ISSN 0280-5316
ISRN LUTFD2/TFRT--5701--SE

Steam Net Simulation with Real Control System

Oscar Ljungkrantz

Department of Automatic Control
Lund Institute of Technology
January 2003

Department of Automatic Control **Lund Institute of Technology** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* January 2003 *Document Number* ISRN LUTFD2/TFRT--5701--SE
Author(s) Oscar Ljungkrantz	*Supervisor* Anders Rantzer, LTH Magnus Holmgren, Solvina *Sponsoring organization*

Title and subtitle

Steam Net Simulation with Real Control System(Ångnätssimulering med verkligt styrsystem)

Abstract

This master thesis has been conducted for Solvina. Solvina is a firm in Gothenburg, which among other things simulate steam nets for the paper process industry. Doing this Solvina has had to simulate not only the steam net itself, but also to simulate the control system. The purpose of this master thesis was to try to build an interface between the controller and the simulated process so that the simulated steam net could be controlled by the real control system.
The process is modelled in Dymola/Modelica. The control system is built in Siemens process control system called Simatic PCS 7 and runs on a PC. The simulated process and the control system run on the same PC and the communication between the process and the control system is a communication between two applications running on the same operative system, which is Windows NT. The two applications communicate with each other by reading from and writing to a common file. The simulated process in Dymola calls a C++ function that reads the control signals from the file and writes the process values to the file, every sampling time. A J++ program communicates with the control system and writes the control signals to the file and reads the process values from the file, every sampling time. The communication is synchronized and the applications access the file using mutual exclusion.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 58	*Recipient's notes*
Security classification		

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Steam net simulation with real control system

Examensarbete av

Oscar Ljungkrantz

för

Solvina

Sammanfattning

Detta examensarbete har utförts för Solvina. Solvina är ett företag i Göteborg som bland annat simulerar ångnät för pappersindustrin. När Solvina gjort sådana simulatorer har de varit tvungna att simulera inte bara själva ångnätet utan även reglersystemet. Syftet med examensarbetet var att programmera ett gränssnitt mellan styrsystemet och den simulerade processen så att det verkliga styrsystemet kan fås att reglera den simulerade processen.

Processen har modellerats i Dymola/Modelica. Styrsystemet har byggts i Siemens styrsystem Simatic PCS 7 och körs på en PC. Den simulerade processen och styrsystemet körs på samma PC och kommunikationen dem emellan är en kommunikation mellan två applikationer som kör under samma operativsystem, vilket är Windows NT. De två applikationerna kommunicerar med varandra genom att läsa från och skriva till en gemensam fil. Den simulerade processen i Dymola anropar en C++ funktion som läser styrsignalerna från filen och skriver processvärdena till filen, varje sampel. Ett J++ program kommunicerar med styrsystemet och skriver styrsignaler till filen samt läser processvärden från filen, varje sampel. Kommunikationen är synkroniserad och applikationerna utnyttjar ömsesidig uteslutning när de tillträder filen.

CONTENTS

1 Introduction

1.1 Background

Solvina is a firm in Gothenburg with customers mainly in the power and process industry. Solvinas business concept is to provide systems engineering solutions for improved profitability to their customers. One way to accomplish this is to build models (primarily dynamic) of the existing system and, on the basis of those models, perform different simulations to try to improve the quality, productivity and reliability of the system. Similar simulations are built of nonexistent systems as well, to investigate how a new or rebuilt system would affect the current activity.

Solvina has had many customers during the last years, who have wanted Solvina to simulate their steam nets. To do this Solvina has been forced not only to simulate the steam net itself but to simulate also the control system. Solvina wants to be able to use the real control system directly upon the simulated process. Doing this would give several advantages:

- The development time will decrease. It takes time to build a simulator of an advanced control system.
- One source of error is eliminated. The model of the control system often becomes inadequate. Some special functions, time delays etc. are very hard to model accurately.
- A new control system can be tested and tuned before put into operation. By doing this you can avoid expensive adjustments during production.
- Improved operators training. The operators can be trained in exactly the same user interface as they are used to, but on the simulated process.

1.2 Purpose

The purpose of this master thesis is to build an interface between the controller and the process so that the simulated process (steam net) can be controlled by the real regulator. The process is modelled in Dymola, which is a tool for the modelling language Modelica and is the program that Solvina uses to simulate steam nets. The control system is developed by Siemens and is called Simatic PCS 7. To be able to build and test the interface, a demo process will be written in Dymola, simulating a real paper mill steam net. However, no validation or verification of the model will be done, since the primary goal with this master thesis is not the model itself. The primary goal is to get a control system that will be written in Simatic PCS7, to control the simulated process. The reason for using PCS 7 is that this is a common process control system and that the mill from which the data is taken, is thinking about upgrading its older control system to Simatic PCS 7.

2 Steam net

Steam is a central part in the paper making process. Steam is used in the paper machines to dry the paper but also in many other phases of the process, for instance when cooking the wood and when drying black liquor. The steam is produced in boilers. The steam producers and the consumers are connected to each other with pipes. The boilers produce steam at quite high pressure and temperature and the consumers often demand steam at different, lower pressures and therefore the steam net is divided in different subnets, with controllable valves in between. The controllable valves controls the flow in the net and the opening of the valves are adjusted so that the pressures in the subnets are kept at the desired levels.

An example of a steam net can be seen in Figure 1. Every component will be explained in more detail below.

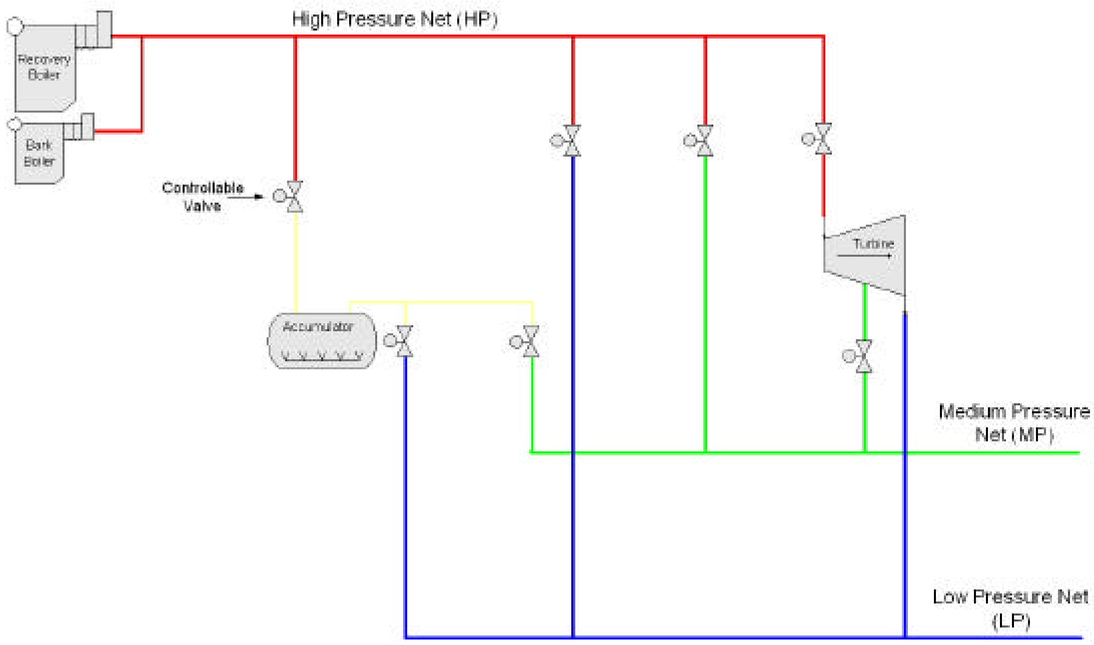


Figure 1, an example of a steam net

2.1 Boiler

The boiler is the steam producer. A broad simplification can be seen in Figure 2. Water to the boiler is pumped into the drum. The drum is a big tank, which contains both water and steam. The water is transported from the drum to the combustion where the water is partly vaporized. The mix of water and steam is transported back to the drum. Steam is separated from the drum and led into a super heater that heats the steam to the desired temperature. [3]

The fuel can be black liquor from the paper production, but usually the recovery boiler is complemented with a power boiler, for instance an oil boiler or a bark boiler. Usually the wood comes to the paper mill like logs. First the bark is taken of the logs. The bark is most likely burned in a bark boiler, possibly together with oil if the bark is too damp. The wood is chipped and cooked with NaOH, which among other things give cellulose and black liquor. The chemicals from the black liquor can be recovered in a boiler, therefore the name recovery boiler.

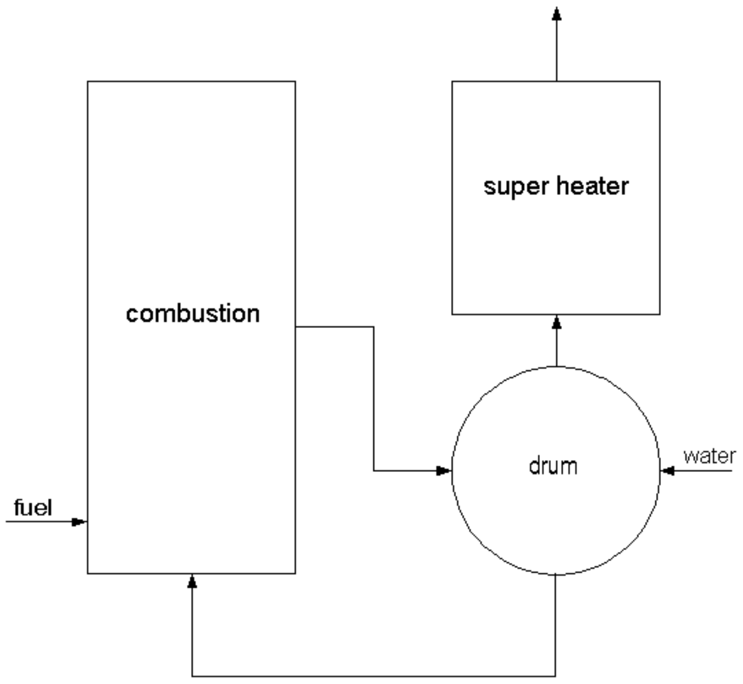


Figure 2, the basic components of a boiler

2.2 Steam Turbine

In the steam turbine the thermal energy of the steam is transformed into rotational energy (which in turn can be transformed into electrical energy in a generator) by making the turbine blades rotating. Steam turbines are used in various contexts and are built in many different ways. One way to classify the different types is by looking at the state and the usage of the outflow of the steam. Then two terms are often used: condensing turbines and backpressure turbines, see Figure 3. In a condensing turbine the steam from the outlet is led into a condenser. The condenser has one or many outlets to reuse the pre-heated water in the boiler. In a backpressure turbine the outlet steam has a relatively high pressure and temperature to be used in some process. Condensing turbines are used when the electricity demand is higher than the steam demand. Backpressure turbines are used when the primary purpose of the steam net is to produce steam, and the electricity production is more or less a way to use the surplus steam and increase profitability. This is the case for the paper process industry. Both condensing and backpressure turbines can have more than one extraction point.[5][7]

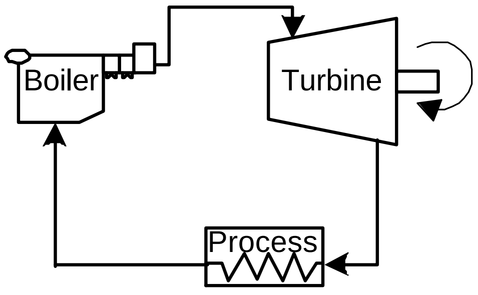


Figure 3a, Backpressure Turbine

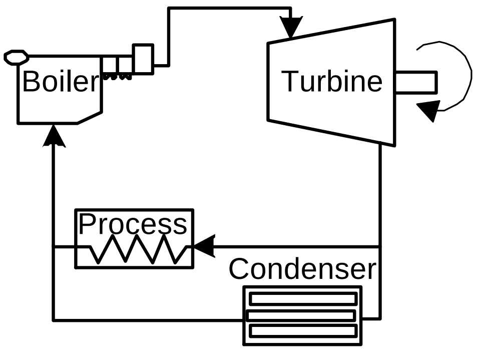


Figure 3b, Condensing Turbine

2.3 Valve

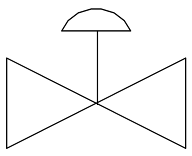


Figure 4, a simple sketch of a controllable valve

In many of the components (e.g. in the boiler and in the turbine) and between the different nets there are valves used to control the flows. The flow, Q, through the valve between nets with pressure P_1 and P_2 is

$$Q = K_v \sqrt{(P_1 - P_2)\frac{r_{H_2O}}{r}} \cdot g \quad \left[\frac{m^3}{h}\right], \tag{2.1}$$

where K_v $\left[\frac{m^3/h}{bar^{1/2}}\right]$ is the capacity of the valve; r is the density of the steam, r_{H_2O} is the density of the water (approximately 1000 kg / m^3) and g ($g \in [0,1]$), the opening parameter, is the parameter to be controlled [6].

In most of the components there are safety valves as well, that simply lets the steam out to the surroundings if the pressure gets too high or some other condition is violated.

2.4 Accumulator

The accumulator is a large tank that contains both water and steam. It can, as indicated by the name, be used to accumulate steam when the pressure is high in both the high pressure net and in the low pressure net. When the low pressure net needs more steam and the steam produced by the boilers isn't sufficient, the accumulator can act as a supplementary steam source for a shorter period of time. When the accumulator is loaded with steam, the pressure in the tank is increased, the steam condenses and the water level is increased. When taking out steam, the pressure is decreased and hence the water is boiling into steam.

3 Introduction to Dymola

Dymola, Dynamic Modeling Laboratory, is a tool for building dynamical models from physical objects. Dymola uses the modelling language Modelica, which is a language developed to describe complex physical systems from a multitude of different areas. Modelica is object oriented and allows hierarchical model composition. Thus, one can easily reuse already written components from existing libraries. [1]

Every object in Dymola can be described by the inputs and outputs used to communicate with the surroundings, along with the variables and the equations which describes the characteristics of the object. No equations needs to be solved by the user, Dymola does that automatically as long as the number of equations and the number of unknowns are equal.

To facilitate the model building Dymola has both a graphical editor and a text editor. The graphical editor is used to join different components in a block diagram. New components can be added to the diagram using "drag-and-drop" from the package browser. The text editor is used to write declarations and equations. Hence new components can be created by reusing and changing components from the library, as well as by creating totally new ones.

In Figure 5 an electrical motor is built up by library components. All connections between the components are made in the graphical editor by simply drawing lines between the connectors.

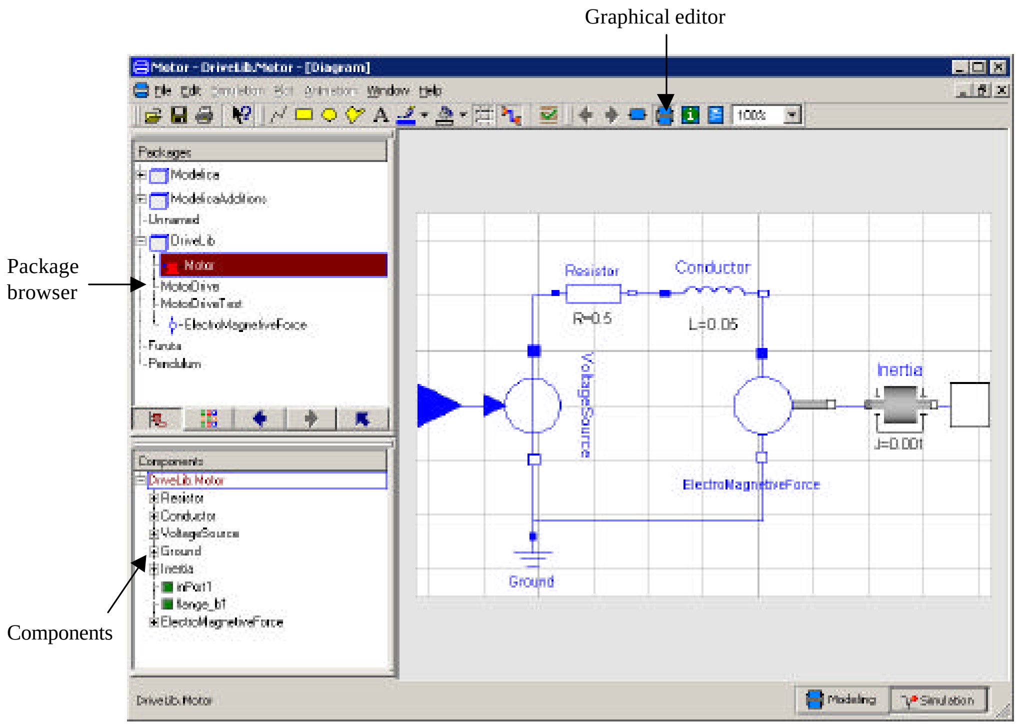
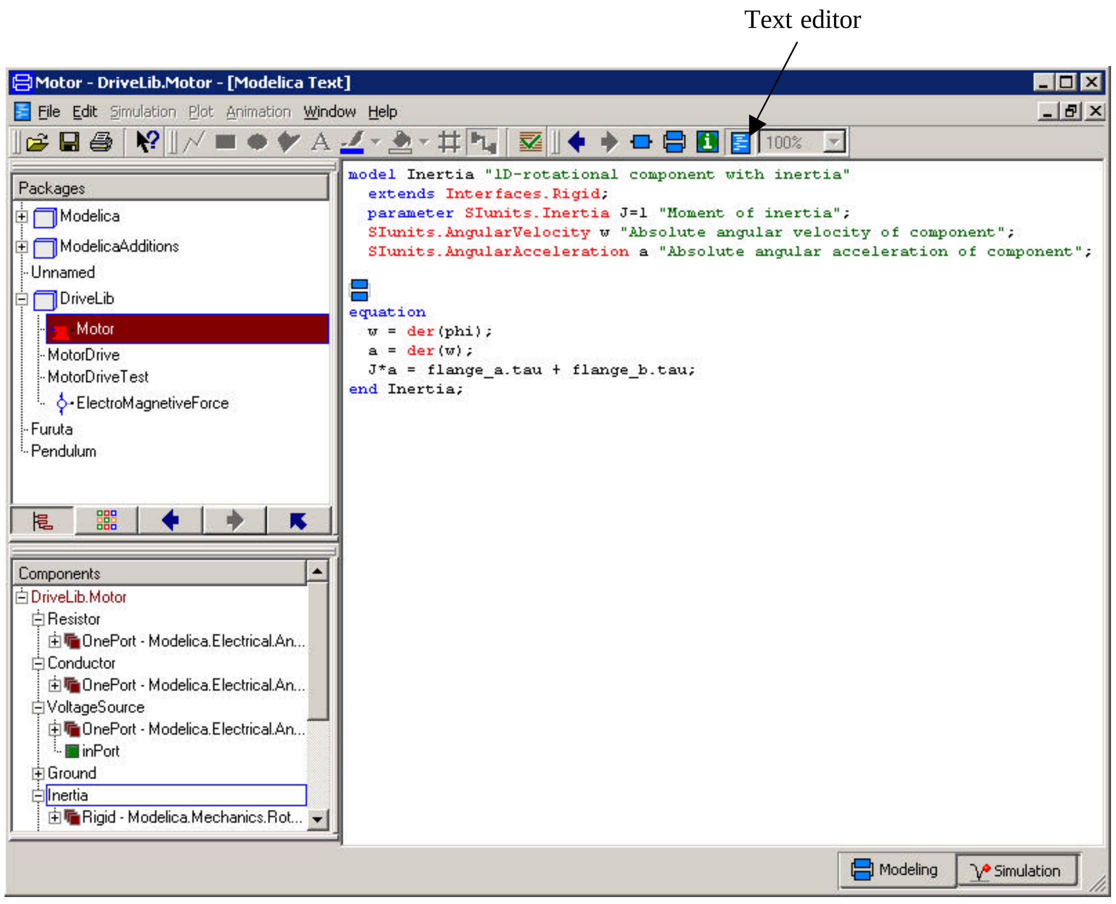


Figure 5, an electrical motor modelled in Dymola

In Figure 6 the declarations and equations of the inertia in the motor, can be seen. The component inherits properties from another component (“extends Interfaces.Rigid;”) and adds some variables and equations specific for this component.


Text editor
Motor - DriveLib.Motor - [Modelica Text]
File Edit Simulation Plot Animation Window Help
100%
Packages
Modelica
ModelicaAdditions
Unnamed
DriveLib
Motor
MotorDrive
MotorDriveTest
ElectroMagnetiveForce
Furuta
Pendulum
Components
DriveLib.Motor
Resistor
OnePort - Modelica.Electrical.An...
Conductor
OnePort - Modelica.Electrical.An...
VoltageSource
OnePort - Modelica.Electrical.An...
inPort
Ground
Inertia
Rigid - Modelica.Mechanics.Rot...
Modeling
Simulation

```
model Inertia "1D-rotational component with inertia"
  extends Interfaces.Rigid;
  parameter SIunits.Inertia J=1 "Moment of inertia";
  SIunits.AngularVelocity w "Absolute angular velocity of component";
  SIunits.AngularAcceleration a "Absolute angular acceleration of component";

equation
  w = der(phi);
  a = der(w);
  J*a = flange_a.tau + flange_b.tau;
end Inertia;
```

Figure 6, declarations and equations of the inertia in the motor

Solvina has in past projects built up a quite extensive library with steam net components, e.g. recovery boiler, power boiler, accumulator, valves, turbines and steam consumers. Those models are based on ThermoFluid, a thermo-hydraulic Modelica library developed by Hubertus Tummescheit and Jonas Eborn at the Department of Automatic Control, Lund Institute of Technology [2]. Solvina has also a library with components for control systems, PID-controllers with anti-windup and tracking, min and max selectors etc.

4 The Steam Net Model

Most of the data in this demo steam net is taken from a steam net in a real paper process industry. The demo process has paper machines that demand steam with two different conditions, and therefore they have two different consumer subnets, one at a medium pressure, MP (approx. 11 bar, 200 °C), and one at low pressure, LP (approx 3.2 bar, 150 °C).

To produce steam, the net has three different boilers, one recovery boiler with black liquor as the fuel and two supplementary power boilers; one two-fuel bark boiler that runs on both bark and oil and one that runs on oil. The recovery boiler and the bark boiler both produce steam to a high pressure subnet, HP1 (approx. 54 bar, 450 °C). The oil boiler produces steam to another high pressure subnet, HP2 (approx. 20 bar, 240 °C).

The net has a backpressure turbine with two extraction points. The turbine has an inlet valve that determines how much high pressure steam (from HP1) that should pass the turbine on the whole. The steam that doesn't go through the discharge valve to MP continues through the remainder of the turbine and onto the LP. A simple sketch of the turbine can be seen in Figure 7.

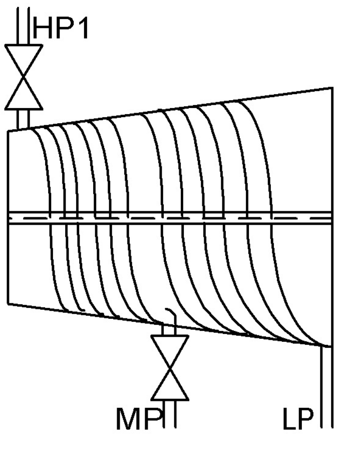


Figure 7, a sketch of the demo process turbine

The steam from HP2 doesn't go though the turbine but goes directly to MP and LP via controllable valves. HP1 is connected to MP and LP via controllable valves too. Both HP1 and HP2 are also connected to an accumulator via controllable valves, which in turn is connected to MP and LP, via controllable valves. The low pressure net, LP, has controllable airblow valves, controlled to let out steam from the net if the pressure gets high. The main parts of the steam net can be seen in Figure 8. The whole steam net, as modelled in Dymola, can be seen in Figure 9. The blue squares are components that represent the volumes of the pipes. The circles with the label "PI" inside are Pressure Indicators.

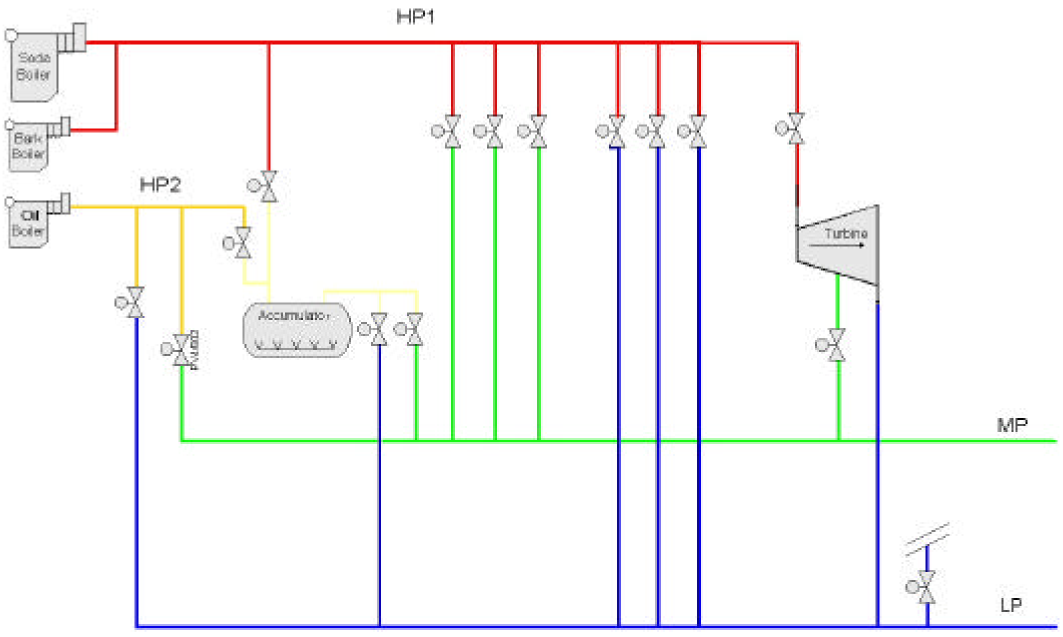


Figure 8, the main parts of the demo steam net

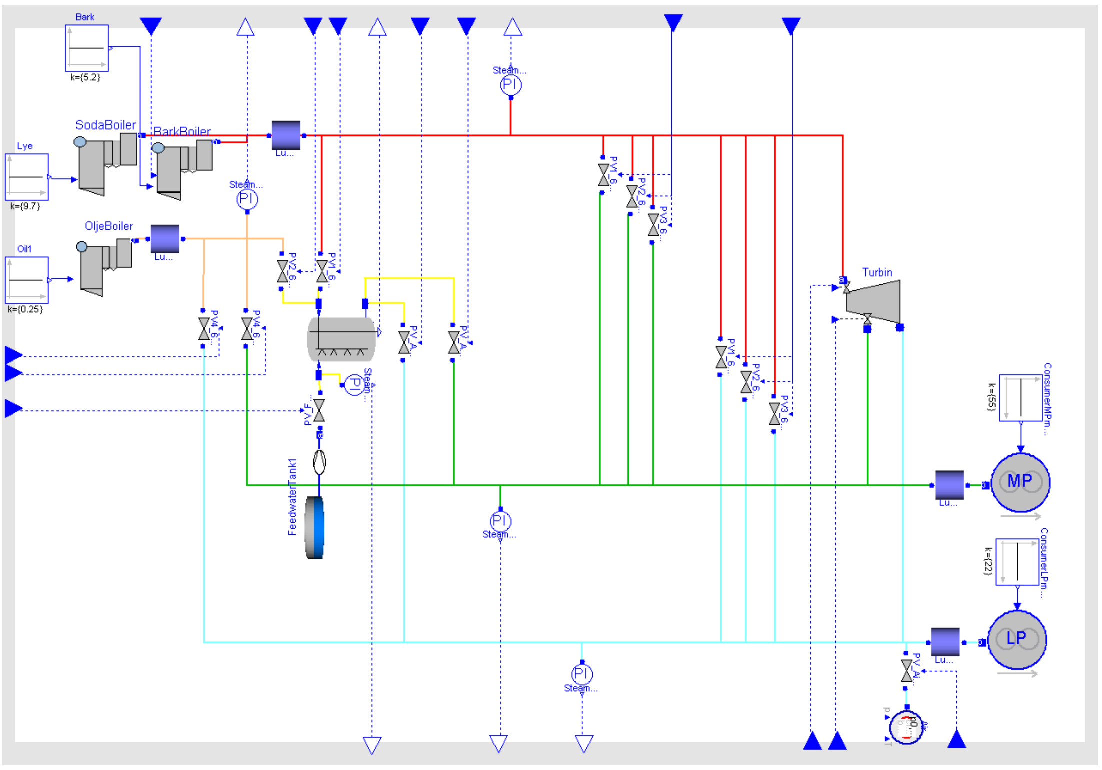


Figure 9, the demo steam net, modelled in Dymola

Almost all components are taken directly from Solvinas steam net and control system libraries with parameters and initial values changed to fit this steam net. The only new component is the valve. The valves in the existing library were described by a simple linearized equation while the valves in this master thesis is described by eq. 2.1 in section 2.3 and has the capacity K_v as parameter.

5 Control of a Steam net

The control of such a steam net described in chapter 2 and 4 above is usually a quite complex control system with many collaborating controllers, often PI-controllers. Typically the opening parameter g of a valve between two nets N1 and N2, where N1 has the highest pressure, is determined as the minimum of two PI-controllers, C1 and C2, see Figure 10. C1 controls the net N1, gets the process value from N1's pressure indicator and has a set point corresponding to the desired pressure of N1. With too low pressure in N1, C1 will of course try to close the valve. Hence the output of C1 is low when the process value is low and vice versa. This is called direct action and is accomplished by having a negative gain in the controller. C2 controls the net N2 and will try to open the valve when the pressure in N2 is too low. This is the way a PI-controller usually works and is called reversed action since a low process value will give rise to a high output and vice versa.

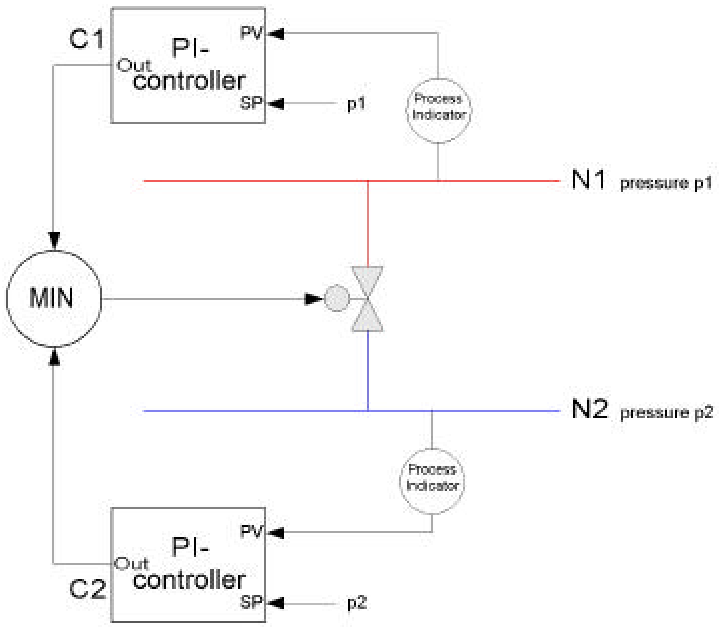


Figure 10, an example of the control of a steam net valve

The output of each controller is limited between two values. Since the opening parameter $g \in [0,1]$, the regulators controlling the valves are limited between 0 and 1. This means that for a reversed P-controller the output will always be 0 if the process value is greater than the set point. For a direct P-controller the output will be 0 if the process value is less than the set point. Adding the integral part this will, of course, not always be true but if the process value is less than the set point for a longer period a direct PI-controller will also have the output signal zero and vice versa.

Some valves have a more difficult control signal, where the signal is a combination of minimums and maximums of different controllers. The design and the tuning (setting the gain, the integration time and the set point of each controller) of the whole control system is an interesting and hard problem due to the way each controllable valve influences the state of the involved subnets, which in turn affects the whole steam net.

As mentioned above, the controllers involved with one net have different set points. As an example, look at Figure 11, where some of the controllers that affects HP1 are outlined. The controllers are described in Table 1. Most of the controllers that get their process values from HP1 are included. In the complete system there will, of course, be other regulators controlling the valves but in this example the focus lies in controlling HP1. Assume that the pressure in HP1 is slightly more than 53 bar and that this pressure has been held for a long period. Since the pressure is above 52.5 bar, PC H1 has an output signal greater than zero, and since this pressure has been held for a long period the integrator will have a large value and the output will be one. In the same way the output signal of PC H2 will be one and the turbine will be fully activated.

The main target of the net is to produce steam to MP and LP. This is taken care of by the controllers PC M1 and PC L1 respectively. They will adjust the valves from HP1 so that the pressure in MP and LP lie around 11.0 bar and 3.2 bar respectively.

Assume now that the pressure in HP1 decreases for some reason, for instance that one of the boilers cannot be run in full operation. Then the pressure in HP1 starts to sink. If the pressure decreases below 53.0 bar the output signal from PC H2 will also start decreasing, trying to get the pressure back to the set point. This measure could be enough, in which case PC H2 will be controlling the pressure to 53.0 bar, but assume that the amount of steam produced is so low that the pressure continues falling. If the pressure decreases below 52.5 bar, the output signal from PC H1 will also start decreasing. When the output signal of PC H1 is less than the output signals of PC L1 or PC M1 the opening of the valve concerned will be decreased. The net will then end up in a state where the turbine is closed (the output of PC H2 will be zero) and PC H1 is controlling the pressure in HP1 to 52.5 bar. Observe that in this state there is no way to keep the desired pressures in MP and LP. The state should never be reached during operation, but during start-up the state is needed.

Assume now instead that the pressure in HP1 increases. The reason for this could for instance be that the amount of MP steam used decreases so that PC M1 decreases the opening of the valve between HP1 and MP. When the pressure in HP1 passes 53.8 bar PC H3 will start opening the valve to the accumulator. If the pressure in HP1 still continues rising and passes 55.5 bar, PC H4 will open up the airblow valve. This will make the pressure in LP decreasing and PC L1 will increase the opening of the valve between HP1 and LP, which in turn will affect the pressure in HP1.

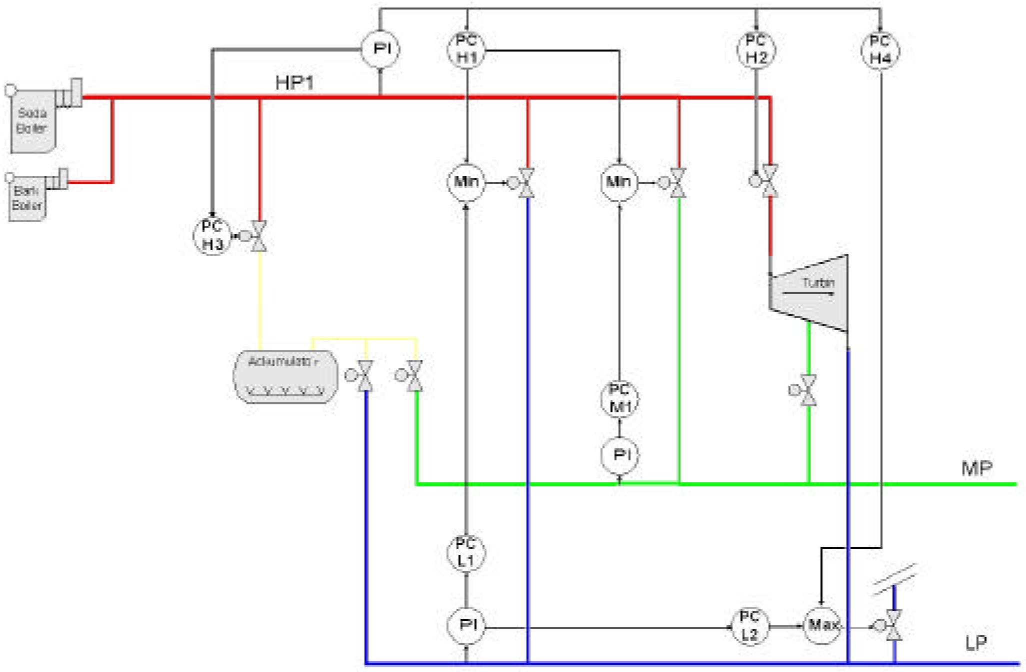


Figure 11, control of HP1

Controller	Set Point	Action
PC L1	3.2	Reversed
PC L2	3.6	Direct
PC M1	11.0	Reversed
PC H1	52.5	Direct
PC H2	53.0	Direct
PC H3	53.8	Direct
PC H4	55.5	Direct

Table 1, the controllers in Figure 11

6 The whole system running in Dymola

One of the purposes of being able to use the real control system to control the simulated process is to get away from having to simulate the control system in Dymola. Still, it comes within the scope of this thesis to investigate the advantages and possible disadvantages of using the real control system instead of a simulated one. Thus, a control system was built in Dymola, to verify the model and to be used later for comparison with the real control system. The data in this demo control system is taken from a real paper process industry; the same as from which the data to the demo steam net was taken.

As mentioned in chapter 3 above, Solvina has built up a library with components for control systems. This library contains PI-controllers and manual controllers developed to behave like the corresponding components in Siemens Simatic PCS7. Using those it was quite straightforward to build up the whole control system. Manual controllers were placed after the min and max selectors so that the opening of the valves can be controlled directly, by simply putting the manual controllers into manual mode and selecting the desired value. To avoid bumping when going back to automatic mode, tracking signals has to be drawn back to the controllers, see Figure 12. In manual mode the output signal of the manual controller can be adjusted by setting the manual value (MV in the picture). In automatic mode the manual controller sends out the value that corresponds to the set point of the manual controller. The output is drawn back to the manual controller as a feed back signal (FB) to guarantee a bumpless change to manual mode. This way the manual value of the controller always starts at the same value as the latest automatic value when changed from automatic to manual mode. The change the other way from manual to automatic mode is also bumpless since all controllers involved have tracked the manual value. [8]

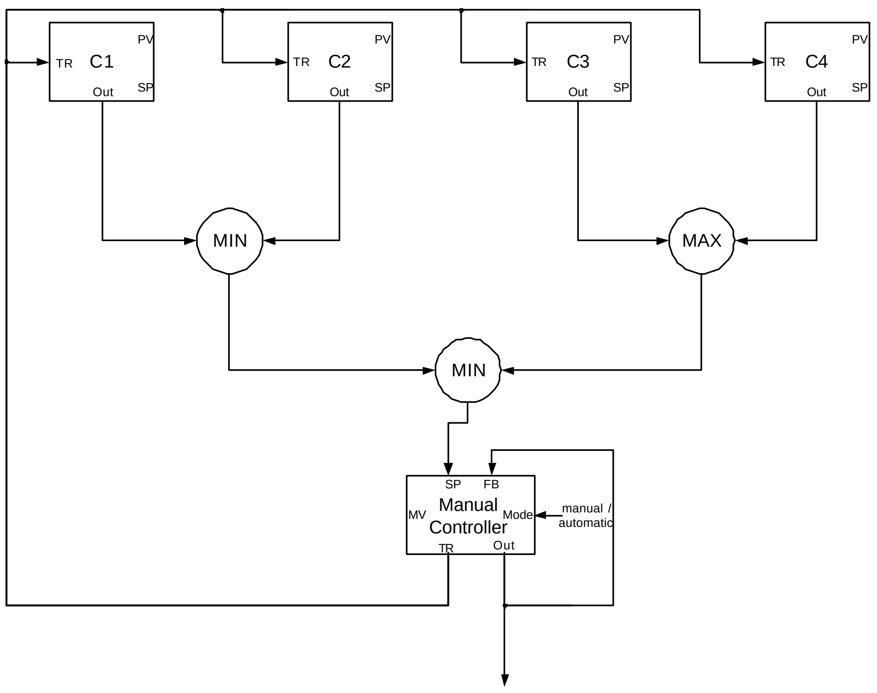


Figure 12, tracking when a manual controller is used to control a signal where several controllers are involved

Using these manual controllers it is possible to test the behaviour of the steam net and the control systems both in the normal operational state and in other special cases for example if the turbine is put out of operation.

In Figure 13 the pressure of the main nets HP1, HP2, MP and LP are shown during normal operation in a test of 4 000 seconds. After some fast changes due to the initial conditions, the pressures of the nets seams to stabilize around very reasonable values, HP1 at almost 54 bar, HP2 at 20 bar and more important MP at 11 bar and LP at slightly more than 3.2 bar. The test is carried on for over an hour (4 000 s) and one would probably not expect the behaviour to be any different when performing a longer test.

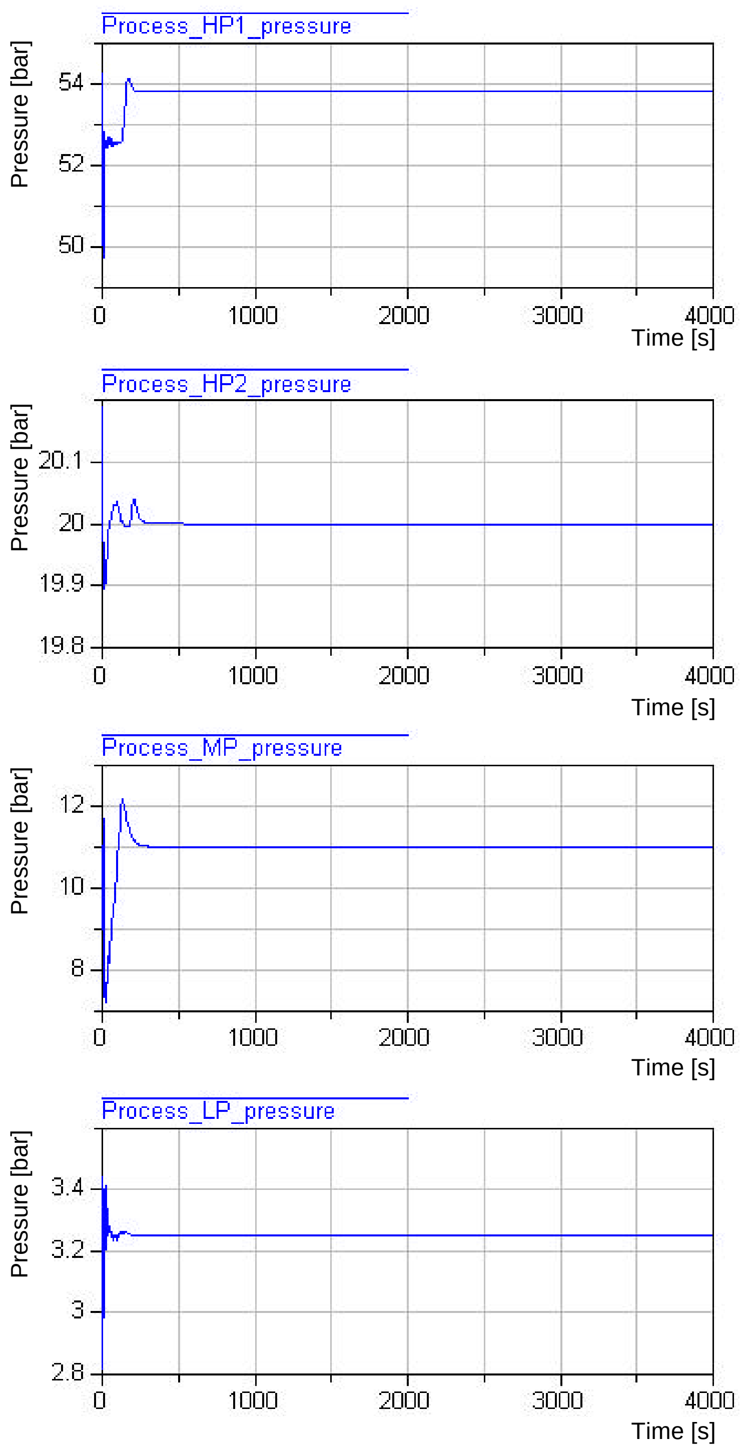


Figure 13, pressure of main nets during normal operation for 4 000 s

Performing a longer test, however, shows an interesting and somewhat more unpredictable behaviour, see Figure 14. Consider the pressure of the high pressure net HP1. The pressure of the net seams to stabilize around some value and then after a while the pressure changes and stabilizes around some new value. This is typical for this kind of control system and is due to the different set points of the controllers of the system. The changes of the pressure can be removed using a master regulator that changes the set points of the system, but it was not used here. Observe that this control system is not tuned to be optimal.

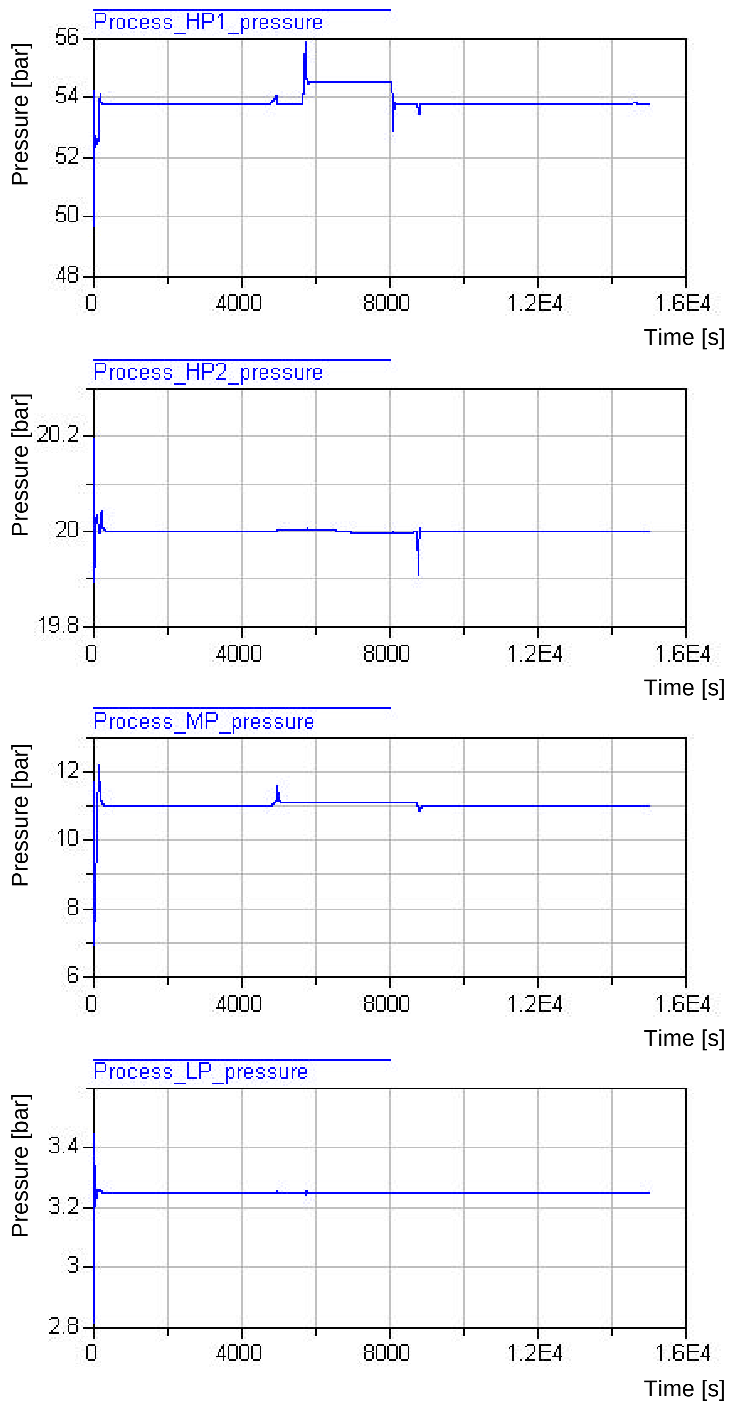


Figure 14, pressure of main nets during normal operation for 15 000 s

Consider, for instance, the changes between 4 000 and 6 000 s. Taking a closer look at the state of the steam net one sees that after about 4 700 s the pressure in the accumulator exceeds 12.5 bar. This is the set point of a controller, PC A1 see Figure 15 and Table 2, controlling the valve between the accumulator and MP. This controller is coupled to the valve via a MIN-selector and the valve is opened. To this MIN-selector another controller, PC M2, is coupled that gets its process value from MP and has the set point 11.1 bar. At the time about 4 800 s the pressure in MP is above 11.1 bar and now this controller limits the opening of the valve between the accumulator and MP and the pressure in MP stabilizes at slightly more than 11.1 bar. The change in MP (from 11.0 to 11.1 bar) has the effect that PC M1, which has the set point 11.0 bar, will close the valve between HP1 and MP. This makes the pressure in HP1 increasing fast, as seen at the top of Figure 14 after about 4 800 s. PC H3 opens up the valve to the accumulator and PC H5 decreases the amount of oil used in the bark boiler. This isn't sufficient at first so the PC H4 also opens up the airblow valve for a while. The signals described can be seen in Figure 16.

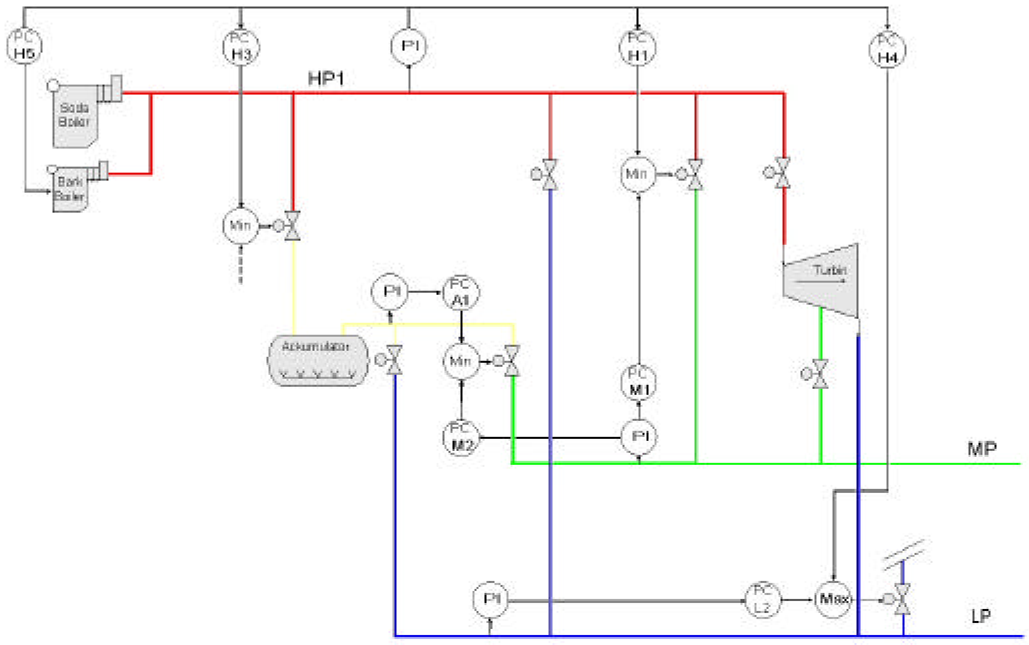


Figure 15, some controllers involved in the experiment presented in Figure 14

Controller	Set Point	Action
PC M1	11.0	Reversed
PC M2	11.1	Reversed
PC A1	12.5	Direct
PC L2	3.6	Direct
PC H1	52.5	Direct
PC H3	53.8	Direct
PC H4	55.5	Direct
PC H5	54.0	Reversed

Table 2, the controllers in Figure 15

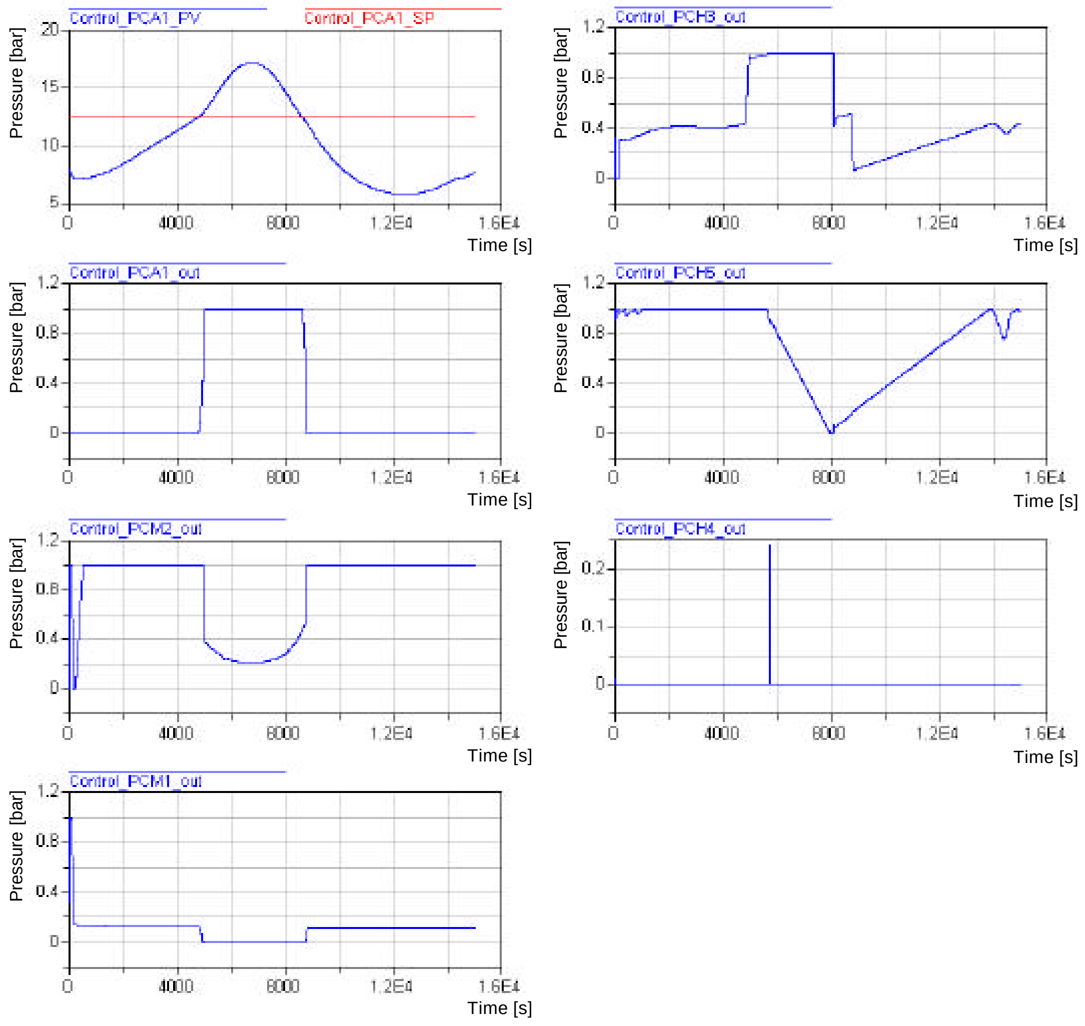


Figure 16, some controllers involved during normal operation

7 Simatic PCS 7

Siemens Simatic Process Control System 7 (PCS 7) is a control system with which you can both control and monitor the running of a process. The basic structure of the system can be seen in Figure 17. Each of the components will be explained below.[9]

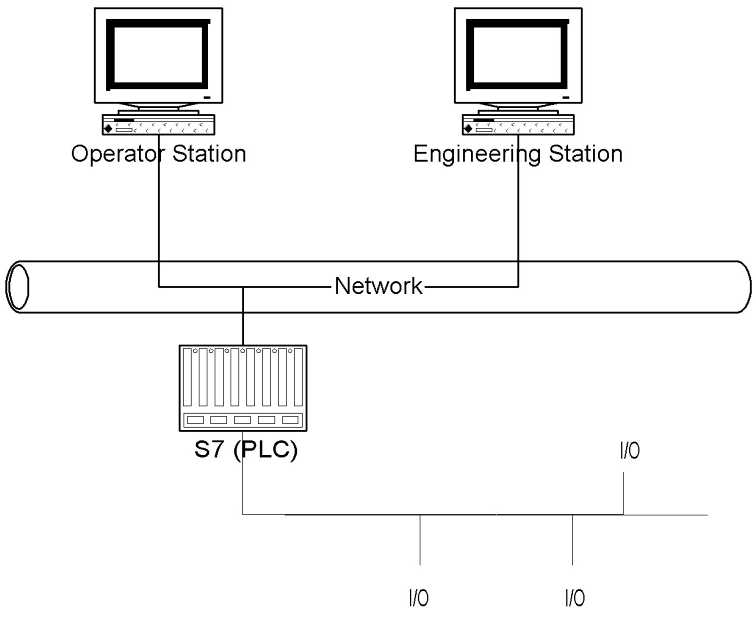


Figure 17, the basic structure of PCS 7

7.1 Operator Station

The Operator Station (OS) is used by the operators to monitor and affect the control of the process. The main program of the OS (besides the operating system which is Windows NT 4.0) is WinCC (Windows Control Center). WinCC is the HMI (Human Machine Interface) of the system. That means that WinCC is the interface between the operators (humans) and the process (machine), and makes communication in both directions possible. WinCC provides tools for developing a graphical user interface for the operators.

The operator can observe the process on the screen, where the process values are presented as plots, values or in any other way, e.g. bar charts. As an extra help the system will automatically signal alarms in the event of critical process status.

The operator can affect the control of the process in two ways. He can affect the automatic control of the system by changing for instance a set point or the gain of a controller. He can also affect the process immediately by, for instance, opening a valve.

7.2 Engineering Station

The engineering station is used to develop the process control system. The graphical user interface can be developed using tools in WinCC. The regulator system can be programmed using different software:

- Simatic Manager

  Represents the platform for all the engineering station components and manages them centrally.

- HW Config

  Tools for configuring the hardware, that is the arrangement of racks and modules.

- CFC

  CFC, Continuous Function Chart, is a graphical tool for building controllers from ready-made blocks in a block diagram. Blocks can easily be inserted into the CFC using “drag-and-drop” from libraries.

- SFC

  SFC, Sequential Function Chart, is a graphical programming language to describe sequential operations. SFC is built of steps, each step representing a state, with conditions in-between that has to be fulfilled to go to the next step.

- SCL

  SCL, Structured Control Language, is a programming language, oriented on Pascal, for programming complex automation tasks. In the engineering station SCL is required for compiling the controllers built in CFC and SFC.

When a control system has been created in the engineering station it can easily be downloaded to the real hardware that is the PLC.

7.3 PLC

The PLC (Programmable Logical Controller) is the computer that runs the real control of the process. The hardware of the PLC consists of a rack (mounting backplane that handles the connections between the inserted modules), a power supply unit, and a CPU with a memory card inserted. Normally the rack has one or more I/O-modules to connect the PLC to the process.

Instead of downloading your regulator system to the real PLC you could download it to Siemens Simatic S7-PLCSIM, which is software that simulates the behaviour of the PLC. It can be used to run and test the entire program and includes most features of the real PLC. The major difference is that the PLCSIM doesn’t run in real time due to the underlying operating system. [11]

7.4 Extensions

The Simatic PCS 7 can be used to build full-scale, complex control systems. Usually there is need for more than one Operator Station. Using an OS server one can connect many (16 per server) OS clients. In the paper process industry it is often insufficient with one PLC and hence they divide the control system into two or more PLCs. Unfortunately PLCSIM can only simulate the behaviour of one PLC.

The network is often an ordinary Industrial Ethernet but depending on the amount of data transferred, the number of nodes and the need for expandability, Fast Ethernet and Profibus could be used as well.

The connections to the process can be done in many ways too. Using I/O modules the process components are directly connected to the process transducers via conventional cabling. This might be good with a simple process with a small number of transducers, but with larger processes there is a better solution. Instead of using central I/O modules placed in the rack, distributed I/O modules, e.g. ET 200M, can be placed at strategical places in the factory. The distributed modules are still connected to the process transducers via conventional cabling but all the I/O modules can be connected to the PLC via a field bus, e.g. Profibus DP. An example of how a more complex and real Simatic PCS 7 can be configured can be seen in Figure 18.

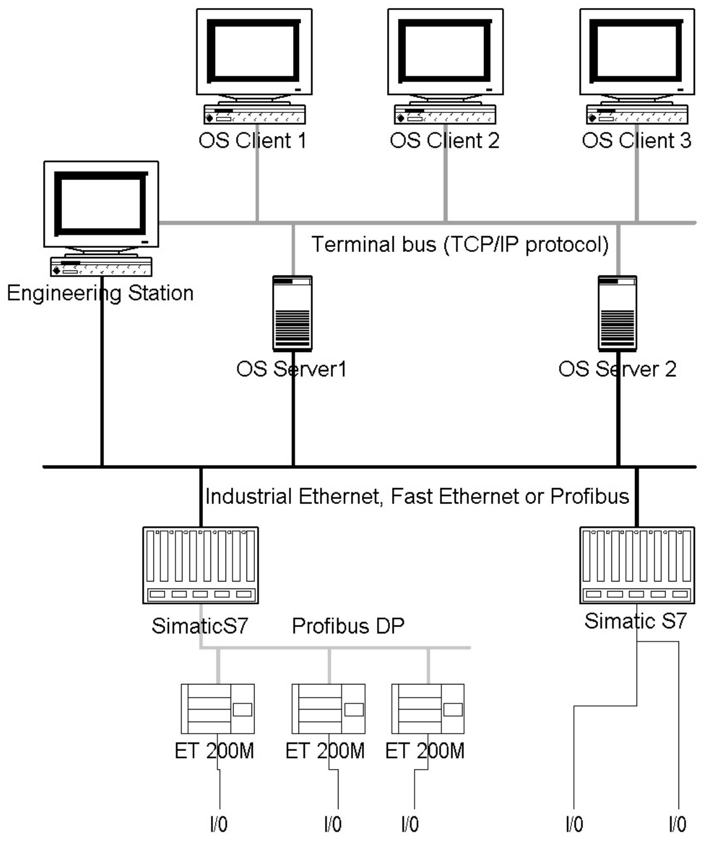
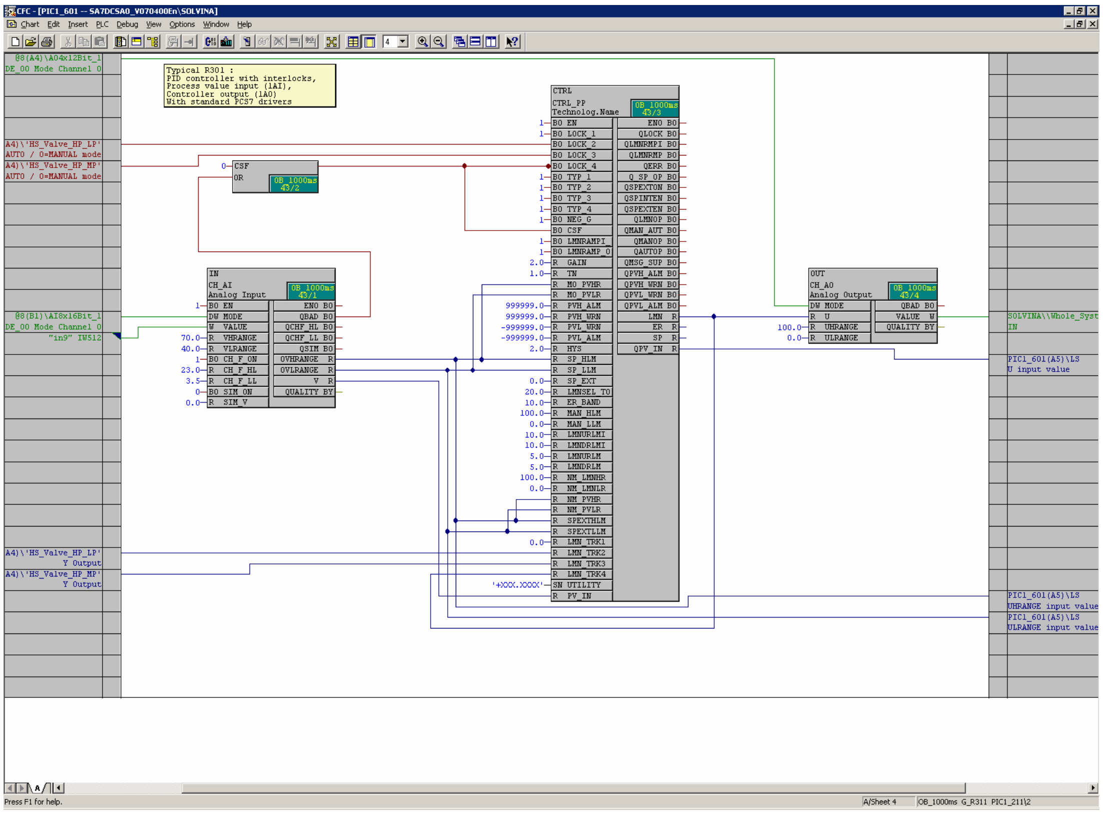


Figure 18, example of a PCS 7 configuration

8 The real control system

Since this master thesis was not a part of a real project and no customer was involved there was no access to the real control system. Hence the control system had to be built once more, now in the real environment of Siemens Simatic PCS 7. This was possible thanks to a lot of help from people at Siemens.

As mentioned in section 7.2 above the engineering station provides tools for building control systems using CFC, Continuous Function Charts. Using this, blocks representing PID-controllers, I/O cards, mathematical operations and many other things could be used and connected to each other. En example of a controller can be seen in Figure 19. An analogue input and an analogue output card are used to convert signals from the desired range. The block “CTRL_PP” is the actual PID-controller providing among others antiwindup and tracking of four different signals with different priorities. Every controller is built as a chart like in Figure 19 and they are then connected to another chart containing the MIN and MAX selectors, as seen in Figure 20. The outputs at the right-hand side of Figure 20 are in turn connected to manual controllers just as described in chapter 6 above.



CFC - [PIC1_601 -- SA7DCSA0_V070400En\SOLVINA]
Chart Edit Insert PLC Debug View Options Window Help
Typical R301 : PID controller with interlocks, Process value input (1AI), Controller output (1AO) With standard PCS7 drivers
CTRL
CTRL_PP
Technolog.Name
OB_1000ms 43/3
IN
CH_AI
Analog Input
OB_1000ms 43/1
OUT
CH_AO
Analog Output
OB_1000ms 43/4
CSF
OR
OB_1000ms 43/2
A4)\'HS_Valve_HP_LP' AUTO / 0=MANUAL mode
A4)\'HS_Valve_HP_MP' AUTO / 0=MANUAL mode
A4)\'HS_Valve_HP_LP' Y Output
A4)\'HS_Valve_HP_MP' Y Output
SOLVINA\\Whole_Syst IN
PIC1_601(A5)\LS U input value
PIC1_601(A5)\LS UHRANGE input value
PIC1_601(A5)\LS ULRANGE input value
Press F1 for help.
A/Sheet 4
OB_1000ms G_R311 PIC1_211\2

Figure 19, some CFC components used in the control system

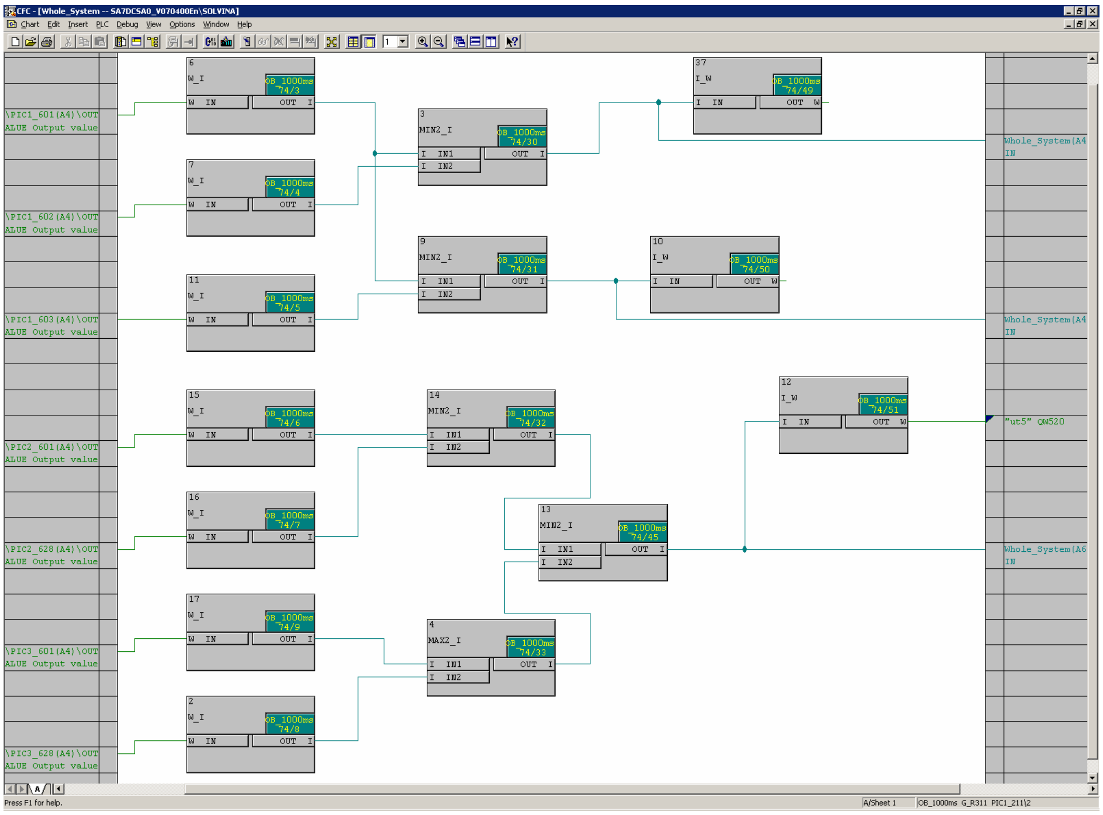

CFC - [Whole_System -- SA7DCSA0_V070400En\SOLVINA]
Chart Edit Insert PLC Debug View Options Window Help
MIN2_I
MAX2_I
W_I
I_W
OB_1000ms
Press F1 for help.

Figure 20, MIN and MAX selectors in PCS 7

Siemens Simatic PCS 7 also provides tools to build a graphical user interface that can be used by the operators in the Operator Station. It is possible to connect pictures to the charts making it possible to interact with the control system in a very convenient way. As mentioned in chapter 7.1, the operator has the ability to both monitor and affect the behaviour of the system in a real Operator Station. The operator gets information about the current state of the steam net from among others indicators and plots. If something is not normal the system will signal an alarm. The operator can then affect the system by for instance changing a set point or putting a controller into manual operation. All of this can be used even if the process is simulated so that the operators can be trained in exactly the same graphical interface as they are used to, but on the simulated process. A view of the graphical user interface can be seen in Figure 21.

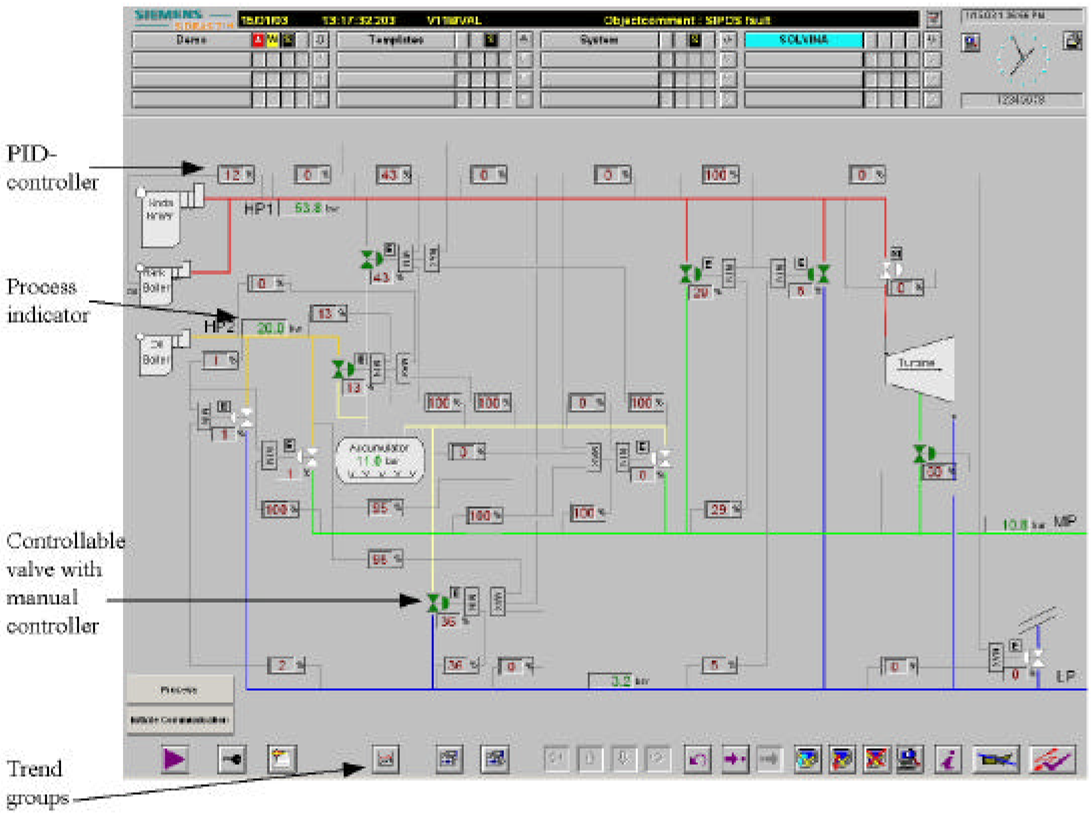
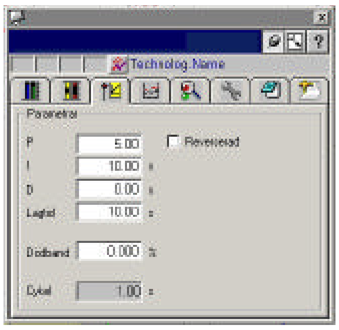


Figure 21, the graphical user interface for the control system

Clicking on a PID-controller opens a window, a group display of the display block CTRL_PP, in which you can see the current state and settings of the controller and change for instance the gain, the integration time and the levels on which to signal alarms, see Figure 22.


Technolog. Name

Parameter

P	5.00	
I	10.00	s
D	0.00	s
Lagtid	10.00	s
Dödband	0.000	%
Cykel	1.00	s

☐ Reverserad

Figure 22, group display of the display block CTRL_PP, Parameters view

The trends of the process values and the outputs of each controller can also be seen in the group display of the controller block, in flap four. It is possible to group many trends so that they can be displayed together, by clicking on the fourth button from the left on the main window, see Figure 21.

9 Communication between Simulated Process and Control System

To be able to control the Dymola process with the PCS 7, output signals from the simulated process has to be transmitted to the control system as input signals and output signals from the control system has to be transmitted back to the simulated process as input signals. This transmission has to be synchronized and happen every sample time.

PLCSIM emulates the real PLC and can be run on the same computer as the process. Letting PLCSIM run the control system thus makes it possible to get an integrated system on one computer. It will not be the real control system that controls the simulated process, from the point of view of hardware, but it will indeed be the real control system when it comes to software and it will behave in the same way. The four advantages of using the real control system, listed in section 1.1 above, will still be valid.

Hence the problem is reduced to communicate between two different applications running on the same operating system, Windows NT. This can be done in many different ways. The way chosen in this master thesis is presented in Figure 23 and is explained below.

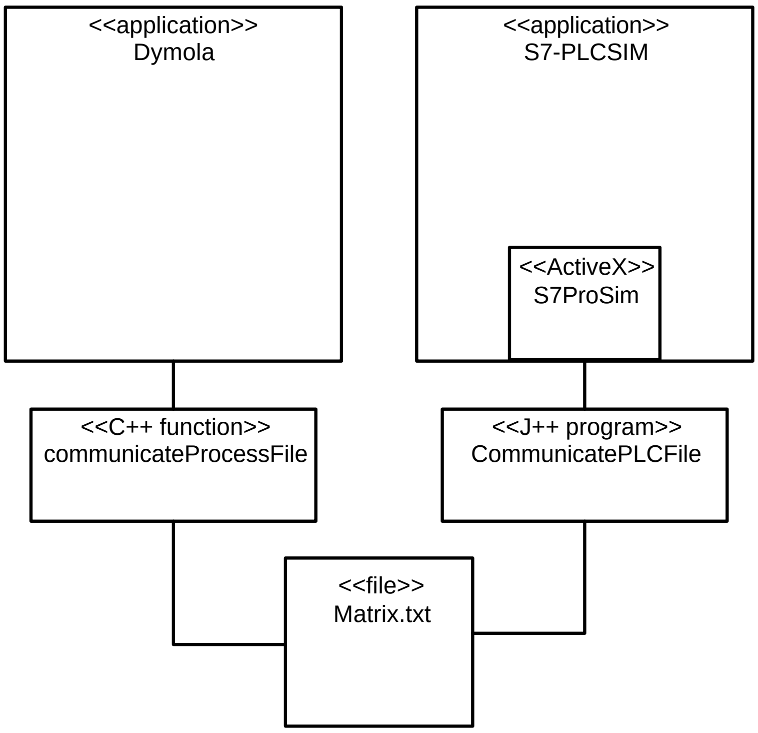


Figure 23, program structure of the communication

Since two applications, running on the same operating system, should communicate with each other, they need to access the same data somehow. A straightforward and general way to do this is to let the data be in a file (an ordinary .txt file). This file can then be written to and read from by two different programs, one transmitting data between the file and Dymola, and one transmitting data between the file and PLCSIM.

Dymola supports linking C/C++ libraries of one's own and using functions in those libraries. Hence, it is possible to write a function

double[] communicateProcessFile(double[] processOutput)

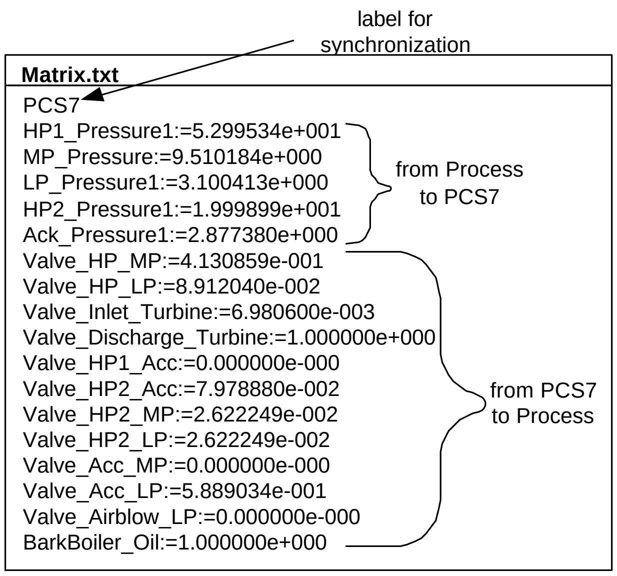
that takes the process output signals from the last sample period as an argument and returns the process input signals for the next sample. The C++ function in turn writes the process output signals to the file and reads the next process input signals from the file.

Simatic PCS 7 on the other hand provides an ActiveX control called S7ProSim. ActiveX is Microsoft's standard for making applications on the same net, running on Windows operating system, share information and interact with each other. ProSim provides programmatic access to the process simulation interface of PLCSIM and has methods for connecting/disconnecting, writing input signals, reading output signals, executing scan cycles etc. It also sends events to the user to report that e.g. a scan cycle is finished.

This was used so that a new program, a J++ program called CommunicatePLCFile, was created that establishes a connection to the control system via the ActiveX control ProSim, fetches the output signals from the control system which it writes to the file and reads next control inputs signals from the file and pushes them to the control system, via ProSim.

9.1 The communication media – the file

The file is called Matrix.txt since it mainly is a matrix, consisting of signals from the process to the control system and from the control system to the process. It also contains a label, for synchronization, that tells which application used the file last. It is left to the applications to access the file using mutual exclusion, that is the applications shall only use the file if no other application is using it simultaneously. When an application have got access to the file it should also check the label to assure that it was not this application that used the file last. If it was it leaves the file and tries again later. If it wasn't, the application changes the label, reads the input signals for the application from the file, prints the output signals from the application to the file and then leaves the file. The contents of the file can be seen in Figure 24. This file was used by PCS7 the last time, as indicated by the label. Otherwise the label would be "Process". The names of the signal are present just to increase legibility, what's interesting are the values. The values are represented in an exponential notation with six decimals and three digits representing the power.


label for synchronization
from Process to PCS7
from PCS7 to Process

```
Matrix.txt
PCS7
HP1_Pressure1:=5.299534e+001
MP_Pressure:=9.510184e+000
LP_Pressure1:=3.100413e+000
HP2_Pressure1:=1.999899e+001
Ack_Pressure1:=2.877380e+000
Valve_HP_MP:=4.130859e-001
Valve_HP_LP:=8.912040e-002
Valve_Inlet_Turbine:=6.980600e-003
Valve_Discharge_Turbine:=1.000000e+000
Valve_HP1_Acc:=0.000000e-000
Valve_HP2_Acc:=7.978880e-002
Valve_HP2_MP:=2.622249e-002
Valve_HP2_LP:=2.622249e-002
Valve_Acc_MP:=0.000000e-000
Valve_Acc_LP:=5.889034e-001
Valve_Airblow_LP:=0.000000e-000
BarkBoiler_Oil:=1.000000e+000
```

Figure 24, the contents of the communication file Matrix.txt

9.2 Communication between Dymola and the file

The communication between the process in Dymola and the file takes some time. In order to make it more realistic this time has to be approximately zero, or at least make it appear as it is zero. The latter is possible in Dymola. The operator *sample*(0,T) returns true at sample instants with sampling time T, and used in a when clause like

```
when sample(0, 1) then
   ControlSignals = communicateProcessFile(ProcessValues)
end when;
```

the result will be the desired. The equations inside the when clause are called at every sampling instant and the evaluation of the expressions is performed in zero simulated time. No matter how much time the C++ function *communicateProcessFile* takes it will still appear as no time to the simulation.

The function *communicateProcessFile* makes use of functions for example to open and close the file, and thus it includes different libraries. To be able to use those libraries in the Dymola model as well, a C++ library, myLibrary.lib, has been built which includes the function *communicateProcessFile* plus the included libraries. This library is used in the Dymola model in the following way:

```
model SteamNet_PCS7
   Process Process1( ); //the model of the process
   ControlSystem ControlSystem1( ); //the model of the control system

   constant Integer nbrPO=5; //number of process outputs
   constant Integer nbrPI=12; //number of process inputs (control signals)

   Real processValues[nbrPO];
   Real controlSignals[nbrPI];

   function communicateProcessFile
      annotation (Library={"myLibrary"});
      input Real fromProcess[nbrPO];
      output Real toProcess[nbrPI];
      external "C" communicateProcessFile(fromProcess, nbrPO, toProcess, nbrPI);
   end communicateProcessFile;

   equation
   when sample(0, 1) then
      processValues[1] = ( Process1.PI_HP1.signal[1] );
      processValues[2] = ( Process1.PI_MP.signal[1] );
      processValues[3] = ( Process1.PI_LP.signal[1] );
      processValues[4] = ( Process1.PI_HP2.signal[1] );
      processValues[5] = ( Process1.PI_Acc.signal[1] );

      controlSignals = communicateProcessFile( processValues );

      Process1.Signal_PV_HP_MP.signal[1] = controlSignals[1];
      Process1.Signal_PV_HP_LP.signal[1] = controlSignals[2];
      Process1.Signal_PV_Inlet_Turbin.signal[1] = controlSignals[3];
      Process1.Signal_PV_Discharge_Turbin.signal[1] = controlSignals[4];
      Process1.Signal_PV_HP1_Acc.signal[1] = controlSignals[5];
      Process1.Signal_PV_HP2_Acc.signal[1] = controlSignals[6];
      Process1.Signal_PV_HP2_MP.signal[1] = controlSignals[7];
      Process1.Signal_PV4_HP2_LP.signal[1] = controlSignals[8];
      Process1.Signal_PV_Acc_MP.signal[1] = controlSignals[9];
      Process1.Signal_PV_Acc_LP.signal[1] = controlSignals[10];
      Process1.Signal_PV_Airblow_LP.signal[1] = controlSignals[11];
      Process1.Signal_PV_Oil2.signal[1] = controlSignals[12];
   end when;
end SteamNet_PCS7;
```

On every call the external C++ function goes through a number of steps:

1. Open the file in a totally unsafe mode, i.e. it will be opened regardless of whether the file is open by another program and it allows the file to be opened by other programs after it is opened by this function.
2. Try to lock a region of the file that reaches from the beginning of the file pass the end of the file. If the attempt is unsuccessful, it means that the file is used by another program. In this case the function waits for a while and tries again. This is repeated until the region is successfully locked.
3. The label of the file is checked. If the label is “Process” it means that this function used the file last, the region is unlocked, the file is closed, the function waits for a while and then goes back to step 2. Otherwise the label is changed to “Process” and the function proceeds to step 4.
4. The control signals are read from the file and the output signals (process values) from the process are written to the file.
5. The region is unlocked and the file is closed.

One might wonder why the file is not opened in a safe mode instead of opening the file in an unsafe mode and then lock the entire region of the file. This will be explained below when describing the communication between the control system and the file. The entire code of the external function *communicateProcessFile* can be seen in Appendix A - communicateProcessFile.

9.3 Communication between S7-PLCSIM and the file

As mentioned above S7-PLCSIM has an ActiveX control that can be used that provides programmatic access to S7-PLCSIM. Although ordinary Java does not support the use of ActiveX, J++, which is Microsoft's tool to create Java code, does. Using this all inputs and outputs of the PLC can be set and read from respectively.

One Java class called *CommunicatePLCFile* performs both the communication with PLCSIM and the communication with the file. It makes use of some private help-methods and of ActiveX classes and other classes for communication.

Microsoft provides a package for communication, com.ms.wfc.io, which among other things contains a file class called File. Using this class it is possible to open a file in a safe mode, that it is will only be opened if no other process is using it and it will allow no other process to read or write to it while this process has access to it. The problem it that the opening and the closing of the file take approximately 1 s for each operation and since both operations have to be performed every sample time this delay is intolerable. Instead the file is opened only once in a totally unsafe mode and every time the file is to be used a region of the file, that reaches from the beginning of the file pass the end of the file, is tried to be locked, just as explained in section 9.2 above. This result is the same as if the file could be opened and closed fast in a safe mode.

The entire code of the J++ program *CommunicatePLCFile* can be seen in Appendix B - CommunicatePLCFile.

10 Putting it all together

The overall communication between the process and the control system has to be performed every sample time. As mentioned in section 9.2 above the simulation waits for the control system to tell that a new clock cycle has begun. Hence the J++ program has to be able to get information about the clock cycle from the control system and communicate with the file when a new clock cycle has begun. PLCSIM updates the output signals every clock cycle, of course, but it doesn't tell exactly when this happens. To get this information a block, that simply contains a counter and increases the counter with one every sample, was added to one of the charts. The value of the counter is placed as an output signal of the PLC, the same way as the control signals, and is used in the J++ program. This block was kindly provided by Mikael Börjesson at Siemens AB.

10.1 The delay of the system

The real control system is discrete with the sampling period one second. The input signals in the current sample are used to calculate the output signals in the next sample. Hence the control signals are delayed one sample period compared to the continuous counterpart. When experimenting with the real control system it seamed that the delay was even greater than one sample. To determine how big the delay was some experiments were made where the pressure of HP1 was sent to the control system, in through an input card and directly back through an output card. Doing this the signal out from the control system should be delayed one sample compared to the signal sent into the control system, but it was actually delayed two samples, as seen in the left part of Figure 25. If the J++ program is changed so that at first it sends the process values to the PLC, then waits for the next clock cycle and finally reads the control signals, the delay can be decreased to the desired one sample, as seen in the right part of Figure 25.

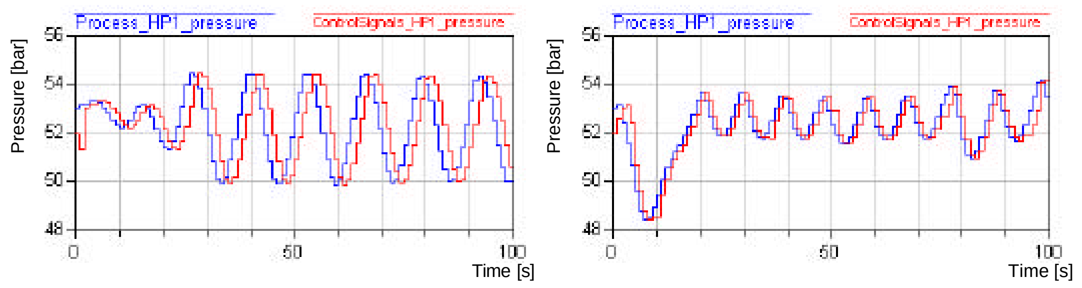


Figure 25, the delay of the control system. In the experiment represented in the left graph the J++ program writes the inputs, reads the outputs and then waits for the next sample. In the experiment represented in the right graph the J++ program writes the inputs, waits for the next sample and finally reads the outputs.

A lot of work has been spent on investigating this extra delay but nothing has shed light on the cause of the delay.

PLCSIM can be run in two different scan modes, continuous scan and single scan. A scan consists of the CPU reading the inputs, executing the program and then writing the results to the outputs. In continuous scan, which was used in the previous example, the CPU executes one complete scan and then automatically starts another scan. In single scan the CPU executes one scan and then waits for the user to initiate another scan. The manual for S7ProSim says that the S7-PLCSIM has to be in single scan mode if it is to be used with an attached process simulation [12], which is the case here, but it doesn't explain why. Letting the J++ program start new scans continually it is possible to let the PLCSIM be in single scan mode, but the delay is the same as for continuous scan, as seen in Figure 26. Using single scans the J++ program has to execute a new scan just after the previous one was executed. This has the effect that the control system cannot be run in real time but is actually a little slower.

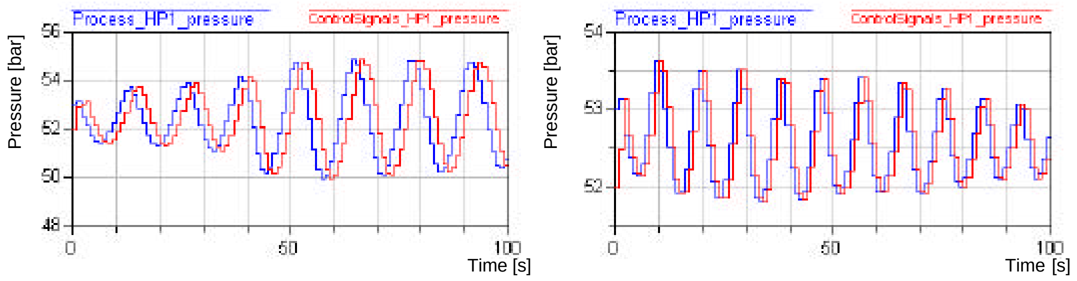


Figure 26, the delay of the control system when the PLCSIM runs in single scan mode. In the experiment represented in the left graph the J++ program writes the inputs, reads the outputs and then ticks to the next sample. In the experiment represented in the right graph the J++ program writes the inputs, ticks to the next sample and finally reads the outputs.

With this knowledge it was decided that the PLCSIM should be run in continuous scan mode and that the J++ program should write the process values to the control system, then wait for the next clock cycle and finally read the control signals. In this way the desired behaviour is achieved.

10.2 Running the system

10.2.1 Prerequisites

To make the entire system work, several programs have to be running. First the control system that is PLCSIM has to be running. Second the J++ program has to be started. The J++ program creates the file Matrix.txt and then waits for the process to connect. When finally the simulation in Dymola is started the whole system is running sending signals in both directions every sampling time and it's possible to experiment with the whole system, monitoring and changing parameters in the operators station.

If the system is to be used to tune the control system or performing other experiments with the system, it might be a good idea to have the entire Dymola program running so the state of the process can be investigated in detail. It might also be a good idea to have Microsoft Visual J++ run the J++ program to be able to get information about the communication. If the system is to be used for operators training both the communication and the simulation can be started from the graphical user interface in the Operators Station. As the reader might have noticed, the graphical user interface, as presented in Figure 21 on page 27 above, contains two buttons at the bottom left with the labels “Initiate Communication” and “Process”. Clicking on the button “Initiate Communication” starts the J++ program as an .exe file. Clicking on the button “Process” starts the simulation. The simulation program, Dymosim, that is used to perform simulations in Dymola, can actually be run as a stand-alone program dymosim.exe. The program reads the experiment inputs from an input file and performs the simulation. The program file dymosim.exe and the file with the experiment inputs, dsinit.txt, thus contain all information necessary to run the simulated steam net. [1]

When performing pure simulations in Dymola it is often desirable to run the simulation as quick as possible. An integration method with variable step-size, for instance DASSL, is then often preferred. Such an integration method is often faster than an integration method with fixed step-size. When the system is changing fast the method with variable step-sizes might however be quite slow. To avoid this change in integration time an integration method with fixed step-size is used for real-time simulations. In the experiments done in this master thesis the integration method Euler was used with a step-size of 0.01 s. [1]

10.2.2 Performance of the communication

The J++ program should read the process values from the file and print the control signals to the file, every sample time. If this isn’t done it writes a message to the standard output. During normal operation this never happens. The behaviour can be forced, by for example dragging a window around really quickly for over a second, but it hasn’t occurred when not really forced.

10.2.3 Performance of the control system

When experimenting with the steam net and the continuous control system made in Dymola the pressures of the nets sometimes oscillates with a quite short period. In the experiments presented in Figure 13, the pressures of both HP1 and LP oscillate in the beginning. This can be seen in Figure 27, where the pressures of HP1 and LP are presented for the first 500 seconds of the experiment. Similar oscillations have been seen in other experiments too, indicating a certain sensitivity of the net.

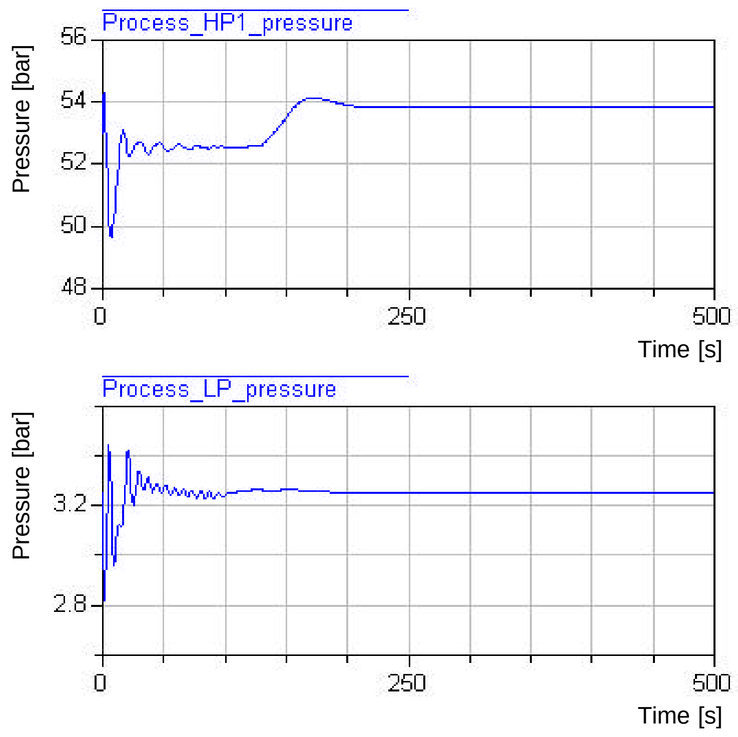


Figure 27, oscillations of the pressures of the net. Dymola continuous control system used.

Doing a similar experiment using the real control system instead results in a very oscillating system as seen in Figure 28.

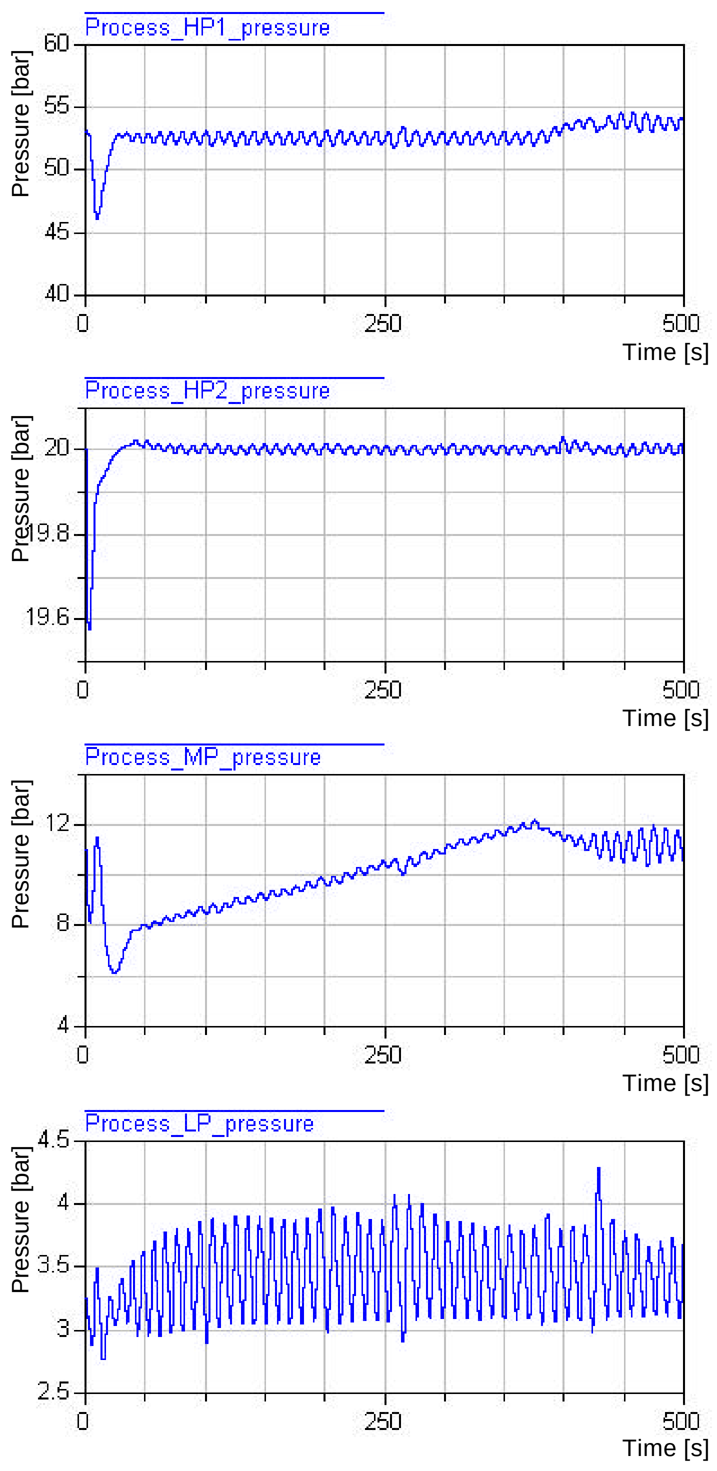


Figure 28, oscillations of the nets using the PCS7 control system

This control system is discrete with the sampling period one second and has a delay of one sample, as mentioned above. Is it the sampling that causes the oscillations? To be able to answer this question unit delays, that both samples and delays the signals, were placed at every control signal in the Dymola control system. Using this control system the same kind of oscillations appeared, as seen in Figure 29. The steam net is started in the same state in both experiments but the integral parts of the controllers are not the same so the two experiments are not expected two give exactly the same results.

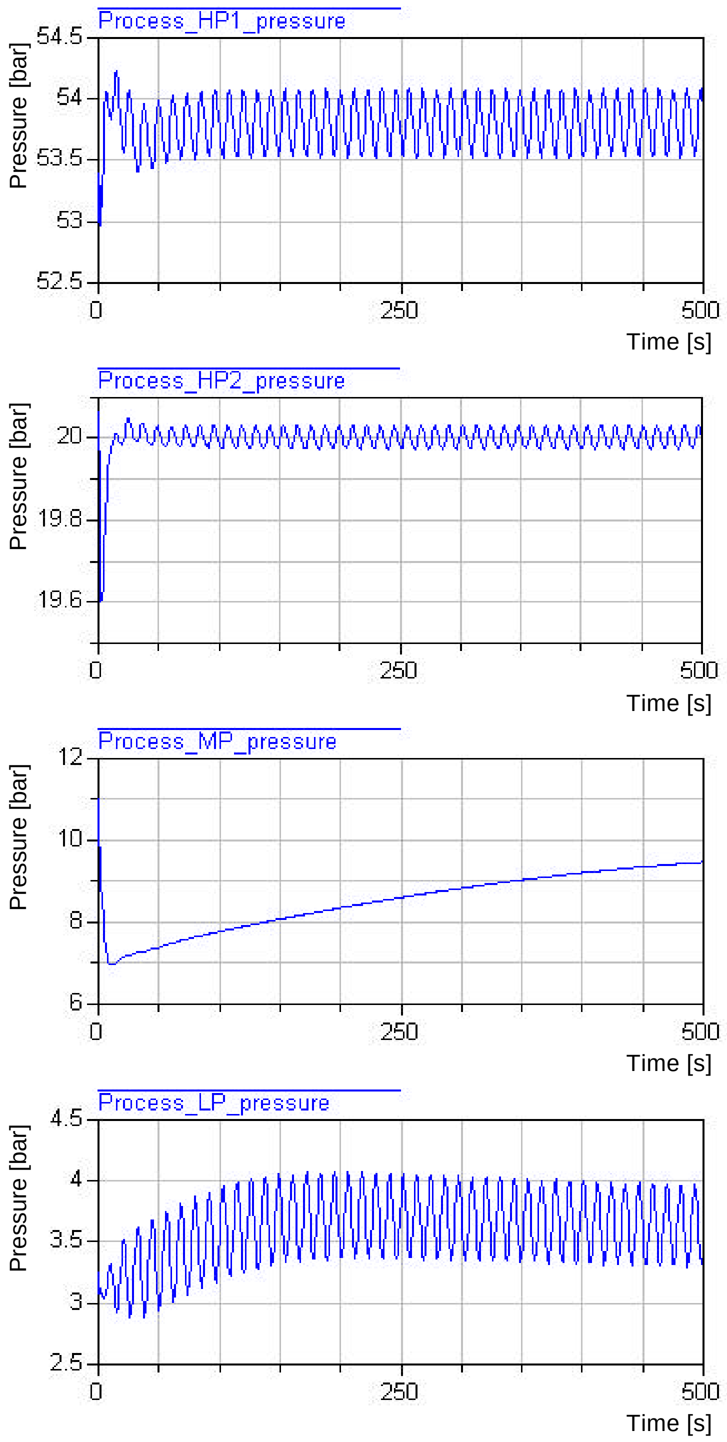


Figure 29, oscillations of the nets using the Dymola control system with unit delays at the control signals

Those oscillations should, of course, not appear in real steam nets. Whether they come from a too sensitive simulated steam net or a poor control system is hard to tell.

The delay of one second, due to the sampling, seams to be a problem but this sampling time is often used in the paper process industry [13], and probably the problem comes from a too sensitively simulated process. In this master thesis there was neither time nor data to make a really good model of the steam net. Nevertheless it is desirable to have a better working system to demonstrate. Making some small changes of the control system, however, makes it possible to get rid of the oscillations, as will be explained below.

The valves between HP1 and MP/LP have great capacities and fluctuating control signals of those will affect the whole net. If they are put into manual control it is possible to get the whole system stable, but when changed back to automatic mode the net starts to oscillate again. This gave rise to the idea to include dead band on some of the controllers. The dead band is used so that the control deviation used in the PID-algorithms, *ER*, is set to zero if the real error, *SP – PV*, is less than the limit *DEADB_W*, see Figure 30.

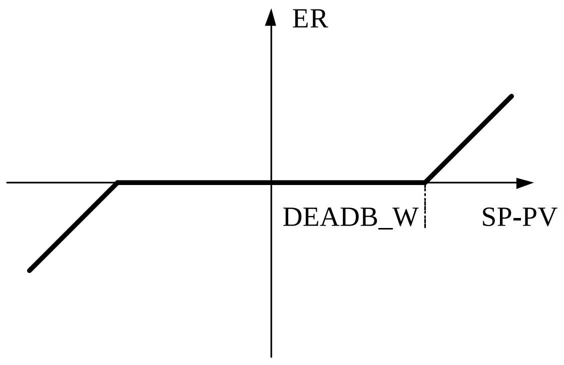


Figure 30, dead band used to generate the control deviation

Dead band has the effect that the controller will not react upon small changes in the process value if the error is small and this could be useful to get rid of the oscillations. Dead band also has the effect that the controller does not try to control the process value to the set point. Thus, dead bands should only be used if necessary; they should be relatively small and only be used on some controllers. The controller block, CTRL_PP, used in this PCS7 control system provides the possibility of using dead bands, which is something that are actually used in process control with the restrictions just mentioned [13]. The reason for using dead bands on a real control system is often to get rid of measurement disturbances. That is not the case here, but yet it could be useful to get better control, without making too big changes in the control system. If a dead band of 0.2 bar is placed on PC M1 and PC L1, see Figure 31, the oscillations could be reduced.

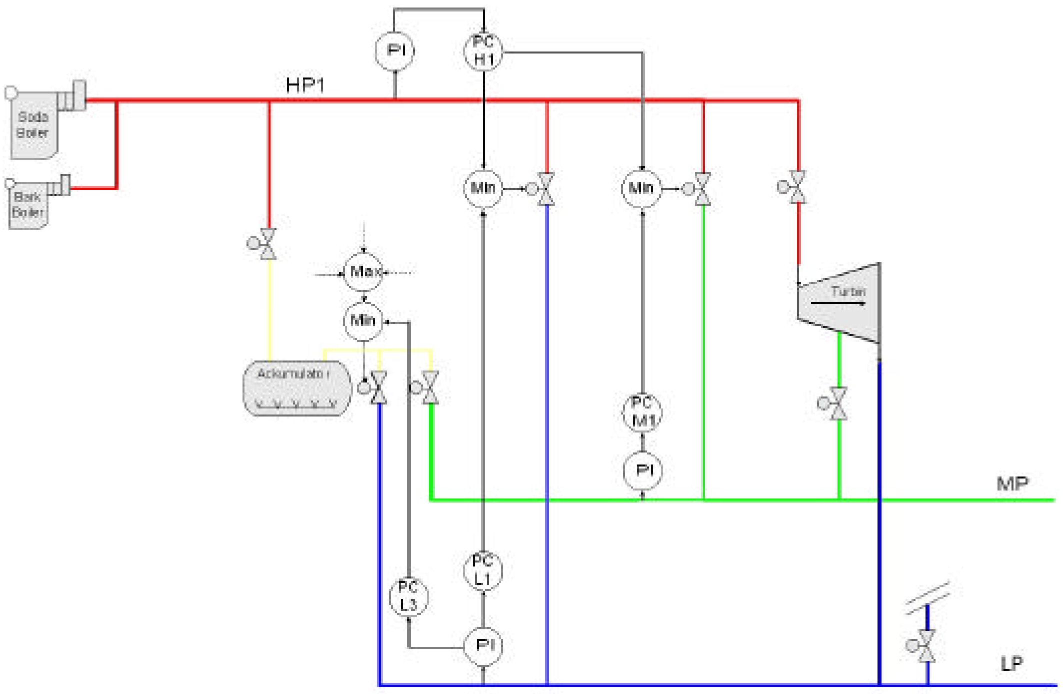


Figure 31, some controllers affected by the dead bands

An experiment in 10 000 seconds, with dead bands on PC L1 and PC M1 is presented in Figure 32.

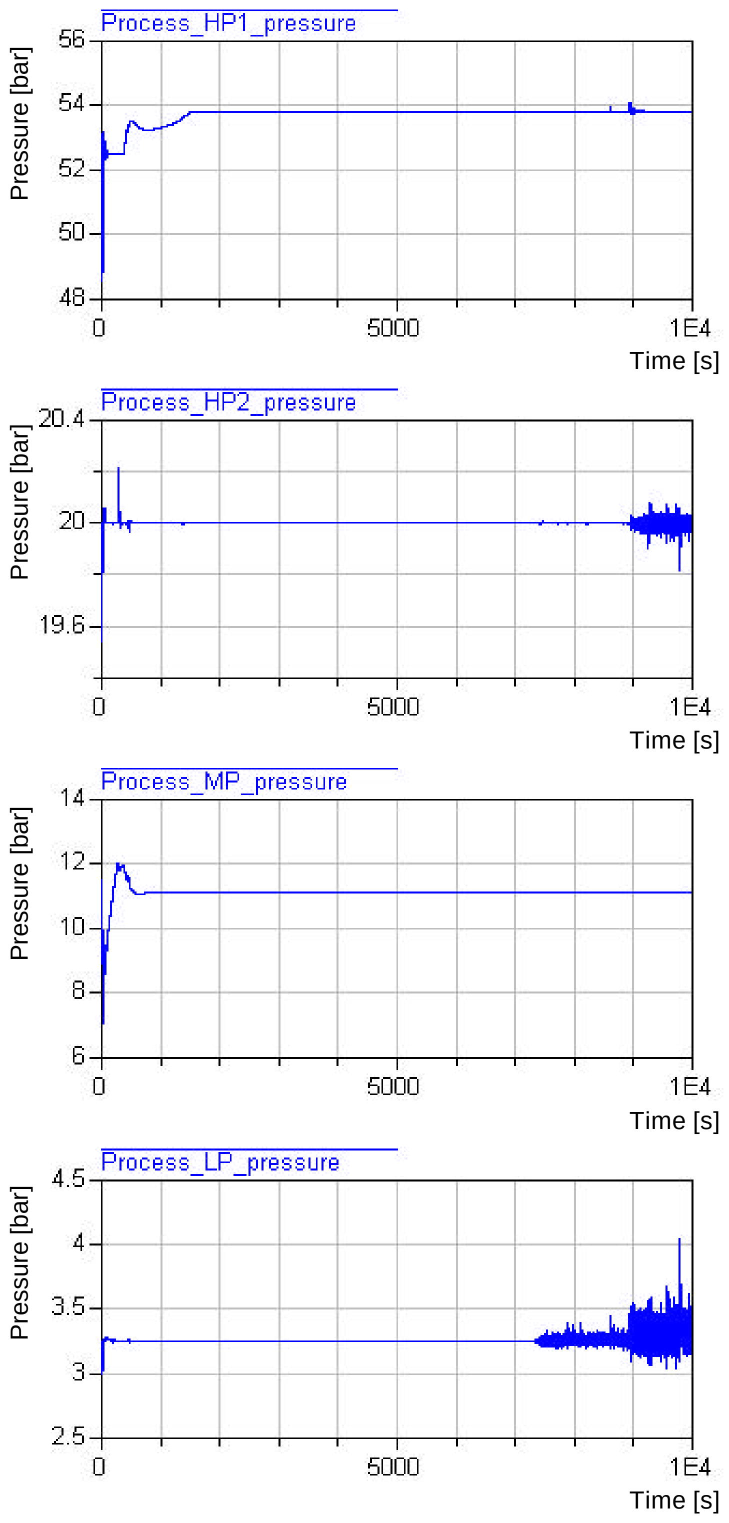


Figure 32, pressures of the net using the PCS7 control system with dead bands on PC M1 and PC L1

The set point of PC L1 is 3.2 bar but since the dead band is 0.2 bar the pressure of LP is instead 3.25 bar, which is the set point of PC L3, see Figure 31 above. After a little more than 7 000 seconds the pressure of LP starts to oscillate again. This can be avoided by putting a dead band on PC L3. The effect of putting a 0.2 bar dead band on PC L3 can be seen in Figure 33, where the dead band is activated after 7 750 s.

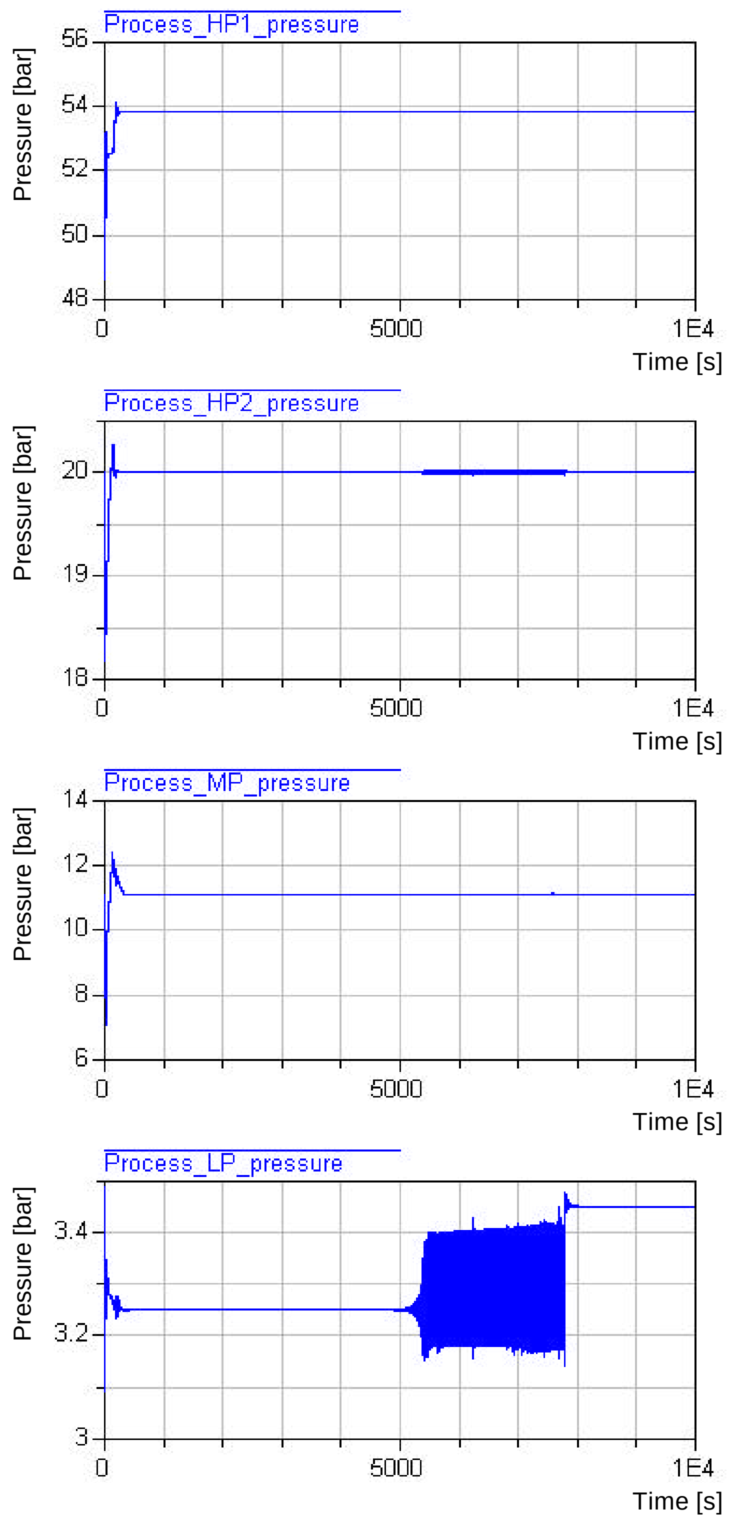


Figure 33, an 0.2 bar dead band on PC L3 is activated after 7 750 s.

The oscillations are expired but the pressure in LP is 3.45 bar, which is a little too high. If the dead band on PC L3 is only 0.05 bar, the pressure of LP is at a more tolerable level and the result of such an experiment can be seen in Figure 34.

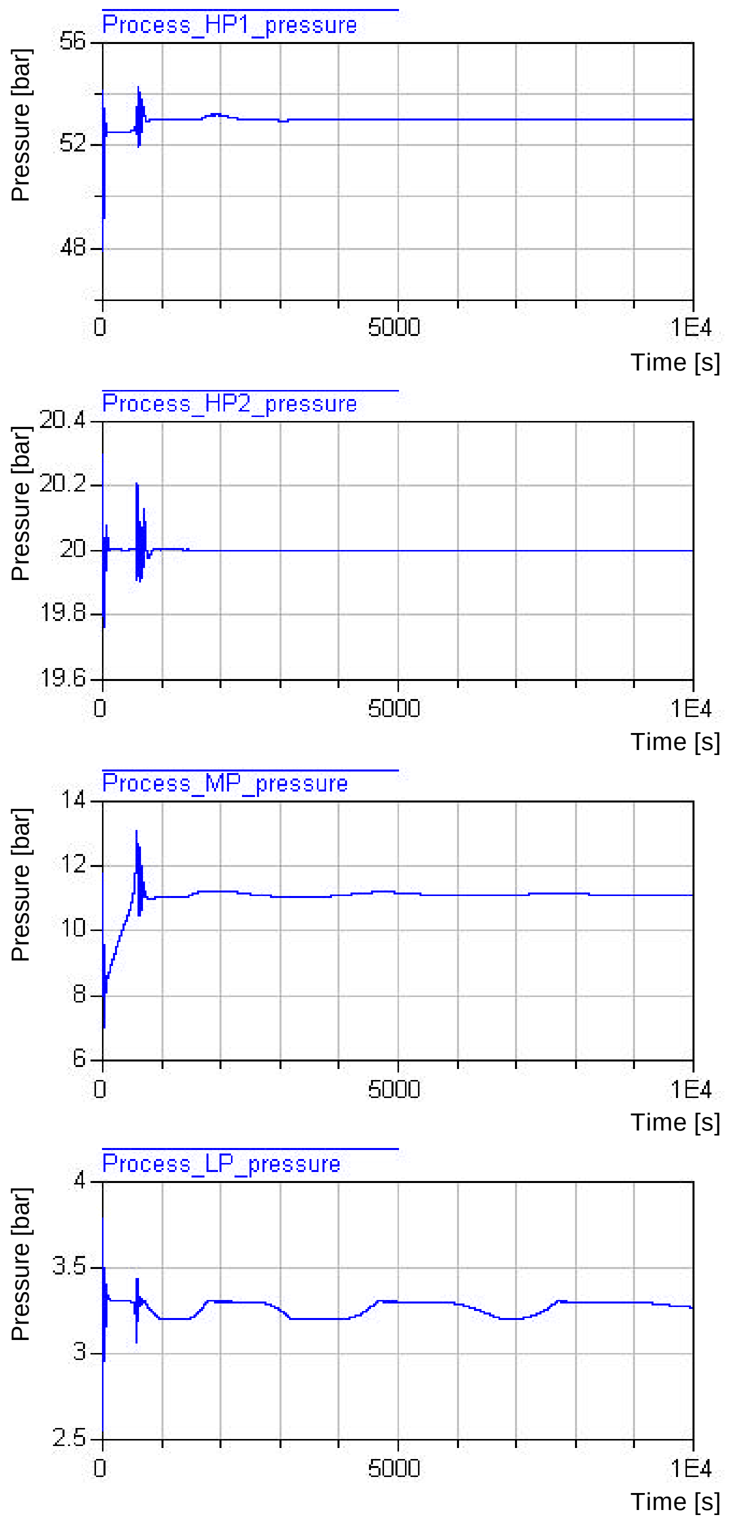


Figure 34, 0.2 bar dead band on PC L1 and PC M1. 0.05 bar dead band on PC L3.

11 Conclusions and suggestions for further investigations

In this master thesis it is shown that it is possible to use a real control system, written in Siemens Simatic PCS 7 and with the sampling period one second, to control a simulated process, written in Dymola. The control system and the process run on the same PC, under Windows NT. The simulated steam net and the control system has both been built to be used in the investigations in this master thesis, but they are based on data from a real control system.

If the solutions presented should be used in a real project some improvements have to be made. At first some work has to be spent on making a better model of the process and to validate and compare with the real process. Solvina has done this in previous projects so it shouldn't be a problem, although it takes some time.

In a real system much more signals than in this demo system are sent between the process and the control system, for instance start signals to the motors, acknowledgement signals and signals to lock the valves. Such signals have not been included in the demo system presented here but they will probably not cause too much problem. The J++ program could perhaps be extended to handle such signals or the control system could be extended, using sequential function charts, to handle acknowledgements of different signals.

S7-PLCSIM can only simulate the behaviour of a single PLC. Since the paper process industry sometimes uses two or more PLC:s to control their process it is desirable to investigate how to handle that situation. Is it possible to divide the system into two, investigating once at a time? Another possibility is to try to communicate with real PLC:s instead using OPC, which is a standardized set of interfaces, properties and methods to be used in applications for process control and automated production [9][15]. This has to be investigated further if it is to be used in such an application.

12 References

[1] Dynasim AB, *Dymola Dynamic Modeling Laboratory User's Manual*, Version 5.0

[2] Tummescheit, H, *Design and implementation of Object-Oriented Model Libraries using Modelica*, Department of Automatic Control, Lund Institute of Technology, 2002

[3] Ekström, F, *Dynamisk modellering av ångpanna*, Control and Automation Laboratory, Department of Signals and Systems, Chalmers University of Technology, 2001

[4] Alwarez, H, *Energiteknik*, Studentlitteratur, 1990

[5] Gustafson, B-A and Nilsson, E, *Kompendium i Strömningsmaskinteknik*, Institutionen för Strömningsmaskinteknik, Chalmers Tekniska Högskola, 1986

[6] Gustafson, B-A, *Kompendium i Turbomaskiner M3*, Institutionen för Strömningsmaskinteknik, Chalmers Tekniska Högskola, 1986

[7] United Nations Economic and Social Commission for Asia and the Pacific, *Guidebook on Cogeneration as a means of pollution control and energy efficiency in Asia, part 1*, http://www.unescap.org/enrd/energy/co-gen/, October 24, 2002

[8] Årzén, K-E, *Real-Time Control Systems*, Department of Automatic Control, Lund Institute of Technology, 2001

[9] Siemens, *Simatic Process Control System PCS 7 Configuration Manual*, Edition 08/2001

[10] Toijer, D, *Distribuerade in- och utgångar för fältbuss*, Automation #8, 2002

[11] Siemens, *Simatic S7-PLCSIM V5.0 User Manual*, Edition 06/2001

[12] Siemens, *Simatic S7ProSim Version: 5.0 User Manual*, Edition 06/2001

[13] *Mikael Börjesson*, Sales and Systems Engineer, Automation and Drives, Siemens AB, personal contacts, autumn 2002

[14] *Jonas Engblom*, Systems Engineer, Industrial Solutions & Services, Metals, Mining & Paper Technologies, Siemens AB, personal contacts, autumn 2002

[15] Pagina.se, *Paginas IT-ordbok*, http://www.pagina.se/itord/, January 20, 2003

Appendix A - communicateProcessFile

```
#ifndef _communicateProcessFile
#define _communicateProcessFile

#include <fstream>
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <tchar.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/locking.h>
#include <fcntl.h>
#include <share.h>
#include <string>



int fileId;
FILE* stream;
char nextFileContents[1000] = "";

/* Declaration that says that this method can be used from files other
than the one in which it's defined */
extern "C" {
    void communicateProcessFile(const double* fromProcess, const int nbrFP, double* toProcess, const int nbrTP);
}

/* Method that give us acces to the file Matrix.txt if no other program is using it
and if it is our time to use it (checks the label of the file). */
void waitForOurTurn()
{
    stream = NULL;
    fileId = _sopen("Matrix.txt", _O_RDWR, _SH_DENYNO ); //open the file "unsafe"

    if ( _locking(fileId, LK_NBRLCK, 4000) == 0 ) //tries to lock the file
    {
        stream = _fdopen( fileId, "r+" );
        //printf("Now we have got access to the file");
    }
    else
        //printf("Seems like the file is locked");

    while (stream == NULL)
    {
        _close(fileId);

        Sleep(10);

        fileId = _sopen("Matrix.txt", _O_RDWR, _SH_DENYNO );
        if ( _locking(fileId, LK_NBRLCK, 4000) == 0 )
            stream = _fdopen( fileId, "r+" );
    }
    char string[255];
    fseek( stream, 0L, SEEK_SET ); //start at the top of the file
    fscanf( stream, "%s", string ); //copy the contents of the file to the string
    int equal = strcmp(string,"Process"); //check the label of the file

    while (equal == 0)
    {
        _locking(fileId, LK_UNLCK, 4000); //unlock the file since its not our time to access it
        fclose(stream);
        stream = NULL;
        Sleep(50);

        fileId = _sopen("Matrix.txt", _O_RDWR, _SH_DENYNO );
        if (_locking(fileId, LK_NBRLCK, 4000) == 0 )
            stream = _fdopen( fileId, "r+" );
            while (stream == NULL)
            {
                  _close(fileId);
                  Sleep(10);
                  fileId = _sopen("Matrix.txt", _O_RDWR, _SH_DENYNO );
                  if (_locking(fileId, LK_NBRLCK, 4000) == 0 )
                        stream = _fdopen( fileId, "r+" );
            }
            fseek( stream, 0L, SEEK_SET );
            fscanf( stream, "%s", string );

            equal = strcmp(string,"Process");
      }

      strcat(nextFileContents,"Process"); //write the new label to the file (to the
                                          //string nextFileContents temporarely)
      strcat(nextFileContents,"\n");
}

/* Method that returns a double representing the control signal of the current
row of the file contents. */
double readNextDouble()
{
      char string[255];
      char* token;
      char* stopstring;
      fscanf( stream, "%s\n", string );

      strcat(nextFileContents,string);
      strcat(nextFileContents,"\n");

      token = strtok(string, "=");

      if( token != NULL )
      {
            token = strtok(NULL, "\n");

            if( token != NULL )
                  return strtod(token, &stopstring);
      }
   return 0.0; //should never happen
}

/* Method that takes the current row of the file contents and replaces the part
of the string that represents the value of the process value with a string
representing the double d. */
void printNextDouble(double d)
{
      char string[1000];
      char newString[1000] = "";

      char* token;
      char number[15];
      long currentPos = ftell(stream);

      fscanf( stream, "%s\n", string );

      token = strtok(string, "="); //go to the character just before the process value

      strcat(newString, token);
      strcat(newString, "=");
      if (token!=NULL)
      {
            //read through the double
            sprintf(number,"%e", d);
            strcat(newString,number);

            strcat(newString, "\n");
      }
      strcat(nextFileContents,newString);
}

/* Extern method that reads control signals from the file and writes process values
to the file. The vector fromProcess shall contain nbrFP numbers of doubles representing
the process values. The nbrTP number of control signals will be written to the
vector toProcess. */
```

```
void communicateProcessFile(const double* fromProcess, const int nbrFP,
                                        double* toProcess, const int nbrTP)
{
    sprintf(nextFileContents,"");
    waitForOurTurn();

    /* print current process values to the file */
    for (int i = 0; i < nbrFP; i++)
        printNextDouble(fromProcess[i]);

    /* read next control signals from the file */
    for (i = 0; i < nbrTP; i++)
        toProcess[i] = readNextDouble();

    fseek( stream, 0L, SEEK_SET );  //go to the beginning of the file
    fprintf(stream, "%s", nextFileContents);
    fflush( stream );
    _locking(fileId, LK_UNLCK, 4000); //unlock the file
    fclose(stream);
}

#endif
```

Appendix B - CommunicatePLCFile

```
import com.ms.activeX.*;
import com.ms.com.*;
import s7wspsmx.*;
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import java.io.*;
import java.lang.Double;
import java.text.DecimalFormat;
import java.text.NumberFormat;


public class CommunicatePLCFile extends Form
{
    private s7wspsmx.S7ProSim.S7ProSim s7ProSim1;
    private FileReader fileReader;
    private FileWriter fileWriter;
    private final String fileName = "C:\\Demo\\Matrix.txt";
    private final int range = 27648;
    private Thread myThread;
    private String nextFileContents;
    private com.ms.wfc.io.File file;
    private static final char CR = 13; //Carriage Return
    private static final char LF = 10; //Line Feed
    private int lastClockCount = 0;
    private long fileStartPosition;
    private final Object fromProcessToPCS7[][] =

    {{"PROCESS VALUES",              "INPUT PORT NUMBER",   "LOWER BOUND",  "UPPER BOUND"},
     {"HP1_Pressure1:=0.000000e-000", new Integer(512),     new Integer(40),   new Integer(70)},
     {"MP_Pressure:=0.000000e-000",   new Integer(514),     new Integer(0),    new Integer(15)},
     {"LP_Pressure1:=0.000000e-000",  new Integer(516),     new Integer(0),    new Integer(5)},
     {"HP2_Pressure1:=0.000000e-000", new Integer(518),     new Integer(0),    new Integer(40)},
     {"Ack_Pressure1:=0.000000e-000", new Integer(520),     new Integer(0),    new Integer(30)}};


    private final Object toProcessFromPCS7[][] =

            {{"CONTROL SIGNALS",                           "OUTPUT PORT NUMBER"},
             {"Valve_HP_MP:=5.000000e-001",                new Integer(512)},
             {"Valve_HP_LP:=5.000000e-001",                new Integer(514)},
             {"Valve_Inlet_Turbine:=5.000000e-001",        new Integer(516)},
             {"Valve_Discharge_Turbine:=5.000000e-001",    new Integer(518)},
             {"Valve_HP1_Acc:=5.000000e-001",              new Integer(520)},
             {"Valve_HP2_Acc:=5.000000e-001",              new Integer(522)},
             {"Valve_HP2_MP:=5.000000e-001",               new Integer(524)},
             {"Valve_HP2_LP:=5.000000e-001",               new Integer(526)},
             {"Valve_Acc_MP:=5.000000e-001",               new Integer(528)},
             {"Valve_Acc_LP:=5.000000e-001",               new Integer(530)},
             {"Valve_Airblow_LP:=5.000000e-001",           new Integer(532)},
             {"BarkBoiler_Oil:=5.000000e-001",             new Integer(534)},
             {"HP1_Pressure1_back:=4.000000e-001",         new Integer(540)}};

    /* The constructor of the class, which actually starts the communication
    using private methods. */
    public CommunicatePLCFile(Thread myThread)
    {
        this.myThread = myThread;
        s7ProSim1 = new s7wspsmx.S7ProSim.S7ProSim();
        createNewFile();
        initControll();
        performWriteandRead();
        s7ProSim1.Disconnect(); //never reached...
    }

    /* Private method that creates a the file Matrix.txt with the contents according to
    fromProcessToPCS7[][] and toProcessFromPCS7[][]. */
    private void createNewFile()
    {
        file = new com.ms.wfc.io.File(fileName,
                                            com.ms.wfc.io.FileMode.CREATE,
                                            com.ms.wfc.io.FileAccess.READWRITE,
                                            com.ms.wfc.io.FileShare.READWRITE);

        String contents = "PCS7" + CR + LF;
        for (int i = 1; i < fromProcessToPCS7.length; i++)
                                contents += (String) fromProcessToPCS7[i][0] + CR + LF;
        for (int i = 1; i < toProcessFromPCS7.length; i++)
                                contents += (String) toProcessFromPCS7[i][0] + CR + LF;
        fileStartPosition = file.getPosition();
        file.setPosition(fileStartPosition);
        file.setLength(contents.length());

        for (int i = 0; i< contents.length(); i++)
            file.write(contents.charAt(i));

        file.flush();
        file.close();
    }

    /* Private method that returns a String with the components of the file. Before
    reading from the file it tries to lock the contents of the file. If the
    attempt is unsuccesfull it sleeps and tries again and so on. */
    private String getContentsFromFile()
    {
        boolean ourFile = true;
        try{ file.lock(0,4000);}
        catch(Exception wioe){System.out.println(wioe);ourFile = false;}
        while (!ourFile)
        {

            ourFile = true;
            try{myThread.sleep(5);}
            catch(InterruptedException ie){System.out.println(ie);}
            try{file.lock(0,4000);}
            catch(Exception wioe){System.out.println(wioe);ourFile = false;}
        }

        file.setPosition(fileStartPosition);
        file.flush();

        String temp = new String("");
        long fileLength = file.getLength();
        for (int i = 0; i < fileLength; i++)
            temp += (char) file.read();
        return temp;
    }

    /* Private method that performs the communication between the file and the
    control system. It reads process values from the file, waits for the control
    system to tick to the next sample and then writes the control signals to the
    file. This is performed repeatedly. */
    private void performWriteandRead()
    {
        file = new com.ms.wfc.io.File(fileName,
                                            com.ms.wfc.io.FileMode.OPEN,
                                            com.ms.wfc.io.FileAccess.READWRITE,
                                            com.ms.wfc.io.FileShare.READWRITE); //open the file "unsafe"
        fileStartPosition = file.getPosition();

        while (true)
        {
            String fileContents = getContentsFromFile();

            int currentIndex = 0;
            String wroteLast = fileContents.substring(currentIndex,
                                            fileContents.indexOf(CR, currentIndex));

            while (wroteLast.equals("PCS7"))
            {
                file.unlock(0,4000);
                try{myThread.sleep(80);}
                catch(InterruptedException ie){System.out.println(ie);}
                fileContents = getContentsFromFile();
                wroteLast = fileContents.substring(currentIndex,
                                            fileContents.indexOf(CR,currentIndex));
        //System.out.println("Not yet our time to write!");
    }

    //Now it is our time

    /* Add the String that says that we entered the file the last time */
    nextFileContents = "PCS7";
    nextFileContents += CR;
    currentIndex = fileContents.indexOf(CR, currentIndex);
    currentIndex = fileContents.indexOf(LF, currentIndex);

    /* Read the input values from the file */
    int nbrOfInputs = fromProcessToPCS7.length;
    for (int i = 1; i < nbrOfInputs; i++)
    {
        Variant input = readNextInput(fileContents, currentIndex,
                            ((Integer) fromProcessToPCS7[i][2]).intValue(),
                            ((Integer) fromProcessToPCS7[i][3]).intValue());
        nextFileContents += fileContents.substring(currentIndex,
                            fileContents.indexOf(CR, currentIndex));
        s7ProSim1.WriteInputPoint(((Integer) fromProcessToPCS7[i][1]).intValue(),0,input);
        nextFileContents += CR;
        currentIndex = fileContents.indexOf(CR, currentIndex);
        currentIndex = fileContents.indexOf(LF, currentIndex);
    }


    /*  wait for the control system to tick to the next sample so we can get the
    corresponding control signals */
    boolean newClockCycle = false;
    while (!newClockCycle)
    {
        Variant clockCount_Variant = new Variant(1);
        s7ProSim1.ReadOutputPoint(538,0,PointDataTypeConstants.S7_Word,clockCount_Variant);
        int clockCount = lastClockCount;

        try{clockCount = clockCount_Variant.toInt();}
        catch(ClassCastException cce){System.out.println(cce);}

        if (clockCount > lastClockCount + 1)
            System.out.println("The communication is too slow!!!!!!!!!");
        if (clockCount > lastClockCount)
        {
            lastClockCount = clockCount;
            newClockCycle = true;
            System.out.println("new ClockCycle: " + lastClockCount);
        }
        else
        {
            try{myThread.sleep(10);}
            catch(InterruptedException ie){System.out.println(ie);}
        }
    }



    // Now it's time to get the next control signals

    /* Read the outputs from the control system and print to the file
    (to the string temporarely) */
    int nbrOfOutputs = toProcessFromPCS7.length;
    for (int i = 1; i < nbrOfOutputs; i++)
    {
        Variant output = new Variant(1);
        s7ProSim1.ReadOutputPoint(((Integer) toProcessFromPCS7[i][1]).intValue(),0,
                            PointDataTypeConstants.S7_Word,output);

        nextFileContents += writeNextOutput(fileContents, output, currentIndex);
        nextFileContents += CR;
        currentIndex = fileContents.indexOf(CR, currentIndex);
        currentIndex = fileContents.indexOf(LF, currentIndex);
    }

    nextFileContents += LF;

    file.setPosition(fileStartPosition);
```

```
            long nextFileLength = nextFileContents.length();
            file.setLength(nextFileLength);

            /* Write the contents of the String nextFileContents to the file */
            for (int i = 0; i< nextFileLength; i++)
                file.write(nextFileContents.charAt(i));

            file.flush();
            file.unlock(0,4000);
        }
}


/* Private method that returns a Variant representing the process value of the current
row (indicated by the integer rowPos) of the file contents. Since the process value is
between low and high in the file but has to be between 0 and range before sent into
the control system it is rescaled */
private Variant readNextInput(String fileContents, int rowPos, int low, int high)
{
        String temp = fileContents.substring(fileContents.indexOf(":=",rowPos)+2,
                                    fileContents.indexOf(CR,rowPos));
        int value = (int) Math.round(((double) java.lang.Double.valueOf(temp).doubleValue()-low)/(high-low)*range);
        if (value < 0)
            value = 0;
        if (value > range)
            value = range;
        Variant k = new Variant((int) value);
        k.changeType(2);
        return k;
}



/* Private method that formats the double d (between 0 and 1) to a String presenting the value
in scientific notation with 6 decimals and three digits representing the exponent.
E.g 0.2345 is formatted to 2.345000e-0.001*/
private String formatToScientificString(double d)
{
        if (d == 0)
            return "0.000000e-000";
        String temp = Double.toString(d/10000);  //toString creates string in scientific notation
                                            //only if the double is < 1e-3 and we need
                                            //to deal with numbers <=1
        int length = temp.length();
        String formattedString = temp.substring(0,Math.min(length-3, 7)); //1.23456789e-2 -> 1.23456, 1.23e-2 -> 1.23
        for (int i = 0; i< 11-length; i++)
            formattedString += "0"; // adding zeros to the end
        if (length == 11)
            formattedString += temp.charAt(7);
        if (length > 11)
            formattedString += String.valueOf(Math.round(Double.valueOf(temp.substring(7,9)).doubleValue()/10));
        int exp = Integer.parseInt(temp.substring(length-2,length))+4;
        if (exp >= 0)
            formattedString += "e+00" + Integer.toString(exp); //assuming exponent is one digit
        else
            formattedString += "e-00" + Integer.toString(exp).substring(1,2); //assuming exponent is one digit

        return formattedString;
}


/* Private method that takes the current row (indicated by the integer rowPos) of the
file contents and replaces the part of the String that represents the value of the control signal
with a String representing the value of the Variant output. The method returns a String
representing the new row. */
private String writeNextOutput(String fileContents, Variant output, int rowPos)
{
        /* format the Variant to String of rigth format */
        output.changeType(Variant.VariantDouble); //to be able to get the double-value
        String formattedString = formatToScientificString((double) output.getDouble()/range);

        /* Construct a new String with the current number replaced */
        String temp = fileContents.substring(rowPos,fileContents.indexOf(":=",rowPos)+2);

        temp += formattedString;
```

```
            return temp;
        }

        /* Private method that connects to the control system by using the ActiveX control S7ProSim*/
        private void initControll()
        {
            this.setNewControls(new Control[] {
                                        s7ProSim1});
            s7ProSim1.begin();
            s7ProSim1.setAutoConnect(false);
            s7ProSim1.setScanMode(1); //1 = Continous Scan
            s7ProSim1.Connect();
        }

        /**
         * The main entry point for the application.
         */
        public static void main(String args[])
        {
            CommunicatePLCFile controller = new CommunicatePLCFile(Thread.currentThread());
        }
    }
```