 1Hz$) and hence we do not have to worry about a too fast process. Since all the filters and much of the controller already are discretisized in SIMULINK, it is only a small part of the code that has to be discretisized and translated into the real-time language.

7.2 Modula-2 Coding

Modula-2 is a real-time language based on the idea that a system is divided into smaller subsystems, called *modules*. Each module has a so called *definition module* where everything to be seen from "outside" is presented, i.e. it will tell the other modules what this module is *exporting* in terms of procedures, types etc.

The Modula-2 language supports the idea of *parallel processes*. The processes are separated from each other and have different priorities. In this way it is possible for each process to get as much processor time as required until a new process with higher priority interrupts.

The controller in this thesis is implemented with the help of a real-time kernel, used in a course in real-time programming. An existing Modula-2 program consisting of a PID controller, operator interface and graphical data presentation has been used. The PID control part has then been replaced by a fuzzy control part. Thus in Appendix C only the control part of the code is found and the rest of the code is omitted.

7.3 Real-Time Simulation

The process of this thesis is run in real-time *SIMNON*, described in Appendix A. The setup for the simulation is illustrated in Figure 7.1.

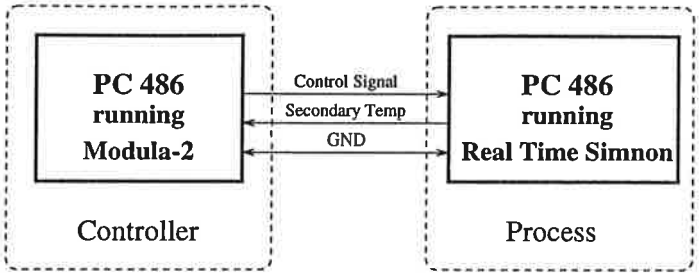


Figure 7.1 An overview of the experimental setup for real-time simulation.

Three cables were connected between two PC's 486:

- The process output signal, i.e. the measured secondary temperature,
- the controller output signal and
- one cable for ground.

The process was sampled with a rate of $20Hz$ and the controller with $50Hz$. A screen dump from the real time simulation and its graphical environment is shown in Figure 7.2.

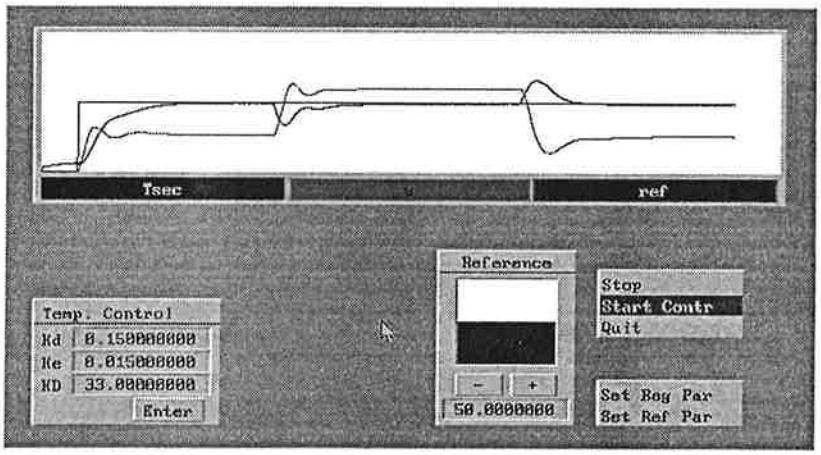


Figure 7.2 A screen dump from the real time simulation. The three plotted values are: *the reference value, the actual value* and *the control signal.*

8. Conclusions

Domestic hot water control is a highly non-linear process which is claimed hard to control due to its non-linear characteristics. Using fuzzy control is one way of handling such processes. In this thesis we have seen that it is possible to achieve a rather good fuzzy controller for this kind of systems.

In Chapter 2 we looked at the problem and the PI-controller currently used by TA. Further in Chapter 3 we looked at some of the mathematical models of the process that we later used for simulation.

Chapter 4 described some basic fuzzy theory and how a fuzzy logic system and a fuzzy controller works.

In Chapter 5 we started to design a linear fuzzy controller in order to imitate the original PI-controller. In order to improve the control action we tuned the fuzzy controller so it became non-linear. Although this worked fine for a certain load level we were not able to derive *one* controller that worked fine for *all* load levels. In order to do so we included the load level as an input to the controller. This worked basically as gain scheduling. Further improvements were shown when we used feed forward of the load level and only let the controller take care of the transients when changing the load level.

In Chapter 6 we used the fuzzy controller as a supervisory controller to an ordinary PI-controller in order to take care of slowly varying parameters such as seasonal variations. In the end of this chapter we saw how it is possible to create an almost arbitrary non-linear mapping with fuzzy control. This could for instance be used as a feed forward block as described in Chapter 5.

Finally we implemented the controller in a real time language, described in Chapter 7 and Appendix C.

If more time was given it would have been interesting to examine a combined DFC and a SFC although this requires at least two fuzzy controllers. The tuning of both the DFC and the SFC is done very rough. Since it is a time consuming work to optimize a controller with so many parameters I have stopped designing the regulator when the result was "satisfactory", which of course is possible to improve.

This thesis has pointed out some of the possibilities and difficulties when using fuzzy control in domestic hot water systems. We have seen that it is possible to continue this work and hopefully this is a beginning of some more detailed experiments and simulations.

9. References

ÅSTRÖM, K. J. and B. WITTENMARK (1989): *Adaptive Control.* Addison-Wesley, Reading, Massachusetts.

ÅSTRÖM, K. J. and B. WITTENMARK (1990): *Computer Controlled Systems—Theory and Design.* Prentice-Hall, Englewood Cliffs, New Jersey, second edition.

DRIANKOV, D., H. H. and M.REINFRANK (1993): *An Introduction to Fuzzy Control.* Springer Verlag, Berlin Heidelberg.

ERICSSON, P. and P. ÖSTBERG (1993): "Dynamisk provning av värmeväxlarsystem.". Master's thesis, Lunds Institute of Technology.

FINSKA VÄRMEVERKSFÖRENINGEN (1983): "Normer för reglerutrustning, granskningsverksamhet och rekommendationskopplingar." Technical Report.

GLAD, T. and L. LJUNG (1989): *Reglerteknik.* Studentlitteratur, second edition.

HALLING, L. (1991): "Utvärdering av värmeväxlarmodell." Technical Report, Tour & Andersson AB, Malmö.

INFORM SOFTWARE CORP. (1993): *FuzzyTECH 3.0, Manual and Reference book*, explorer edition.

JOHANSSON, M. (1993): "Interpolation methods in fuzzy control.". Master's thesis, Lunds Institute of Technology.

JOHANSSON, M. (1994): *FuzzyCODE, A Fuzzy COntroller Design Environment.*

JOHANSSON, R. (1992): *Process Identifiering.* Sigma Tryck.

LING, K. and A. DEXTER (1994): "Expert control of air-conditioning plant." *Automatica*, **30:5**, pp. 761–773.

THE MATHWORKS, INC. (1992a): *Matlab Reference Guide.*

THE MATHWORKS, INC. (1992b): *Simulink Reference Guide.*

NESLER, C. (1986): "Adaptive control of thermal processes in buildings." *IEEE control systems magazine*, August.

NIELSEN, L. (1991): "Computer implementation of control systems." Report TFRT-7476, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

PALMERTZ, H. (1993): *Ventilhandbok, Teori och praktik.* Berlings.

RADKE, F. and R. ISERMANN (1987): "A parameter-adaptive PID-controller with stepwise parameter optimasation." *Automatica*, **23**, pp. 449–457.

SSPA SYSTEMS (1990): *Simnon, User's Guide for MS-DOS Computers*, 2nd edition.

WATCOM PUBLICATIONS LTD. (1988): *Maple Reference Manual*, 5th edition.

WOLLERSTRAND, J. (1993): "Parameteridentifiering i en dynamisk modell av genomströmningsberedare för tappvarmvatten." Technical Report ISRN LUTFD2/TFRT--LUTMDN/TMVK–3156–SE--SE, Värme och kraftteknik, LTH.

A. Simnon Code

SIMNON is a program that is used for simulation of dynamical systems. It is developed at the Department of Automatic Control at Institute of Technology, Lunds University.

The program supports simulation of systems composed of many subsystems. This makes it a useful tool when dealing with control problems where division into smaller subsystems is a common way of handling a more complex model of a system.

The existing controller from TA is implemented in SIMNON although I have used SIMULINK for my simulations. Thus the SIMNON model is the model which is considered to be the *original* one in this thesis.

The system is composed of four different subsystems excluding the controller and one *connecting system* which can be considered as the main program in SIMNON. The four subsystems are:

- The valve
- The heat exchanger
- The recirculation circuit
- The sensor

The code for these systems, beginning with the connecting system, is presented on the following pages.

```
CONNECTING SYSTEM con

" Heat water control (one step heat exchanger).
"
" File name : convvx.t
"
" Author: Lars Halling /920709
"
" Note! The actuator dynamics is not included in the model. It is
" assumed that the actuator mainly introduces a rate limit in the
" control signal. To model this, the controller must be equipped
" with a rate limit corresponding to the actuator stroke time.


TIME t

yr[reg] = yr " set point
u[sensor] = T4[vvx] " domestic hot water temperature
y[reg] = y[sensor] " sensor signal
u[valve]= u[reg] " control signal
m1[vvx] = q[valve] " primary flow

load = if t<tstep1 or t>tstep2 then lowlev*100 else (lowlev+amp)*100
m2a = m2max*load/100

m2c = if m2a<clim*m2max then m2max*circlev else 0  " circulation flow

m2[vvx] = m2a+m2c

T1[vvx] = Tp
u[recirc] = T4[vvx]
T3[vvx] = (Tsek*m2a + y[recirc]*m2c)/(m2a+m2c)


" parameters ------------------------------------------------------------

lowlev:0.25 " low level load disturbance (0=0% of max, 1=100% of max)
m2max:0.2 " max domestic water flow (l/s)
circlev:0.1 " ratio (recirculated flow)/(max domestic water flow)
amp:0.5  " load disturbance
yr:50 " set point
Tp:65 " primary input temperature [degC]
Tsek:5 " secondary input temperature [degC]

tstep1:200
tstep2:350
clim:0.1

END
```

```
CONTINUOUS SYSTEM valve

" Valve model.
"
" Valve authority included in model.
"
" File name: valve.t
"
" Author: Lars Halling /900926
"
" Modification record:
"

INPUT u  " actuator position, 0-100%
OUTPUT q " flow through valve, l/s


Kvmin= Kvmax/s     " minimum Kv-value
h = u/100     " valve position

fi = if h<h0 then fileak else if h<h1 then b*h else s^(h-1)

pn = p/100     " normalized pressure, P in kPa
pnv = pn*1/(1 + (fi)^2*(1/a-1))     " normalized pressure over valve
q = Kvmax*fi*sqrt(pnv)/3.6     " flow through valve, l/s


" parameters ----------------------------------------------------

"Kvmax: 2.5        " maximum Kv-valve
Kvmax: 1.0

s: 30        " control range
p: 500        " total pressure over circuit, kPa
a: 0.93         " valve authority
h0: 0.05        "
h1: 0.294        " limit for linear characteristics = 1/ln(s)
b: 0.308        " slope linear part ln(s)*s^(1/ln(s)-1)
      " Note that the slope of the linear part is calculated
      " under the assumptions that it crosses origin and that
      " the valve characteristics and its derivative is
      " continuous
fileak: 0.001        " fi leak value

END
```

```
CONTINUOUS SYSTEM vvx

" Heat exchanger model with 5 compartments on each side.
" Heat transfer coefficient varies with the flow.
" Component data for arbitrary heat exchanger may be used.
"
" File name: vvx6.t
" Author: Lars Halling/910701
"
" The model is documented in the report:
"
" 'Utvrdering av vrmevxlarmodell' by Lars Halling.
"

INPUT m1    " Primary mass flow [kg/s]
INPUT m2    " Secondary mass flow [kg/s]
INPUT T1    " Primary inlet temp [degC]
INPUT T3    " Secondary inlet temp [degC]

OUTPUT T2    " Primary outlet temp [degC]
OUTPUT T4    " Secondary outlet temp [degC]

STATE x11 x12 x13 x14 x15 " Compartment temperatures [degC]
STATE x21 x22 x23 x24 x25

DER dx11 dx12 dx13 dx14 dx15
DER dx21 dx22 dx23 dx24 dx25

TIME t

INITIAL
  dC1 = C1/5    " compartment heat capacities
  dC2 = C2/5
SORT

" dynamics ----------------------------------------------------

Ah1 = k*m1        " heat transfer coefficient primary side
Ah2 = k*m2        " heat transfer coefficient secondary side
Ah = 1/(1/Ah1+ 1/Ah2 + d/(A*lambda))" total heat transfer coefficient
dAh = Ah/5
m1cp = m1*cp1
m2cp = m2*cp2

x10 = T1      " inlet temperatures
x20 = T3

dtm1 = (x10+x11-x25-x24)*0.5 " compartment mean temp diff
dtm2 = (x11+x12-x24-x23)*0.5
dtm3 = (x12+x13-x23-x22)*0.5
dtm4 = (x13+x14-x22-x21)*0.5
```

```
dtm5 = (x14+x15-x21-x20)*0.5

Qh1 = dAh*dtm1   " compartment heat transfer energy flows
Qh2 = dAh*dtm2
Qh3 = dAh*dtm3
Qh4 = dAh*dtm4
Qh5 = dAh*dtm5

dx11 = (m1cp*(x10-x11)-Qh1)/dC1    " primary side
dx12 = (m1cp*(x11-x12)-Qh2)/dC1
dx13 = (m1cp*(x12-x13)-Qh3)/dC1
dx14 = (m1cp*(x13-x14)-Qh4)/dC1
dx15 = (m1cp*(x14-x15)-Qh5)/dC1

dx21 = (m2cp*(x20-x21)+Qh5)/dC2   " secondary side
dx22 = (m2cp*(x21-x22)+Qh4)/dC2
dx23 = (m2cp*(x22-x23)+Qh3)/dC2
dx24 = (m2cp*(x23-x24)+Qh2)/dC2
dx25 = (m2cp*(x24-x25)+Qh1)/dC2

T2 = x15       " outlet temperatures
T4 = x25

" parameters ------------------------------------------------


"C1: 10.7E3   " total primary heat capacity [J/K]
"C2: 10.7E3   " total secondary heat capacity [J/K]
C1: 2.45E3
C2: 2.45E3

"d: 0.0004   " heat exchanger wall thickness [m]
d:0.0005

lambda: 14   " heat conducting coefficient [W/(m*K)]

"A: 1.805   " area [m2]
A:0.5

"k: 17158   " Polynomial fit. See the file fit4.m or fit5.m.
k: 1.27E4

m: 0.65
cp1: 4180   " specific heat primary side [J/Kg*K]
cp2: 4180   " specific heat secondary side [J/Kg*K]

END
```

```
CONTINUOUS SYSTEM recirc
"
" Simple model of thermal dynamics in recirculation loop in
" domestic water process.
" The dynamics is modeled as a first order system with time delay:
" Transfer function:
"
"              K1
"    G(s) = ------ * exp(-s*Td)
"           1+s*Tc
"
" Author: Lars Halling. 910620
"
INPUT  u     " input temperature [degC]
OUTPUT y     " output temperature [degC]
STATE  x
DER    dx
TIME   t

ud = DELAY(u,tdel)
udd = if t<tdel then u else ud
dx = (K1*udd-x)/Tc
y = x

" parameters ------------------------------------------------

K1: 0.9       " filtered term gain
Tc: 20     " time constant [sec]
tdel: 20     " time delay [sec]

END
```

```
CONTINUOUS SYSTEM sensor

" Dynamics of temperature sensor
"
" File name: sensor.t

INPUT u       " temperature of surroundings

STATE x       " measured temperature

OUTPUT y      " measured temperature

DER dx

TIME t

dx=1/ts*(u-x)
y = x

" parameters ------------------------------------------------------------

ts: 1.5       " sensor time constant, s

END
```

B. Using Simulink

SIMULINK is a toolbox working under MATLAB. The package is used for simulations of dynamic systems and has a graphical interface for model development. The program is easy to use but can be a bit slow when simulating larger systems.

The SIMULINK toolbox has been the working environment for this thesis and nearly all simulation has been done here. Although it might be a bit slow it is easy to build up large and very complex systems and still have a nice overview in order to quickly change parameters etc.

The FuzzyCODE toolbox, earlier described in Section 4.5, is relies heavily on the SIMULINK package. Since the fuzzy controllers are developed in the FuzzyCODE environment it is easy to extend or change the system.

First of all the original controller has been directly translated into a SIMULINK model. Some of the subsystems are non standard ones and hence they had to be implemented in another way.

SIMULINK supports a class of custom designed functions, called *S-functions*, where one are allowed to specify such things as initial states and values, output and updating-states-procedures. This makes it possible to implement almost any dynamic system. More to read about this is found in the reference manual for SIMULINK [Mat, 1992b].

The model has been implemented in blocks and the system has the form of the block schemes in Figure B.1.

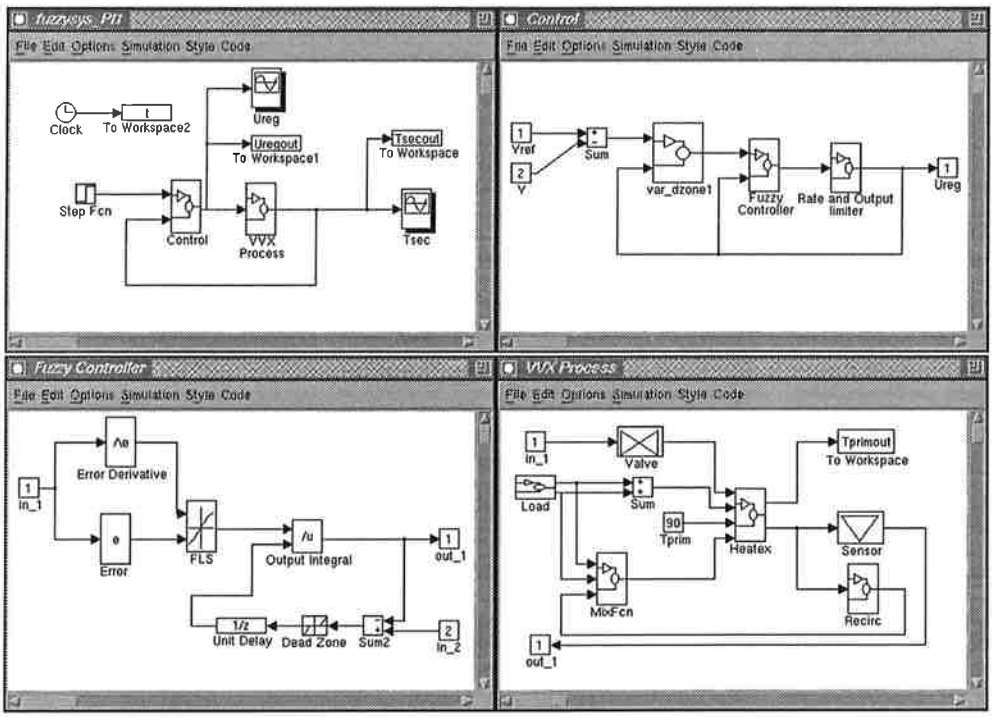


Figure B.1 The main blocks implemented in SIMULINK. From upper left: An overview of the system, the controller, the fuzzy block and the domestic hot water system.

The valve, the variable deadzone and the heat exchanger had to be implemented as S-functions. The code for these blocks is described on the following pages.

```
function [sys,x0] = valve(t,x,u,flag,Kvmax,s,p,a,h0,h1,b,fileak)
% Valve dynamics

%%% Parameters characteristic to the particular valve

 Kvmax= 1;        % maximum Kv-valve
 s= 30;         % control range
 p= 150;         % total pressure over circuit, kPa
 a= 0.93;         % valve authority
 h0= 0.05;        %
 h1= 0.294;        % limit for linear characteristics = 1/ln(s)
 b= 0.308;        % slope linear part ln(s)*s^(1/ln(s)-1)
      % Note that the slope of the linear part is calculated
      % under the assumptions that it crosses origin and that
      % the valve characteristics and its derivative is
      % continuous
 fileak= 0.001;        % fi leak value

if flag == 3,
  % if flag=3, return system output

  Kvmin=Kvmax/s;
  % minimum Kv-value

  h=u/100;
  % valve posistion

  if h<h0,
    fi=fileak;
  elseif h<h1,
    fi=b*h;
  else
    fi=s^(h-1);
  end

  pn = p/100;
  % normalized pressure, P in kPa

  pnv = pn*1/(1+(fi)^2*(1/a-1));
  % normalized pressure over valve

  sys = Kvmax*fi*sqrt(pnv)/3.6;
  % flow through valve, l/s

elseif flag == 0,
  sys = [0,0,1,1,0,1];
  x0 = [];
else
  sys = [];
end
```

```
function [sys,x0] = act_switch(t,x,u,flag,ulow1,ulow2,T1,dzoneh,dzonel)

% The active switch for high and low control action.
% Part of the variable deadzone block implemented by Lars Halling, TA.

if flag == 1,
  if u<ulow1,
    sys=-x/T1;
  elseif u>ulow2,
    sys=(1-x)/T1;
  else
    sys=0;
  end

elseif flag == 3,
  if x>0.5,
    sys=1;
  else
    sys=-1;
  end

elseif flag == 0,
  sys = [1,0,1,1,0,1];
  x0 = [0];

else
  sys = [];

end
```

```
function [sys,x0] = vvx_5(t,x,u,flag,C1,C2,d,lambda,A,k,m,cp1,cp2)

% Heat exchanger model with 5 compartments on each side.
% Heat transfer coefficient varies with the flow.
% Component data for arbitrary heat exchanger.
%
% File name: vvx6.t
% Author: Lars Halling/910701
%
% The model is documented in the report:
%
% 'Utvrdering av vrmevxlarmodell' by Lars Halling.
%
% MATLAB version for SIMULINK created 940321 by Jerker Sjogren.
%


%%% Parameters for vvx_5

 C1= 2450;   % total primary heat capacity [J/K]
 C2= 2450;   % total secondary heat capacity [J/K]
 d= 0.0005;   % heat exchanger wall thickness [m]
 lambda= 14;   % heat conducting coefficient [W/(m*K)]
 A= 0.5;    % area [m2]
 k= 1.27E4;   % Polynomial fit. See the file fit4.m or fit5.m.
 m= 0.65;
 cp1= 4180;   % specific heat primary side [J/Kg*K]
 cp2= 4180;   % specific heat secondary side [J/Kg*K]

 dC1= C1/5;
 dC2= C2/5;


if flag == 1,

  m1=u(1);   % Primary mass flow [kg/s]
  m2=u(2);   % Secondary mass flow [kg/s]
  T1=u(3);   % Primary inlet temp [degC]
  T3=u(4);   % Secondary inlet temp [degC]

  % T2   Primary outlet temp [degC]
  % T4   Secondary outlet temp [degC]

  % STATE x(1) x(2) x(3) x(4) x(5)  Compartment temperatures [degC]
  % STATE x(6) x(7) x(8) x(9) x(10)


  % dynamics ------------------------------------------------------

  Ah1 = k*m1^m;            % heat transfer coefficient primary side
  Ah2 = k*m2^m;             % heat transfer coefficient secondary side
```

```
    Ah = 1/(1/Ah1+ 1/Ah2 + d/(A*lambda)); % total heat transfer coefficient
    dAh = Ah/5;
    m1cp = m1*cp1;
    m2cp = m2*cp2;

    x10 = T1;      % inlet temperatures
    x20 = T3;

    dtm1 = (x10+x(1)-x(10)-x(9))*0.5;    % compartment mean temp diff
    dtm2 = (x(1)+x(2)-x(9)-x(8))*0.5;
    dtm3 = (x(2)+x(3)-x(8)-x(7))*0.5;
    dtm4 = (x(3)+x(4)-x(7)-x(6))*0.5;
    dtm5 = (x(4)+x(5)-x(6)-x20)*0.5;

    Qh1 = dAh*dtm1;                % compartment heat transfer energy flows
    Qh2 = dAh*dtm2;
    Qh3 = dAh*dtm3;
    Qh4 = dAh*dtm4;
    Qh5 = dAh*dtm5;

    sys(1) = (m1cp*(x10-x(1))-Qh1)/dC1;     % primary side
    sys(2) = (m1cp*(x(1)-x(2))-Qh2)/dC1;
    sys(3) = (m1cp*(x(2)-x(3))-Qh3)/dC1;
    sys(4) = (m1cp*(x(3)-x(4))-Qh4)/dC1;
    sys(5) = (m1cp*(x(4)-x(5))-Qh5)/dC1;

    sys(6) = (m2cp*(x20-x(6))+Qh5)/dC2;     % secondary side
    sys(7) = (m2cp*(x(6)-x(7))+Qh4)/dC2;
    sys(8) = (m2cp*(x(7)-x(8))+Qh3)/dC2;
    sys(9) = (m2cp*(x(8)-x(9))+Qh2)/dC2;
    sys(10) = (m2cp*(x(9)-x(10))+Qh1)/dC2;

elseif flag == 3,

    T2 = x(5);                              % outlet temperatures
    T4 = x(10);

    sys(1)=T2;
    sys(2)=T4;

elseif flag == 0,

    sys= [10,0,2,4,0,0];   % Sizes
    x0 = zeros(10,1);

else

    sys = [];

end
```

C. Modula-2 Code

The real-time implementation of the controller has been made in MODULA-2, a real time language developed at ETH, Zürich. The code for the controller has been inserted into an existing piece of code made for a realtime project at the Department of Automatic Control in Lund. Hence I only present the code for the fuzzy controller which is a rather small part of the total code.

The definition module and the implementation module is presented on the following pages. It should be stressed that these implementations are made only to verify the functionality of the controller and are not made in a very proper way.

```
(************************** FUZZY.DEF ****************************)

DEFINITION MODULE Fuzzy;

TYPE BvType      = RECORD
     bv     : ARRAY [1..7] OF REAL;
     length : INTEGER;
   END;
     FuzzyParType = RECORD
      Coeffs           : ARRAY [1..16] OF
   ARRAY [1..8] OF REAL;
      SysConFig        : ARRAY [1..5] OF INTEGER;
      Bvec             : ARRAY [1..7] OF
   ARRAY [1..3] OF REAL;
      KNe,KNde,KNload,
      KDu              : REAL;
      N,umax,umin,h    : REAL;
      Cut,Tr,T1,
      Dzonel,Dzoneh,
      Kx1,Tc,Hl,Hyst,
      Rtimel,Steps,
      Wmin             : REAL
    END (* ParType*);

     FuzzyType   = RECORD
     Parameters  : FuzzyParType;
     a,b,c,d,e       : REAL;
     Dold,eold,I   : REAL;
     Hiact,
     Windup,Windupold      : REAL;
     bv1,bv2,bv3 : BvType;
     u,vold,v,v3,du   : REAL;
     Ti1,
     ulow1,ulow2 : REAL;
   END (* FuzzyRecord*);

PROCEDURE InitReg(VAR FuzzyReg: FuzzyType);

PROCEDURE Dispose(VAR FuzzyReg: FuzzyType);

PROCEDURE SetFuzzyPar(VAR Reg: FuzzyType; VAR Pars: FuzzyParType);

PROCEDURE GetFuzzyPar(VAR Reg: FuzzyType; VAR Pars: FuzzyParType);

PROCEDURE GetExtraVar(VAR Reg: FuzzyType):REAL;

PROCEDURE ComputeOutput(VAR Reg: FuzzyType; yr,y: REAL): REAL;

PROCEDURE UpdateState(VAR Reg : FuzzyType);

END Fuzzy.
```

```
(************************ FUZZY.MOD ****************************)

IMPLEMENTATION MODULE Fuzzy;

FROM Console IMPORT PutChar;
FROM Storage IMPORT ALLOCATE;

PROCEDURE InitReg(VAR Reg: FuzzyType);
VAR i:INTEGER;
    ap:POINTER TO BOOLEAN;
BEGIN
  ap:=NIL;
  WITH Reg DO
  WITH Parameters DO
    IF SysConFig[5]=2 THEN
      FOR i:=1 TO SysConFig[1]+2 DO
bv1.bv[i]:=Bvec[i,1];
      END;
      bv1.length:=i-1;
      FOR i:=1 TO SysConFig[2]+2 DO
bv2.bv[i]:=Bvec[i,2];
      END;
      bv2.length:=i-1;
    ELSE
      IF ap^ THEN END;
      (* Fel Fuzzyregulator *)
    END;
    N:=10.0;
    u:=0.0;
    v:=0.0;
    v3:=0.0;
    du:=0.0;
    Hiact:=0.0;
    Windup:=0.0;
    Dold:=0.0;
    eold:=0.0;
    I:=0.0;
    b:=h/10.0;
    a:=b*2.0;
    c:=1.0/(1.0+N*h);
    d:=N*c;
    e:=0.0;
    Ti1:=(Cut/100.0)*Rtime1*Steps;
    ulow1:=Hl-Hyst/2.0;
    ulow2:=Hl+Hyst/2.0;
  END;END;
END InitReg;

PROCEDURE SetFuzzyPar(VAR Reg: FuzzyType; VAR Pars: FuzzyParType);
BEGIN
  Reg.Parameters:=Pars;
```

```
END SetFuzzyPar;

PROCEDURE GetFuzzyPar(VAR Reg: FuzzyType; VAR Pars: FuzzyParType);
BEGIN
  Pars:=Reg.Parameters;
END GetFuzzyPar;

PROCEDURE GetExtraVar(VAR Reg: FuzzyType):REAL;
BEGIN
  RETURN Reg.e;
END GetExtraVar;

PROCEDURE Sat(v,umin,umax: REAL): REAL;
BEGIN
  IF v<umin THEN
    RETURN umin
  ELSIF v>umax THEN
    RETURN umax
  ELSE
    RETURN v
  END;
END Sat;

PROCEDURE FindIndex(Vec:BvType; In:REAL):INTEGER;
(* Hittar ett index i en dimension i coeffs fr en given insignal *)
VAR i:INTEGER;
BEGIN
  i:=1;
  WHILE ((Vec.bv[i]<=In) AND (i<>Vec.length)) DO
    i:=i+1;
  END;
  RETURN i-1;
END FindIndex;

PROCEDURE Deadzone(dzone,error:REAL):REAL;
VAR e1:REAL;
BEGIN
  IF error<-dzone THEN
    e1:=error+dzone;
  ELSIF error>dzone THEN
    e1:=error-dzone;
  ELSE
    e1:=0.0;
  END;
  RETURN e1;
END Deadzone;

PROCEDURE VarDeadzone(VAR Reg:FuzzyType; error:REAL):REAL;
VAR e1,dzone:REAL;
BEGIN
  WITH Reg DO
    IF Hiact>0.5 THEN
```

```
      dzone:=Parameters.Dzoneh;
    ELSE
      dzone:=Parameters.Dzonel;
    END;
    e1:=Deadzone(dzone,error);
    RETURN e1;
  END;
END VarDeadzone;

PROCEDURE UpdateVarDeadzone(VAR Reg:FuzzyType);
VAR dHiact:REAL;
BEGIN
  WITH Reg DO
    IF u<ulow1 THEN
      dHiact:=-Hiact/Parameters.T1;
    ELSIF u>ulow2 THEN
      dHiact:=(1.0-Hiact)/Parameters.T1;
    ELSE
      dHiact:=0.0;
    END;
    Hiact:=Hiact+Parameters.h*dHiact;
  END;
END UpdateVarDeadzone;

PROCEDURE RateLim(VAR Reg:FuzzyType):REAL;
BEGIN
  RETURN Reg.v3;
END RateLim;

PROCEDURE UpdateRateLim(VAR Reg:FuzzyType);
VAR v1,v2:REAL;
BEGIN
  WITH Reg DO
  WITH Parameters DO
    v1:=((v*Kx1)-v3)/Tc;
    v2:=Sat(v1,-Cut,Cut);
    v3:=v3+v2*h/Ti1;
  END;END;
END UpdateRateLim;

(************************** Filters **************************)

PROCEDURE ComputeInFilters(VAR Reg:FuzzyType;
 VAR denorm,enorm:REAL);
BEGIN
  WITH Reg DO
    Dold:=c*Dold+d*(e-eold);                (* dError *)
    eold:=e;
    denorm:=Parameters.KNde*Dold;

    enorm:=Parameters.KNe*e;                (* Error *)
    (* LoadNorm:=Parameters.KNload*Load; *) (* Load *)
```

```
  END;
END ComputeInFilters;

PROCEDURE ComputeOutFilters(VAR Reg:FuzzyType):REAL;
BEGIN
  RETURN Reg.v;
END ComputeOutFilters;

PROCEDURE UpdateInFilters(VAR Reg:FuzzyType);
BEGIN
  (* Nothing to update *)
END UpdateInFilters;


PROCEDURE UpdateOutFilters(VAR Reg:FuzzyType);
VAR e2,W:REAL;
BEGIN
  WITH Reg DO
    e2:=u-v;
    W:=Deadzone(Parameters.Wmin,e2);
    Windup:=W*1.0/Parameters.Tr;
    v:=v+Parameters.KDu*Parameters.h*du;
    Windupold:=Windup;
  END;
END UpdateOutFilters;

(*************************************************************)

PROCEDURE ComputeOutput(VAR Reg: FuzzyType; yr,y: REAL): REAL;

VAR ind,jx,jy,jz    :INTEGER;
    e1,denorm,
    enorm, LoadNorm,
    u1              :REAL;
    in1,in2,in3     :REAL;

BEGIN
  WITH Reg DO
    e1:=yr-y;
    e:=VarDeadzone(Reg,e1);

    ComputeInFilters(Reg,denorm,enorm);

    (* Input Saturations *)

    in1:=Sat(denorm,-1.0,1.0);
    in2:=Sat(enorm,-1.0,1.0);

    (* in3:=Sat(LoadNorm,-1.0,1.0); *)

    (* Output Calculations *)
```

```
      jx:=FindIndex(bv1,in1);
      jy:=FindIndex(bv2,in2);
      (* jz:=FindIndex(bv3,in3); *)

      ind:=(jx-1)*bv2.length+jy;
      WITH Parameters DO
        du:=(in1*    Coeffs[ind,1]+
         in2*Coeffs[ind,2]+
     in1*in2*Coeffs[ind,3]+
     Coeffs[ind,4])/
     (in1*   Coeffs[ind,5]+
         in2*Coeffs[ind,6]+
     in1*in2*Coeffs[ind,7]+
     Coeffs[ind,8]);
      END;
      v:=ComputeOutFilters(Reg);
      u1:=v;
      u1:=RateLim(Reg);
      u:=Sat(u1,Parameters.umin,Parameters.umax);
      RETURN u;
    END;
END ComputeOutput;

PROCEDURE UpdateState(VAR Reg : FuzzyType);
BEGIN
  UpdateRateLim(Reg);
  UpdateOutFilters(Reg);
  UpdateInFilters(Reg);
  UpdateVarDeadzone(Reg);
END UpdateState;

BEGIN
END Fuzzy.
```