Master’s Thesis 2013

Candidate: Stian Krogstad

Title: Heating of buildings with focus on measurement and control

Telemark University College
Faculty of Technology
M.Sc. Programme

MASTER'S THESIS, COURSE CODE FMH606

Student:	**Stian Krogstad**		
Thesis title:	**Heating of buildings with focus on measurement and control**		
Signature:	. . . .		
Number of pages:	**149**		
Keywords:	. . . .		
	. . . .		
	. . . .		
Supervisor:	Nils Olav Skeie	sign.:	. . . .
2nd Supervisor:	Carlos Pfeiffer	sign.:	. . . .
Censor:		sign.:	. . . .
External partner:		sign.:	. . . .
Availability:	Open		

Archive approval (supervisor signature)**:** sign.: . . . . **Date :** . . . .

Abstract:

In Norway 60% of all energy is commonly used for heating, during cold winters this number is seen to rise even higher. Any reduction in the standby power used for heating will result in large power savings. In this thesis it has been proven that a good BAS system can reduce the energy usage with at least 20%. This means large savings can be made both in the power bill each month but also in a more global environmental perspective.

The first principle house model is found to inadequately predict house heating times. Augmenting the model with a Kalman filter for estimating disturbances is greatly improving the estimations. Straight forward OLS regression shows good results during experiments using similar conditions.

Three controllers are analyzed designed and implemented in Visual Studio (MPC,PID,and LQR). The Linear Quadratic Regulator is prosed as the optimal controller for the BAS MIMO system.

Table of Contents

Preface

This dissertation is based on a research program at Telemark University College regarding smart houses and energy savings and the BAS master project [1]. None of the thesis text or experimental data is taken directly from previously published material.

This thesis was written as a completion of the Systems and Control Master's program at Telemark University College (TUC). The thesis is a summary of the last two years culminated knowledge during the master's program.

My first word of thanks goes to the thesis supervisor Nils Olav Skeie for keeping me on schedule and some good on (and off) topic discussions.

A second word of thanks goes to co. supervisor Carlos Pfeifer for advice on the Riccati equations and the Kalman filter implementation algorithm.

Third, I would like to thank Whatsala Perera for creating the non-linear house model that has been tested in this thesis.

Fourth, I would like to thank Anders Theodorsen for some help with the Matrix library.

Last but definitely not least I would like to thank Ole P. Kordahl for proof reading the thesis, gathering the statistical energy data from his workplace at NVE and always giving encouraging advice.

The fully dressed use cases, FURPS+ sheets and MATLAB scripts are found in the Appendix sections of this thesis, including excerpts of the main code algorithms.

The complete source code has been given to thesis supervisor Nils Olav Skeie, but is available upon request.

Notodden 31th May 2013

Stian Krogstad

Nomenclature

This chapter gives a list of symbols, abbreviations, and subscripts used in the thesis.

ADC	Analogue to Digital Converter
BAS	Building Automation System
BB	Battery Board (ZigBee Pro Development kit)
dSSM	Discrete State Space Model
eSSM	Extended (augmented) State Space Model
ED	End Device See BB
ENOB	Effective Number of Bits
FURPS+	Functional Usability Reliability Performance Supportability +
GUI	Graphical User Interface
GW	Gateway
IO	Input / Output
LP	Low Pass
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
MIMO	Multiple input Multiple Output
NaN	Not a Number
OLS	Ordinary Least Squares
PRO	Professional
SS	Steady State
SSM	State Space Model
SISO	Single Input Single Output
UART	Universal asynchronous receiver/transmitter
XML	Extensible Markup Language
VS	Visual Studio
WF	Windows Forms

1 Introduction

1.1 Background

The cold winter months in Northern Europe create a high demand for energy, and thus energy savings are highly prioritized. In Norway about 60% of the energy used is for heating purposes, and with prices exceeding 1kr/kW during winter time energy savings are prioritized [2, 3]. The high prices and the large amount of power needed for heating both contribute to a high demand for new and smarter ways to save energy.

The Building Automation System (BAS) use sensors to monitor and a model to predict the heating time. A modeled approach will let the power be turned completely off when the house is empty and turned back on at the correct time thus saving the most amount of energy [1]. Current available systems only reduce the inside temperature with 5°C at the maximum [1].

This thesis will both be an evaluation on using a model in the BAS system and which control methods to use in a finished system.

The BAS sensors are needed at several places both inside and outside the building in question and create the need for a wireless sensor network [1].

1.2 Market overview

In the BAS master project work [1] there was proven that the most power savings to an automated system could be made by turning the power completely on and off again. There was done research into this particular area and what was available on the market. This research concluded two things. Firstly there are very few vendors on the market, and secondly there seems to be a misconception on how to save the most amount of energy. The few vendors are due to the relatively small market mainly Scandinavia. Existing methods from NOBØ [4] work by lowering the temperature during set intervals. The lowering schemes exist to prevent large heating times and ensure the comfort temperature is reached. Currently there exists no adaptable BAS system saving the most energy [1].

The basis for this project is to evaluate such an adaptable BAS system using a model to predict heating time ensuring comfort temperature at the correct time and maximizing the energy savings.

1.3 Previous work

A master project titled Building Automation Systems was concluded in the fall of 2012 [1] and creates the basis for the work done in this master thesis. During this project a house model was created together with Ph.D. student Whatsala Perera [5]. This model will be implemented in this thesis. Using a house model makes the BAS adaptable to changes. Adaptable BAS systems are a relatively new area and little information on the subject is currently available. The high usage of electric energy to heating is also primarily a concern in the northern regions which is a small market seen in a global perspective.

Klaus Kaae Andersen, Henrik Madsen and Lars H. Hansen published a paper called '*Modeling the heat dynamics of a building using stochastic differential equations'* in 1998. The procedure used in this dissertation is a combination of using the laws of physics and statistical data for modeling the heating of houses [6]. Their model performed reliably to their specific conditions. The basis of the model is time consuming since statistical data needs to be acquired for each building.

Bertil Thomasa, Mohsen Soleimani-Mohsenib and Per Fahle published a paper called "*Feed-forward in temperature control of buildings*" where the approach the control problem with focus on rapid changes in the outside temperature. The feed forward controller model is proven to increase the inside temperatures stability [7].

The Norwegian building standards have standards on the minimum amount of insulation in buildings and are denoted using the U[1] value [8]. U value properties and measurements are discussed in several on-line communities for energy savings [9].

The wireless sensor network was set up and tested in a summer job at TUC by the author [10]. The ZigBee ZStack code created in the BAS master project will be the basis for the communication to from the ZigBee nodes/ sensors to the ZigBee Coordinator /gateway[11] [12].

[1] A conduction and convection property

1.4 New work

The new work in this thesis is to employ a model in a BAS creating an adaptable system estimating heating time based on current environmental data. The model will be from the BAS master project [1] [5].

In addition the thesis will propose which control algorithms will be best suited for house temperature control. The controllers are the feedback control (PID) the Model Predictive Control (MPC), and Linear Quadratic Regulator (LQR). All controllers will be designed using suitable parameters for the BAS, simulated and tested.

There will be created a data acquisition program used as a ZigBee gateway parsing the environmental sensor data to file. The ZigBee gateway will be created using Visual Studio (VS) and C#. The gateway will be used in to gather experiment data from to validate the model.

A prediction model will be created based on the house model and a Kalman filter for estimating the disturbances. The prediction model will be tested using the experiment data before implementation in VS. The BAS control system utilizing the sensor data the controllers and the predictor will be created in VS and tested.

An interesting perspective is in having a good house model the BAS system will be invertible, estimating the cooling down period. This would open for energy savings in a much larger global market.

1.5 BAS system description

The Building Automated System (BAS) is a complete house temperature monitoring and control system. Therefore sensors are needed in order to measure present conditions, a data acquisition system functioning as a gateway to transform the sensor signals to readable formats, and a control system using these sensors for controlling the temperature. In the BAS master project it was proven that shutting the heaters completely off when the house was empty was the most energy efficient [1]. This creates the need for a prediction model estimating the time to reach the set point temperature. The complete setup of the BAS system can be seen in Figure 1-1. The data acquisition is done by the wireless sensor network sent through a gateway to the prediction system. This data is then used by the predictor to estimate the heating time based on the remaining time to a comfort interval. The control system uses the same data to keep the temperature at constant level.

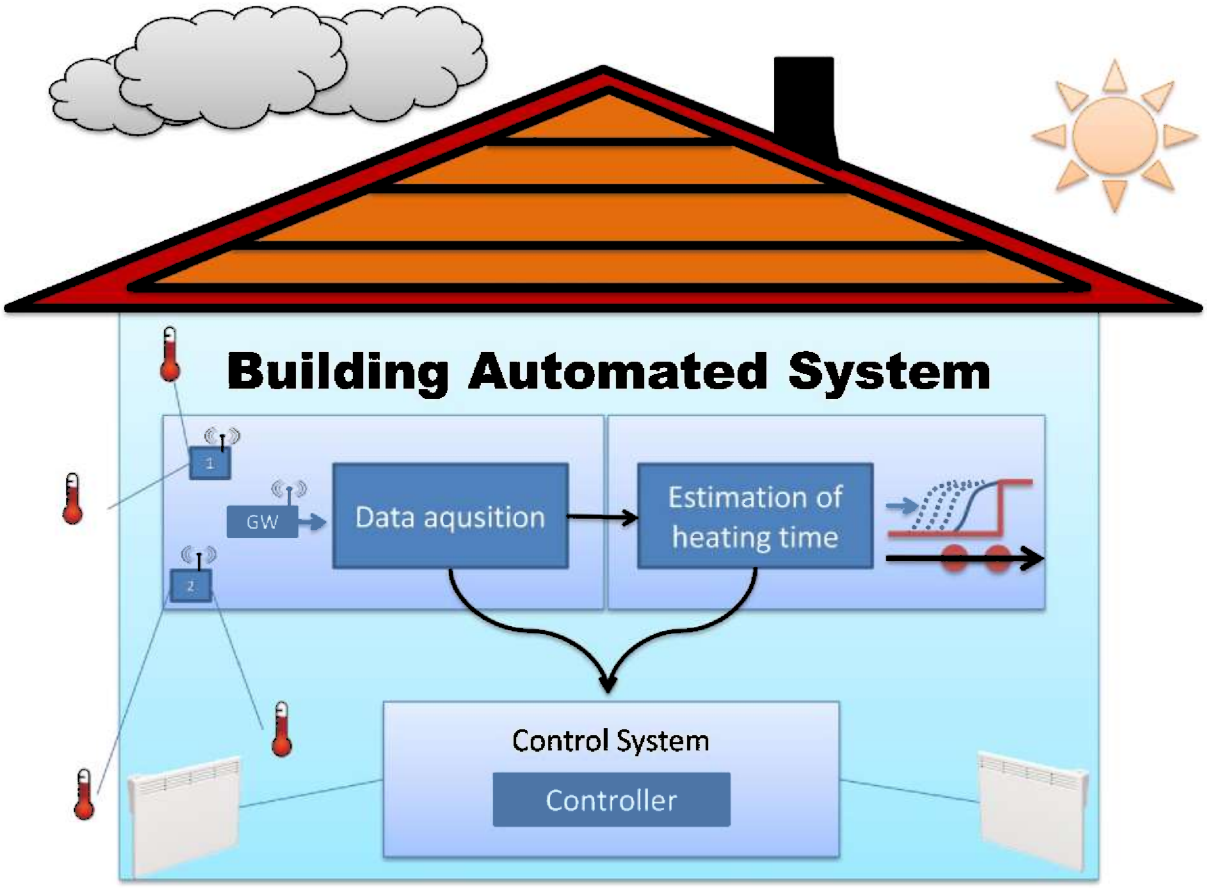


Figure 1-1 BAS thesis setup showing the three parts of the thesis and their main purpose.

The wireless sensor network has been selected as a ZigBee network due to the fact of interoperability between different vendors and the low power performance [11]. The ZigBee nodes will be coded in IAR EW using embedded C [1]. The main system code will be in Visual Studio and C#. The first principle model will be augmented using a Kalman filter and used for estimating heating time in paragraph 3.2.3.1. The control output will be run through software Pulse Width Modulator seen in paragraph 4.5, and the heater power will be set by a DAQ-6008 device [13]. MATLAB will be used to do data and system analysis.

1.6 Report structure

The BAS system will be a comprehensive system and can be viewed as the combination of the three major parts.

1. Data acquisition and house measurements, the gateway
2. Prediction model and heating time estimates, the predictor
3. Control system implementation, the controller

The parts will be individually completed before the next part is started and added to the system in the way of Object Oriented Design and Analyses [14]. This way the complete system can be viewed as the three use cases seen in Figure 1-2.

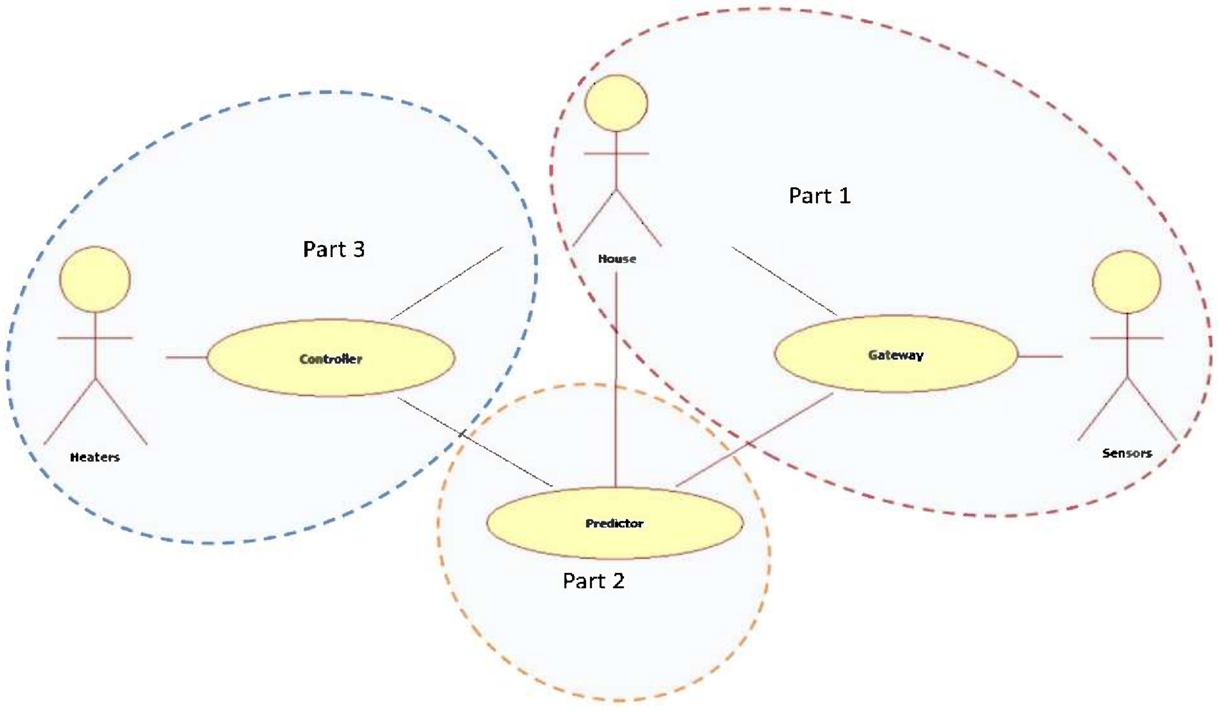


Figure 1-2 Thesis overview seen as use case diagram

All parts will follow the same basic structure and start with the theoretical background and end with a completed system test. MATLAB software will be used to do data analysis and control method simulations before implementation in Visual Studio (VS). The report structure can be seen in Figure 1-3.

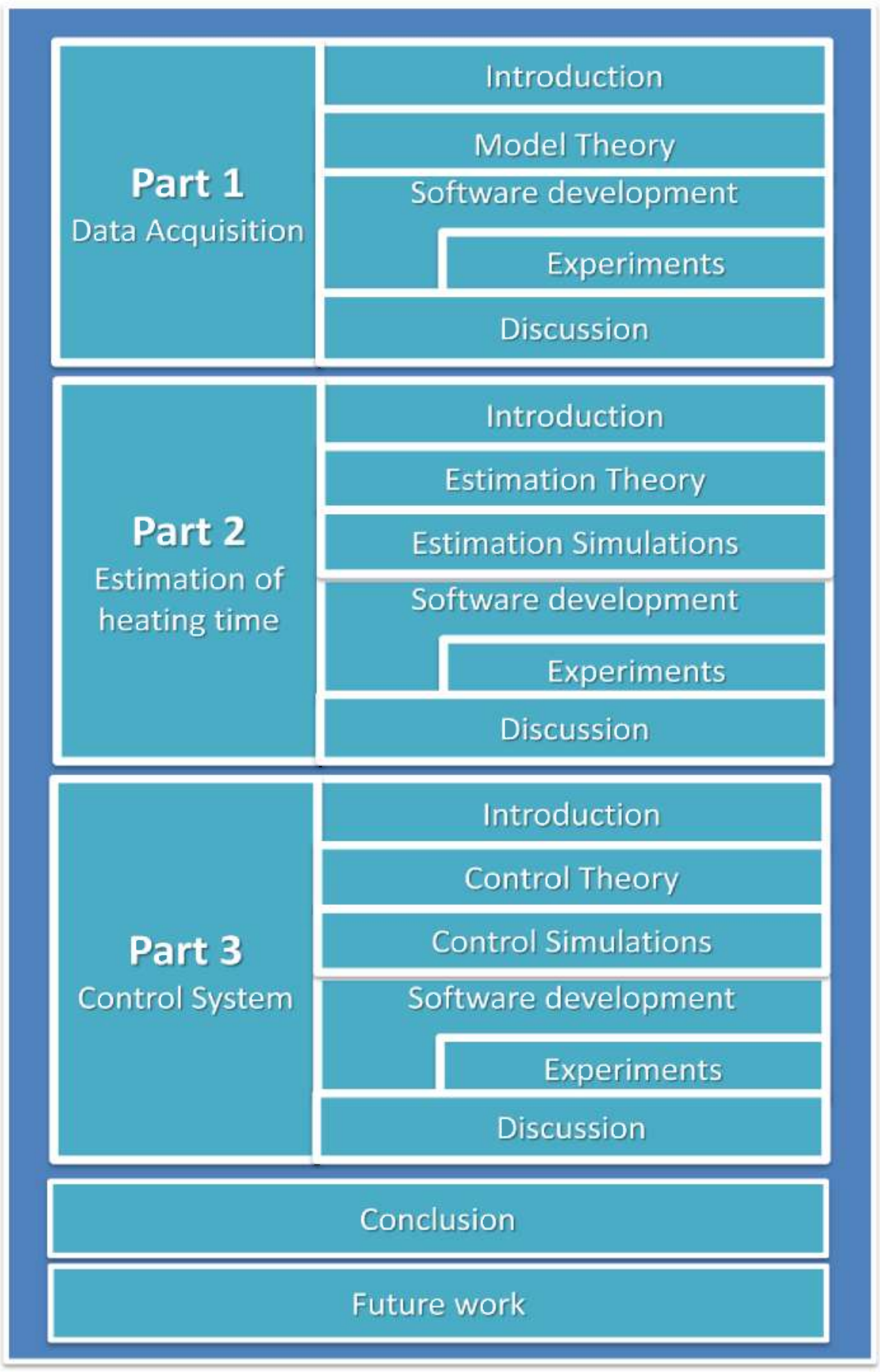


Figure 1-3 BAS thesis structure seen with major parts on the left and sub parts and sub chapters on the right.

Part 1

2 Data acquisition

2.1 Introduction

In order to monitor the temperatures inside a house there is the need for several sensors located both inside and outside the house measuring temperatures, humidity and ventilation [1]. To minimize the amount of cabling needed to all these sensors a wireless network has been selected for communication. The ultra-low power ZigBee will be used as the wireless sensor communication platform. The control system and data analysis software will be based on the Windows OS platform.

In order to achieve communication between the Windows OS and the ZigBee protocols a gateway is needed, where the gateway will work as a translator joining together the two networks. The gateway should convert the ZigBee sensor information to readable data in the MATLAB environment and the Windows OS based control system. There are two main reasons for the need of this software gateway:

1. The ZigBee devices lack the memory and the computational power to compute optimal control strategies.
2. MATLAB and other programs are needed to do sufficient data analysis.

2.1.1 System description

The Gateway system will be based on data from the sensors connected to the ZigBee nodes, end devices, and coordinator. The ZigBee Professional development kit from Texas Instruments has been used to read the temperature sensors in the end devices and send these to a ZigBee coordinator in the BAS master project [1]. The ZigBee coordinator is connected to the computer system using the COM port as the communication medium. This can be seen in Figure 2-1

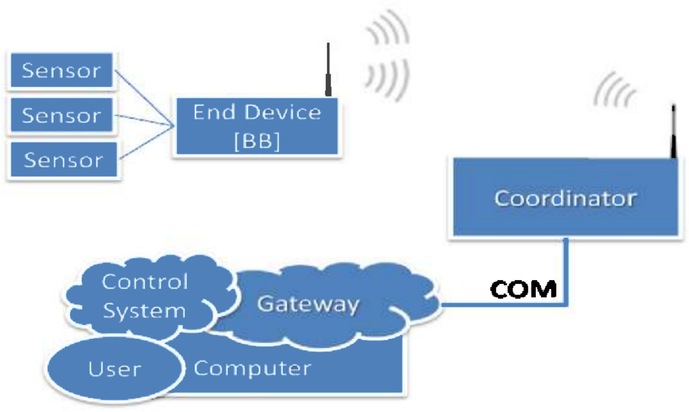


Figure 2-1 Part one system description - gateway

2.1.2 Part 1 Structure

The first part in this thesis will address the house measurements and data acquisition software. In order to understand the system to be monitored some theoretical background is needed. This will be gathered by analyzing the most important and measurable parameters of the house model. At the end of part 1 there will be real life experiments further testing the gateway and the validity of the model. The gateway part of this thesis will follow the structure seen in Figure 2-2 where the main chapters are seen on the right and the sub chapters are seen on the left.

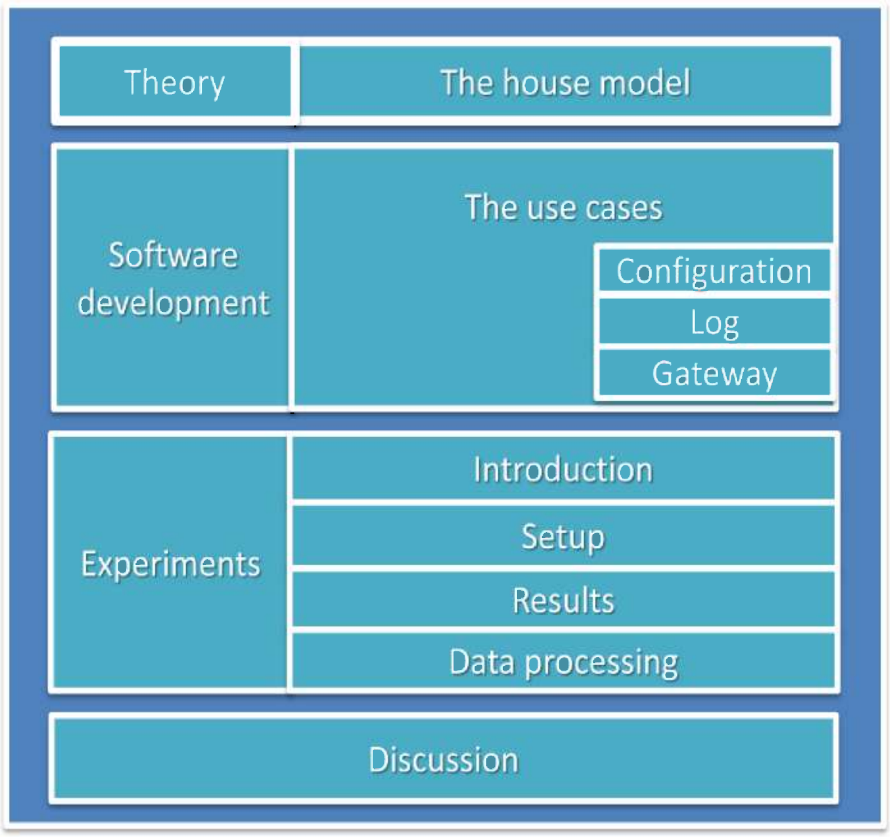


Figure 2-2 Part 1 structure progressing in a downwards fashion

2.2 Theory

2.2.1 The house model

In order to best approach the data acquisition software some theoretical background is needed. The house model will give a good understanding to what needs to be measured, and what should be set based on fixed values or theoretical data. The house model was created by PhD student Degurunnehalage Wathsala Upamali Perera [5] and used in the BAS master project [1]. It is based on two differential functions, the change in inside temperature seen in Equation (2-1) and the change in inside air density seen in Equation (2-2). The outside air density is assumed constant.

$$\frac{dT}{dt} = \frac{\left(\rho\dot{V}_o - \rho_i\dot{V}_i\right)}{\rho V}T + \frac{1}{\rho V(\widehat{c_p} - \frac{R}{M})}\left(\rho_i\dot{V}_i\widehat{H}_i - \rho\dot{V}_o\widehat{H}_o + \dot{Q}\right) \qquad \textit{(2-1)}$$

$$\frac{d\rho}{dt} = \frac{N}{3600}\cdot(\rho_i - \rho) \qquad \textit{(2-2)}$$

The model parameters are seen in Table 2-1.

Table 2-1 Model configuration parameters

Notation	Type	Unit
ρ	$Inside\ density$	$[kg/m^3]$
ρ_i	$Inlet\ density$	$[kg/m^3]$
$\dot{V}_i$	Volumetric flow rate of inlet air	$[m^3/s]$
$\dot{V}_o$	Volumetric flow rate of outlet air	$[m^3/s]$
M	Molar mass of outgoing moist air	$[kg/mol]$
$\hat{c}_p$	Specific heat of moist air at constant pressure	$[J/kgK]$
T	Temperature inside the room	$[K]$
$\widehat{H}_i$	Specific enthalpy of inlet air	$[J/kg]$
$\widehat{H}_o$	Specific enthalpy of outlet air	$[J/kg]$
$\dot{Q}$	Net heat energy transported into the system $\dot{Q} = \dot{Q}_{supply} - \dot{Q}_{loss}$	$[J/s]$
V	Volume of house	$[m^3]$
R	Gas constant	$[J/mol\ K]$
N	$Number\ of\ air\ changes\ per\ hour$	$[m^3/h]$

The house model parameters base on inside and outside conditions are visualized in Figure 2-3.

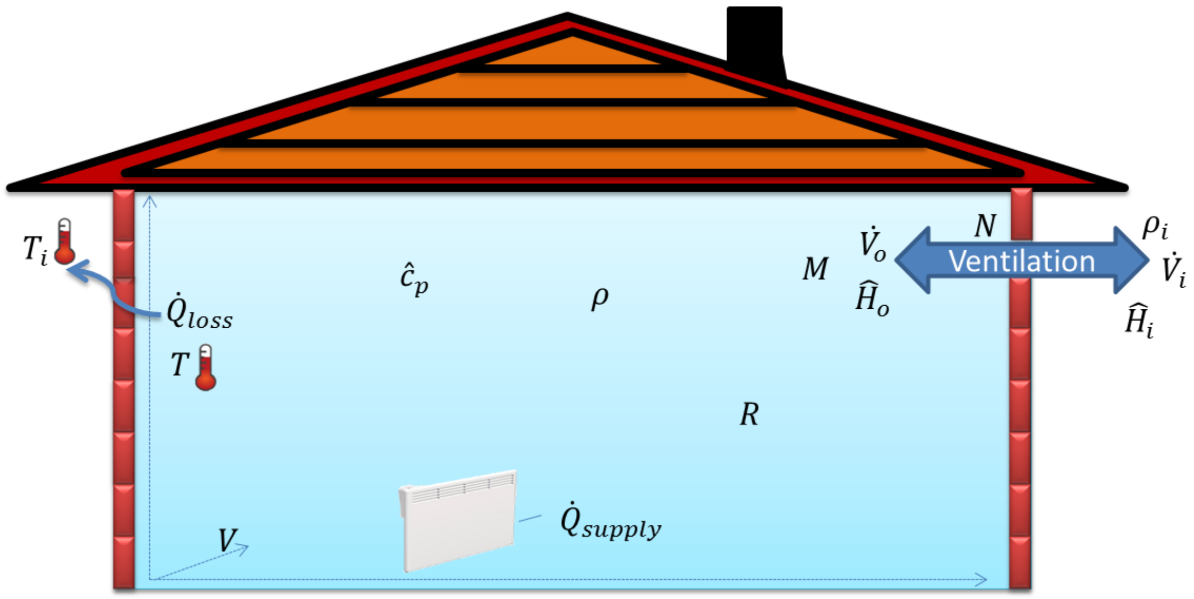


Figure 2-3 Visualization of model parameters

For more specifics on the model the reader is advised to read the BAS master project [1] or the house model paper [5].

The house model depends on a set of parameters from the building. Understanding these parameters is important to implement a good model based temperature control system. The most important parameters will be discussed in the next section.

2.2.1.1 House and model parameters

The model depends on several parameters from a specific building in order to emulate that building properly. Many of these parameters should be measured directly by a sensor network. The inside temperature, outside temperature, density, ventilation and pressure are such parameters. The U- value, the overall heat transfer coefficient, might however be easier to estimate from tables using known materials and known U-values.

The overall heat transfer coefficient U

One of the major parameters in the building model is the heat loss through convection and conduction known as the overall heat transfer coefficient U. This value is a measure of how much heat is lost from building elements to the environment.

A wall with a high U value means that it is leaking a lot of heat, while a low U value means a high degree of insulation. Figure 2-4 visualizes the difference between a well-insulated low U-value wall, and a poorly insulated high U-value wall.

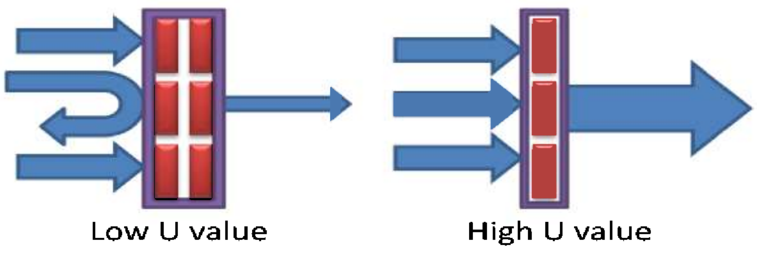


Figure 2-4 Low and high U values

The heat loss equation seen in Equation (2-3) is thus based on the U-values, the conduction and convection through composite materials elements walls, windows, floors, doors, roof and are based on the difference in inside and outside temperature. The heat loss equation can be seen in Equation (2-3) where the parameters are seen in Table 2-3.

$$\dot{Q}_{loss} = UA\Delta T \tag{2-3}$$

Table 2-2 Heat loss equation parameters

Notation	Type	Unit
ΔT	Overall temperature difference	[K]
A	Area of the element	$[m^2]$.
U	Overall heat transfer coefficient	$[w/m^2 \cdot K]$

The U values are useful in predicting the behavior of composites materials with regards to total heat loss from the complete element instead of each of the materials. The U value for a wall, floor, roof, door, and window will be specific to that wall and that house [8]. The Norwegian building standards have set regulations for the maximum recommended U values [8]. Together with a house model these values can be used to predict if a house is up to the Norwegian standards assuming known or measured ventilation temperature, pressure and density. If the temperature in the house drops faster than simulated by the model this would indicate that the elements of the building has a higher U value then specified. The U-values may also be measured and this is discussed in the Appendix section 7.2.

The most practical method is to use the standard values for the buildings elements either from the construction or the Norwegian standards, TEK-10 regulations on technical requirements for construction [8]. The TEK-10 maximal recommended values will be used further in the thesis and are seen in Table 2-3

Table 2-3 TEK-10 U values.

U-Wall	U-Window/door	U-Floor	U-Roof
0.18 W/(m^{2}K)	0.12 W/(m^{2}K)	0.15 W/(m^{2}K)	0.13 W/(m^{2}K)

In order to test the validity of the model and further test the functionality of the created Gateway system some experiments were performed. The experiments are found in section 2.4.

2.3 Software Development

The main objective of the gateway is to read several sensors values sent from the ZigBee end devices to the ZigBee coordinator[2] and save these values to file. The maximum number of sensors for each device is 7 given by the maximum number of inputs on the End Devices (ED) [15]. The number of end devices, types of sensors, IO channels configuration and the name of the device should be stored in a configuration file. The configuration file will be parsed using extensible markup language (XML), which is used to keep the data structured, organized and promote easy access.

The log file will be a text file with the sensor data and time stamp using the Norwegian CSV, separating values with a semicolon [16].

The Gateway will be created in Visual Studio, C#, and the Graphical User Interface will be based on Windows Forms, the ZigBee nodes are coded in IAR workshop and embedded C. All the requirements of the gateway have been considered using the FURPS+[3][14] method, the FURPS+ sheet can be found in Appendix 3 FURPS+ paragraph 7.3.1

2.3.1 The Use Case Diagram

The FURPS+ analysis of the Gateway are then made into a use case diagram. The use case diagram gives a good graphical overview of the functionality, and requirements of the system. The created use case diagram can be seen in Figure 2-5.

[2] The ZigBee gateway device where the UART/COM port is located.

[3] Functional, Usability, Reliability, Performance, Supportability +Additional.

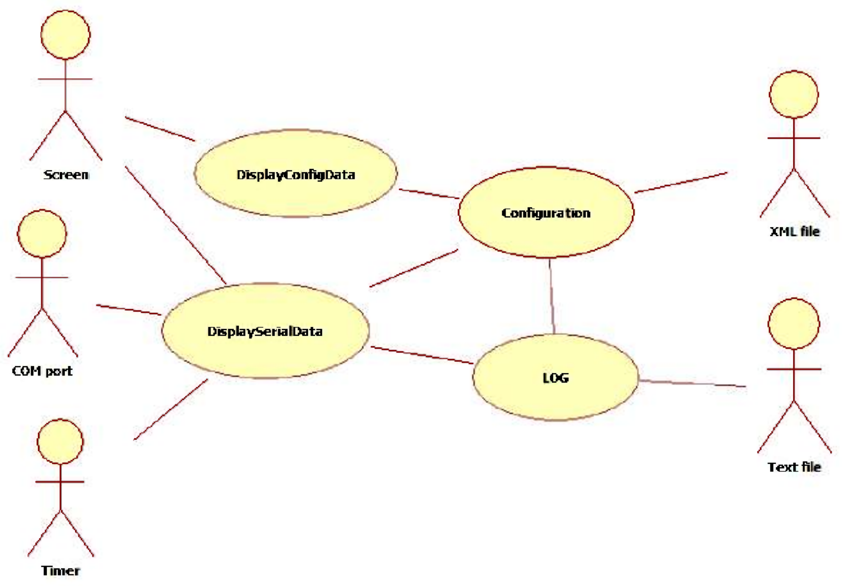


Figure 2-5 Use case diagram of the Gateway

In order to better visualize the inner workings of the gateway, a layered architecture design diagram has been created and can be seen in Figure 2-6. The layered architecture shows which use cases are communicating with each other, the operating system (OS) and the user through the graphical user interface (GUI).

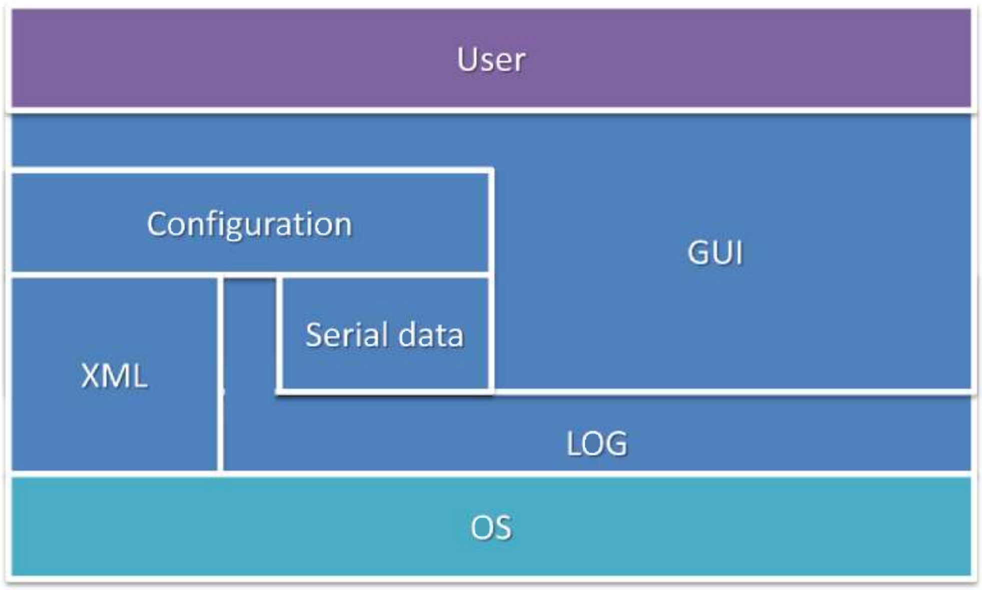


Figure 2-6 Layered architecture of the gateway system

The next step in the software process is to further analyze, design, code and test a selected use case, often the one with the highest risk or importance. The configuration use case is needed by all other use cases in the program, making it important and convenient to finish first. The progress of the four use cases, following the Unified Process (UP) [14]can be seen in Figure 2-7.

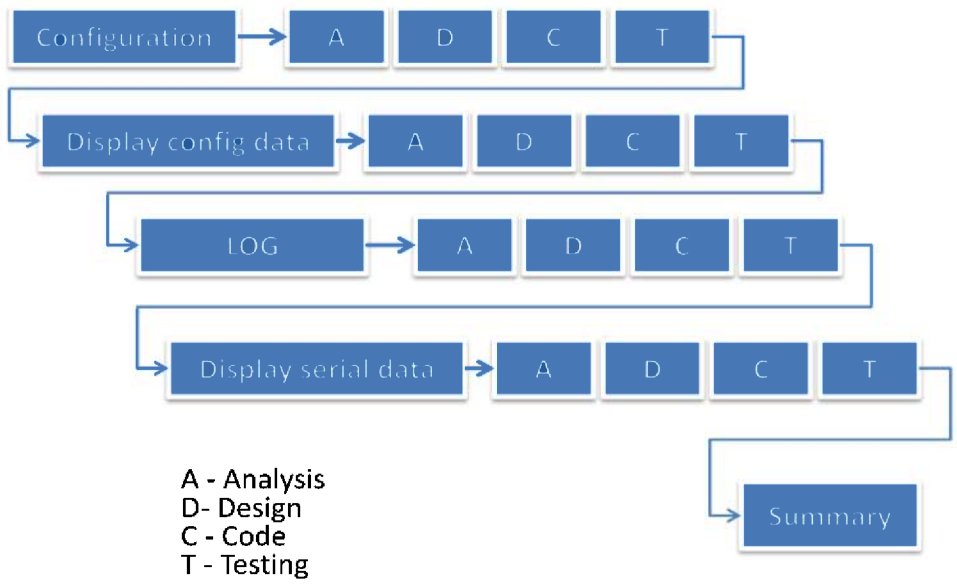


Figure 2-7 Software process following the UP

All the *classes* will start with the same names as their *use case* and a separate *.cs* file is created for each use case to simplify debugging and updating.

2.3.2 The configuration use case

The configuration class is responsible for storing and retrieving the program configuration. The configuration will be parsed using the Extensible Markup Language (XML). A fully dressed use case document will be created this gives good documentation and a good starting point for the programming. The fully dressed use case document can be found in Appendix 4: Fully dressed use case documentsparagraph 7.4. Following will be the design of the use case and its parameters.

2.3.2.1 Designing the configuration use case

The configuration use case is where all the data with changeable parameters are stored. The gateway program needs to store information about each sensor node, and the serial port settings. There might also be needed to change the timer saving data to disc, the sample time, so the timer should also be changeable. The main elements[4] of information needed can be seen in Figure 2-8.

[4] An XML element is consists of a start tag, and end tag, and the content in between.

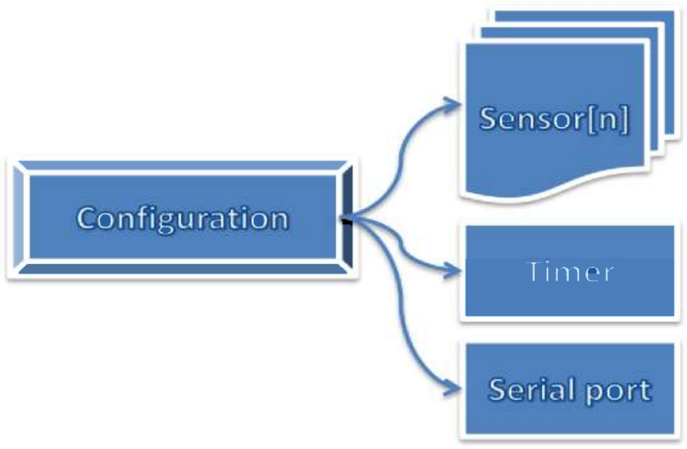


Figure 2-8 Configuration nodes

The elements specific information, nodes, has to be chosen. For the sensors there is needed information about the network address, name in order to know which End Device, ED is sending the data. In addition there is needed an Input Output (IO) channel numbering to sort the sensors connected to each ED. Finally the location and the measured value from the sensor should be set. For further use it might also be a good idea to add the range and uncertainty of the sensor and the installed battery date of the ED connected to this sensor. Lastly there should be a miscellaneous column in order to set additional information not thought of at the present time. The available sensor information, for one typical sensor, are summarized in Table 2-4.

Table 2-4 Typical example of one sensor settings in the configuration

Address / name	IO channel	Type	Location	Measureand	Range	Uncertainty	Battery install date	MISC
0AAA	00	PT1000	Bedroom	Temperature	-50°C to +100°C	0.02%	1/1-2013	Additional information

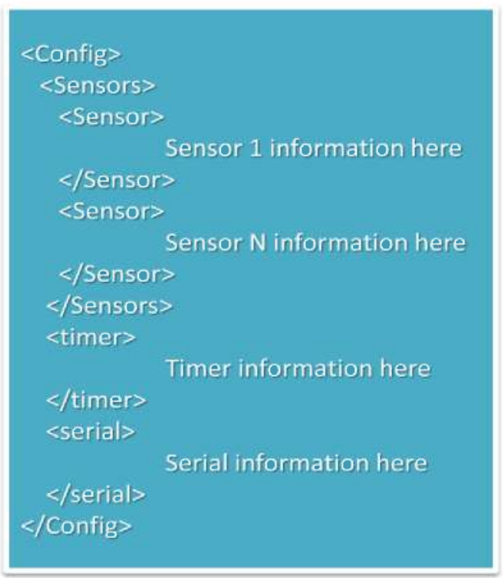
The serial link properties should be changeable to make the program run on different computers with different setups. The best way to do this let the user choose from the current available COM ports and COM port settings in Visual Studio (VS). An example of the serial information needed to run the program can be seen Table 2-5.

Table 2-5 Typical example of COM port settings

COM port	Baud Rate	Parity	Data bits	Stop Bits	Handshake	RTS enable
COM1	34800	none	8	one	disabled	enabled

The selected functions for creating this XML based configuration is the *XML serializer* function contained in .NET, which is a straight forward way of creating text based XML files. For reading, and writing to file the *filestream* function will be used [17].

The configuration file format will be XML following the template seen in Figure 2-9.



```
<Config>
  <Sensors>
    <Sensor>
                Sensor 1 information here
    </Sensor>
    <Sensor>
                Sensor N information here
    </Sensor>
  </Sensors>
  <timer>
                Timer information here
  </timer>
  <serial>
                Serial information here
  </serial>
</Config>
```

Figure 2-9 XML script template

Excerpts of the configuration codes most important algorithms are found with commentary in Appendix 5 – Source Code from paragraph 7.5.1.1 through 7.5.1.3.

2.3.3 Display Configuration Data use case

The next use case to be further analyzed, designed and added to the code is the display configuration use case. This use case contains the interface between the configuration data and the user. The configuration will be entered in a program configuration editor which will work during runtime, and remove any erroneous type errors from using a text based editor. The *DisplayConfigData* use case is created as a windows Form GUI to connect the user to the XML configuration file without the need for any external editing programs. The created fully dressed use case document can be found in Appendix 4: Fully dressed use case documents paragraph 7.4.2.

2.3.3.1 Designing the configuration GUI, *DisplayConfigData*

The *DisplayConfigData* use case needs to give a good and simple way to add, edit or remove sensors from the configuration file. Some sensors might break down or for other reasons need to be changed or new sensors added. The GUI will be made in VS and Windows Forms. The information in the GUI is as discussed in the configuration use case. The configuration display will use a *data grid view* for the sensor information, *combo boxes* for the serial configuration and a *text box* for the timer. The serial link properties should be selectable from available parameters in the .NET environment and the computer hardware. In addition an information button should be included to supply the user information about the configuration, and the correct way of inserting data. Lastly there should be an exit button and a button for saving the changes. The created configuration GUI can be seen in Figure 2-10.

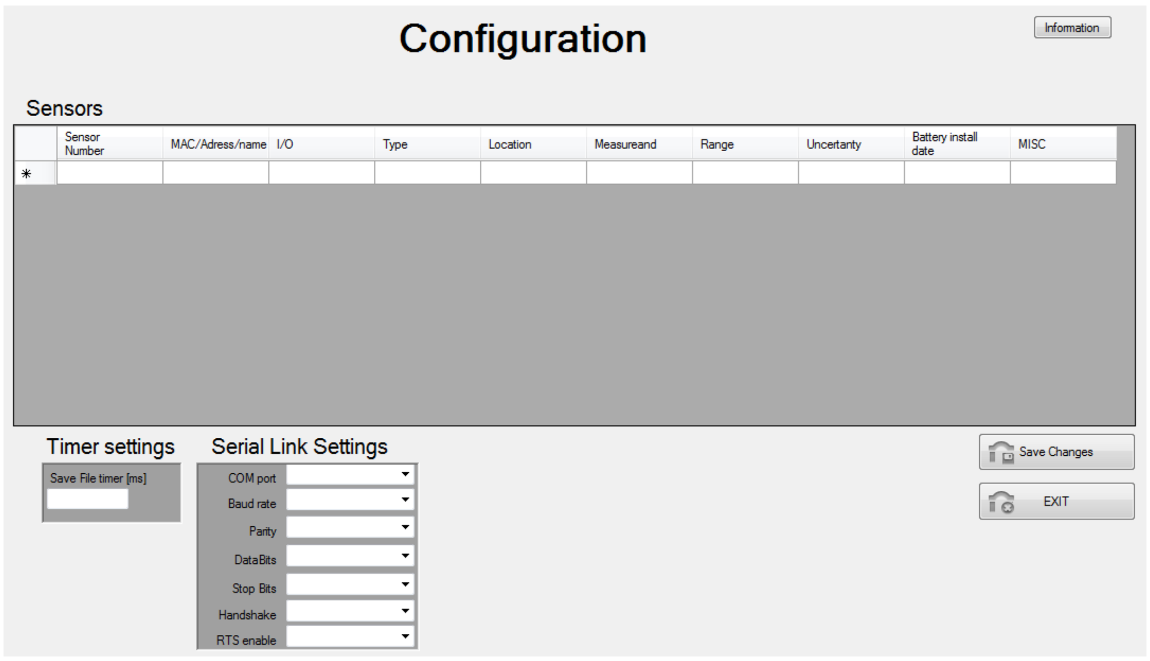


Figure 2-10 Configuration GUI.

The code is based on reading and saving the configuration data using the *config* class and the main code excerpts with explanation, results, testing and error handling can be found in Appendix 5 – Source Code paragraph 7.5.1.7and 7.5.1.9. Everything was found as working correctly and should be further checked in the log use case, for this reason the next use case to be further analyzed designed and added to the code will be the *LOG* use case.

2.3.4 LOG use case

The log use case main purpose is to parse the raw serial data, add a time stamp and save the data to file. The text received from the serial port will be a stream of characters that need to be redistributed in a readable format for MATLAB and other applications. The distribution of the sensor values should be based on the sensors configuration in the config.xml file. The LOG fully dressed use case document can be found in Appendix 4: Fully dressed use case documents paragraph 7.4.

2.3.4.1Designing the log, the *LOG* use case

The main function of the LOG use case is to parse the serial data into columns containing the date and time for the message and one column for each of the sensors values. The received raw data from the serial link contains the message between a start data sign, <, and a stop data sign. The message itself first consists of the network address or name of the ZigBee end device, followed by the IO address for that specific sensor. The reason for this is that some end device may have several different sensors connected, but only one sensor for each IO channel. The maximum numbers of sensors IO addresses are eight, but they are notated in the same way as in the Texas Instruments ZigBee ZStack[5] v 2.5.1 as 00 to 07 [15]. In Figure 2-11 a sensor message from the end devices is seen divided up into the specific parts.

$$\underbrace{<}_{Start}\ \overbrace{0AAA}^{Adress}\underbrace{00}_{IO}\ \overbrace{782}^{Value}\ \underbrace{>}_{Stop}$$

Figure 2-11 Example of sensor data sent from the end devices

The separating character between the columns should be a semicolon. This makes the parsed data easily readable by MATLAB and other data analysis software. The flow of the LOG use case can be seen in Figure 2-1.

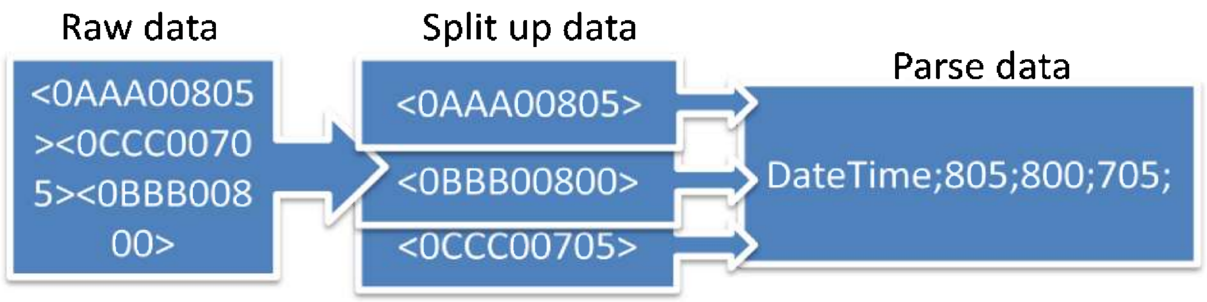
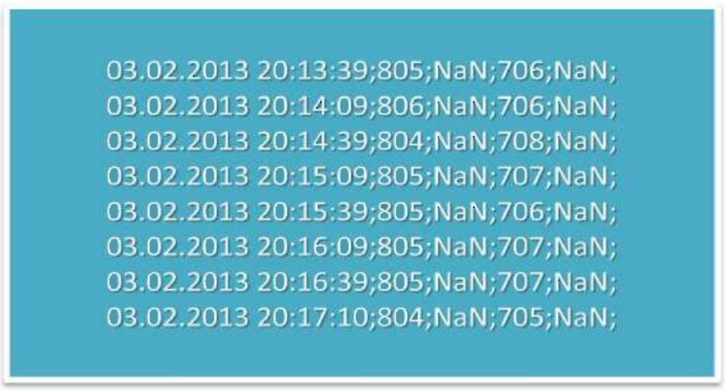


Figure 2-12 Log message flow

There needs to be created an algorithm in order to split the message data into packets with the information between the two separating signs. This algorithm will work by searching the incoming data for the end message sign > in order to be sure a complete message has been sent. Then it will check if the start message sign is the first part of the message. If both are valid a complete message has been recorded and it will be separated into an array based on the length between the message start and the message stop sign. This is done until the end of the message and the new created message array is ready for further processing. The parsing method will search through all the messages in the message array and pair it with the correct sensor from the configuration. If a sensor does not have any messages the Not a Number

[5] For more information on the ZStack and ZigBee reader is adviced to read[10] S. Krogstad, "ZigBee PRO development kit set up guide," ed, 2012, [15] T. Instruments, "CC253x System-on-Chip Solution for 2.4-GHz, IEEE 802.15.4 and ZigBee® Applications, CC2540/41 System-on-Chip Solution for 2.4-GHz Bluetooth® low energy Applications User's Guide " 2012.

(NaN) value will be set. Receiving several values from one sensor will only result in the last value being overwritten. The file will be in TXT format following the template seen in Figure 2-13.



```
03.02.2013 20:13:39;805;NaN;706;NaN;
03.02.2013 20:14:09;806;NaN;706;NaN;
03.02.2013 20:14:39;804;NaN;708;NaN;
03.02.2013 20:15:09;805;NaN;707;NaN;
03.02.2013 20:15:39;805;NaN;706;NaN;
03.02.2013 20:16:09;805;NaN;707;NaN;
03.02.2013 20:16:39;805;NaN;707;NaN;
03.02.2013 20:17:10;804;NaN;705;NaN;
```

Figure 2-13 Log data template

2.3.4.2The LOG code

The log code is made up of the algorithm used to split up the raw serial data, and methods to parse the serial data and save it to file. The main parts of the functions will be gone through more in detail and testing with error handling will follow the code in *Appendix 5 – Source Code* paragraph 7.5.1.7 through 7.5.1.9. The space required for log file saving has been calculated and can be found in 7.5.1.10

2.3.5 DisplaySerialData use case

The *DisplaySerialData* use case handles the serial port information and the visual interface between the received serial data and the user. The use case main property is to read the current data on the serial port. The current configuration should be available from the configuration XML file. The user should be prompted for saving a log file, and have the availability to both view and save the parsed data. There should also be an option for saving the raw data for debugging purposes. The documentation for the *DisplaySerialData* use case can be found in the Appendix 4: Fully dressed use case documents paragraph 7.4.

*2.3.5.1*Designing serial data GUI, *DisplaySerialData*

Using a serial port in C# is fairly straight forward it by dragging the serial port from the toolbox to the windows form. The use case needs two timers, one for the read serial data and one for saving the log files to disk. Both timers are used directly from the toolbox in VS. Since the gateway should be running continuously the exit cross action should be changed to hiding the application in the windows tray rather than closing the application. The created GUI for the main form of the gateway application can be seen in Figure 2-14.

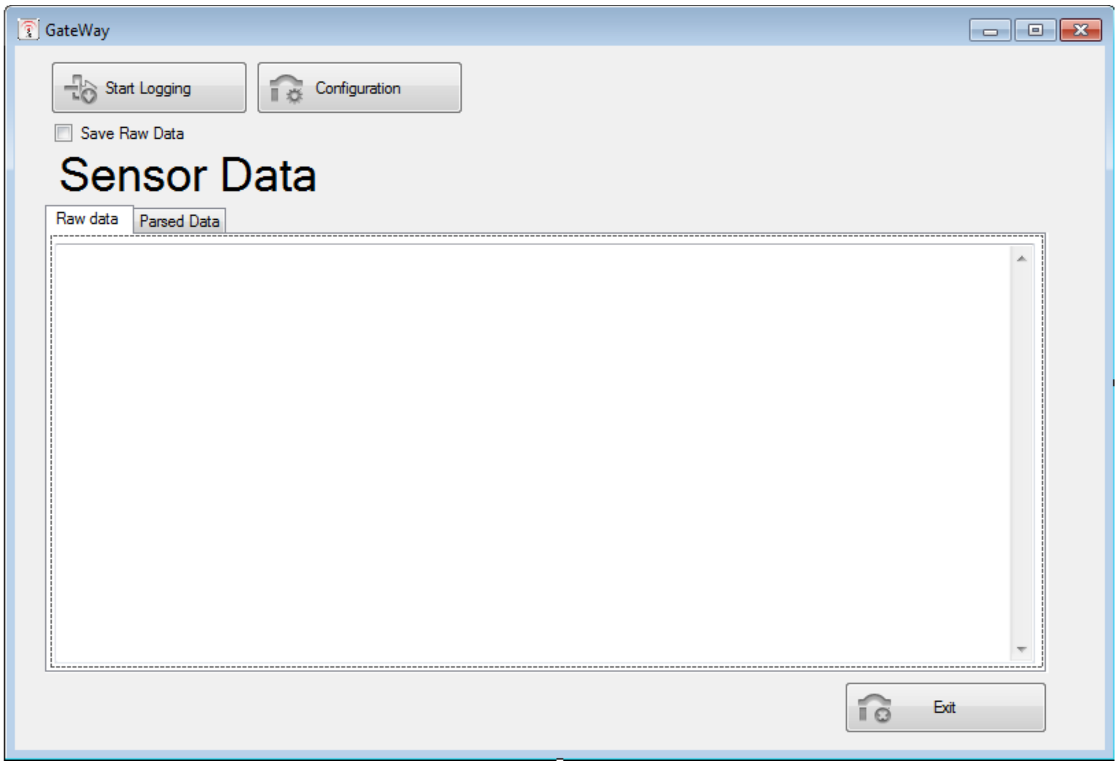


Figure 2-14 GUI of the read serial data

The function to read the serial port will be the *SerialPort.ReadExisting*. The ReadExisting method works by reading all the available bytes from the serial port, before returning a string. In addition the save file dialog method will be used to prompt the user for file name and location.

2.3.5.2GUI and extra functions

In order to create a better GUI several icons were found from www.iconfinder.com freely available for commercial use. In addition a main gateway icon was modified to fit this program. The icons can be seen in Figure 2-15.

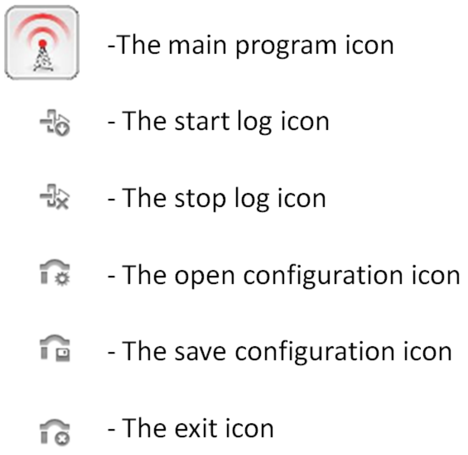


Figure 2-15 Icons used in the GUI of the gateway program

2.3.5.3Testing and error handling in the gateway system

The complete gateway system was logging data for 14 days consecutively and no problem arose, and the gateway will be further testing in the experiment part.

All the error handling in the gateway is done by saving the error to an error log containing the time, date and type of the error including the method name where the error occurred. A typical error log message in the split message method can be seen in VScode 2-1

```
LogSave("error.log", DateTime.Now.ToString() +  e.Message + e.Source +
"@splitmessage");
```

VScode 2-1 Error handling

2.4 Experiments

2.4.1 Introduction

In order to test the validity of the model experimental temperature data are needed. This data has been logged using the created gateway and can as such be directly imported to the MATLAB environment for further processing. In the MATLAB environment the data is easily plotted and compared to the models output with the same circumstances. This will also introduce additional testing of the data acquisition software. A sketch showing the model validation process can be seen in Figure 2-16 the simulated data given from the model is compared to the experimental data.

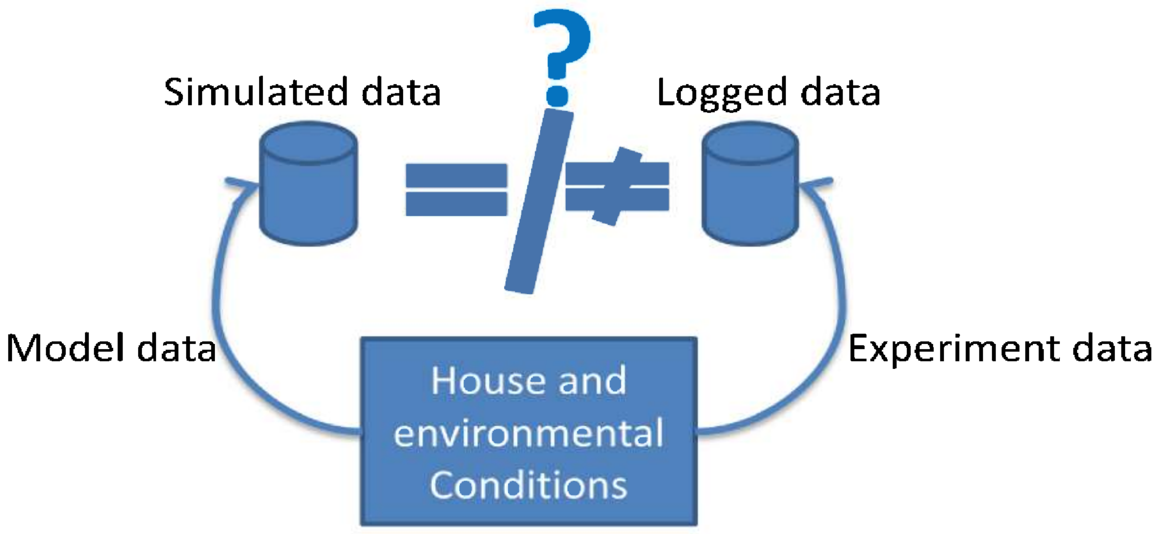


Figure 2-16 Model verification process

The created gateway has only been tested in simulations and these experiments will also be a test on how it will work in a real house monitoring situation. A house was made available for the experiments from the 21th of March to the 30th March 2013, and there was created nine low power temperature sensors for the temperature monitoring. The experimental data acquisition hardware consists of a computer with the Gateway software and a COM port for the connection to the ZigBee Coordinator. The ZigBee Coordinator acts as a hub receiving all the data from the wireless ZigBee sensor network. The sensor network consists of three ZigBee end devices each with three temperature sensors. The heaters are used as pure on off devices, and the used power is read off the power meter. The experiment procedure can be visualized in Figure 2-17 where the experiment values are read manually from the power meter and by the computer for the temperature sensors. The bottom level is visualized as either the heater connected to the power meter or the sensors connected to the end devices. Connections are noted with an arrow and a label denoting wired or wireless communication.

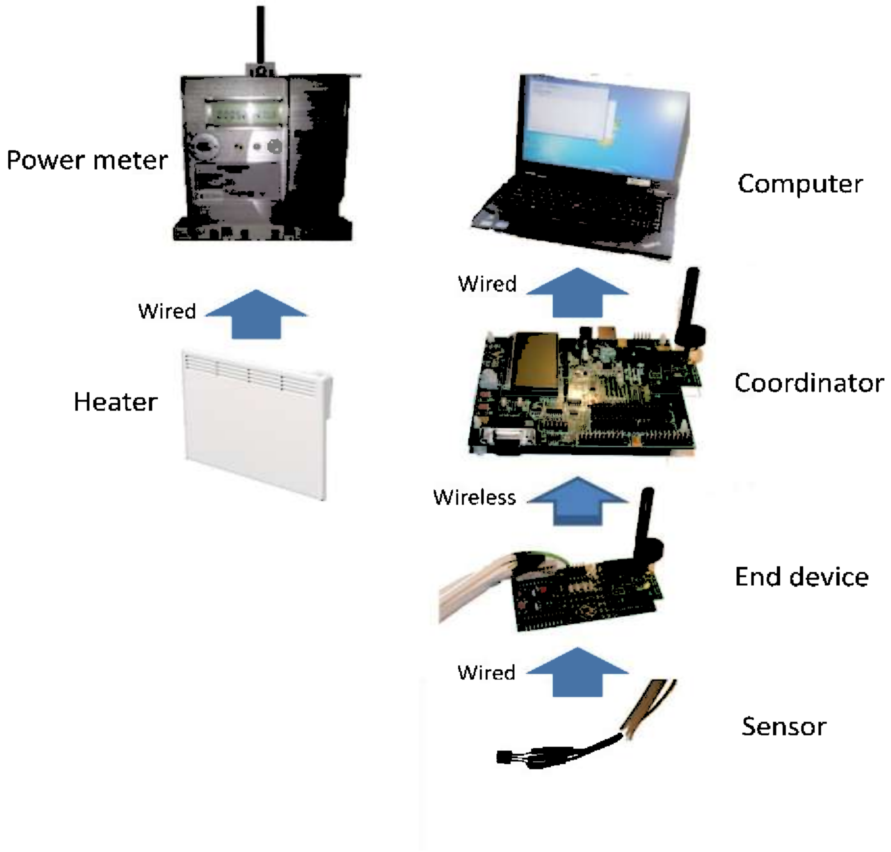


Figure 2-17 Experimental hardware

The IAR EW ZigBee based BAS gateway system created for the BAS Master Project was used as the ZigBee to COM communication, and for more theory on the ZigBee devices the reader is advised to read [15] and [10].

2.4.2 Experiment setup

The nine temperature sensors should be spread out with at least one sensor in each of the main rooms in the house, large rooms should use several sensors in order to get a correct room average. In addition there should be at least two sensors outside on different sides of the house to have one always in the shadow. The temperature sensors are silicon devices of type TMP36 [18].The sensor placement is seen in the building drawing seen in Figure 2-18 In addition the heaters are marked with name, where red box heater indicates a panel heater and the tiles indicate a floor heater. The arrows, Vent, indicates where there is a ventilation opening.

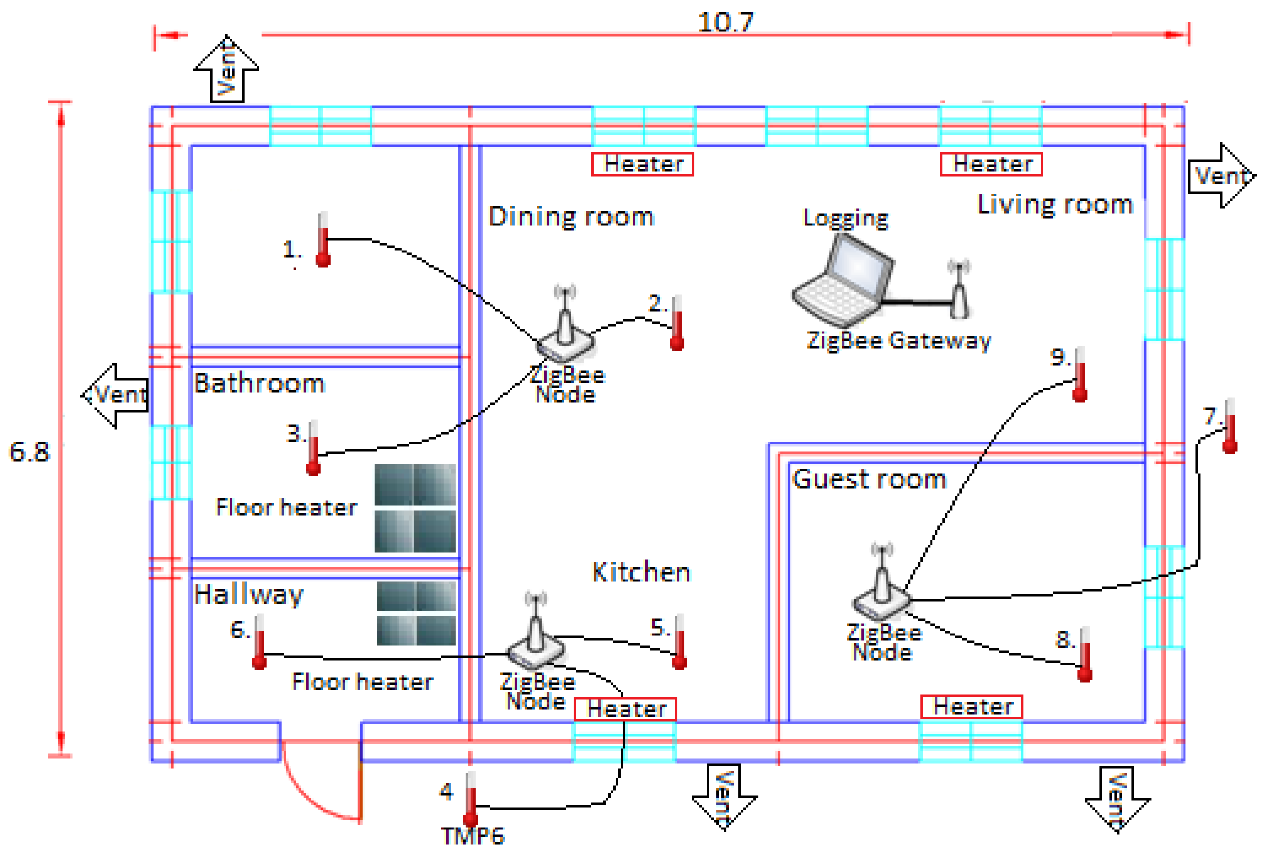


Figure 2-18 Temperature sensor locations and house setup

The temperature sensors are seen as the silicon devices connected to the ZigBee nodes /end devices by 3 cables of 2 and 4 meters. The sensors are numbered according to their numbered setup in the gateway configuration file. The sensors were placed at approximately 1.5 meter location from the floor. A picture of such typical placing can be seen in Figure 2-19.

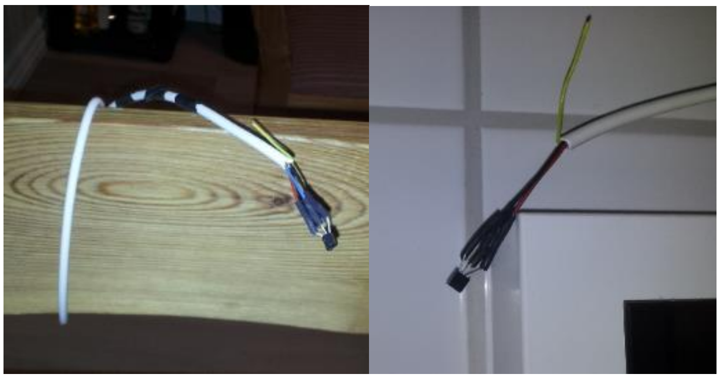


Figure 2-19 Typical sensor placement, sensor 2 on the left and 3 on the right

The outside sensors outside were placed as seen in Figure 2-20.

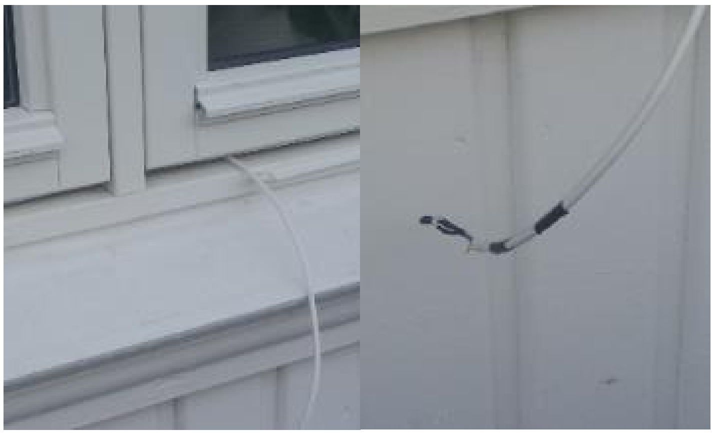


Figure 2-20 Outside sensor placement, sensor 7 seen in image

A typical location of the ZigBee node can be seen in Figure 2-21.

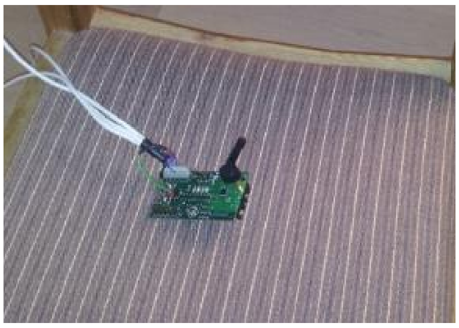


Figure 2-21 ZigBee end device placement, dining room node

The 9 temperature sensors were connected to the ZigBee nodes by connectors that fit the IO header B port 15 on the ZigBee battery board. The slots were connected to the analogue digital converter (ADC) channel 02, 04 and 07 on the ZigBee nodes. The connection can be seen in Figure 2-22.

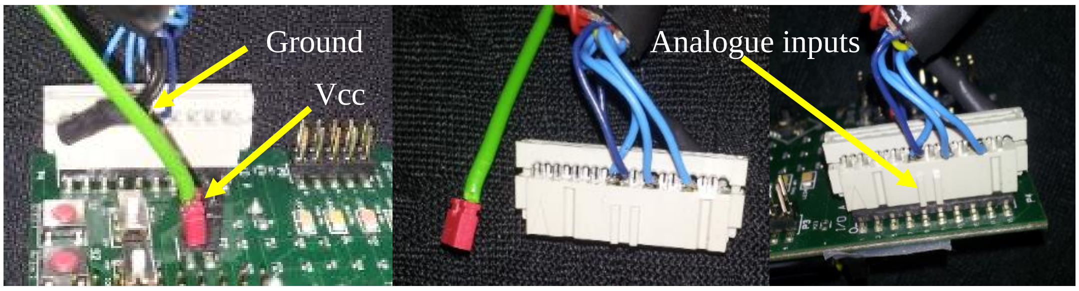


Figure 2-22 Connection to the ZigBee node (end device)

In addition there was one vent in each room, the vents could be closed and all was of the type seen in Figure 2-23.

Figure 2-23 Ventilation locations, living room vent seen in image

The uncertainties of the measurements are important to know in order to do any analysis, and calculating the temperature sensors uncertainty needs to be done. The ADC have 12 effective number of bits (ENOB) for ADC conversion. The range is from -3V to 3V which gives 2048 bins available on the positive side 0V to +3V .This gives as seen in Equation (2-4) and Equation (2-5)

$$\frac{3V}{2048} = 0.0015 \approx \frac{1.5mV}{bin} \tag{2-4}$$

$$\frac{1.5mV/bin}{3V} * 100 \approx 0.05\% \tag{2-5}$$

TMP6 Temperature sensor accuracy using worst case scenario will then as seen in Equation (2-6).

$$\frac{\pm 2°C}{156°C} * 100 \approx 1.3\% \tag{2-6}$$

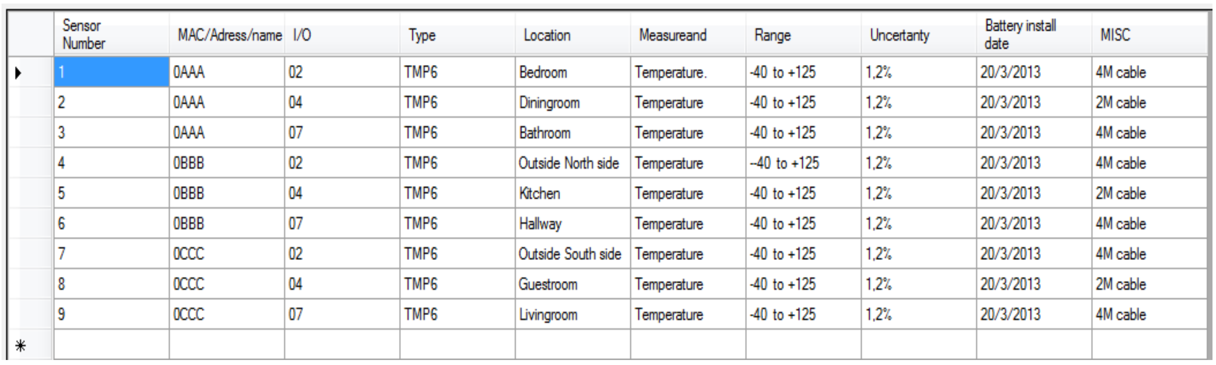
The complete uncertainty budget can be seen in Table 2-6 where the main sources of uncertainty is the TMP36 device and the manually read power meter.

Table 2-6 uncertainty budget

Type	Sensor	Range	Accuracy	Note
CC2530 EM	ADC	-3V to 3V	$\pm$0.05%	
Temperature	TMP36	-25°C to 140°C	$\pm$1.3%	
Power meter	N/A	N/A	$\pm$0.9kWh	The meter only shows kWh

2.4.3 Gateway setup, and data acquisition

The data was gathered using the created gateway organizing the sensor data. The gateway store all the information in text based log files that can be directly imported into MATLAB. The sensor data are included in columns separated by a semicolon where the first column is the Date time stamp. The sensor setup for the house experiments in the gateway can be seen in Figure 2-24.



Sensor Number	MAC/Adress/name	I/O	Type	Location	Measureand	Range	Uncertanty	Battery install date	MISC
1	0AAA	02	TMP6	Bedroom	Temperature.	-40 to +125	1,2%	20/3/2013	4M cable
2	0AAA	04	TMP6	Diningroom	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
3	0AAA	07	TMP6	Bathroom	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
4	0BBB	02	TMP6	Outside North side	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
5	0BBB	04	TMP6	Kitchen	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
6	0BBB	07	TMP6	Hallway	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
7	0CCC	02	TMP6	Outside South side	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
8	0CCC	04	TMP6	Guestroom	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
9	0CCC	07	TMP6	Livingroom	Temperature	-40 to +125	1,2%	20/3/2013	4M cable

Figure 2-24 Gateway configuration for running the experiments

The sampling time set in the save file timer is set to 3600ms or 6 minutes which should be a high enough sampling time due to the large time constants of a house.

2.4.4 Data processing

The raw sensor data saved to file using the gateway need some processing in order to be correctly represented when analyzed. The raw data should be filtered through a low pass filter, all non-values, NaN, should be removed and there should also be a check for gross outliers. Several MATLAB functions were created in order to accommodate the data processing and verification needed and will be further discussed in 2.4.4.1 to 2.4.4.3.

2.4.4.1The NaNremove.m function

The data acquisition system should interpolate between missing values and give the user information about the number of NaNs that are removed. There are efficient methods for this in the basic MATLAB setup. Using the *find.m* function with the *isnan.m* function any NaN values position is found. The MATLAB environment 2d interpolation function *interp1.m* is used to interpolate between the known values and the found NaN indexes. The interpolate function requires values before and after the missing value in order to function. This means the data needs to be checked for missing values at the start of the data set, these rows are simply removed from the data set. This is seen in an excerpt of the *NaNremove.m* function seen in MATLAB script 2-1.

```
while (find(isnan(data(j,:)))>0 & j<length(data(:,1)))
      j=j+1;
    end
    %remove all first rows with NaN data
    data=data(j:length(data(:,1)),:)
```

MATLAB script 2-11check of first row NaNs

Then all the NaN values that can be interpolated are found and the data is interpolated. This is seen in the excerpt of the *NaNremove.m* seen in MATLAB script 2-2

```
for i=1:length(data(1,:))
    Non=data(:,i)
    NonNan(:,i)=interp1(find(~isnan(Non)),Non(~isnan(Non)),1:length(Non))';
    NaNs=length(find(isnan(data)));
End
```

MATLAB script 2-2 interpolate between missing values

Finally the data needs to be checked for NaN values at the end of the data file. If found these end rows are removed following the same principle as removing any NaNs contained in the first rows. This is seen in the *NaNremove.m* function excerpt in MATLAB script 2-3

```
if (find(isnan(NonNan)) > 0 )
     [row,col,vals]=find(isnan(NonNan));
     EndNaNsRemoved=length(vals);
     NonNanData=NonNan(1:min(row)-1,:); %Remove end rows with NaNs
```

MATLAB script 2-3 removal of end rows containing NaN

The function is then tested with all the three parameters the result can be found in Appendix 6: MATLAB scripts paragraph 7.6.2.

2.4.4.2Sensor value to Temperature conversion

In order to have the data in degrees Celsius the Analogue to Digital Converters (ADC) values needs to be converted. The ADC converter receives the voltage output from the TMP36 temperature sensor which is directly correlated to the current temperature. The ADC on the ZigBee nodes has ENOB of 12bits, ranged from -2048 to 2048, or -3V to 3V [15], and the conversion is done based on the TMP36 sensor scaling. This is seen in Table 2-7.

Table 2-7 TMP36 temperature sensor scaling parameters

Sensor	Offset voltage (V)	Scaling voltage (mV/°C)	Output voltage at 25°C (mV)
TMP36	0.5	10	750

For the conversion the data is first converted back to the sensors voltage from the ADC value as seen in Equation (2-7).

$$Volts = ADC_{data} * \frac{3}{2048} \tag{2-7}$$

The voltage data is then converted into °C seen in Equation (2-8).

$$Temperature = (Volts - 0.5) * 100 \tag{2-8}$$

2.4.4.3The outlier marking function

The outlier removal function will also use MATLAB to interpolation between the gross outliers. Gross outliers have been selected as values that lay 2 standard deviations from the mean of the entire data set. This is however an input to the function, and the size of this standard allowed deviation should be set regarding the length of the sample in question. Since the *Interp1.m* function will return NaN if there are outliers found in the end and start of the data file, this function should be run before the NaN remover. This will ensure that all outliers and all NaNs are removed before smoothing the data. The standard deviations and the mean of the data sets are found using the *repmat.m* function. An excerpt of the outlier detection and removal function can be seen in MATLAB script 2-4 the complete source code is found in Appendix 6: MATLAB scripts paragraph 7.6.3

```
mu = mean(data); %Create a matrix of mean value
sigma = std(data);%Get the standard deviation of the data
[n,p] = size(data);%Get the size of the data matrix
MeanMat = repmat(mu,n,1); % replicating the mu vector for n rows
SigmaMat = repmat(sigma,n,1); % replicating the sigma vector for n rows
outliers = abs(data - MeanMat) > 2*SigmaMat;% Create a matrix of zeros and
ones, where ones indicate the location of outliers
```

MATLAB script 2-4 outlier removal excerpt

The outlier removal function was tested with some noisy data gathered during the experiments and the resulting plots before and after outlier removal can be seen in Figure 2-25.

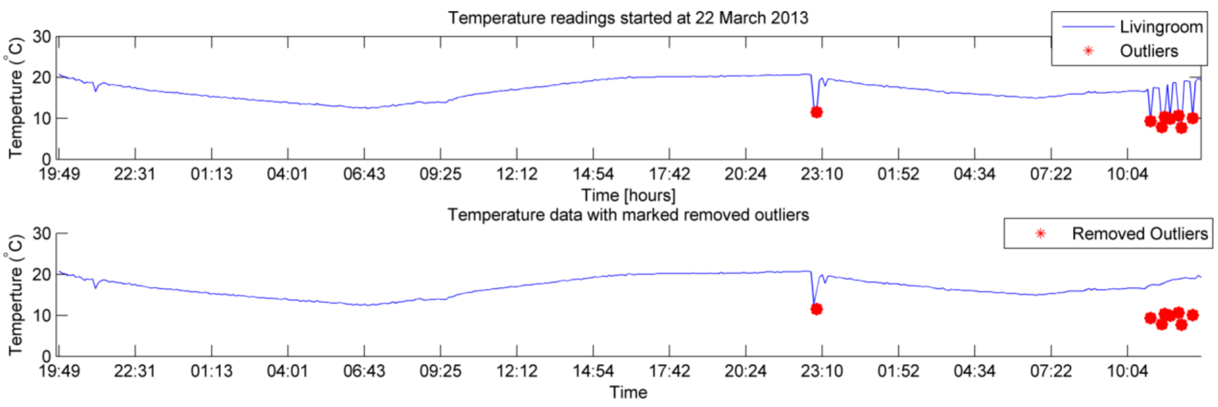


Figure 2-25 Removed outliers using a standard deviation of two, top graph shows the data before outliers are removed and bottom shows after removal.

In this plot it may seem as one outlier is remaining at 23:10 March 23th and this may be discussed, however since it is not removed this means that there are at least 3 consecutive low samples. With a 6 minutes sampling time this means the samples are over an 18 minute period on the March 23th and as such should not be seen as outliers. The reason for these samples seeming erroneous is probably due to a door being kept open for too long letting cold air inside. The last points that have been interpolated are just single samples and as such can be seen as outliers. The data should also be run through a LP smoothing function that will take care of this. The LP filter function can be found in Appendix 6: MATLAB scripts paragraph 7.6.4.

The plots before and after all the data acquisition functions can be seen in Figure 2-26 and Figure 2-27 respectively. The script loading the log data, running the data processing functions and creating the correct axis and output format can be found in Appendix 6: MATLAB scripts paragraph 7.6.1.

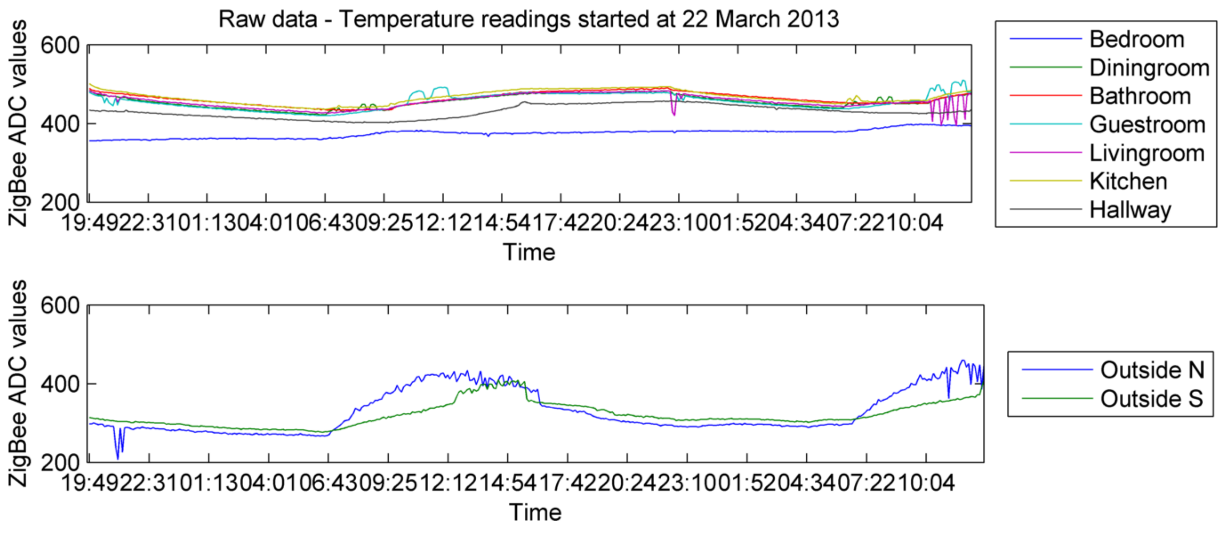


Figure 2-26 Raw data received from the ADC of the ZigBee nodes, before data processing

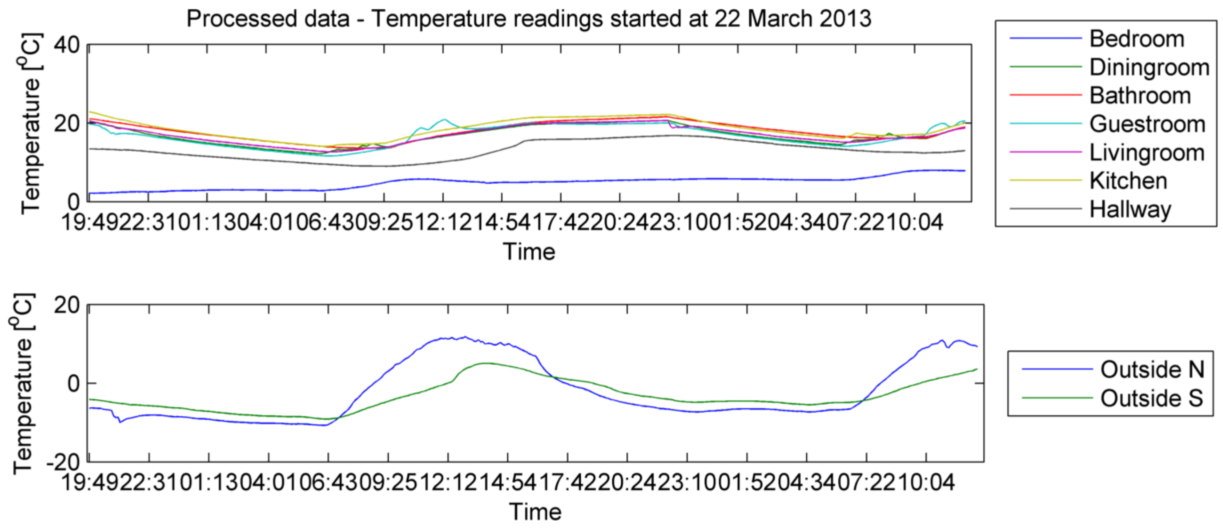


Figure 2-27 Temperature data after data processing

2.4.5 Discussion

The experiments clearly indicated that the model has some problems with estimating the correct time constants of the house. The cooling and heating times lasted both much longer in the real house than model simulations [5]. In order to better visualize this difference, the sensor values has been averaged to yield one inside temperature, and one outside temperature. Both the "*all power on*" and "*all power off"* experiments data will be compared with the model simulations using the same environmental conditions and house parameters. In the simulations the temperature in the ground is seen as 5°C higher than in the air for simplicity.

The data used for the all power off simulation is from March 28th and 29th and gives representable data from all the "*all power off"* experiments. Some data colored by noise should still be removed before comparing with the model simulations. From 07:00 to 09:00 on 29 March the inside temperature data is too colored by the sun which is seen from the outside temperatures in Figure 2-28. From 18:33 to 19:30 the data is also somewhat colored by unknown disturbances, probably cold wind from opening the door when leaving the house. The processed data for the entire power off interval can be seen in Figure 2-28.

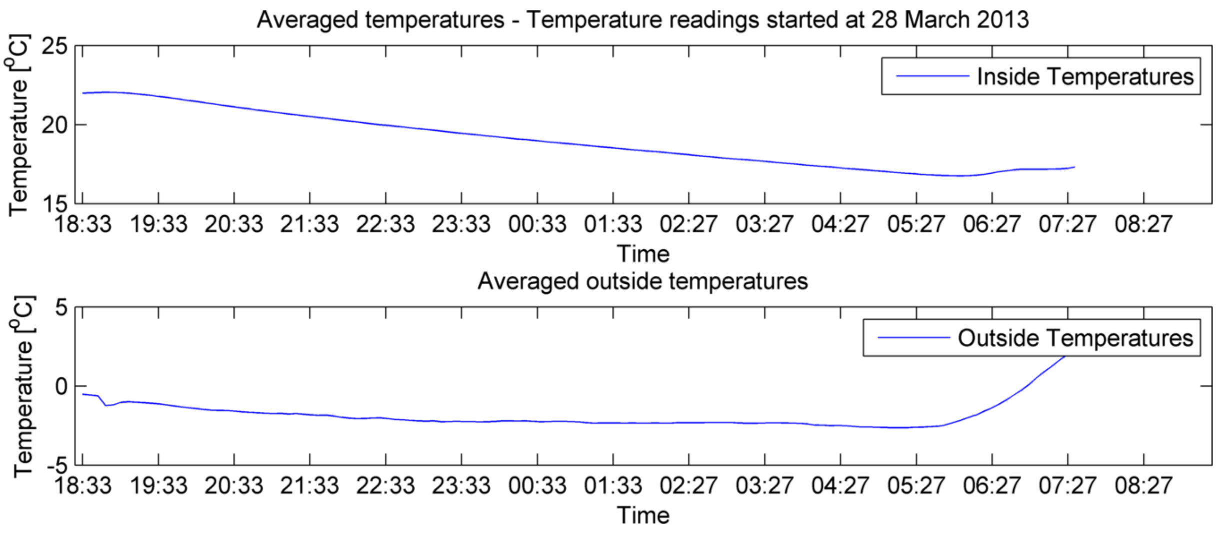


Figure 2-28 Averaged temperatures, inside temperature seen in top graph and outside temperature seen in bottom graph.

The removal of this noise colored data results in a 10 hour period of good data quality ready for comparison. This temperature data is seen plotted together with the non-linear model in Figure 2-29.

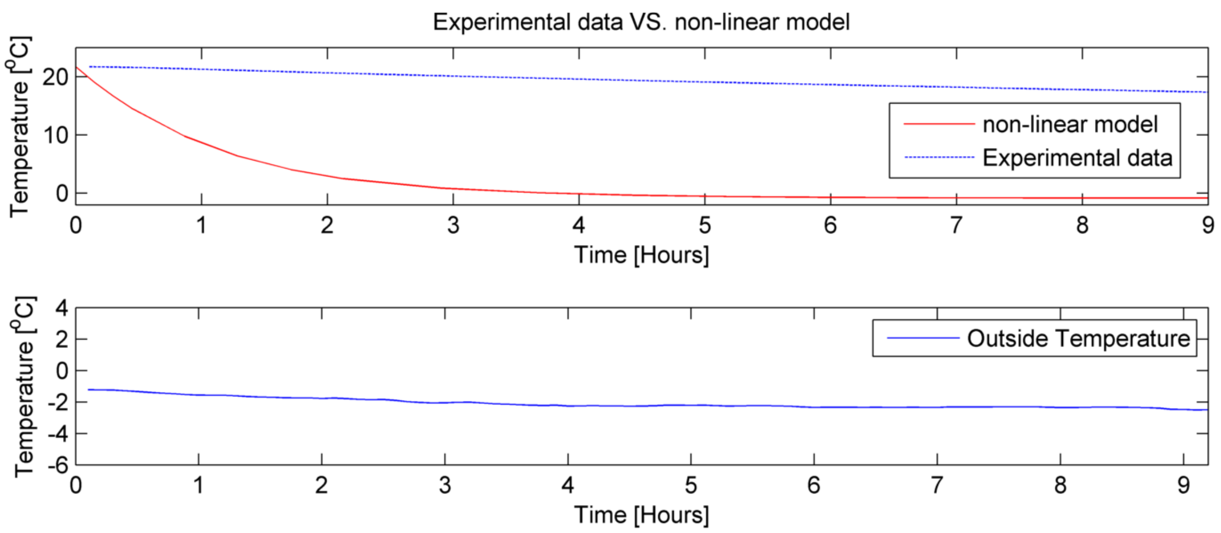


Figure 2-29 experiment data "all power off" VS non- linear model, inside temperature seen in top graph and outside temperature seen in bottom graph.

In the same manner an all power on representable data set was found at March 22th from 09:30 to 19:00 and is plotted next to the non-linear model seen in Figure 2-30.

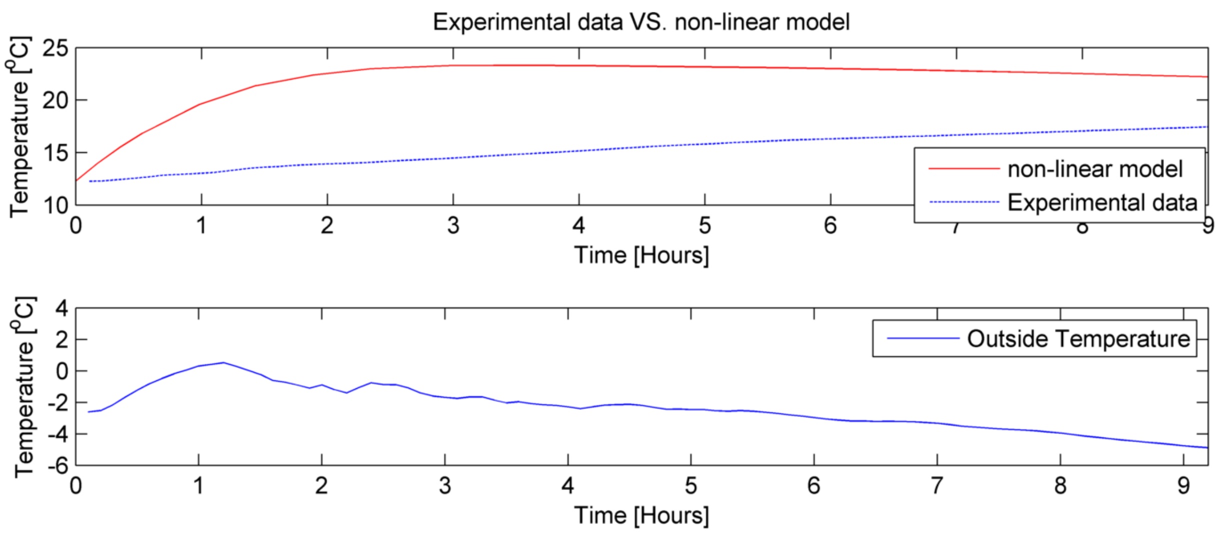


Figure 2-30 Experiment data "all power on" VS non-linear model, inside temperature seen in top graph and outside temperature seen in bottom graph.

In Figure 2-29 and Figure 2-30 as the non-linear model fits the experimental data poorly. There are several reasons for this:

1. The model does not take into account the mass of the house the walls the floor the furniture etc. All which has a lot of mass and much specific heat capacity.
2. A statistical U- value has been used TEK10.
3. The Ventilation is set to a statistical, TEK10, value and is not measured
4. The effect of the sun i.e. the solar rays are not measured
5. The temperature of the ground is unknown

The statistical data for the house blocks U-values and the ventilation might contribute significantly to the erroneous of the non-linear model. Both these parameters are changeable in the model and an experimental test was devised in order to test the correctness of the statistical data: *All known ventilation was closed and the inside house temperature was kept at steady state during which the power consumption was monitored.*

At March 29th the outside and inside temperatures reached something close to steady state conditions, this can be seen in Figure 2-31.

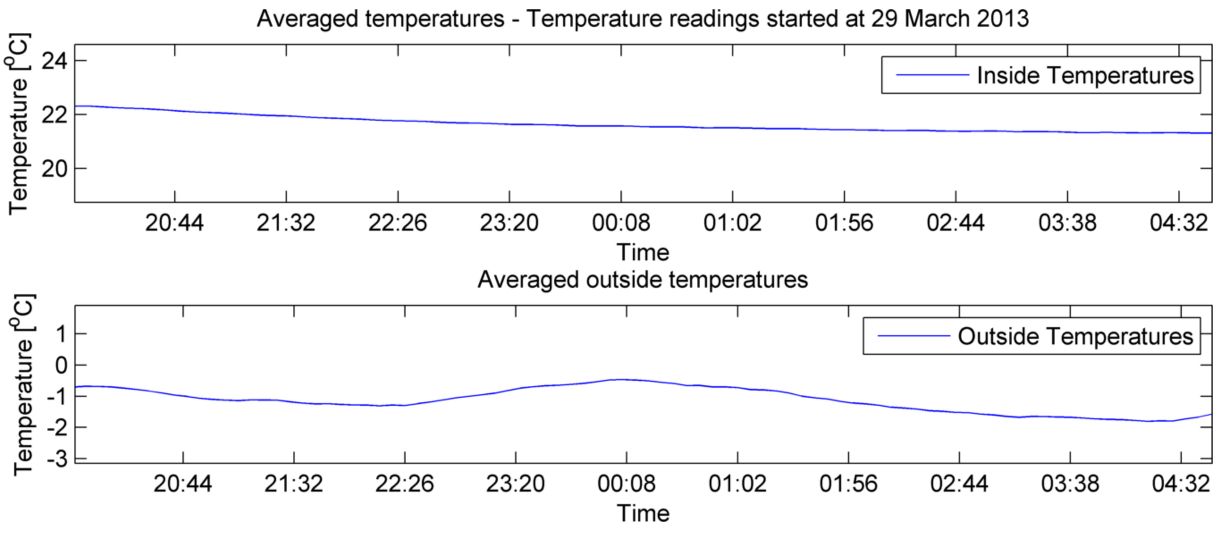


Figure 2-31 «Steady state» conditions for temperatures

Even though the temperatures seen in Figure 2-31 are not in a completely steady state they should give a good approximation to the total house heat leakages. The inside temperature only changes from 21.5 ± 0.6, and the outside temperature change is -1 ± 0.5 over the interval of eight hours, and nine kWh of power was used which is approximately 1.1kW per hour. The temperature difference is taken as the mean of the differences over the length of the time interval. This gives $\Delta T \approx 22.6$, and the total area of the house is $A = 218.6m^2$. The total estimated leakage factor can then be estimated using equation (2-3), under paragraph 2.2.1.1 House and model parameters. The result is seen in equation (2-9).

$$\bar{U}_T = \frac{Q_{work}}{A\Delta T} = \frac{1125W}{218.6m^2 * 22.6°C} \approx 0.228W/m^2K \qquad (2\text{-}9)$$

The total house U value result of approximately 0.23 was set in the model with the total house area and the model was run and compared to the TEK 10 U values results. This can be seen in Figure 2-32.

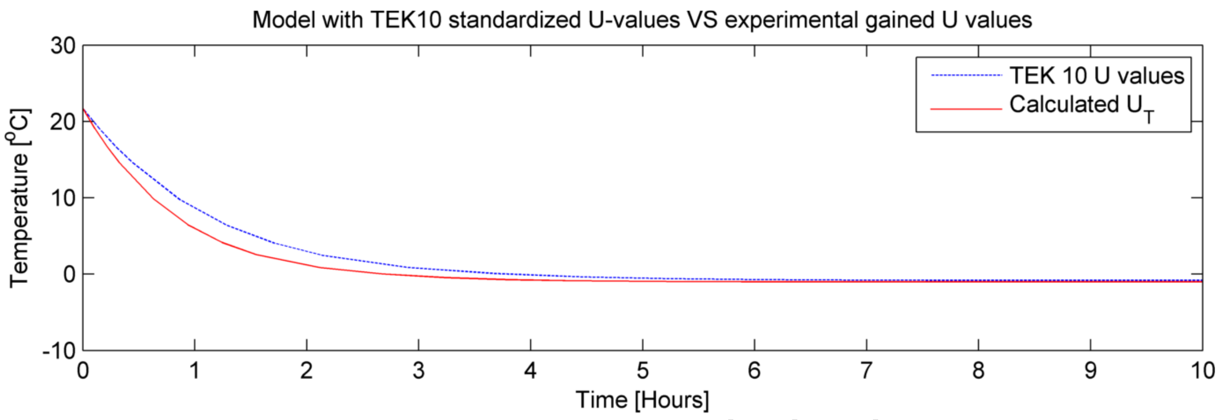


Figure 2-32 TEK10 vs. estimated total U-value

The TEK 10 standardized values show somewhat longer cooling time than the experimentally gained U – value. This means the TEK-10 used values are probably quite accurate since the total U-value also will include other leakages[6]. This also mean that the house mass is extremely important when creating a house model usable for control, and this heat capacity factor needs to be included in the model. This will be further discussed in the next part on creating the heating time estimation in section 3.

[6] Leakages between elements (door and walls, window walls, etc) and in corners .

Part 2

3 Estimation of heating time

3.1 Introduction

The main reason for using a model in this BAS system is to have a prediction of how the inside house temperature will change over time. Specifically the model should be used to estimate the heating time with current environmental conditions. This heating time estimation should be accurate in order to both minimize the amount of power used and reach the correct temperature. In 2.4.5 the house model did not correspond to the experimental data. The house mass heat capacity will prolong the time used for heating and make any heating time estimation too short. Therefore a new prediction model needs to be created.

3.1.1 System description

The heating time estimation is based on a temperature reference named comfort intervals. The comfort intervals are references to when the residents are at home, and not sleeping.

Heating time estimation is needed to reach the comfort temperature when the residents are home from work or getting up in the morning. The estimation will be made with a prediction model calculating the present heating time estimate using the environmental conditions from the gateway seen in part 1 Data acquisition. This heating time estimation will be run each sampling time and when estimated time corresponds with the time remaining until the comfort interval, the heaters should be turned on. This is visualized in Figure 3-1 where heating estimations are seen as the blue dotted lines and the solid blue line is when the estimation is the same as the time remaining till comfort interval. The red line denotes the temperature reference.

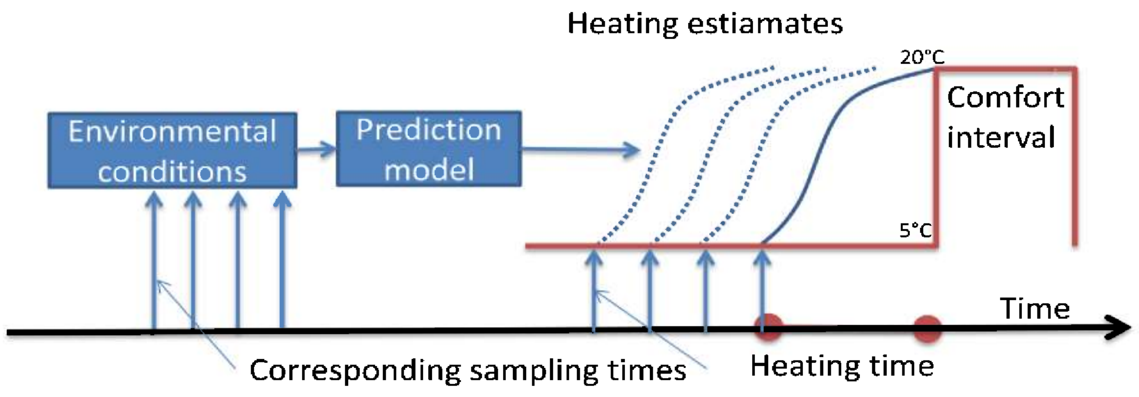


Figure 3-1 Heating time estimation showing heaters turned on at the correct time based on the current environmental conditions.

3.1.2 Contents and structure

Part two will first consider the theory behind the temperature references and prediction model, before simulations are performed in MATLAB. The temperature prediction system will be implemented in Visual Studio and then the finished system is tested in an experiment. Lastly the results from this chapter will be discussed. The structure of part two is seen in Figure 3-2 where the main chapters are seen on the left and the sub chapters are on the right.

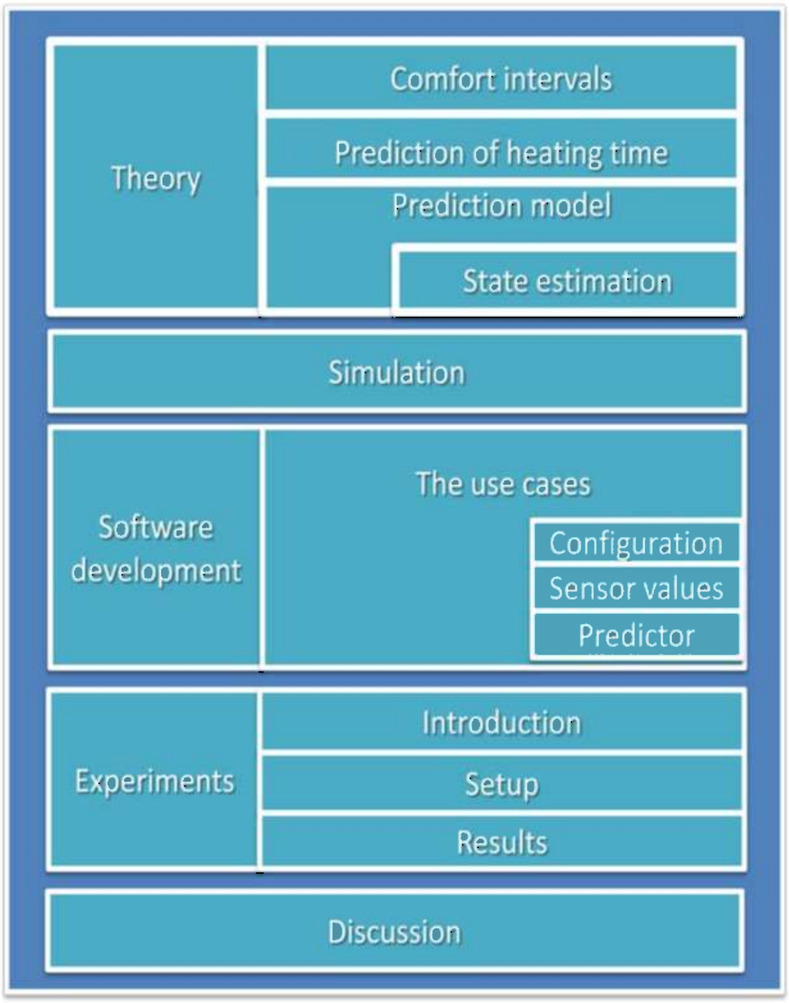


Figure 3-2 Part 2: heating time predictions structure main chapters on the left and sub chapters on the right, and progress proceeds in a downwards fashion.

3.2 Theory

3.2.1 Comfort Intervals

In order to control the inside temperature of a house there is need for a control reference. In a home this control reference will be based on when the occupants are at home and what comfort temperature is preferred. In a completed system the comfort intervals will be based on the specific residents work schedule, and bed time. In a completed system this can be learned using the sensors, for this analysis and simulation purposes however "typical" family workweek will be invented: *The residents will go to bed at 23.00 get up at 07.00 go to work at 08.00 and get home from work at 16.00.*

When the house is in use the resident will want a constant comfort temperature of 20°C, this means on a "typical" weekday for this invented resident will follow this schedule. For a typical weekend Saturday and Sunday, the residents may get up a little later around 09.00 and go to bed a little later around 01.00. The temperature references can be seen in Figure 3-3 where the weekday temperature references are seen on the left and weekend on the right.

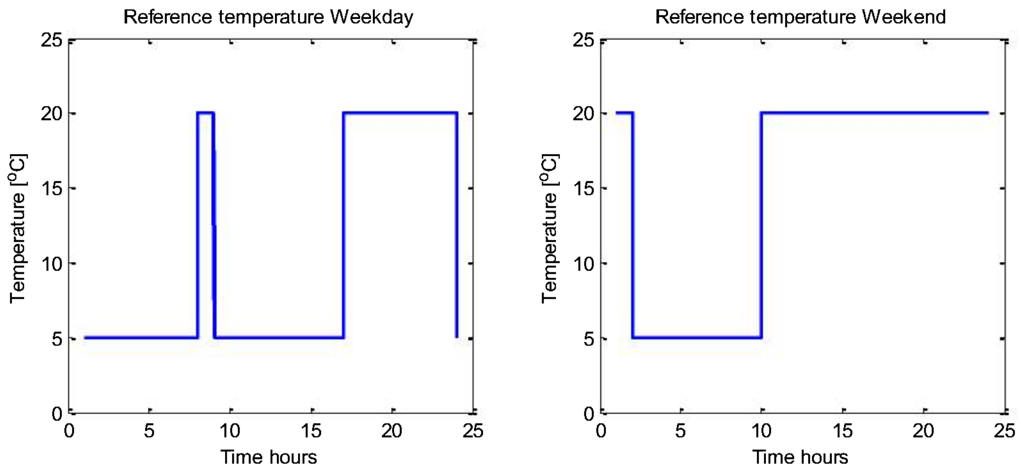


Figure 3-3 Temperature comfort intervals, reference zones weekday on the left and weekend on the right

In the case of when the temperature may be lower the heater can be turned completely off saving the most amount of energy, however there should be some limitations making sure the temperature never reaches below 5°C. The Comfort intervals are created in a MATLAB function to be used as reference vectors in in part 3 Control System simulations.

In Figure 3-3 the amount of time where the temperature needs to be at comfort level, 20°C, at weekdays is less than when the heaters can be turned off. This means a lot of power can be saved using temperature prediction in a control system. Some straight forward calculation gives that for one complete week there are 72 hours in which the temperature should be at comfort level seen in Equation (3-1).

$$Weekdays\ 1hour\ +\ 7hours * 5 = 40$$
$$Weekends\ 16hours * 2 = 32 \qquad (3\text{-}1)$$
$$168hours\ pr\ week - 72\ hours\ heater\ on = 96$$

This means that there is 96 hours a week, about *7 months* of the year where the heaters can be turned completely off saving power. Even though the heaters will need to be turned on more than these 44%, this is still a good visualization on how much power can be saved.

3.2.2 Prediction of heating time

In order to save the maximum amount of power in a house a heating time prediction model is needed, the model will be used to predict future behavior based on future known references and current environmental data. The temperature estimator will be used as a reference into the future. The reference will be r_{k+L} where L the heating time is calculated by the prediction model, and k is present time. The future references are based on the residents comfort intervals.

Turning the power completely off and then on again without using a heating time estimation would result in the temperature being too low in the comfort interval. This is visualized in Figure 3-4

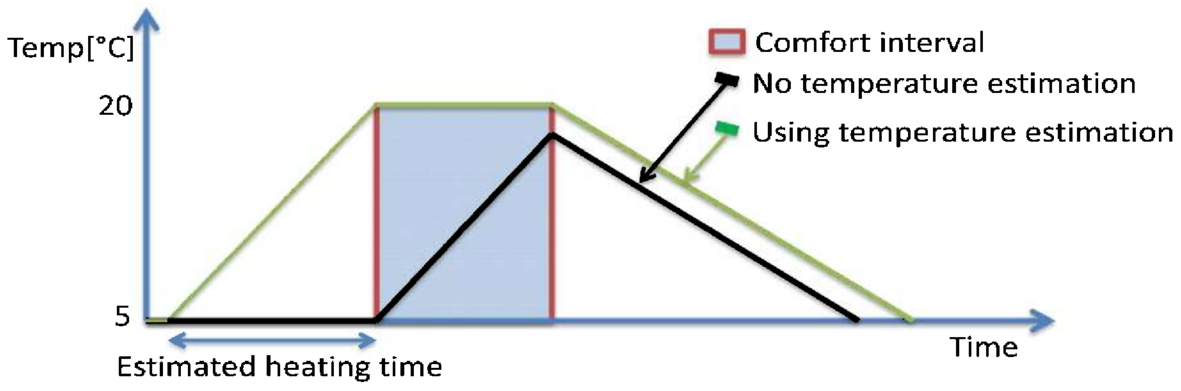


Figure 3-4 Why there is needed a temperature estimator to reach the set point temperature in time

Another way would be to use a fixed heating time based on the fixed condition parameters, often included as the temperature lowering systems currently available [4]. This will not be the most efficient as the set point would be reached too soon in almost all cases. In Figure 3-5 this is visualized where the comfort temperature is reached too soon and power is lost.

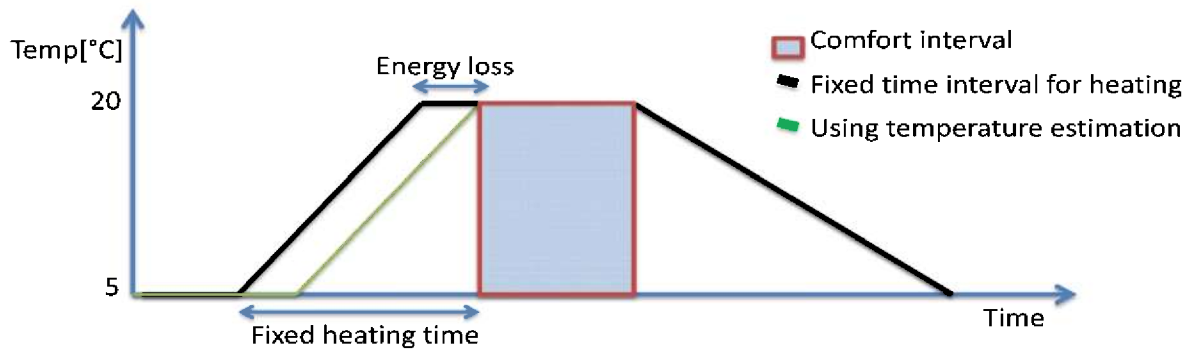


Figure 3-5 Energy loss using fixed heating time, the lowering scheme [4]

This last power saving scheme is not optimal, and the main reason for this being implemented in current power saving systems [4] is that there is no house model to estimate the heating time. This thesis is based on a house model and an approximate heating time can be calculated, resulting in saving the most amount of power.

The house model was shown to be inadequate in its present form since there is no implementation of the house mass and the house heat capacity. The heat capacity of the house will be very hard to measure in any direct way, and should if possible be estimated by an observer. In addition the air densities are not currently measured and should for this reason be added to the unknown disturbance factor. This will be further discussed in the next section.

3.2.3 Temperature prediction model

The only known parameters for each sampling time is the inside and outside temperature. In order to have the best possible fit to the experimental data there should be used two states one for the inside temperature and one for the disturbances. The outside temperature will be viewed as a slowly varying disturbance and included in the state calculation at each sampling interval. The remaining disturbance value, mainly the mass heat capacity, should if possible be estimated by an observer.

The estimation of the heating time is based on the first differential function seen in Equation (2-1), the house parameters and the current measured values from the sensors.

The house mass heat capacity and other disturbances are added as an extra state, ϑ_m to the temperature part of the house model, the outside temperature is denoted v_T, and the inside temperature is denoted x_1. The disturbances are viewed as constant or slowly varying $\dot{\vartheta}_m = 0$, $\dot{v}_T = 0$. The differential function for inside temperature, $f_1 = \dot{x}_1$ augmented with the disturbance mass, $\vartheta_{m,}$ can be seen in Equation (3-2).

$$f_1 = \frac{\left(v_T\dot{V_o} - \rho_i\dot{V_\iota}\right)}{v_T V} x_1 + \frac{1}{v_T V(\widehat{c_p} - \frac{R}{M})}\left(\rho_i\dot{V_\iota}\widehat{H_i} - v_T\dot{V_o}\widehat{H_o} + u - UA * \Delta T\right) + \vartheta_m \qquad (3\text{-}2)$$

The heat loss equation (2-3) has been expanded into the first differential equation (2-1) where the difference between the inside and the outsider temperatures are seen as ΔT, and the heater power as u, and house area as A. The other parameters are denoted in Table 2-1.

The model is then transformed into a linear state space representation (SSM) in order to implement the augmented changes and using the model for prediction. The complete set of differential functions is then seen in Equation (3-3)

$$f(x, u, v_T) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} Inside\ temperature \\ Distrubances \end{bmatrix} \qquad (3\text{-}3)$$

The state matrix A is found as the two first (linear terms) of a Taylor series expansion of the right hand side around the points x_0 and u_0 seen in Equation (3-4) [19].

$$A = \left.\frac{df}{dx^T}\right|_{ss} = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{d\vartheta_m} \\ \frac{df_2}{dx_1} & \frac{df_2}{d\vartheta_m} \end{bmatrix}_{ss} \qquad (3\text{-}4)$$

The transition matrix B is found as the Jacobin seen in Equation (3-5)

$$B = \left.\frac{df}{du}\right|_{ss} = \begin{bmatrix} \frac{df_1}{du} \\ \frac{df_2}{du} \end{bmatrix}_{ss} \qquad (3\text{-}5)$$

The model in Equation (3-2) is then expanded with all parameters [1], and deviated using symbolic MATLAB the results of this derivation operation can be seen from Equations (3-6) through (3-11). The complete expanded model can be found in Appendix 7: Expanded model.

$$\frac{df_1}{dx_1} = \frac{\frac{N\dot{V_\iota}}{3600}\left(\rho_i - \frac{PM}{Rv_T}\right)}{\rho_i} + \frac{-\dot{V_o}\left(C_{pa} + f_r Cpw\right) - UA}{V_r\rho_i C_p} \qquad (3\text{-}6)$$

$$\frac{df_1}{d\vartheta_m} = 1 \qquad (3\text{-}7)$$

$$\frac{df_2}{dx_1} = 0 \qquad (3\text{-}8)$$

$$\frac{df_2}{d\vartheta_m} = 0 \qquad (3\text{-}9)$$

$$\frac{df_1}{du} = \frac{1}{v_T V(\widehat{c_p} - \frac{R}{M})} \tag{3-10}$$

$$\frac{df_2}{du} = 0 \tag{3-11}$$

Evaluated at the steady state of $x_{1|ss} = 20°\mathrm{C}$, $v_T = -5°\mathrm{C}$ $u = 910W$ the SSM is found as seen as in Equations (3-12) through (3-14)

$$A_c = \begin{bmatrix} -8.8817e^{-4} & \vartheta_m \\ 0 & 0 \end{bmatrix}_{ss} \tag{3-12}$$

$$B_c = \begin{bmatrix} 0.4392730484e^{-5} \\ 0 \end{bmatrix} 0_{ss} \tag{3-13}$$

$$D = [1 \quad 0] \tag{3-14}$$

The system is discretized using zero order hold and sampling time corresponding to the experiments sample time of six minutes. The calculation of the discrete matrices using the zero order hold numerical method can be seen in Equations (3-15) through (3-19) , the numerical method is based on [20].

$$A_d = I + Ah + \frac{A^2h^2}{2!} + \frac{A^3h^3}{3!} + \cdots \frac{A^nh^n}{n!} \tag{3-15}$$

$$= I + A\left(\underbrace{I\,h + \frac{Ah^2}{2!} + \frac{A^2h^3}{3!} + \cdots \frac{A^{n-1}h^n}{n!}}_{S}\right) \tag{3-16}$$

$$= I + AS \tag{3-17}$$

$$B_d = \left(\underbrace{I\,h + \frac{Ah^2}{2!} + \frac{A^2h^3}{3!} + \cdots \frac{A^nh^n}{n!}}_{S}\right)B \tag{3-18}$$

$$= SB \tag{3-19}$$

After running the discretization function using the first ten polynomials the discrete SSM (dSSM) is seen in Equation (3-20) and Equation (3-21).

$$A_d = \begin{bmatrix} 0.7263 & 308.12 \\ 0 & 1 \end{bmatrix} \tag{3-20}$$

$$B_d = \begin{bmatrix} 0.0014 \\ 0 \end{bmatrix} \tag{3-21}$$

Observability

In order to introduce an observer into the system the system needs to be fully observable. A SSM system will be fully observable if the rank of the observability matrix is equal to the number of states in the system [19]. The observability matrix for this two state system is calculated from the state matrix, A_d in Equation (3-20) and the output matrix D in Equation (3-14). The result can be seen in Equation (3-22) and Equation (3-23).

$$O = \begin{bmatrix} D_d \\ D_d A_d \end{bmatrix} \tag{3-22}$$

$$O = \begin{bmatrix} 1 & 0 \\ 0.7263 & 308.12 \end{bmatrix} \tag{3-23}$$

As seen in Equation (3-23) the rank of the observability matrix is two and equal to the number of states in the system which means the system is fully observable. This means the system can implement an observer for the house mass heat capacity and extra disturbances.

The optimal observer Kalman filter will be used for the state estimation and will be discussed in the next section.

3.2.3.1State estimation

The Kalman filter is a well-known model based algorithm useful to estimate unmeasured variables. It is seen as an optimal observer in the sense that the variance of the measurement error is minimized. The Kalman filter will implemented on the apriori-apostriori form and the block diagram can be seen Figure 3-6.

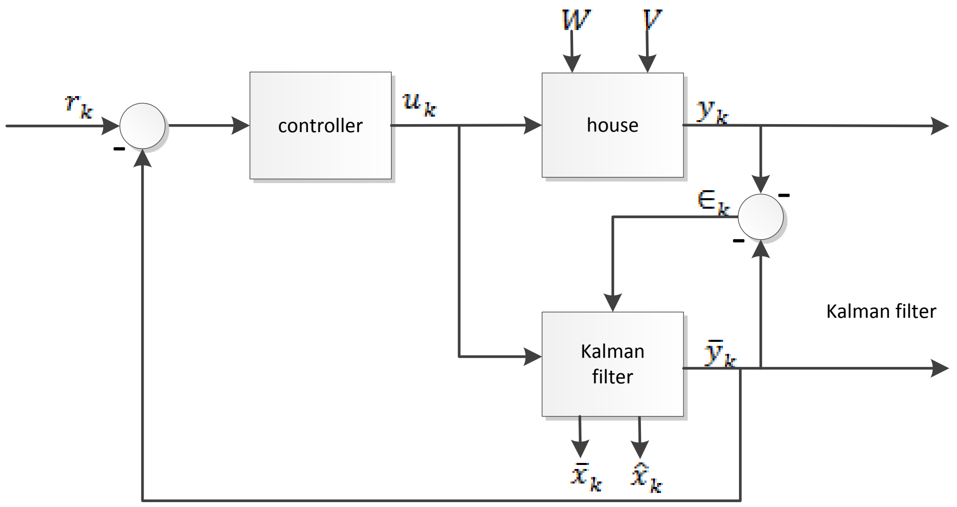


Figure 3-6 Kalman filter block diagram

In Figure 3-6 $\bar{x}_k$is the apriori state estimate, $\hat{x}_k$ is the apostriori state estimate, $\bar{y}_k$ is the predicted output, $r_k$is the reference and $\in_k$ is the error between the predicted output and the measured output. W is assumed to be slowly varying stochastic process noise and V is assumed to be slowly varying stochastic measurement noise. The state space model can be seen in Equation (3-24) and Equation (3-25).

$$x_{k+1} = Ax_k + Bu_k + W \tag{3-24}$$

$$y_k = Dx_k + V \tag{3-25}$$

In order to get in the algorithm the apriori state estimate initial values, needs to be specified.

$$\bar{x}_{k=0} \tag{3-26}$$

Then the rest of the algorithm should be run in all other time instances seen in Equation (3-27), (3-28), and (3-29).

1. Calculate the predicted output measurement

$$\bar{y}_k = D\bar{x}_k \tag{3-27}$$

2. Calculate the aposteriori state estimate

$$\hat{x}_k = \bar{x}_k + K\,\underbrace{(y_k - \bar{y}_k)}_{\in} \tag{3-28}$$

3. Update the apriori state estimate

$$\bar{x}_{k+1} = A\hat{x}_k \tag{3-29}$$

K is the constant Kalman filter gain calculated by minimizing x seen in Equation (3-30) , where X is the solution to the Riccati equation seen in Equation (3-31).

$$K = X * D^T * W^{-1} \tag{3-30}$$

$$AX + XA^T - XD^TW^{-1}DX + V = 0 \tag{3-31}$$

The constant Kalman filter gain can be found by the *kalman.m* function in MATLAB, since however the Kalman filter should be implemented in a C# control system the best way is to use an iteration based solution. Constant Kalman filter gain converges fast and is found within a few iterations [21]. The Kalman filter gain is found iteratively by the following algorithm seen in Equations (3-32) through (3-35).

1. Specify initial values for the states covariance matrix

$$\bar{P}_{k=0} \tag{3-32}$$

2. Obtain an estimate of the states covariance matrix before reading the output

$$\bar{P}_k = A\hat{P}_{k-1}A^T + Q \tag{3-33}$$

3. Obtain the Kalman filter gain matrix

$$K_k = \bar{P}_kD^T(D\bar{P}_kD + R)^{-1} \tag{3-34}$$

4. Correct the state covariance matrix after reading the output

$$\hat{P}_{k+1} = (I - K_k D)\bar{P}_{k+1} \tag{3-35}$$

Where Q is the covariance matrix for the process noise, R is the covariance matrix for the measurement noise, K is the Kalman filter gain, and A and D are the SSM matrices with proper dimensions. The Kalman filter gain iterative calculation is based on [22]. The Kalman filter will be implemented and tested in the next section and further tested in the experiments in section 3.5.

3.3 Implementation and simulation

Before coding the Kalman filter and using it directly in the temperature predictor some more analysis is needed. This will be based on implementing the Kalman filter in MATLAB and simulation using the experimentally gained values from section 2.4. First the weighting parameters need to be defined. The covariance matrix for the measurement noise R is normally found as the variance of the time series [21]. Using the *var.m* function in MATLAB the result can be seen in Equation (3-37).

$$R = [2.32] \tag{3-36}$$

The covariance matrix for the process noise Q is seen as a tuning parameter with each component responding to the variance of that specific state. The Q matrix is tuned to the highest value not causing too noisy measurements [19]. The found values can be seen in Equation (3-37).

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{3-37}$$

The Kalman filter algorithm can be seen implemented in MATLAB script 2-1

```
%Kalman gain matrix calculation-------------------------------------------
    Phat=A*Phat*A'+Q;        %Estimate of covariance matrix
    K=Phat*D'/(D*Phat*D'+R)  %Kalman filter gain matrix
    Phat=(I-K*D)*Phat;       %Correct the state covariance matrix
%State estimation---------------------------------------------------------
    Ykbar=D*x;               %Predicted output measurement
    Yk=AvgInTemp(k);         %Read output vector
    Xhat=Xbar+K*(Yk-Ykbar)   %Aposterori state estimate
    Xbar=Ad*Xhat+Bd*u;       %Update the apriori state estimate
```

MATLAB script 3-1 Kalman gain calculation and state estimation

After the simulation has run four iterations the steady state Kalman filter gain is found as seen in Equation (3-38).

$$K = \begin{bmatrix} 0.998 \\ 0.0032 \end{bmatrix} \tag{3-38}$$

The resulting plots from the predicted output measurement using the estimated states from the Kalman filter can be seen in Figure 3-7.

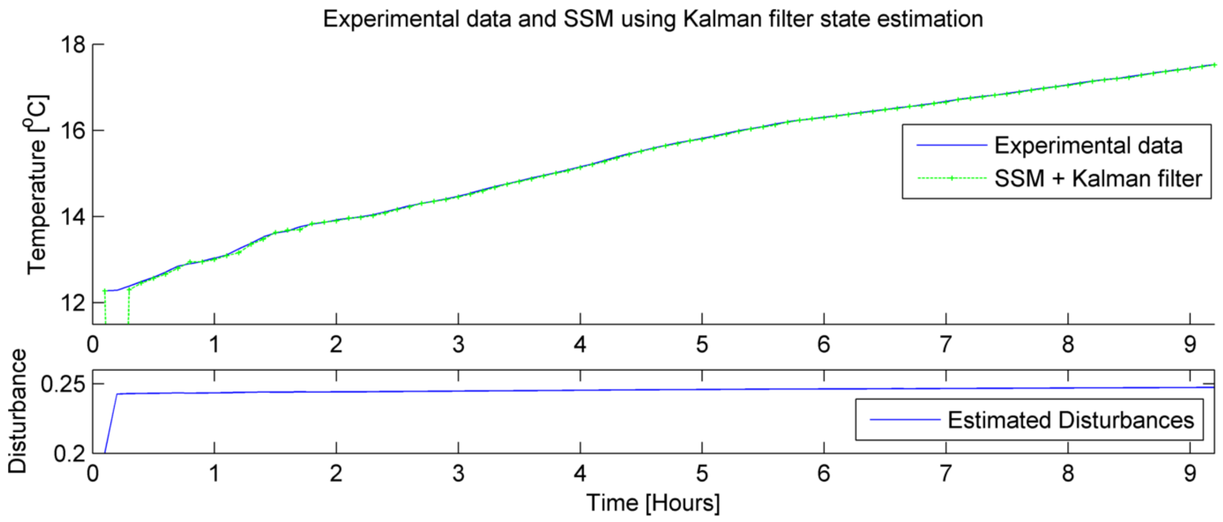


Figure 3-7 SSM model simulation using Kalman filter state estimation

In Figure 3-7 the SSM model with estimated disturbance states fits the experimental data nicely. The disturbance estimation is increasing with the increasing inside temperatures probably due to the increasing amount of thermal energy being stored.

This increase is not negligible due to the large weight the disturbance has on the temperature state [2.4.5]. The increase is however close to linear and can be fitted using one polynomial ordinary least squares regression (OLS). In Figure 3-8 the disturbance change is seen plotted together with the fitted least squares line.

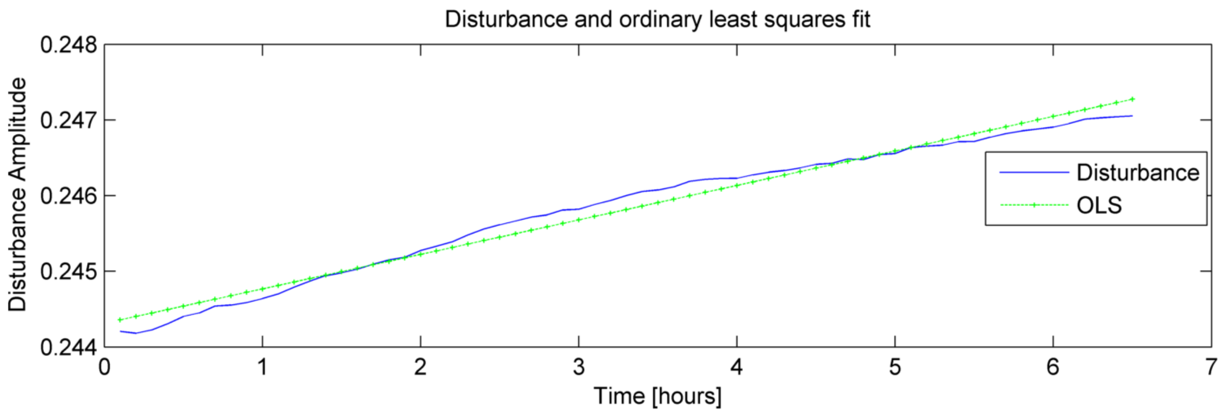


Figure 3-8 Disturbance and least square fit

In order to create the fit line and have a good heating time estimation based on the model there is needed a first heating run where the disturbance parameters can be learned. This should be done by keeping the temperature at the low point before turning all heaters to the maximum storing all the values until the set point temperature is reached. The new Kalman OLS State Space predictor (K-OLS-SSM) together with the non-linear model and pure linear regression estimation (OLS) can be seen estimating heating time as in Figure 3-9.

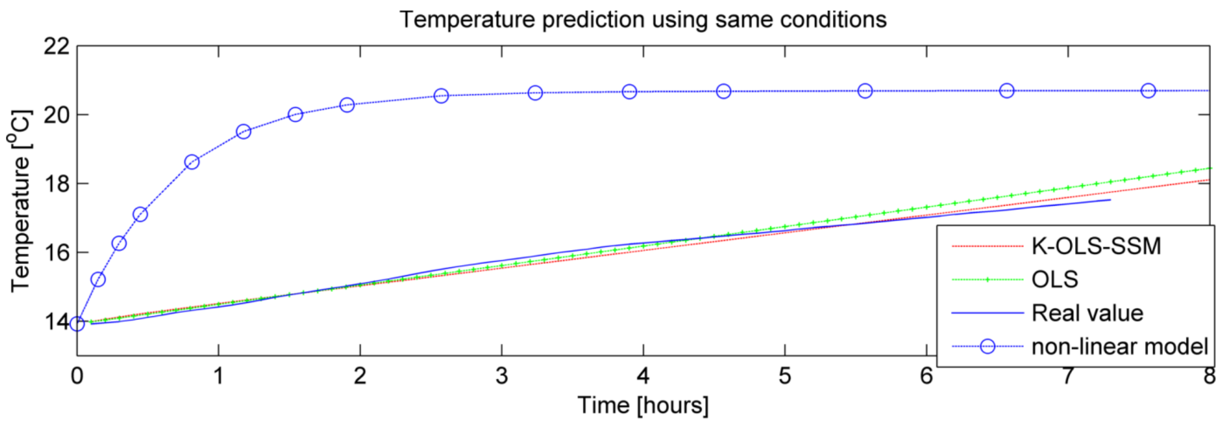


Figure 3-9 Temperature predictions using three methods

The temperature estimation is on the same data as used to create the disturbance vector and the close fitting is to be expected, this is also evident when seen at the close fit to the pure linear regression fit (OLS). A new test set is gathered from the 24th of March in order to verify the prediction model on a different data set with different environmental conditions[7]. For the test set the two best estimations will be used, the purely OLS estimate and the K-OLS-SSM. The heating predictions can be seen in Figure 3-10.

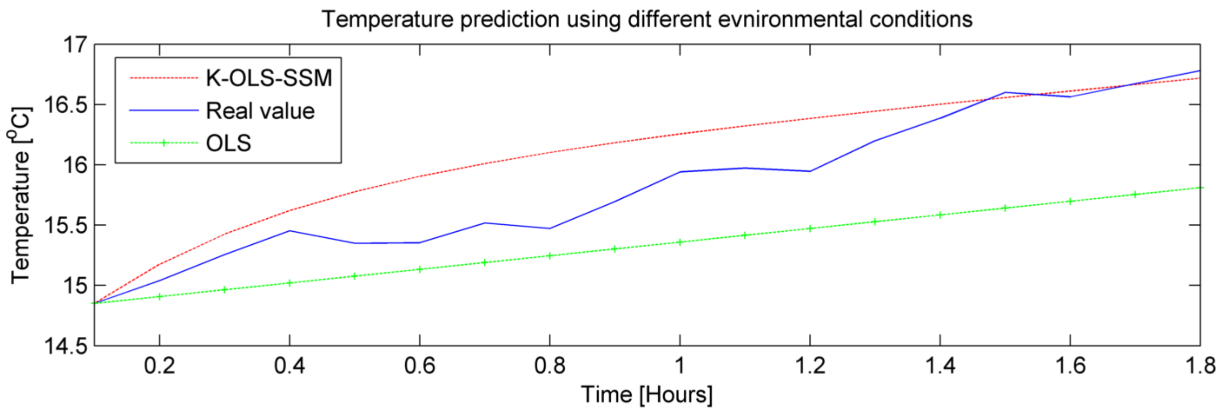


Figure 3-10 Heating predictions using a real test set

The results shows that the Kalman filter and OLS based estimate produce the best heating predictions. The SSM model is derived from the present conditions and will make the predictor more adaptable to change. There should be noted that the test set period only two hours and the functions should be more tested in the experiments part found in section 3.5. For this reason both the linear OLS regression and the Kalman estimated predictor will be implemented in VS.

[7] The test sets on all other occasions of heating time is too much colored by the sun. Heating tests should for this reason in the future be set to run at night.

3.4 Software Development

The temperature predictor needs to have the current measurements, house and model data in order to predict the heating time. These parameters are needed to create the discrete state space model using the current environmental conditions. In addition the temperature reference, the comfort intervals are needed to know when to send the start heating signal to the controller. The temperature references, model and house parameters should be stored in the XML configuration file in the same manner as in paragraph 2.3.2. The user should be able to set the comfort interval reference from a weekly table and the temperature reference system should get the current time date information in order to do the prediction into the future. In order to handle both Single Input Single Output (SISO) and Multiple Input Multiple Output (MIMO) systems, matrix calculation methods are needed. The FURPS+ sheet can be found in the Appendix 3 FURPS+ paragraph 7.3.2.

3.4.1 The use case diagram

The use case diagram is made to get a visual representation of the requirements to the temperature prediction system. The use case diagram can be made based on the found requirements can be seen in Figure 3-11.

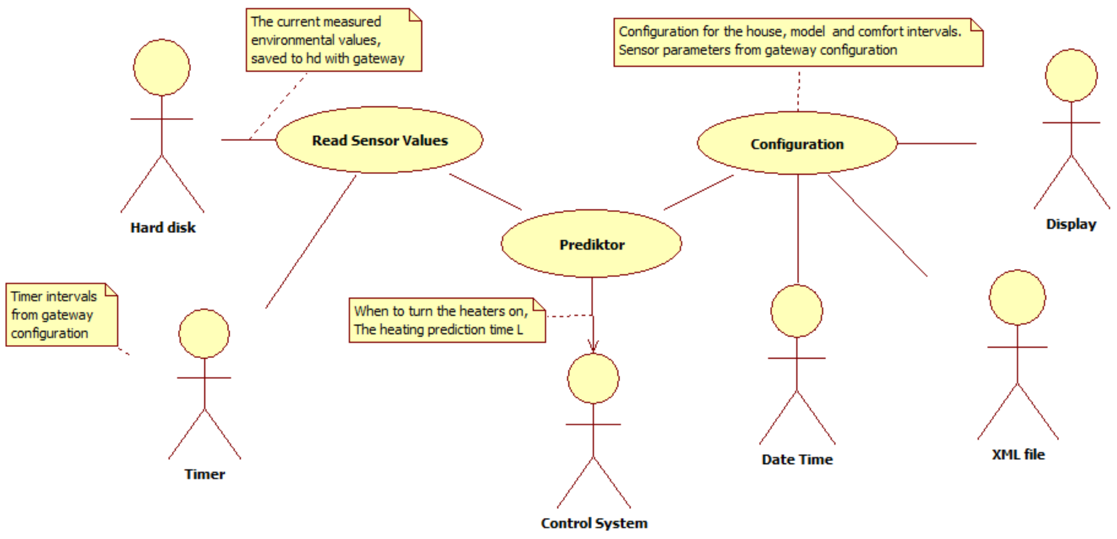

The current measured environmental values, saved to hd with gateway
Configuration for the house, model and comfort intervals. Sensor parameters from gateway configuration
Hard disk
Read Sensor Values
Configuration
Display
Prediktor
Timer intervals from gateway configuration
When to turn the heaters on, The heating prediction time L
Timer
Date Time
XML file
Control System

Figure 3-11 Use case diagram of the Temperature Predictor

There are several classes in some use cases and to have a better visualization of these classes a layered architecture has been created in Figure 3-12 to visualize the classes in each use case. The configuration use case consists of three classes and the predictor use case consists of two classes.

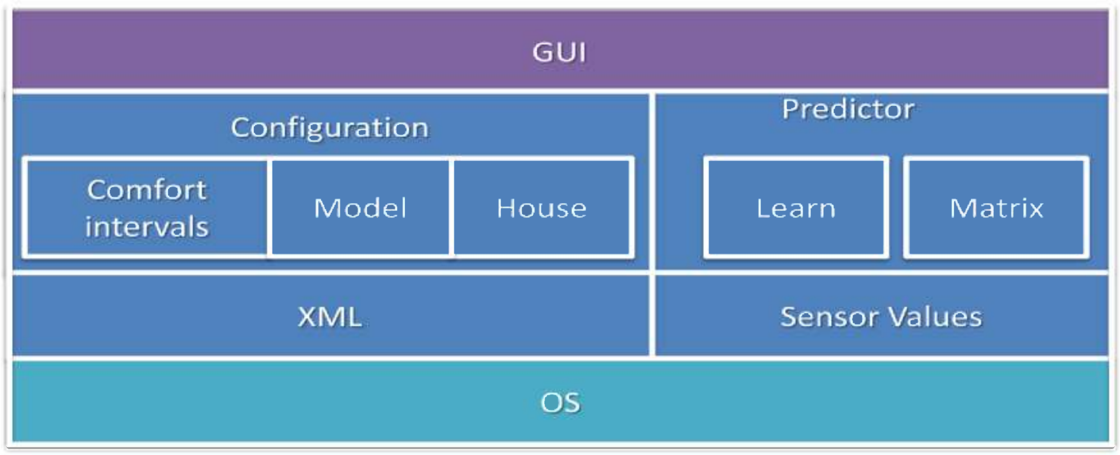


Figure 3-12 Layered architecture of the heating time estimation

3.4.2 The Configuration use case

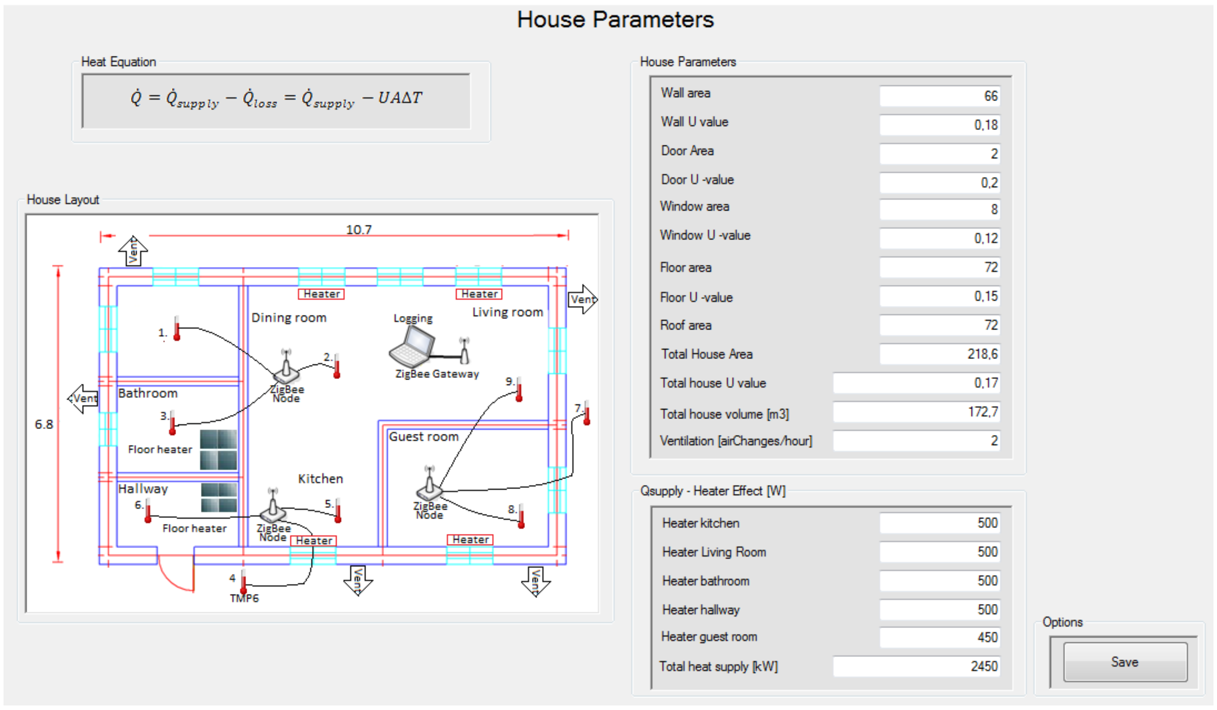
The configuration contains the house and model parameter and are responsible for the GUI between the user and these settings. The configuration is also responsible for storing the parameters to a configuration file. The configuration file will be based on the XML configuration used in the gateway from part 1 Data acquisition with additional elements for the new information.

There are needed many configuration parameters and the GUI is divided into two main Windows Forms (WF), the house and heater parameters and the model parameters.

The fully dressed use case document can be found in Appendix 4: Fully dressed use case documents paragraph 7.4.2.

3.4.2.1Designing the house parameters Windows Form

The house parameters WF will give the user a visual representation of the model parameters needed in the calculation of the inside temperature state. The house parameters are the U-values, areas and heater power. The house and sensors locations are seen in the house layout image box. A save button is used to save the new values to the XML file. The GUI of the house parameters can be seen in Figure 3-13.


House Parameters

Heat Equation

$$\dot Q = \dot Q_{supply} - \dot Q_{loss} = \dot Q_{supply} - UA\Delta T$$

House Layout

House Parameters

Wall area	66
Wall U value	0,18
Door Area	2
Door U -value	0,2
Window area	8
Window U -value	0,12
Floor area	72
Floor U -value	0,15
Roof area	72
Total House Area	218,6
Total house U value	0,17
Total house volume [m3]	172,7
Ventilation [airChanges/hour]	2

Qsupply - Heater Effect [W]

Heater kitchen	500
Heater Living Room	500
Heater bathroom	500
Heater hallway	500
Heater guest room	450
Total heat supply [kW]	2450

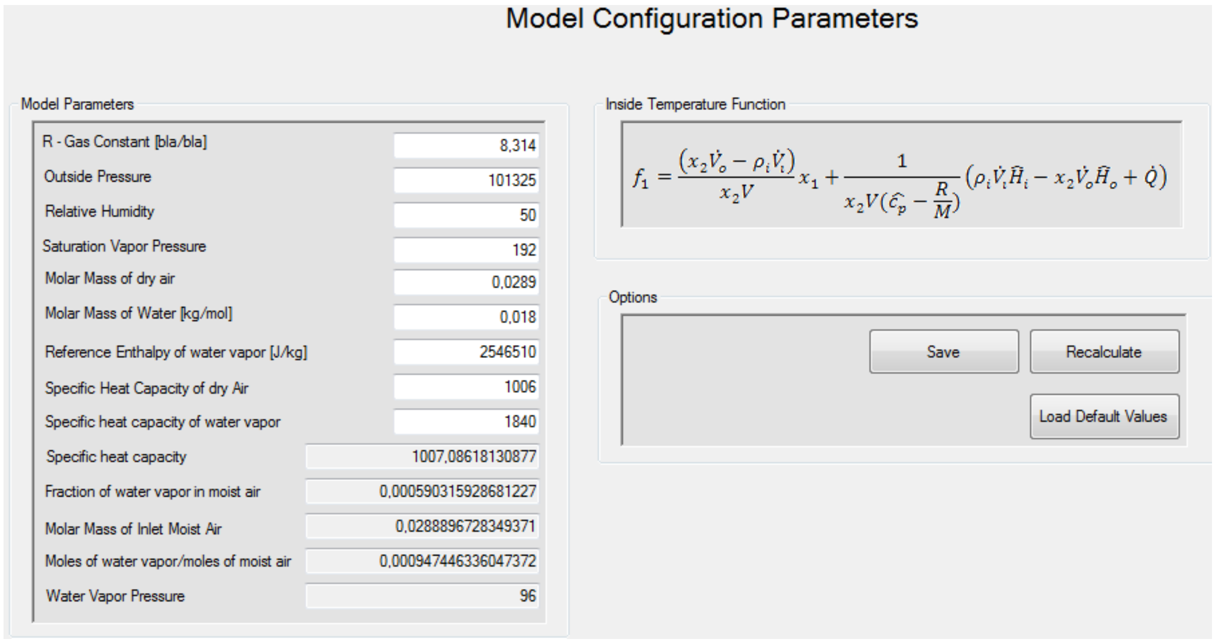
Options

Save

Figure 3-13 GUI of house configuration parameters

3.4.2.2Designing the Model parameters Windows Form

The model parameters file name is *ConfigModel.cs* and should give a visual representation of the models parameters. The default values should be set in the .cs file if the user wants to reload these. The recalculate button will recalculated the parameters based on the new values. In Figure 3-14 the created GUI can be seen.


Model Configuration Parameters

Model Parameters

R - Gas Constant [bla/bla]	8,314
Outside Pressure	101325
Relative Humidity	50
Saturation Vapor Pressure	192
Molar Mass of dry air	0,0289
Molar Mass of Water [kg/mol]	0,018
Reference Enthalpy of water vapor [J/kg]	2546510
Specific Heat Capacity of dry Air	1006
Specific heat capacity of water vapor	1840
Specific heat capacity	1007,08618130877
Fraction of water vapor in moist air	0,000590315928681227
Molar Mass of Inlet Moist Air	0,0288896728349371
Moles of water vapor/moles of moist air	0,000947446336047372
Water Vapor Pressure	96

Inside Temperature Function

$$f_1 = \frac{(x_2\dot V_o - \rho_i \dot V_i)}{x_2 V} x_1 + \frac{1}{x_2 V(\hat c_p - \frac{R}{M})}(\rho_i \dot V_i \hat H_i - x_2 \dot V_o \hat H_o + \dot Q)$$

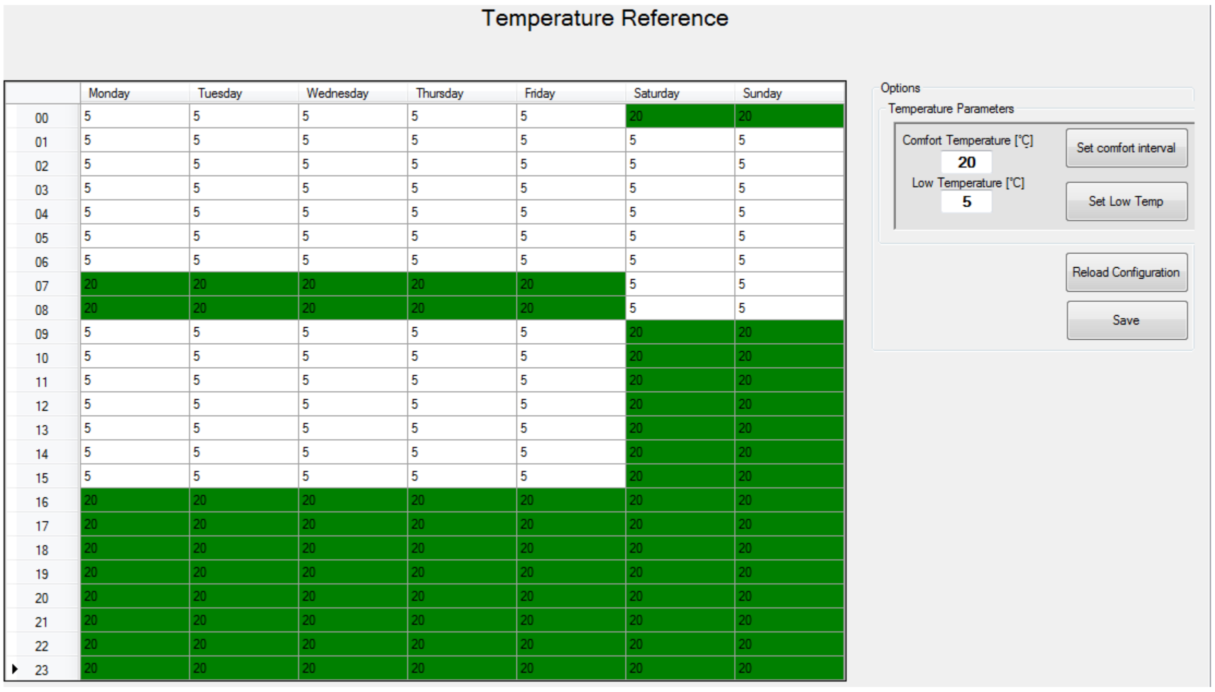
Options

Save | Recalculate | Load Default Values

Figure 3-14 GUI of model configuration parameters

3.4.2.3 Designing the comfort intervals use case

The comfort intervals use case is mainly a GUI for setting the comfort intervals schedule of the residents. The user should be able to set his preference for comfort and low temperature, and when these temperatures are wanted. The intervals at comfort temperature should be set in a green color for easy visualization. The comfort intervals should for simplicity only accept integers and should give the user a notice if the temperature is set lower than 5°C or higher than 25°C. The WF file name is *ConfigTemperatures.cs*. In Figure 3-15 the GUI created is seen with the typical invented workweek comfort intervals.



	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
00	5	5	5	5	5	20	20
01	5	5	5	5	5	5	5
02	5	5	5	5	5	5	5
03	5	5	5	5	5	5	5
04	5	5	5	5	5	5	5
05	5	5	5	5	5	5	5
06	5	5	5	5	5	5	5
07	20	20	20	20	20	5	5
08	20	20	20	20	20	5	5
09	5	5	5	5	5	20	20
10	5	5	5	5	5	20	20
11	5	5	5	5	5	20	20
12	5	5	5	5	5	20	20
13	5	5	5	5	5	20	20
14	5	5	5	5	5	20	20
15	5	5	5	5	5	20	20
16	20	20	20	20	20	20	20
17	20	20	20	20	20	20	20
18	20	20	20	20	20	20	20
19	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
21	20	20	20	20	20	20	20
22	20	20	20	20	20	20	20
23	20	20	20	20	20	20	20

Comfort Temperature [°C] 20
Low Temperature [°C] 5

Figure 3-15 Comfort intervals configuration GUI

The user may use the buttons to set comfort temperature or the low temperature or using the context menu strip, right mouse click option as seen in Figure 3-16.

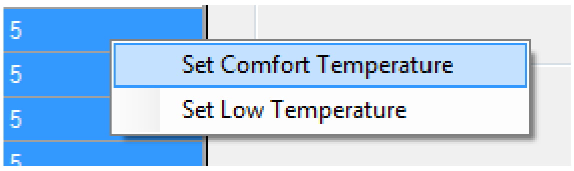


Figure 3-16 Context menu strip right click option

3.4.3 Read sensor values use case

The sensor values are stored to a log file containing the time stamp and the sensors values in a numbered order as seen in part 1 paragraph 2.3.4. The get sensor values use case should convert the ADC values to temperature and pass the values through a low pass filter. The complete fully dressed use case document can be found in Appendix 4: Fully dressed use case documents paragraph 7.4.2

3.4.3.1Designing the read sensor values use case

In C# there is no option for reading a specific line since this is not how the reader works on the lower levels. For this reason all lines will be read into an indexed array using the current index to remember which line to read. This might cause problems with very high sample times needed for fast systems (small time constants), however with the house system this is not a problem. The method should also return the date time stamp in order to plot the correct time values and have some information on the exact time of the specific sample.

The low pass filter is added as a new class filtering the values based on the previous value, the filter time constant and the sampling time. Each sensor is filtered in turn.

3.4.4 The predictor use case

The predictor use case takes care of the heating time predictions, the predictions are done in the manner previously specified in paragraph 3.3. The predictor will use the Kalman filter to calculate the disturbance vector, and a least squares regression to get the direction of this vector. The estimated disturbance function should be calculated one the first run time from a stable Kalman filter gain has been achieved to set point. In addition the purely linear regression estimate will be also be implemented in parallel as a second heating time estimate.

The complete fully dressed use case document can be found in Appendix 4: Fully dressed use case documents paragraph 7.4.2.

3.4.4.1Designing the predictor use case

The predictor use case utilizes a Kalman filter algorithm and need a Matrix calculation class to function. The C# and .NET libraries do for some reason not contain any inbuilt Matrix manipulation functions. A functional Matrix library using the *Strassen* algorithm for large matrix manipulations have been created by *Ivan Kucirc* [23]and will be used for the matrix manipulations in this thesis. Some additions were made to the library since it could not facilitate matrix and scalar values together: *Scalar values times matrix values, matrix values times scalar values, matrix divided on scalar values, and scalar values added to matrix values.* The ordinary least squares regression is calculated using the matrix equation found in [24].

3.5 Experiments

3.5.1 Introduction

The prediction model and the heating estimation program should be further tested to ensure the functionality in a real life experiment. Telemark University College has an air heater system that is based on the same rules as a complete house. The time constant will be much smaller than a real house due to the very small volume, area, and large U-value. There will also be a high degree of ventilation and a very large heater in relations to the size. These are all changeable parameters in the model and the air heater should therefore give a good real house simulation. The experiment is visualized in Figure 3-17.

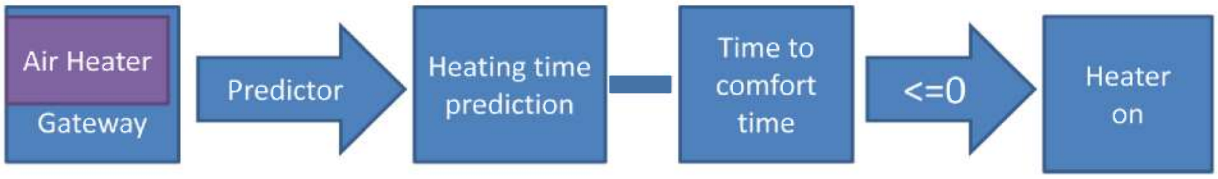


Figure 3-17 Heating time experiment

If the comfort intervals are reached at the correct time the heating time estimation will be working properly.

3.5.2 Experiment setup

The experiments should utilize the code already created in the gateway together with the new created code from the temperature prediction. The time constant of the air heater will be much smaller and the sampling time should be set to a much lower value. In addition two temperature sensors are use inside the air heater to get a good estimation of the inside temperature. All data will be saved to disk for plotting in MATLAB, and the comfort interval reference is changed from hour to second. The air heater has a length of 1m and a radius of 7.5cm, the heater power is set by the USB DAQ-6008 device [25]. For more specific information on the air heater see [13].

The setup for the heating time prediction using the gateway and the air heater can be seen in Figure 3-1.

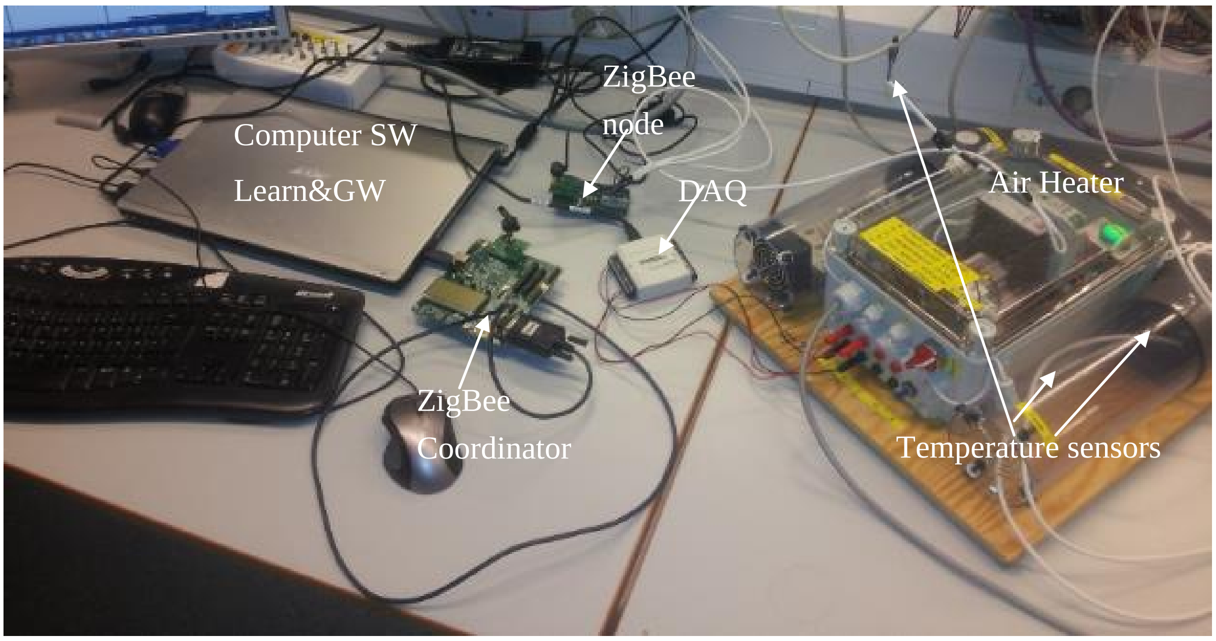


Figure 3-18 Heating time prediction setup

The uncertainty budget for this experiment can be seen in Table 3-1, where the temperature sensors are the main source of uncertainty. The temperature uncertainty was calculated in paragraph 2.4.2.

Table 3-1 Uncertainty budget

Type	Sensor	Range	Accuracy	Note
CC2530 EM	ADC	-3V to 3V	$\pm0.05\%$	
Temperature	TMP6	-25°C to 140°C	$\pm1.3\%$	
NI-DAQ 6008	ADC	0-5V	$\pm0.01\%$	

Prediction setup

The U value for plexiglas of 5mm thickness is about 14 [26], and the heater is 15W [13], the ventilation parameter is unknown and set as a tuning parameter, with the starting value of ten air changes each second. This parameter creates the continuous model SSM model of the air heater seen in (3-39).

$$A = -0.02274, B = 4\ D = 1 \tag{3-39}$$

The Comfort intervals were setup to be between 25 and 40 degrees Celsius since the inside temperature in the room of 20 is the outside temperature when seen in regards to the air heater. The predictor will work as an on off controller turning the heater off when the temperature is at the comfort level, and back on again when the temperature is below.

3.5.3 Results

When the system was connected the gateway was started at the same time as the learn function. The learn function is set to wait until the Kalman filter estimates has stabilized. The learn function saves the data together with both the comfort interval reference and the predicted reference from the heating time estimations. The saved data is then imported into MATLAB for analysis. For the straight forward OLS regression model the results were correct and the predicted reference was hit at exactly the same time as the comfort interval reference seen in Figure 3-19

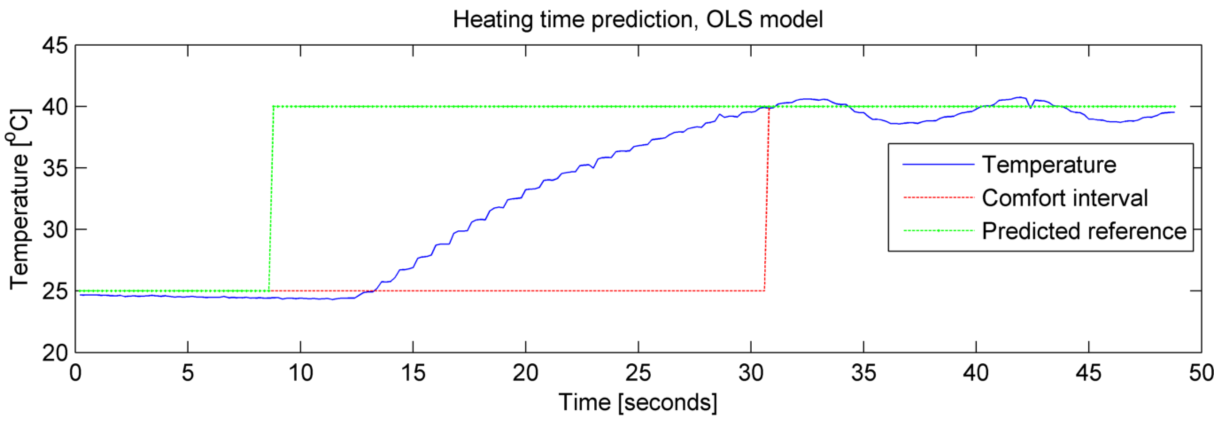


Figure 3-19 Heating time estimates using linear OLS model

For the K-OLS-SSM model the heating time estimate was too small and resulted in the comfort interval reference being reached to soon seen in Figure 3-20.

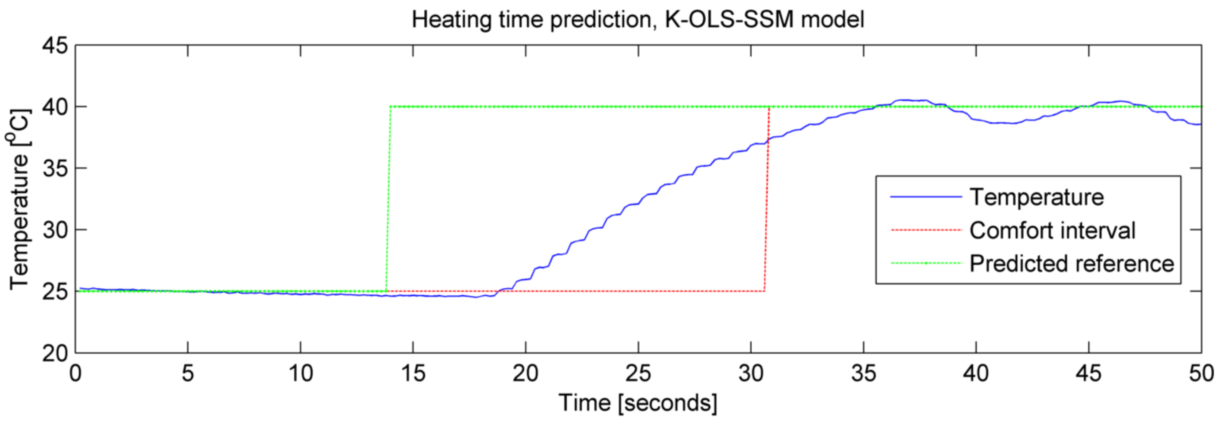


Figure 3-20 Heating time estimates with K-OLS-SSM prediction model

The K-OLS-SSM model is under predicting because of the time delay. This time delay lasts about two seconds and is not modeled in the SSM. The linear OLS model does however take this time delay into consideration when creating the regression line which resulted in a better prediction. This time delay factor can be calculated in the learn function from heating is started to the temperature is seen increasing and added to the K-OLS-SSM model creating a better and adaptable estimate. The prediction models can be found in Appendix 8: Regression models for predictor.

3.6 Discussion

The learn function was tested several times and the prediction models tested. Each time the best prediction was from the straight forward linear OLS regression model. The K-OLS-SSM heating time estimate was close but needed an added time delay factor in order to predict correctly. Testing the model with different outside temperatures was not tested, since the Air heater should be kept inside the school with a set comfort temperature. Testing with the same prediction models and different outside temperatures might prove the K-OLS-SSM model to be more adaptable to change as in the simulation part seen in paragraph 3.3.

From the real house experiments and the comfort intervals there can be created an estimate of the real power savings. The house heating time was found as roughly 1/3 of the cooling down time with the same environmental conditions. From the comfort intervals found in paragraph 3.2.1 the heaters can be off 64 hours each week or roughly 60% of the time. In 2/3's of these 64 hours the heater can be turned off, during the on time however the heater will use more power than it would during steady state. The experiments steady state power usage was 1.1kW seen in part 1 Data acquisition 2.4.5, and the heating is found to be 2.1kW from the experiment data. The power savings can then be calculated as seen in equations (3-40) and (3-41).

$$steady\ state\ consumption = 96hours * 1125W = 108kWh \tag{3-40}$$

$$power\ off\ and\ on\ consumption = 96 * \frac{2}{3} * 0 + 96 * \frac{1}{3} * 2100 = 67.2kWh \tag{3-41}$$

This means that approximately 22% of all the power used for heating can be saved utilizing a good model based control system in this specific house. This could be increased by adding more heaters and more power lowering the heating time.

The on-off controller seen in Figure 3-19 and Figure 3-20 is not very good when seen in regards to both efficiency and comfort. The oscillations indicate the need for a controller to keep the temperature at set point and avoid oscillations and over or under shooting. For this reason the next and last part of this thesis will be on designing and testing several controllers for the BAS system.

Part 3

4 Control System

4.1 Introduction

In part 2 [3.5] a straight forward and common on-off heater controllers are seen as inadequate for both resident comfort and power savings. A more advanced control algorithm is needed to keep the temperature at the comfort level and to avoid oscillations. This controller will use all the parts of the BAS system to function and is for this reasons included in a main program GUI binding the system together in a functional control system.

4.1.1 System description

The control system is responsible for reaching the comfort intervals with minimum overshoot, minimizing power usage. Predicted heating time will be added to the current reference gathered from the comfort intervals [3.2.2]. The control system will keep the inside temperature at comfort level, and be responsible for the complete GUI of the BAS.

Several control methods will be tested; *the feedback PID controller, the Linear Quadratic Regulator (LQR) and the Model Predictive controller (MPC).*

MPC and LQR rely on a good model for optimal control and for the experiments part a new model will be created. The new model will be created using the DSR subspace identification algorithm [27]. The control algorithms will be made applicable both for MIMO and SISO systems. The complete control system can be seen in Figure 4-1.

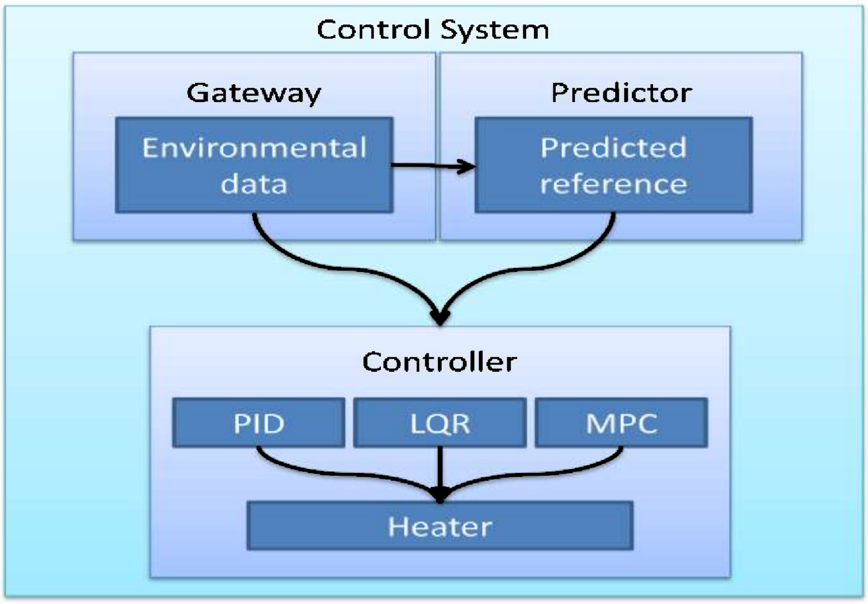


Figure 4-1 Control system description

4.1.2 Part 3 Structure

This part follow the same basis as the previous parts and in the object oriented programming sense each part will be finished before the next is started. The model and control theory will be explained before performing simulation using MATLAB. The different controllers will be discussed and compared. This gives a thorough analysis of the needed algorithms before implementation in VS. After implementation a complete system test will be performed using the air heater and the created BAS Control System. The progress of this part can be seen in Figure 4-2 where each main chapter is seen on the left and the sub chapters are seen on the right.

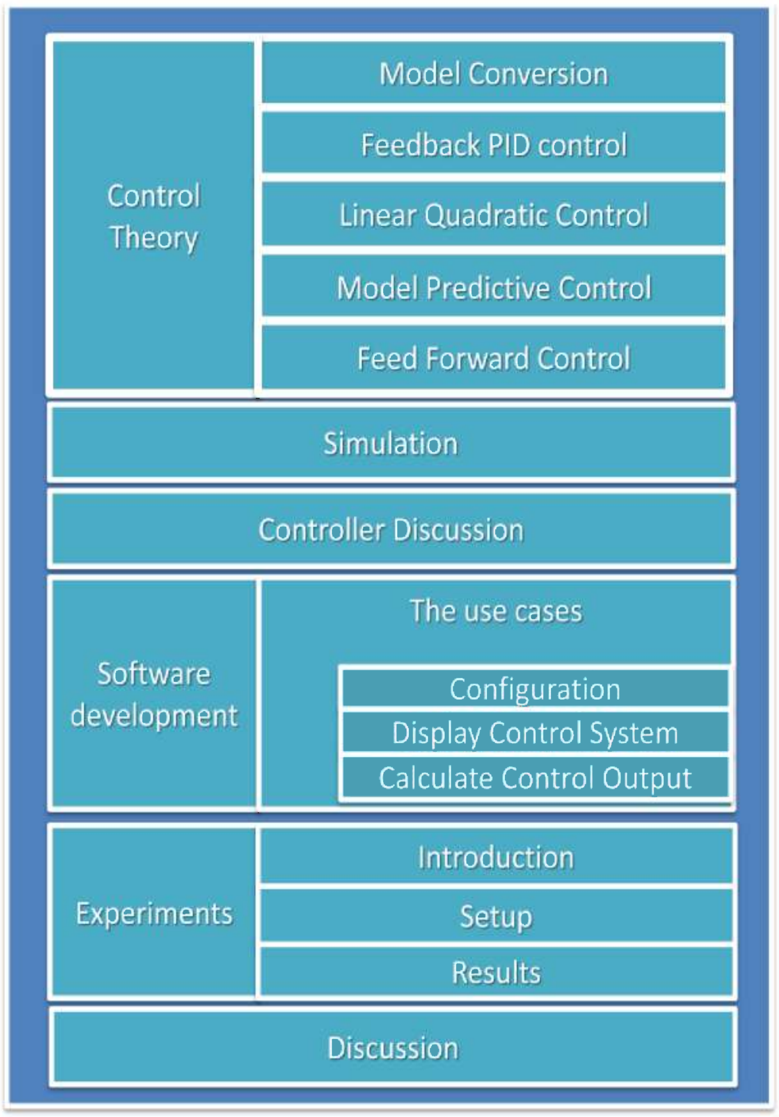


Figure 4-2 -Chapter progress all main chapters on the right and sub chapters on the left, progress will move in a downwards fashion.

4.2 Control Theory

4.2.1 Model conversion

For utilizing known control methods the model is converted into linear discrete state space form as was seen in 3.2.3 *Temperature prediction model*. The simulations will be based on a scalar one state system. The discrete state space model is seen in Equation (4-1) and Equation (4-2)

$$\dot{x} = Ax + Bu + v \tag{4-1}$$

$$y = Dx + w \tag{4-2}$$

Where x is the state vector, $x \in R^{n \times n}$, u is the control input vector, $u \in R^{n \times m}$, y is the output vector, $y \in R^{n \times m}$, and A, B and D are known system matrices of appropriate dimensions. The disturbances v and w are both unknown disturbances. The model is discretized in the same manner as in [3.2.3] resulting in the scalar state space system seen in Equations (4-3), (4-4) and (4-5)

$$A_d = 0.7263 \tag{4-3}$$

$$B_d = 0.0014 \tag{4-4}$$

$$D = 1 \tag{4-5}$$

Observability and Controllability

In order to make sure the model can be controlled a study is done on the controllability matrix. The observability of the system has already been tested found to be fully observable in [3.2.3]. The controllability matrix defines if the system can be controlled by the control input to the system. The controllability matrix is found in MATLAB using the *ctrb.m* function and the rank is found to be 1. This means the inside temperature state is controllable though B, which makes sense.

4.2.1.1The state-space model on deviation form

The process noise term v and the measurement noise w are assumed to be constant or slowly varying. For this reason the model can be reformed using velocity, deviation form removing the unknown and constant or slowly varying noise terms from the equation. The Linear Quadratic Regulator (LQR) with integral action also needs the state space matrix on velocity form and the PID controller is normally included in the velocity form [19].

The state equation is seen in its discrete form in Equation (4-6)

$$x_{k+1} = Ax_k + Bu_k + v \tag{4-6}$$

By using the last time instant, k=k-1 the state equation becomes as in Equation (4-7)

$$x_k = Ax_{k-1} + Bu_{k-1} + v \tag{4-7}$$

Inserting Equation (4-6) into (4-7) gives Equation (4-8) and (4-9)

$$x_{k+1} - x_k = Ax_k + Bu_k + \cancel{v} - Ax_{k-1} - Bu_{k-1} - \cancel{v} \tag{4-8}$$

$$\Delta x_{k+1} = A\Delta x_k + B\Delta u_k \tag{4-9}$$

Where delta x is denoted as the change from last sample time seen in Equation (4-10).

$$\Delta x_{k+1} = x_{k+1} - x_k, \text{and } \Delta u_k = u_k - u_{k-1} \tag{4-10}$$

The output equation on deviation form following the same principles becomes as seen in Equations (4-11) through (4-14).

$$y_k = Dx_k + w \tag{4-11}$$

$$y_{k-1} = Dx_{k-1} + w \tag{4-12}$$

$$y_k - y_{k-1} = Dx_k + \cancel{w} - Dx_{k-1} - \cancel{w} \tag{4-13}$$

$$y_k = y_{k-1} + D\Delta x_k \tag{4-14}$$

The model has now been transformed into a well-known strictly proper state space model and several control methods can be implemented and simulated. First the different control strategies theory will be explained, and designed starting with the most common the PID feedback control.

4.2.2 Feedback control

The first control method to be tested is the most common of all the control methods: the Feedback PID control. This controller is very robust and stable, and is not model dependent. The standard feedback control block diagram with BAS specific notations can be seen in Figure 4-3

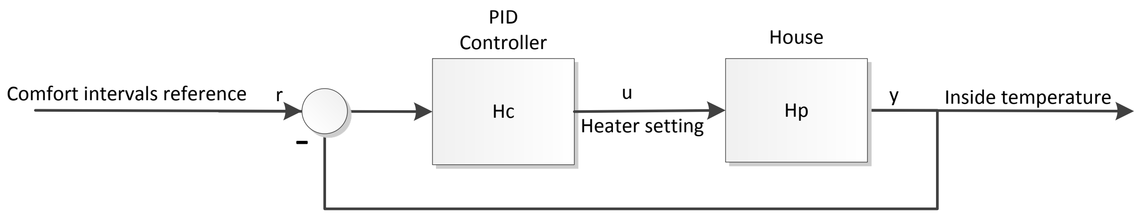


Figure 4-3 Standard feedback control block diagram

The discrete PID controller discretized using explicit Euler on velocity form can be seen in Equations (4-15) and (4-16).

$$u_k = u_{k-1} + g_o e_k + g_1 e_{k-1} + g_2(y_k - 2y_{k-1} + y_{k-2}) \tag{4-15}$$

$$g_o = K_p, \quad g_1 = -K_p * \left(1 - \frac{\Delta t}{T_i}\right), \quad g_2 = -\frac{K_p T_{d,}}{\Delta t} \ , \ e_k = r_k - y_k \tag{4-16}$$

Where u_k is the control signal, u_{k-1} the previous control signal $y_k$the output and y_{k-1}, y_{k-2} are the two previous outputs, Δt is the sampling time, K_p is the proportional gain T_i is the integral time and $T_{d,}$ is the derivation time.

The SIMC settings will be used to tune the controller [28], and MATLAB will be used to do test the controller in simulations before implementing in the control system. The SSM model should be transformed to its transfer function equivalent for tuning purposes.

The PID control parameters will be found in continuous time, we have the continuous state space model as seen in Equation (4-17) with the scalar numerical values from Equation (3-20) $A_c = -8.88e^{-4} \quad , B_c = 4.39e^{-6}$.

$$\dot{x} = Ax + Bu \tag{4-17}$$

The transfer function is found as in Equation (4-18)

$$h_p = k \frac{1}{1 + Ts} \tag{4-18}$$

Where $T = -\frac{1}{A}$ and $k = B * T$ which gives Equation (4-19)

$$h_p = 0.0049 * \frac{1}{1 + 1125s} \tag{4-19}$$

The SIMC tuning rules then gives the PID parameters seen in Equation (4-20) [28].

$$K_p = \frac{T}{k + T_c} = \ , T_i = T \ , T_d = 0 \tag{4-20}$$

Where T_c is the user specified time constant from the closed loop set point response.

4.2.3 Linear Quadratic Regulator

In a typical house there will be several rooms and each room will probably have one or more heater. This means the BAS system will be a Multiple Input Multiple Output (MIMO) system. The PID controller which only has output feedback does not have state feedback and several individual tuned PID controllers would have to be used in a buildings control system. This might prove both advanced to tune correctly and inherently unstable.

The Linear Quadratic Regulator (LQR) controller has both output and state feedback and can guarantee nominal stability of a MIMO system [29]. The LQ controller is suitable for use on non-linear systems when a linear state space model is available [29]. The LQR is model dependent, but can be compared with the feedback PI controller on velocity form. Di Rusccio [29] proposes a method to obtain integral action on the LQ controller. The state space model is augmented with the output equation, this means the output is included as a state in the model. The augmented SSM model on deviation form is seen in Equation (4-21) and Equation (4-22).

$$\left[\underbrace{\begin{matrix}\Delta x_{k+1} \\ y_k\end{matrix}}_{\tilde{x}_{k+1}}\right] = \left[\underbrace{\begin{matrix}A & 0_{n*m} \\ D & I_{m*m}\end{matrix}}_{\tilde{A}}\right]\begin{bmatrix}\Delta x_k \\ y_{k-1}\end{bmatrix} + \left[\underbrace{\begin{matrix}B \\ 0_{m*r}\end{matrix}}_{\tilde{B}}\right]\Delta u_k \tag{4-21}$$

$$y_k = \underbrace{[D \quad I_{m*m}]}_{\tilde{D}}\begin{bmatrix}\Delta x_k \\ y_{k-1}\end{bmatrix} \tag{4-22}$$

Introducing the augmented model matrices the model is on strictly proper state space form as seen in Equation (4-23) and Equation (4-24).

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\Delta u_k \tag{4-23}$$

$$y_k = \tilde{D}\tilde{x}_k \tag{4-24}$$

The LQ regulator needs a cost function, or a cost objective to be minimized, the cost objective is denoted J_i. This cost function is the squared sum of all future output deviations times the cost factor $\tilde{Q}$ plus the squared sum of all future control outputs times the cost factor P. The cost objective in its matrix form using the augmented SSM can be seen in Equation (4-25).

$$J_i = \frac{1}{2}\sum_{k=i}^{\infty}\left[\tilde{x}^T\tilde{Q}\tilde{x} + \Delta u_k^T P \Delta u_k\right] \tag{4-25}$$

The matrix $\tilde{Q}$is seen as a weighting matrix weighting the cost of the deviation from the reference set point. In this BAS system the cost of deviation from the set point is seen as high and the Q matrix should be given a high value. The $\tilde{Q}$ matrix is seen in both weighting the output and the states.

$$\tilde{Q} = \begin{bmatrix} Q_k & 0 \\ 0 & Q_z \end{bmatrix} \tag{4-26}$$

The cost factor R denotes the cost of the control outputs and in the BAS system where power savings is the important factor this factor should also be high, however a high cost of both the control outputs and reference deviations are not obtainable at the same time, and since the reference deviations are seen as the primary concern at the comfort intervals the cost factor Q should be weighted the highest.
Together the performance index seen in Equation (4-25) and the state-space model in Equations (4-23) and (4-24) defines a standard LQR optimal control problem. A solution to this optimal control problem, minimizing the cost objective, will exist if P > 0, the pair $(\tilde{A};\ \tilde{B})$ is stabilizable and that the pair $(C; \tilde{A})$ is detectable where C is the square root matrix of $\tilde{Q}$ such that $\tilde{Q} = C^T C$.
Minimizing the performance index in Equation (4-25) with respect to the control deviation is given by the state feedback seen in Equations (4-26) and (4-27) [29].

$$\Delta u_k^* = G\tilde{x}_k \tag{4-27}$$

Where the feedback matrix G is obtained as seen in Equation (4-28), and R is the solution to the discrete time algebraic Riccati Equation (4-29) [29].

$$G = -(P + \tilde{B}^T R\tilde{B}\big(P + \tilde{B}^T R\tilde{B}\big)^{-1}\tilde{B}^T R\tilde{A} \tag{4-28}$$

$$R = \tilde{Q} + G^T PG + \big(\tilde{A} + \tilde{B}G\big)^T R(\tilde{A} + \tilde{B}G) \tag{4-29}$$

The solution to the discrete time algebraic Riccati equation can be solved simply with the MATLAB *dlqr.m* function, however since there is the need for implementation in Visual Studio (VS) there has been created a recursive solver. The recursive solver needs only be run at startup and will run until the steady state solution has been found, when the error between the new value and the previous value is below the set error limit. This is seen in MATLAB script 4-1.

```
      while (error>1e-10 & it<<=maxit);
        G0=    (P+B'*R0*B)\(B'*R0*A);
        R1=     A'*R0*A + Q -(A'*R0*B)*G0;
        G1=    (P+B'*R1*B)\(B'*R1*A)

        error=max(max(abs(G1-G0)));
        it=it+1
      R0=R1;
       end;

         G=-G1;
         G1=G(:,1:size(A));
         G2=G(:,size(A)+1:size(A)+size(D,1));
```

MATLAB script 4-1 Recursive Riccati solver

The output from the MATLAB function the constant feedback gain matrix G can be viewed as seen in Equations (4-30) and (4-31).

$$\Delta u_k = [G_1 \quad G_2]\begin{bmatrix} \Delta x_k \\ \underbrace{y_k - r_k}_{e_k} \end{bmatrix} = u_k - u_{k-1}, \qquad u_k = \Delta u_k + u_{k-1} \tag{4-30}$$

$$u_k = u_{k-1} + G_1 \Delta x_k + G_2 * e_k \tag{4-31}$$

The created LQR regulator can be seen in block diagram of the optimal LQ controller is seen with the optimal LQR gain parameters G1 and G2 in Figure 4-4.

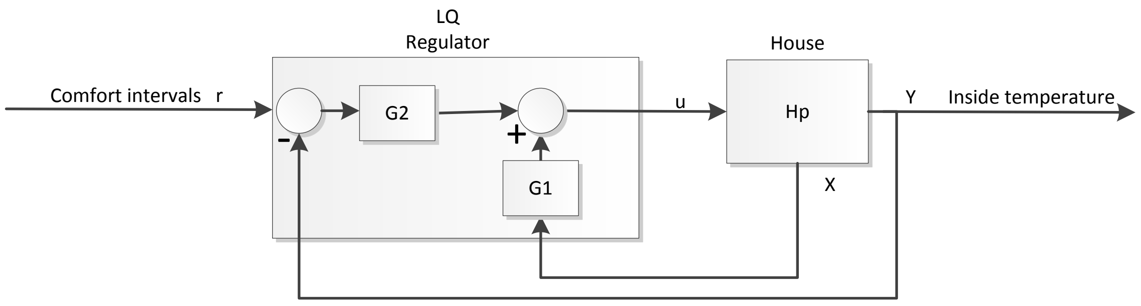


Figure 4-4 Linear quadratic regulator block diagram

This means the controller structure is equivalent with the velocity form of the PI controller seen in Equations (4-32) and (4-33) [29].

$$u_k = u_{k-1} + g_o e_k + g_1 e_{k-1} \tag{4-32}$$

$$g_o = K_p, \quad g_1 = -K_p * \left(1 - \frac{\Delta t}{T_i}\right), \tag{4-33}$$

4.2.4 Model predictive control

Model based Predict Control (MPC) is a very important control method and can be found implemented in MATLAB, LabVIEW, CENIT and Honeywell’s Profit to mention some [30]. The MPC works with both SISO and MIMO systems making it useful for a complete house control and there exists also non-linear methods for MPC. MPC works by calculating the optimal control output based on a specified reference vector into the future [31]. This length of this reference trajectory into the future is called the prediction horizon, L. The optimal control output is based on minimizing a cost function based on a prediction model, similar to the Linear Quadratic Regulator (LQR). The main differences are that LQR uses an infinite prediction horizon, and that the MPC controller has inbuilt constraints handling [31]. In order to minimize the cost function based on the given prediction model and constraints there is needed an optimizer. The MPC controller can be seen in Figure 4-5.

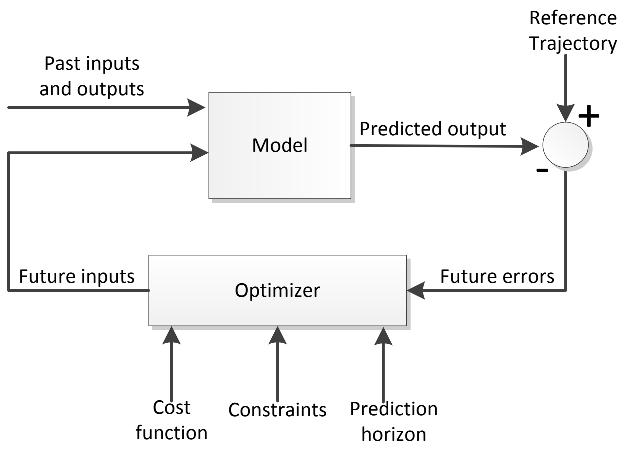


Figure 4-5 MPC block diagram

The Quadratic Programming problem with constraints can be solved by *quadprog.m* in MATLAB or by and an active set method [31].The solution to the QP problem in VS can be done using the Microsoft Solver Foundation [32] however the computing time to find a solution might be very high especially in a large house MIMO system. Another solution is to handle the constraints using if and else statements at a higher level. This can be done in this BAS system since the only constraints are the maximum and minimum heater capacity saving computing time and greatly simplify the optimizer. The only drawback of MPC is then the model dependency; *a good model is needed to sustain good control solutions*.

The MPC controller needs a prediction model to be minimized subject to a cost function. This prediction model can be found from the augmented SSM seen in Equations (4-21) and (4-22) in order to give the controller integral action [33].

4.2.4.1Finding a prediction model

The prediction model may be obtained from the augmented state space model (eSSM) on deviation form. The present time is predicted into the future with the prediction horizon L. To find the prediction model the prediction horizon L is set to be three. Using this prediction horizon the prediction model is calculated as seen from Equations (4-34) through (4-37).

$$y_{k+1} = \tilde{D}\tilde{x}_{k+1} = \tilde{D}\left(\tilde{A}\tilde{x}_k + \tilde{B}\tilde{u}_k\right) = \tilde{D}\tilde{A}\tilde{x}_k + \tilde{D}\tilde{B}\Delta u_k \qquad (4\text{-}34)$$

$$y_{k+2} = \tilde{D}\tilde{x}_{k+2} = \tilde{D}\left(\tilde{A}\tilde{x}_{k+1} + \tilde{B}\Delta u_{k+1}\right) = \tilde{D}\tilde{A}^2\tilde{x}_k + \tilde{D}\tilde{A}\tilde{B}\Delta u_k + \tilde{D}\tilde{B}\Delta u_{k+1} \qquad (4\text{-}35)$$

$$\begin{aligned} y_{k+L} &= \tilde{D}\tilde{x}_{k+L} = \tilde{D}\tilde{A}^2\tilde{x}_{k+1} + \tilde{D}\tilde{A}\tilde{B}\Delta u_{k+1} + \tilde{D}\tilde{B}\Delta u_{k+1} \\ &= \tilde{D}\tilde{A}^3\tilde{x}_k + \tilde{D}\tilde{A}^2\tilde{B}u_k + \tilde{D}\tilde{A}\tilde{B}\Delta u_{k+1} + \tilde{D}\tilde{B}\Delta u_{k+2} \end{aligned} \qquad (4\text{-}36)$$

$$\begin{bmatrix} y_{k+1} \\ y_{k+2} \\ y_{k+L} \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{D} \\ \tilde{D}\tilde{A} \\ \tilde{D}\tilde{A}^2 \end{bmatrix}}_{O_3} \tilde{A}\tilde{x}_k + \begin{bmatrix} \underbrace{\begin{matrix} \tilde{D}\tilde{B} \\ \tilde{D}\tilde{A}\tilde{B} \\ \tilde{D}\tilde{A}^2\tilde{B} \end{matrix}}_{O_3\tilde{B}} & \underbrace{\begin{matrix} 0 & 0 \\ \tilde{D}\tilde{B} & 0 \\ \tilde{D}\tilde{A}\tilde{B} & \tilde{D}\tilde{B} \end{matrix}}_{H_3^d} \end{bmatrix} \begin{bmatrix} u_k \\ u_{k+1} \\ u_{k+2} \\ u_{k+3} \end{bmatrix} \qquad (4\text{-}37)$$

Where O_3 is the extended observability matrix$\tilde{A}$, $\tilde{B}$ and $\tilde{D}$ are the extended state space models and H_3^d is the lower block triangular Toeplitz matrix for the triple $(\tilde{D}, \tilde{A}, \tilde{B})$ Error! Bookmark not defined.. The prediction model with a prediction horizon of three is seen in Equation (4-38).

$$y_{k+1|3} = O_3\tilde{A}\tilde{x}_k + [O_3\tilde{B} \quad H_3^d] * \Delta u_{k|3} \qquad (4\text{-}38)$$

The prediction model can then be defined in the standard form as seen in Equation (4-39)

$$y_{k+1|L} = P_L + F_L\Delta u_{k|L} \qquad (4\text{-}39)$$

Where the prediction model parameters are as seen in Equation (4-40) and the extended observability matrix O_L and the deterministic Toeplitz matrix H_L^d can be seen in (4-41).

$$P_L = O_L\tilde{A}\tilde{x}_k \ , \qquad F_L = [O_L\tilde{B} \quad H_L^d] \qquad (4\text{-}40)$$

$$O_L = \begin{bmatrix} \tilde{D} \\ \tilde{D}\tilde{A} \\ \vdots \\ \tilde{D}\tilde{A}^{L-1} \end{bmatrix} , and\ H_L^d = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \tilde{D}\tilde{A}^{L-1}\tilde{B} & \cdots & \tilde{D}\tilde{B} \end{bmatrix} \qquad (4\text{-}41)$$

The prediction model has been specified and the control objective needs to be set.

4.2.4.2MPC control objective

In the MPC controller there is also needed a control objective to be minimized with respect to the found prediction model. A typical control objective guaranteeing stability may be seen in Equation (4-42) [31]

$$J_k = \frac{1}{2}\sum_{i=1}^{L}[(y_{k+i} - r_{k+i})^T Q_i (y_{k+i} - r_{k+i}) + \Delta u_{k+i-1}^T R_i \Delta u_{k+i-1}] \tag{4-42}$$

The control Objective seen in Equation (4-42) may conveniently be written in more compact form as seen in Equation (4-43), where $y_{k+1|L}$, $\Delta u_{k|L}$, and $r_{k+1|L}$ are seen in Equation (4-44). Q and R are the triangular weighting matrices seen in Equation (4-45).

$$j_k = \left(y_{k+1|L} - r_{k+1|L}\right)^T Q \left(y_{k+1|L} - r_{k+1|L}\right) + \Delta u_{k|L}^T R \Delta u_{k|L} \tag{4-43}$$

$$y_{k+1|L} = \begin{bmatrix} y_{k+1} \\ y_{k+2} \\ \vdots \\ y_{k+L} \end{bmatrix} , \quad \Delta u_{k|L} = \begin{bmatrix} \Delta u_k \\ \Delta u_{k+1} \\ \vdots \\ \Delta u_{k+L-1} \end{bmatrix} , \quad r_{k+1|L} = \begin{bmatrix} r_{k+1} \\ r_{k+2} \\ \vdots \\ r_{k+L-1} \end{bmatrix} \tag{4-44}$$

$$Q = \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_L \end{bmatrix} , \quad ,R = \begin{bmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & R_L \end{bmatrix} \tag{4-45}$$

The prediction model in Equation (4-39) is inserted into the control objective in Equation (4-43) resulting in a cost objective to be minimized seen in Equation (4-46).

$$j_k = \left(P_L + F_L \Delta u_{k|L} - r_{k+1|L}\right)^T Q \left(P_L + F_L \Delta u_{k|L} - r_{k+1|L}\right) + \Delta u_{k|L}^T R \Delta u_{k|L} \tag{4-46}$$

Some rearranging gives the objective in standard from seen in Equation (4-47), where the quadratic term Hessian matrix , H, is as seen in Equation (4-48), the linear term , f_k^T, is seen in Equation (4-49), and the scalar term, J_0, can be seen in Equation (4-50).

$$J_k = \Delta u_{k|L}{}^T H \Delta u_{k|L} + 2f^T \Delta u_{k|L} + J_0 \tag{4-47}$$

$$H = P_L + F_L^T Q F_L \tag{4-48}$$

$$f_k^T = F_L^T Q \left(P_L - r_{k+1|L}\right) \tag{4-49}$$

$$J_0 = \left(P_L - r_{k+1|L}\right)^T Q \left(P_L - r_{k+1|L}\right) \quad (8) \tag{4-50}$$

This control problem is a linear quadratic problem and there exists only one solution. The optimal unconstrained MPC control u^*_k is found where J_k is minimized, where the derivative is equal to zero $\frac{dJ}{d\Delta u_{k|L}} = 0$. This calculation can be seen in Equations (4-51) and (4-52).

$$\Delta u^*_{k|L} = \frac{dJ}{du_k} = 2H\Delta u_{k|L} + 2f_k^T + 0 = 0 \tag{4-51}$$

$$\Delta u^*_{k|L} = -H^{-1}f_k = -H^{-1}F_L Q(P_L - r_{k+1|L}) \tag{4-52}$$

The optimal control deviation can then be calculated as seen in Equation (4-53), where the gain matrix $G = -H^{-1}F_L Q$ may be calculated in advance.

$$\Delta u^*_{k|L} = G(P_L - r_{k+1|L}) \tag{4-53}$$

4.2.5 Feed forward control

In this BAS system a rapidly decrease in outside temperature is quickly measured by the outside temperature sensors, however the feedback controllers will not react until the temperature decrease/increase has affected the inside temperature. A feed forward (FF) controller will keep the influence of the outside temperature to a minimum by modeling the effect of the disturbance on the system. The FF control has previously been tested successfully in house applications [7]. The controller needs to know when to act based on the systems time delay, and how much to act based on the disturbance model. In this case the time delay of the system is not known, but a FF controller can be designed from the current nonlinear model. The FF block diagram together with a common feedback controller can be seen in *Figure 4-6*.

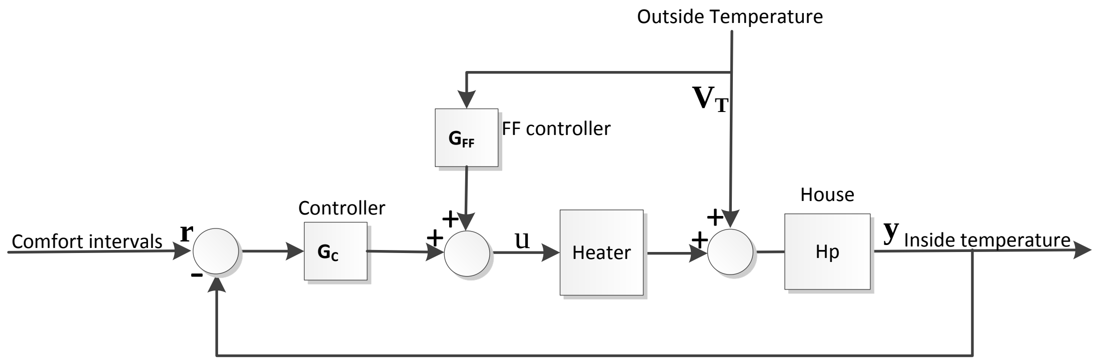


Figure 4-6 Feed forward controller block diagram

In Figure 4-6 the FF controller gain G_{FF} is seen added to the controller gain from the PID, LQR or MPC, G_c .The FF gain is based on the modeled disturbance outside temperature V_T.

The FF controller needs to be designed based on the model, this means the model must be solved for the gain u based on the outside temperature. The solution is calculated assuming a constant reference $\dot{x} = 0$ seen in Equations (4-54) through (4-58).

$$0 = \frac{\left(x_2\dot{V}_o - \rho_i\dot{V}_i\right)}{x_2V}x_1 + \frac{1}{x_2V(\widehat{c_p} - \frac{R}{M})}\left(\rho_i\dot{V}_i\widehat{H}_i - x_2\dot{V}_o\widehat{H}_o + \dot{Q}\right) \tag{4-54}$$

$$-\frac{\dot{Q}}{x_2V(\widehat{c_p} - \frac{R}{M})} = \frac{\left(x_2\dot{V}_o - \rho_i\dot{V}_i\right)}{x_2V}x_1 + \frac{1}{x_2V(\widehat{c_p} - \frac{R}{M})}\left(\rho_i\dot{V}_i\widehat{H}_i - x_2\dot{V}_o\widehat{H}_o\right) \tag{4-55}$$

$$\dot{Q} = -\left(x_2\dot{V}_o - \rho_i\dot{V}_i\right)\left(\widehat{c_p} - \frac{R}{M}\right)x_1 - \rho_i\dot{V}_i\widehat{H}_i + x_2\dot{V}_o\widehat{H}_o \tag{4-56}$$

From the heat Equation (7-1) using steady state conditions the FF gain will be as seen in Equation (4-57)

$$\dot{Q} = \dot{Q}_{supply} - \dot{Q}_{loss}, \quad \dot{Q}_{supply} = u_{ff} \tag{4-57}$$

Inserting Equation (4-57) into Equation (4-56) gives the control solution seen in Equation (4-58).

$$u_{ff} = -\left(x_2\dot{V}_o - \rho_i\dot{V}_i\right)\left(\widehat{c_p} - \frac{R}{M}\right)x_1 - \rho_i\dot{V}_i\widehat{H}_i + x_2\dot{V}_o\widehat{H}_o + \dot{Q}_{loss} \tag{4-58}$$

Since the model was found inadequate in the previous sections this should only be a basis on how to create a feed forward controller when the important mass factor is added. Some experiments on the time delay in real house systems should also be performed to time the FF controller correctly.

4.3 Control Simulations

Before implementing the more advanced controllers in VS a good idea is to have them function in the MATLAB environment. The controller calculations are done based on the integrator model in Equations (4-3), (4-4) and (4-5) before the outputs are fed to the non-linear model. The specific tuning parameters are seen in Table 4-1.

Table 4-1 Controller parameters

PID controller	$K_p = 375$	$T_i = 1126$	$T_d = 0$
LQR controller	$q = 1$	$r = 0.01$	N/A
MPC controller	$L = 10$	$q = 10 * eye(L)$	$r = 0.01 * eye(L)$

Where the user specified time constant is three, and $eye(L)$, denotes an identity matrix with the size of the prediction horizon L.

The controller simulations can be seen in Figure 4-7.

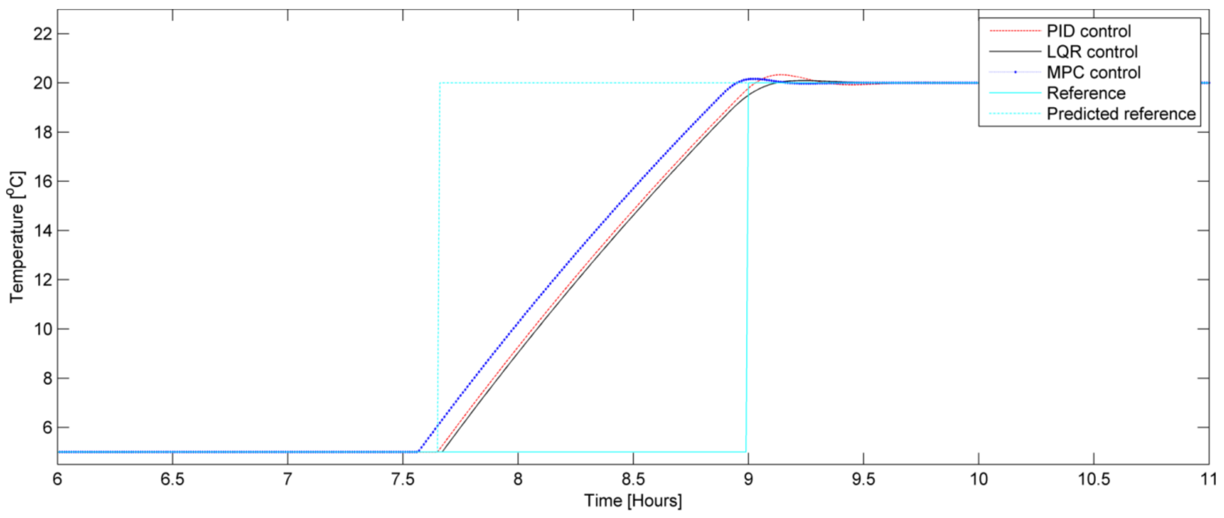


Figure 4-7 Controller simulations

In Figure 4-7 all controllers and the predicted reference are working properly, the MPC controller is starting somewhat earlier due to the prediction horizon. Since the predicted reference will take care of the predictions into the future the MPC prediction horizon should be set to the smallest stable value on implementation. It is however also seen that the LQR especially reaches the comfort intervals a little too late due to the need to minimize the overshoot. This is handled by increasing the predicted reference with a percentage that should be set based on experimentally gained knowledge, 10% percent is adequate in this case.

The advantage over the LQ regulator over the MPC controller is that because of the infinite prediction horizon the Controller feedback gain matrix G needs only be computed once, and not at each sampling interval. This makes the optimal controller much faster than the MPC controller. The only disadvantage seen when in regards to the control problem is that no set basis to handle process constraints. Due to the simplicity of the constraints in the BAS system this does not propose any problems. The controller simulation function is found in the Appendix 6: MATLAB scripts paragraph 7.6.6

4.4 Controller discussion

All the control methods have been analyzed, designed and simulated and simulated based on a SSM model and the non-linear first principles house model seen in Part 1 paragraph 2.2.1. The finished BAS system will be MIMO systems which should remove the common feedback PI control as any option. MPC and LQR controller methods are model dependent but the basis of the thesis is to evaluate a model based BAS system and as such a working model is expected on implementation.

The MPC controller is much more advanced than the LQR controller and the main reason to choose MPC over LQR would be the straight forward way of handling constraints. These constraints can however be handled simply using if-else statements. An overview of the control methods tested can be seen in Table 4-2.

Table 4-2 Control methods properties overview, (X) included, (-) not included

Properties	Feedback PID control	Linear Quadratic Regulator	Model Predictive Control
SISO	X	X	X
MIMO	-	X	X
Optimal control	-	X	X
Output feedback	X	X	X
State feedback	-	X	X
Constraints Handling	-	-	X
Model dependent	-	X	X
Complexity	Low	Medium	High
Computing time	Low	Low	High

An important note not is that the only control not model dependent and thus not being limited by the correctness of the model is the feedback control.

Feed forward control has been found to be a viable option in order to reduce the influence of the outside temperature [7] . The feed forward controller can in theory be added to all the controllers mentioned above. Care should be taken when introducing the feed forward controller to a MIMO system.

4.5 Heater control

For the BAS control system to be a viable commercial application the control system must work with existing heaters. In most cases these heaters will probably be simple on-off thermostat controlled with some power settings. The control output delivered from the controllers is however a scalar value between zero and the maximum available heater power. The simplest solution would be to use a socket on-off switch and turning all the heaters to the maximum settings and letting the on-off controller handle when the heaters should be on. This is not the best solution as was evident in part 2 paragraph 3.5.3. A better result is achieved by using a binary on/off Pulse Width Modulation (PWM) signal. This makes the heaters applicable for all variations in output power settings. In Figure 4-8 the PWM is visualized with the connection to the heaters.

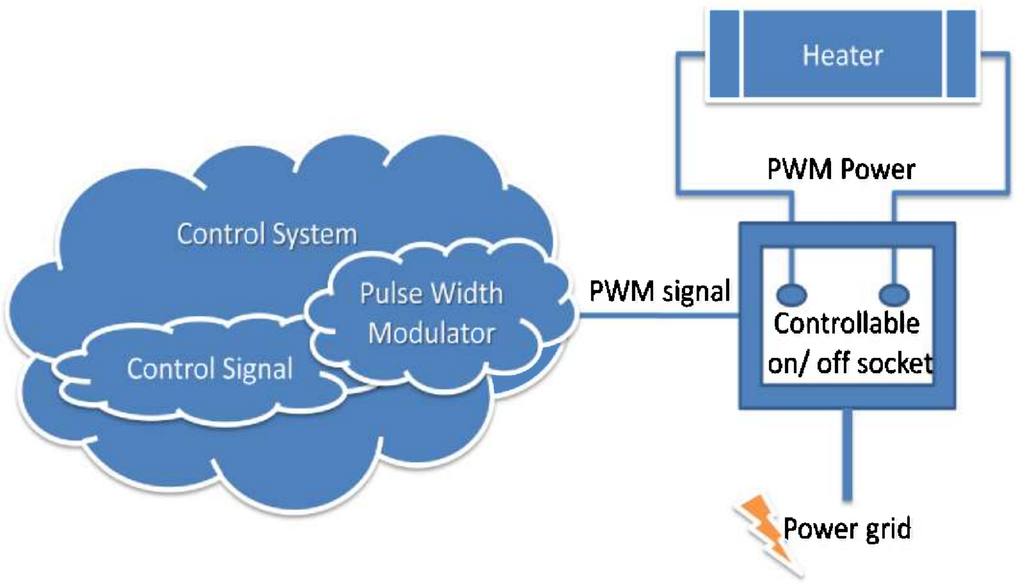


Figure 4-8 PWM control layout

The Pulse Width Modulation of a signal usually refers to rapid pulsing of a digital signal in order to simulate varying voltage. In this case the PWM is used a little differently, as each sampling time the PWM is used to have the correct power output from the heaters. The sampling time interval, (six minutes), is divided into a carrier signal frequency of 10 shifts per interval; this means 36 seconds' intervals or 0.0278Hz.

The saw tooth waveform is used to set the correct on-time based on the duty cycle. For a 1000W heater a 20% duty cycle means that the heater will be 1000W 20% of the time and 0W 80% of the time making it average 200W over the sampling period. An 80% duty cycle means that the heater will be 1000W 80% of the time and 0W 20% of the time averaging 800W over the sampling period. The 200W and 800W examples can be seen in Figure 4-9, on the left and right respectively.

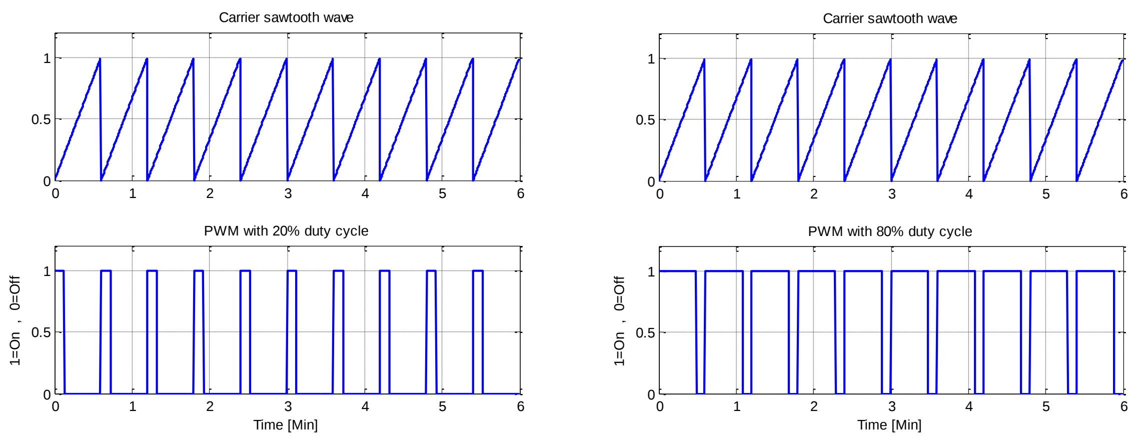


Figure 4-9 PWM with 20% duty cycle on the left and 80% duty cycle on the right

In addition to the PWM software there needs to be a hardware on-off device connected to the control system between the heater and the power grid. This controllable socket device should be made to handle relatively high frequency on/off signals, however if the heaters use a too high PWM frequency this will create a problem on the power grid. In a large house the BAS system will control a lot of heaters, utilizing much power. If these loads are turned on and off at very frequent intervals they will chop up the 50Hz Sinus. This creates a non-linear load and Harmonics on the power grid which might create severe problems on the power grid. In order to resolve this issue the heaters will not be turned on and off at any speed near to the 50Hz net frequency. The selected low frequency carrier signal of 0.0287Hz will be adequate to ensure no harmonics occur. These low frequency shifts should not be a problem due to the large time constants of the model. Some additional control should be included to avoid very small duty cycles. This can be done by dividing the PWM duty cycles into minimum regions, i.e. 5%.

The MATLAB function creating the PWM signal based on the maximum heater power and the output from the controller can be found in the Appendix 6: MATLAB scripts paragraph 7.6.7.

4.6 Software development

The control system should be the main GUI of the BAS monitoring and control system using the predictor and the control methods explained in paragraph 4.2, and the sensor data saved to file from the gateway system. The control algorithms have several tuning settings that should be changeable in a configuration, together with the controller type. The SSM model can be calculated from the model and house configurations or entered directly in the configuration. The MPC and LQR control algorithms should applicable for MIMO systems, however the main control system will use the SISO implementation from paragraph 4.1.2.

The heater output will be through a DAQ 6008 device and the experiments will be run on the air heater using the built in PWM [13]for this reason the PWM will not be included in VS.

All configurations from the control system and the predictor seen in paragraphs 4.6.2 and 3.4.2 are added as tabbed forms to a main configuration form creating a straight forward GUI, and saved in XML using the XML class from part 1 paragraph 2.3.2.1 appended with new parameters.

The sensor values will be plotted based on name and location into one outside temperature graph and one inside temperature graph. The outside sensors will be chosen on the basis that the location should include outside. The analysis sheet can be found in Appendix 3 FURPS+ paragraph 7.3.3.

4.6.1 The use case diagram

The functionality of the control system is made into a use case diagram to give a good visual representation. In Figure 4-10 the control systems use cases are seen with the gateway configuration seen in paragraph 2.3.2, predictor 3.4 and log files 2.3.4 as inputs to the system. The main outputs to the system are the heaters and the display.

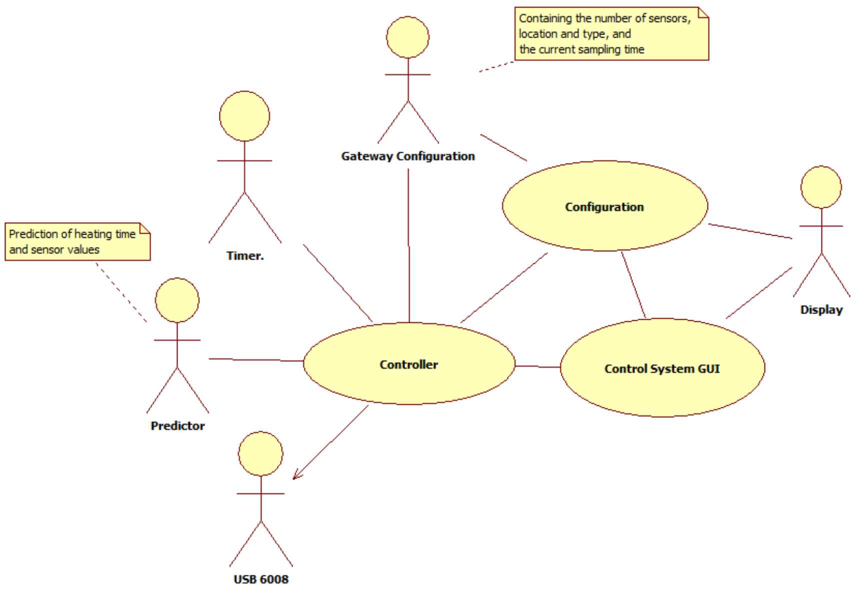

Containing the number of sensors, location and type, and the current sampling time
Gateway Configuration
Timer.
Configuration
Display
Prediction of heating time and sensor values
Predictor
Controller
Control System GUI
USB 6008

Figure 4-10 Control system use case diagram

The control system layered architecture is seen in Figure 4-11, where all main classes in the system are visualized and connected to the physical layers. The operating system for the sensor values and configurations. The arrows show the information flow between the predictor the sensor values and the controllers.

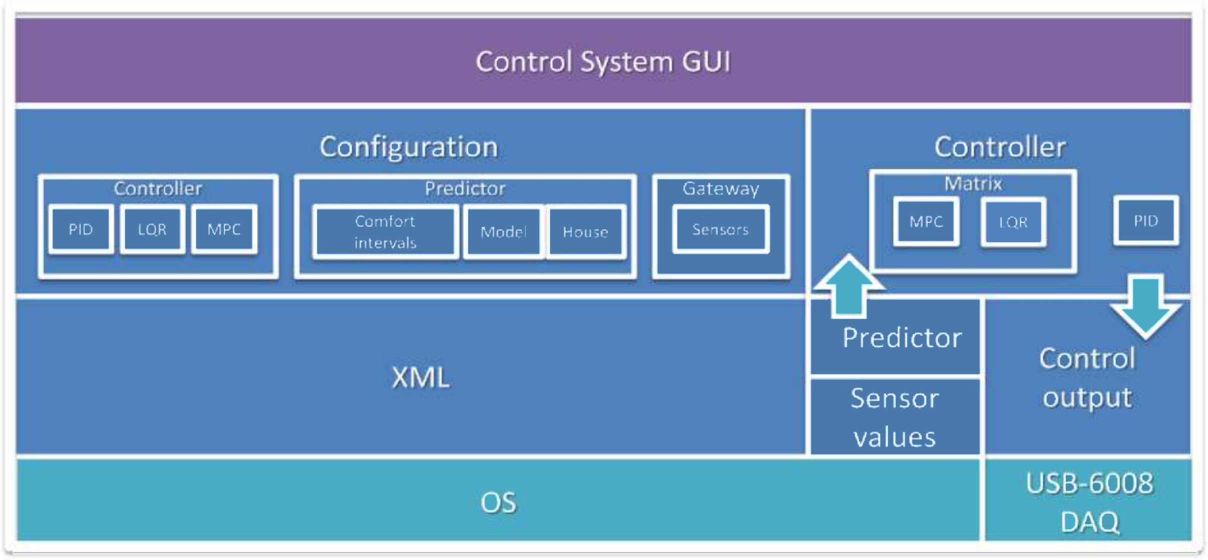


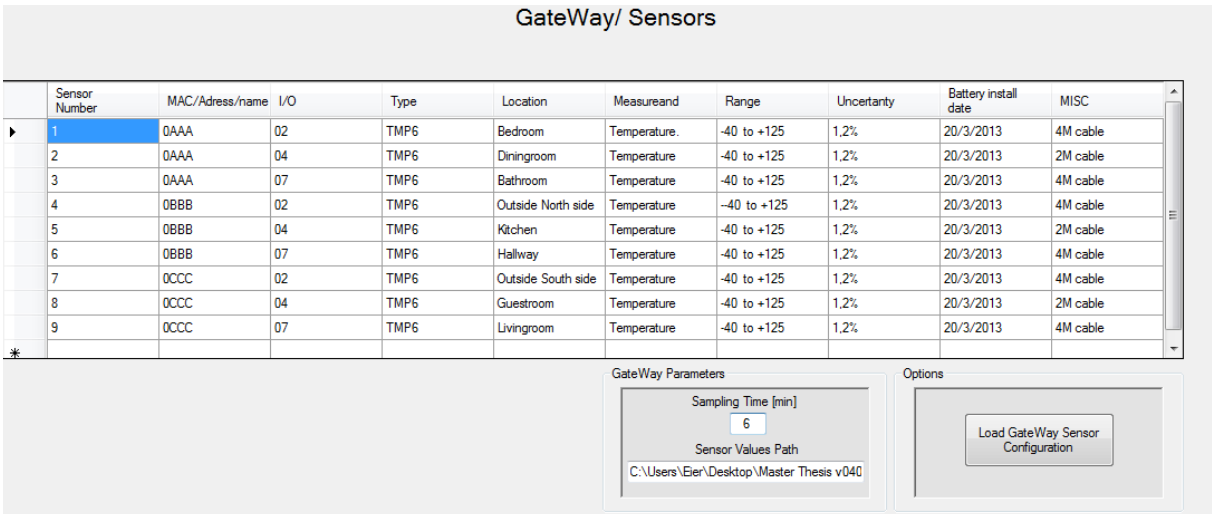
Figure 4-11 Layered architecture of control system

4.6.2 Configuration use case

The configuration use case specific for this last part control system will consist of two main forms, the first will be the control configuration and the second the sensor configuration. The sensor configuration will function as a copy of the gateway configuration used to get the correct sampling time, and sensor specifications. All configuration parameters will be saved to the control system config.XML file. The analysis documentation of the configuration is found in Appendix 4: Fully dressed use case documents paragraph 7.4.3

4.6.2.1Designing the sensor configuration form

The sensor configuration form will read the current configuration from the gateway and store this in the control system configuration. The sensor configuration should not be changeable in the control system and is set to read only. The sensor configuration GUI will contain the table with the sensors and the sampling time and the path of the sensor values. There is a button to prompt the user to select the path of the gateway configuration file and the path of the sensor log file. The created GUI in the sensor configuration form can be seen in Figure 4-12.


GateWay/ Sensors

	Sensor Number	MAC/Adress/name	I/O	Type	Location	Measureand	Range	Uncertanty	Battery install date	MISC
▸	1	0AAA	02	TMP6	Bedroom	Temperature.	-40 to +125	1,2%	20/3/2013	4M cable
	2	0AAA	04	TMP6	Diningroom	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
	3	0AAA	07	TMP6	Bathroom	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
	4	0BBB	02	TMP6	Outside North side	Temperature	–40 to +125	1,2%	20/3/2013	4M cable
	5	0BBB	04	TMP6	Kitchen	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
	6	0BBB	07	TMP6	Hallway	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
	7	0CCC	02	TMP6	Outside South side	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
	8	0CCC	04	TMP6	Guestroom	Temperature	-40 to +125	1,2%	20/3/2013	2M cable
	9	0CCC	07	TMP6	Livingroom	Temperature	-40 to +125	1,2%	20/3/2013	4M cable
*										

GateWay Parameters

Sampling Time [min]: 6

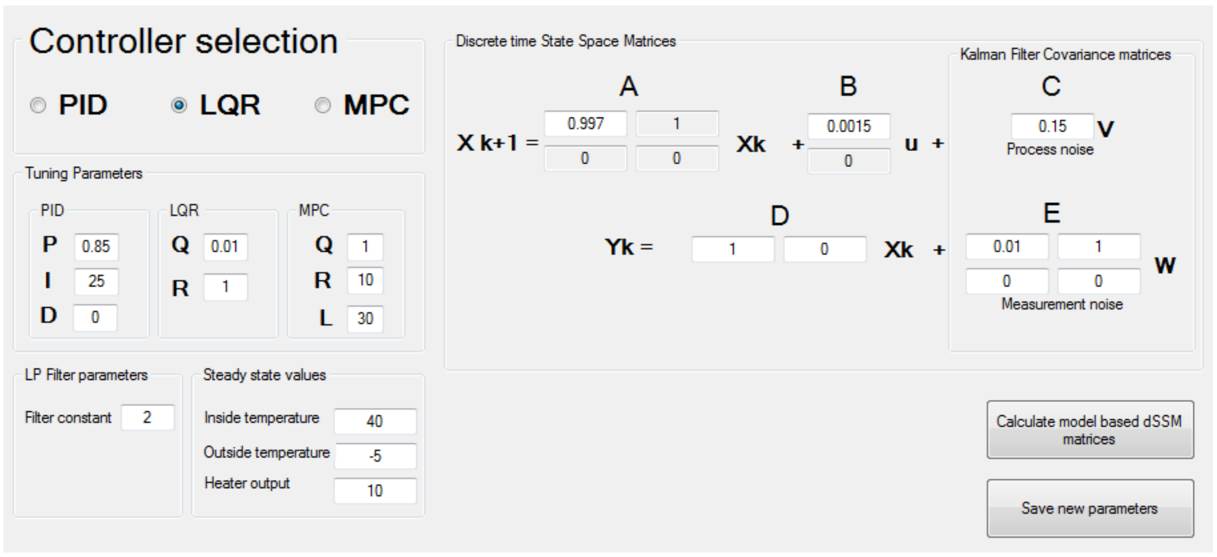
Sensor Values Path: C:\Users\Eier\Desktop\Master Thesis v040

Figure 4-12 Gateway / sensor configuration form GUI seen with sensor and sampling time values from Part 1 paragraph 2.4

Since the system needs the sensor location file in order to function the main system will not start if this file is missing and the user will be prompted to enter the location and name of the file.

4.6.2.2Designing the controller configuration form

The controller configuration should include all the setup needed to calculate the steady state gains from the Kalman filter seen in paragraph 3.2.3.1, the LQR 4.2.3 and the MPC matrices 4.2.4. These setup algorithms are made into the Kalman filter and controller's individual classes. The user should be able to select which controller to use and set the controllers tuning parameters, the LP filter constant, and the steady state values. The steady state values are used together with the house and model data to create the SSM model, however the SSM can also be entered directly in its discrete form. All this is included in the controller configuration GUI seen in Figure 4-13


Controller selection

PID [✓ LQR] MPC

Tuning Parameters

PID		LQR		MPC	
P	0.85	Q	0.01	Q	1
I	25	R	1	R	10
D	0			L	30

LP Filter parameters

Filter constant: 2

Steady state values

Inside temperature	40
Outside temperature	-5
Heater output	10

Discrete time State Space Matrices

$$X_{k+1} = \underbrace{\begin{bmatrix} 0.997 & 1 \\ 0 & 0 \end{bmatrix}}_{A} X_k + \underbrace{\begin{bmatrix} 0.0015 \\ 0 \end{bmatrix}}_{B} u + \underbrace{\begin{bmatrix} 0.15 \end{bmatrix}}_{C} V$$

$$Y_k = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{D} X_k + \underbrace{\begin{bmatrix} 0.01 & 1 \\ 0 & 0 \end{bmatrix}}_{E} W$$

Kalman Filter Covariance matrices

C: Process noise

E: Measurement noise

Figure 4-13 Controller configuration showing all configurations when run using air heater and parameters from Part 2 paragraph 3.5

The MPC controller setup needed some new matrix manipulation functions added to the Matrix library. These extra functions are found in Appendix 5 – Source Code paragraph 7.5.2.5. All parameters are saved in the config.XML file under the added controller element. New elements are added in the same manner as in paragraph 2.3.2.

4.6.2.3Control system configuration GUI

In order to handle all the configurations needed for the created control system the configurations were added to a main configuration forms tab control. The BAS logo and the current date and time are visualized above the tab control. This can be seen in the screen dump of the configuration at run time in Figure 4-14

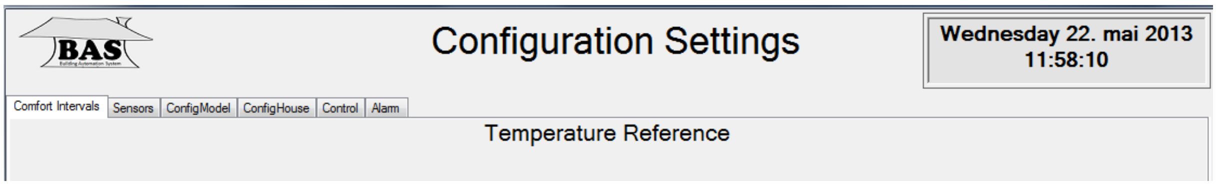


Figure 4-14 Main control system GUI showing all the configurations needed for the control system in tab selections

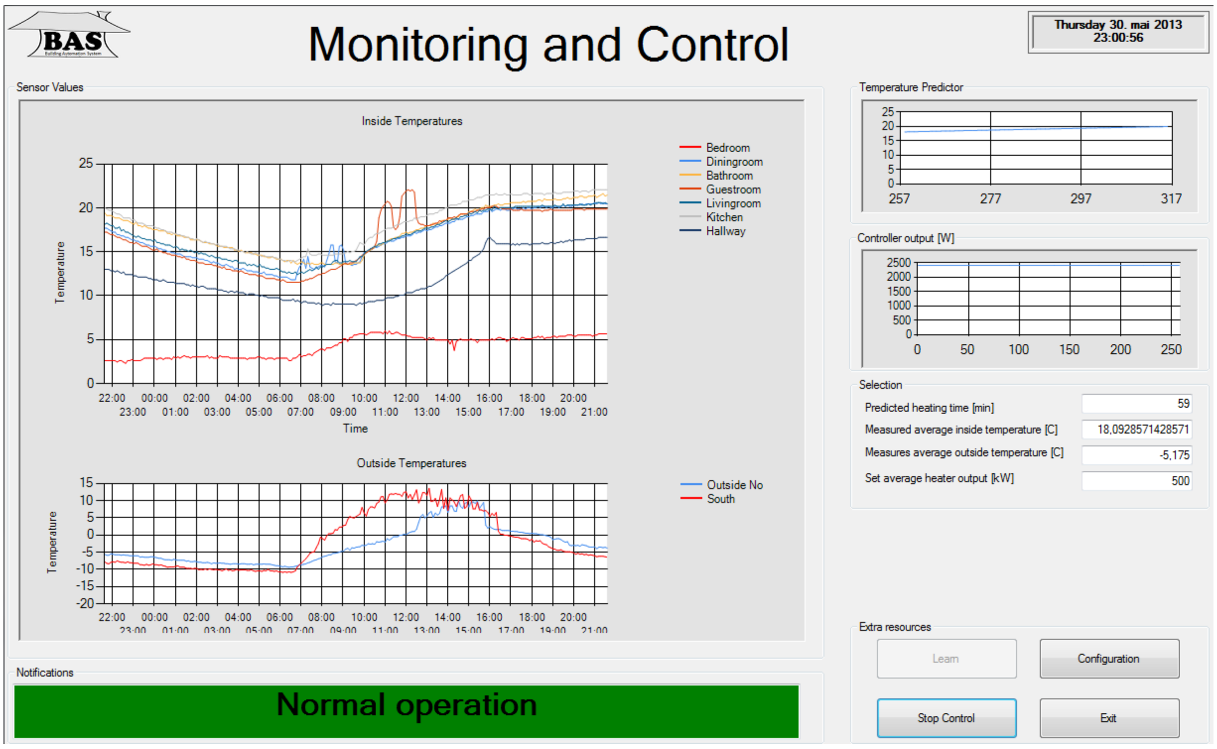
Each configuration parameter is included with a context menu strip giving a right click help button information option, and can be seen in the Appendix 5 – Source Code paragraph 7.5.3.2. Excerpts of the configuration code primarily the control algorithm setup can be found with commentary in Appendix 5 – Source Code paragraph 7.5.3.

4.6.3 Calculate control output use case

The controller use case is based on the parameters set in the configuration and the control theory in paragraph 4.2. Excerpts of the main code algorithm with commentary can be found in Appendix 5 – Source Code paragraphs 7.5.3.3 through 7.5.3.3.4.

4.6.4 Display control system use case, the main GUI

The main GUI of the control system should give the user the necessary information currently available. The main control system GUI is seen in a screen dump in Figure 4-15 while reading the log file from the data acquisition experiments done in part 1 paragraph 2.4. In Figure 4-15 there are two main graphs; one for the inside and one for the outside temperature. Two smaller graphs contain the control output and the predicted heating time. The calculated predicted heating time, average inside temperature and average outside temperature is set in text boxes on the lower left side. The predicted heating time is seen as 59 minutes using the average inside temperature of 18°C for prediction. Average heater output from the five heaters is seen as 500W.



Thursday 30. mai 2013
23:00:56

Selection

Predicted heating time [min]	59
Measured average inside temperature [C]	18,0928571428571
Measures average outside temperature [C]	-5,175
Set average heater output [kW]	500

Notifications

Normal operation

Figure 4-15 Control system GUI while reading sensor values from log file

4.7 Experiments introduction

The BAS system should be tested with the control methods and all relevant software. The MPC and LQR controllers are model dependent and a poor model will result in poor control. In order to make sure the controllers are working correctly a better model of the air heater will be gathered using the DSR subspace system identification method [27].

4.7.1 Experiment setup

The setup to the completed BAS experiments is the same as in part 2 Estimation of heating time and can be found under paragraph 3.5 Experiment setup. The control system is added to the computer software and including the predictor. Some changes were made to the control system for these experiments: The outside temperatures graph was changed to show all the controller outputs at the same time. The steps in the comfort intervals references were set to be 30 seconds, instead of an hour and only Mondays references was used for simplicity. The comfort temperature was set to be 40°C and the low temperature was set to be 30°C. The time stamps were used as the 0.2 second sample time and the MPC prediction horizon was set to 30 samples, or two seconds

To get the best model the air heater was excited using a pseudo random binary reference between the minimum 0V and the maximum 5V outputs [13]. The MATLAB dsr toolbox resulted in the following SSM matrices seen in Equation (4-59)

$$A = 0.9997 \quad B = 0.001459 \tag{4-59}$$

The experiments tuned weighting parameters are seen in Table 4-3.

Table 4-3 Experiments controller parameters

PID controller	$K_p = 0.8$	$T_i = 23$	$T_d = 0$
LQR controller	$q = 0.01$	$r = 100$	
MPC controller	$L = 30$	$q = 1.5 * eye(L),$	$r = 250 * eye(L)$

4.7.2 BAS experiment results

Using the new model the controllers showed some specific categories. The PID controller was simple to tune and produced good results, the LQR was somewhat harder to tune but gave the best results. The MPC controller proved very hard to tune and produced the least good results. The LQR controller can be seen working in the control system screen dump in Figure 4-16 and all three controller experiments results can be seen plotted together using MATLAB in Figure 4-17.

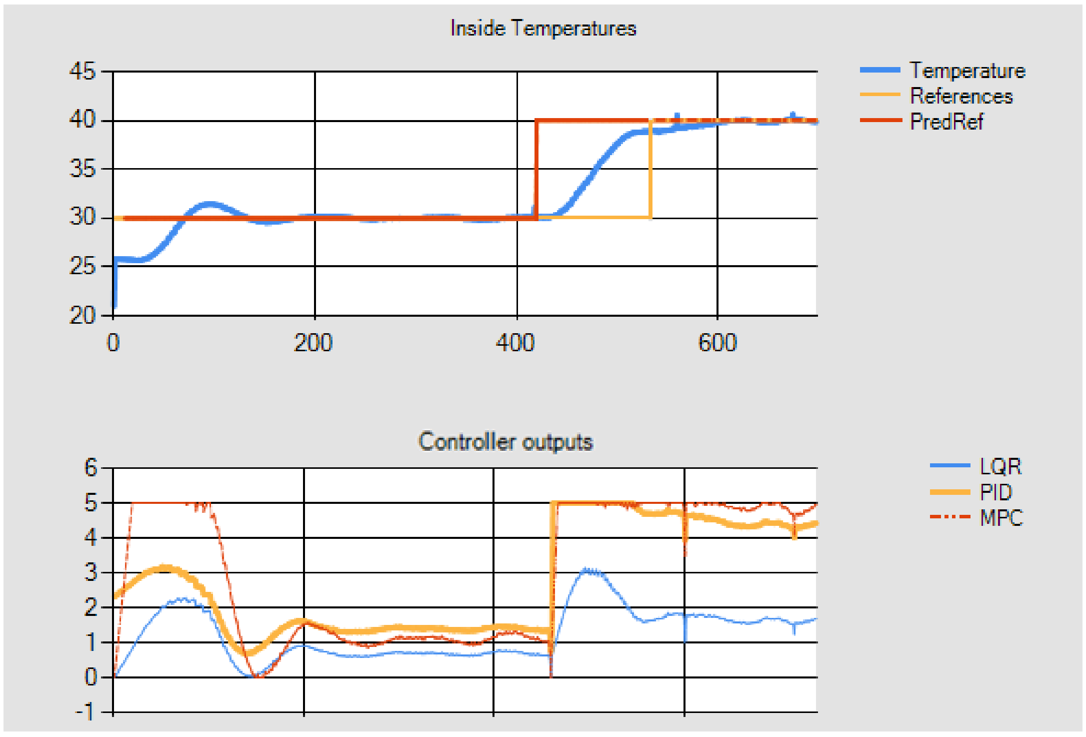


Figure 4-16 Screen dump of control system working with LQR controller; output temperature seen in top graph and all controller outputs seen in bottom graph.

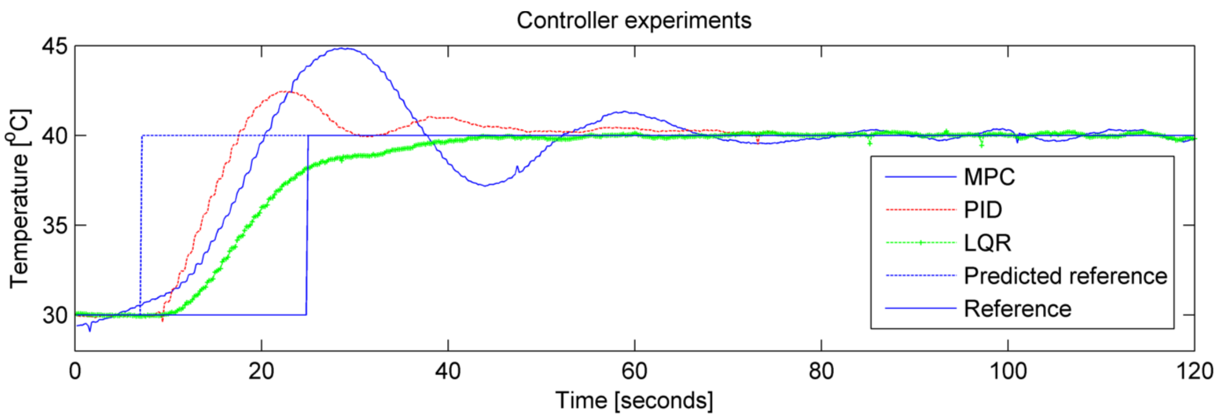


Figure 4-17 All controller experiments results plotted together in MATLAB

4.8 Discussion

The control system experiments showed the standard feedback controller as a viable option for the air heater SISO system producing good control results. A complete house will however be a MIMO system and the correct tuning of several inputs (heaters) and several outputs (room temperature) will be complex [19]. The two controllers applicable for these MIMO systems are the LQR and the MPC. The MPC proved however hard to tune correctly and use complex matrix algorithms in the calculation of the control output. This makes the MPC controller both harder to understand code and debug. The LQR regulator however showed great promise during the experiments. Even though the comfort intervals were reached a little too the LQR both stabilized the system the fastest and also prevented overshoot.

The MPC controller was shown in simulation to produce as good results in paragraph 4.3; however the added complexity of tuning the MPC resulted in poorer performance. The reasons to select the MPC over the LQR is the constraints handling and the future predictions contained in the prediction horizon. In this BAS system however the future predictions are taken care of by the predictor and the only constraints in the system are the maximum and minimum heater power. These constraints are proven handled just as good using if an else statements.

Adding a feed forward controller to the BAS system will prove an advantage in response to sudden changes in the outside temperature. The added controller will also increase the complexity and will rely on a good model. The feed forward controller should be included to the system when a suitable model is obtained and new experiments should be run to answer the need for this controller’s added complexity.

5 Conclusions

The gateway has been thoroughly analyzed, coded and tested. The gateway was running correctly for 12 consecutive days. The theory behind the main parameters of the house model was discussed and during the practical experiments part some ways of determining the U value has been tested.

The model has been proven unsatisfactory in regards to the real experiments data. The model does not take into account the house mass and heat capacity the time constant will be too small. Augmenting the model using a Kalman filter has been shown to largely improve the estimations. The pure OLS regression model proved best when handling systems with time delayed reactions to the control output. Any time delay in the system added to the Kalman filter disturbance estimated SSM model should be added as an offset to the heating time estimations.

Using a simple on-off controller for keeping the temperature at comfort interval was found as inadequate. Three controllers were tested for reliability, complexity and handling of MIMO systems. The MPC controller was found as unnecessary complex, the PID controller will prove advanced to tune in a MIMO system. The optimal controller for the MIMO BAS system is proposed as the LQ regulator. This selection is based on the fact that this BAS system will have a working house model.

A subspace system identification method is simple to implement and will create a very good model when conditions are stable. A first principle model will be more adaptable to changes and the house model should be augmented with the house mass.

During this thesis it has been proven that an adaptable BAS system will result in large energy savings for a common working household.

6 Future work

In order to have a building model predicting correctly the current house model should be augmented with the house mass heat capacity using first principles. The house model should also be adapted to handle several rooms, floors and heaters (MIMO). The house mass is constant and should then be estimated correctly using the Kalman filter.

Implementing a way for measuring the solar radiation is important to get an accurate estimation of heating time. In addition the sensor network should add sensors for measuring the ventilation, air density, and pressure.

The ZigBee gateway should include a send method and tests should be performed using the software PWM together with heaters and the gateway in future experiments.

The control system should add the possibility of MIMO systems configurations, and different temperature settings in different rooms. In addition a better way of sorting the inside and outside temperature sensors should be implemented.

A feed forward controller should be implemented to work during the comfort intervals to minimize the influence of the outside temperature.

The DSR subspace system identification algorithm should be implemented in VS. This algorithm can be used in parallel with the first principle house model to further test the functionality.

References

i

[1] X. G. Stian Krogstad, Moa'atasim Amer, Terje Nordal, "Smart House: BAS Design for Electrical Heating," 2012.

[2] L. H. G. Birger Bergesen, Benedicte Langseth, and D. S. Ingrid H. Magnussen, Jun Elin Wiik Toutain. (2012). *Energibruksrapporten 2012 - Energibruk i husholdninger*. Available: http://webby.nve.no/publikasjoner/rapport/2012/rapport2012_30.pdf

[3] S. Sentralbyrå. (2011). *Priser på elektrisk kraft, 4. kvartal 2010*. Available: http://ssb.no/elkraftpris/arkiv/art-2011-01-11-01.html

[4] Nobø. (2013). *Energikontroll gjort enkelt*. Available: http://www.nobo.no/

[5] C. P. D.W.U. Perera, N.-O. Skeie "Modelling and simulation of heat dynamics of a single room with ventilation under Norwegian building regulations," 2013.

[6] K. K. Andersen, H. Madsen, and L. H. Hansen, "Modelling the heat dynamics of a building using stochastic differential equations," 1998.

[7] M. S.-M. Bertil Thomasa, Per Fahle, "Feed-forward in temperature control of buildings," 2004.

[8] I. Forening. *Forskrift om krav til byggverk (Byggteknisk forskrift) TEK10*. Available: http://www.ipf.as/forskrifter.htm

[9] RIBA. (2013). *U-Values*. Available: http://www.architecture.com/SustainabilityHub/Designstrategies/Earth/1-1-1-10-Uvalues(INCOMPLETE).aspx

[10] S. Krogstad, "ZigBee PRO development kit set up guide," ed, 2012.

[11] Z. Alliance, "ZigBee Specification," 2007.

[12] T. Instruments, "A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications," 2009.

[13] F. Haugen. (2010). *Lab Station: Air Heater*. Available: http://home.hit.no/~finnh/air_heater/

[14] N.-O. Skeie, "Object-Oriented Analysis, Design, and Programming using UML and C#," 2011.

[15] T. Instruments, "CC253x System-on-Chip Solution for 2.4-GHz, IEEE 802.15.4 and ZigBee® Applications, CC2540/41 System-on-Chip Solution for 2.4-GHz Bluetooth® low energy Applications User's Guide " 2012.

[16] Wikipedia. (2011). *CSV*. Available: http://no.wikipedia.org/wiki/CSV

[17] D. Obasanjo and M. Corporation. (2003). *XML Serialization in the .NET Framework*. Available: http://msdn.microsoft.com/en-us/library/ms950721.aspx

[18] A. Devices. (2010). *Low Voltage Temperature Sensors TMP35/TMP36/TMP37*. Available: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf

[19] D. D. Ruscio, *System theory, State Space Analysis and Control Theory*, 1996.

[20] T. Finn Haugen. (2005). *Discrete-time signals and systems*. Available: http://www.scribd.com/doc/26420589/39/Discretization-with-zero-order-hold-element-on-the-input

[21] D. D. Ruscio, *Subspace System Identification Theory and applications*, 1995.

[22] C. F. Pfeiffer, "Title," unpublished|.

[23] I. Kuckir. (2011). *Lightweight fast matrix class in C# (Strassen algorithm, LU decomposition)*. Available: http://blog.ivank.net/lightweight-matrix-class-in-c-strassen-algorithm-lu-decomposition.html

[24] H. P. L. Aslak Tveito, Bjørn Frederik Nielsen, Xing Cai, "Elements of Scientific Computing," ed, 2010.

[25] H. P. Halvorsen. (2013). *NI USB-6008 DAQ Device*. Available: http://home.hit.no/~hansha/documents/lab/Lab%20Equipment/NI%20USB-6008%20DAQ%20Device/NI%20USB-6008%20DAQ%20Device.pdf

[26] Wikipedia. (2013). *What is the U value of plexiglass?* Available: http://wiki.answers.com/Q/What_is_the_U_value_of_plexiglass

[27] D. D. Ruscio. (2013). *D-SR Toolbox for MATLAB*. Available: http://www-pors.hit.no/tf/fag/sce2206/d-sr/d-sr_e.html

[28] S. Skogestad, "Probably the best simple PID tuning rules in the world," 2001.

[29] D. D. Ruscio, "Discrete LQ optimal control with integral action:
A simple controller on incremental form for MIMO
systems," 2012.

[30] H. P. Halvorsen. (2011). *Model Predictive Control in LabVIEW*. Available: http://home.hit.no/~hansha/documents/labview/training/Model%20Predictive%20Control%20in%20LabVIEW/Model%20Predictive%20Control%20in%20LabVIEW.pdf

[31] D. D. Ruscio, *MODEL PREDICTIVE CONTROL
and optimization*, 2001.

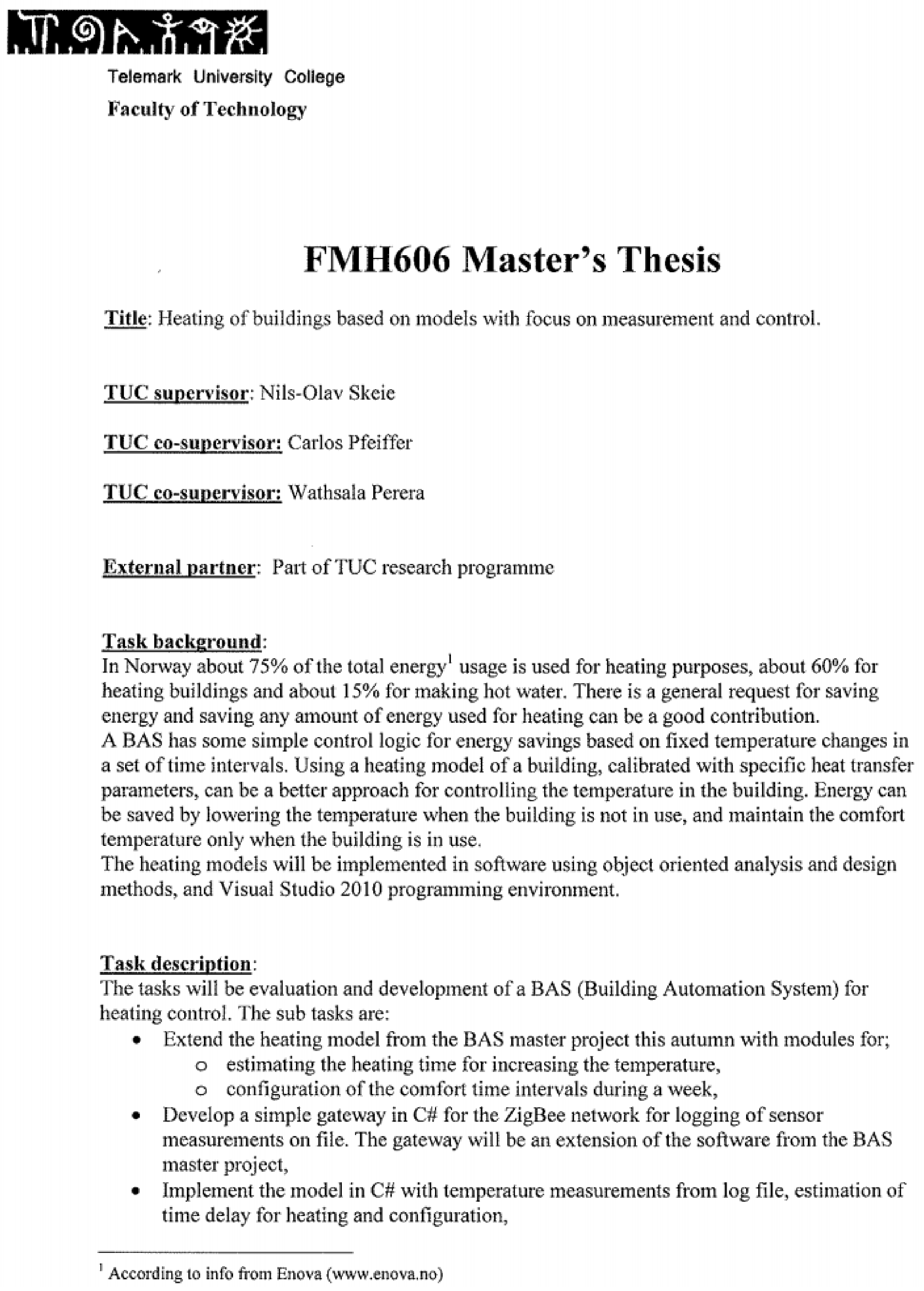
[32] Microsoft. (2013). *Microsoft Solver Foundation*. Available: http://msdn.microsoft.com/en-us/devlabs/hh145003.aspx

[33] D. D. Ruscio, "Model Predictive Control with integral action," 2011.

7 Appendices

Appendices table of contents

7.1 Appendix 1- Thesis text



Telemark University College

Faculty of Technology

FMH606 Master's Thesis

Title: Heating of buildings based on models with focus on measurement and control.

TUC supervisor: Nils-Olav Skeie

TUC co-supervisor: Carlos Pfeiffer

TUC co-supervisor: Wathsala Perera

External partner: Part of TUC research programme

Task background:
In Norway about 75% of the total energy[1] usage is used for heating purposes, about 60% for heating buildings and about 15% for making hot water. There is a general request for saving energy and saving any amount of energy used for heating can be a good contribution.
A BAS has some simple control logic for energy savings based on fixed temperature changes in a set of time intervals. Using a heating model of a building, calibrated with specific heat transfer parameters, can be a better approach for controlling the temperature in the building. Energy can be saved by lowering the temperature when the building is not in use, and maintain the comfort temperature only when the building is in use.
The heating models will be implemented in software using object oriented analysis and design methods, and Visual Studio 2010 programming environment.

Task description:
The tasks will be evaluation and development of a BAS (Building Automation System) for heating control. The sub tasks are:

- Extend the heating model from the BAS master project this autumn with modules for;
    - estimating the heating time for increasing the temperature,
    - configuration of the comfort time intervals during a week,
- Develop a simple gateway in C# for the ZigBee network for logging of sensor measurements on file. The gateway will be an extension of the software from the BAS master project,
- Implement the model in C# with temperature measurements from log file, estimation of time delay for heating and configuration,

[1] According to info from Enova (www.enova.no)

- The model will depend of a set of parameters from the building, explain these parameters and make some suggestions how these can be verified by a measurement system,
- Describe different methods for controlling the inside temperature in the building,
- Discuss possible controller strategies that can be used for existing type of heaters for buildings,
- Develop a controller in C#, based on the model and configuration, to maintain the inside temperature in the building according to the comfort time intervals from configuration,

Student category:
SCE student who is familiar with;

- the BAS master project,
- developing software for ZigBee sensor network,
- Visual Studio 2010 for developing software.

Practical arrangements:
A ZigBee sensor network with software development kit will be available.

Signatures:

Student (date and signature): [illegible]

Supervisor (date and signature): [illegible] 18-FEB-13

7.2 Appendix 2: Measuring the U value

In order to measure the U values directly with the ZigBee sensor network the house and outside temperatures are needed in a steady state. This might however happen in some meteorological instances and the U-values can be measured by using a tile with known resistance and three temperature sensors. This is seen in Figure 7-1.

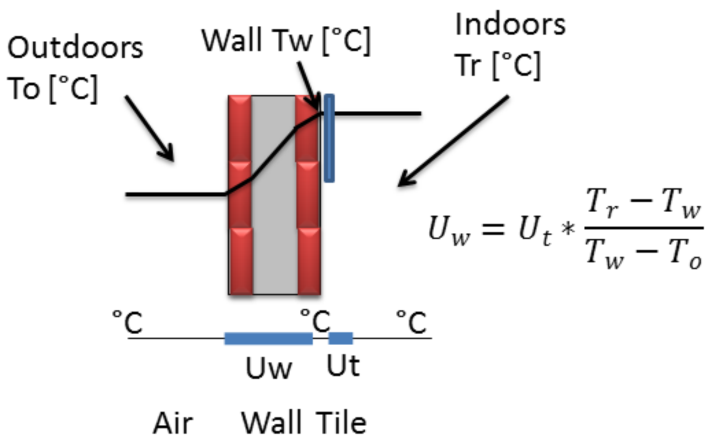


Figure 7-1 Measuring the U value experimentally

Another way to measure the total house heat leakage value, $\overline{U}_T$, or energy leakage would be to measure the amount of power used over a period of steady state conditions[8], with all ventilation closed, and use that in steady state we have as seen in (7-1) and (7-2).

$$Q_{loss} = Q_{work} = \overline{U}_T A \Delta T \tag{7-1}$$

$$\overline{U}_T = \frac{Q_{work}}{A\Delta T} \tag{7-2}$$

Where A is the total surface area of the house, and $\Delta T$is the difference between inside and outside temperatures and Q_w is the heater power used to keep the inside at constant temperature.

Both these estimations rely heavily on steady state conditions and while steady state conditions in the inside temperature is obtainable. Steady state in the outside temperature rarely happens over any large amount of time. The best way to measure the U value would be to use a heat flux based measurement as `TRSYS01` from Houseflux Thermal sensors. The heat flux based measurement does not need steady state conditions.

[8] Steady state means

7.3 Appendix 3 FURPS+

7.3.1 Gateway FURPS+

Functional	Get sensor values from the ZigBee sensors, and display them to a LCD Log to file on disc, parsed data and raw data if needed, containing tag information, current date and time Configuration; Specify time for saving the sensor data, holding all the needed information about the sensors, and all the serial link properties
Usability	Language English Keyboard and mouse Display (current values, parsed and raw and configuration) Hard Disk for saving the sensor data, and keeping the configuration Configuration file should be XML v 1.0 format containing sensors, sampling time and serial link configuration. LOG standard is in text format with extension .log as: [datetime;sensor1value;sensor2value;sensor3value….]
Reliability	The system will run 24x7.
Performance	Save file timer set in configuration Serial read existing parameter timer set in program. [50ms].
Supportability	
+	The gateway will run on windows based OS (32/64bit) using C# and windows forms

7.3.2 Predictor FURPS+

Functional	Read sensor values from the log file created by gateway Predictor: Learn the system by logging the data and turning the heater on full. Learn both using OLS and K-OLS-SSM estimates. Predict the heating time based on current environmental values, and the prediction models created from the learn function. Configuration of the house model, house parameters, and comfort intervals
Usability	Language English Keyboard and mouse Display (current values, parsed and raw and configuration) Hard Disk for reading the sensor data, and keeping the configuration Configuration file should be XML v 1.0 format containing sensors, sampling time and serial link configuration. Set heater output using DAQ 6008 device
Reliability	The system will run when user specified and automatically the first time
Performance	Use sampling time from gateway to get correct prediction regression models Maximum output values in the DAQ-6008 device is 5V
Supportability	
+	The gateway will run on windows based OS (32/64bit) using C# and windows forms

7.3.3 Control System FURPS+

Functional	Control the temperature and calculate the controller outputs based on the model and the configurations Display the complete control system GUI, plot the inside temperature outside temperature and heater output, and heating time estimation as graphs, give a clear indication to which controller method has been selected. Configuration of controllers and sensors including the Predictor configurations in a main configuration form. All configuration kept in a static data object and XML file.
Usability	Language English Keyboard and mouse Display Hard Disk for the configuration Configuration file should be XML v 1.0 format Controller output will be set using DAQ 6008 device
Reliability	The system will run when user specified and automatically the first time
Performance	Run each sampling time set in the gateway Maximum output values in the DAQ-6008 device is 5V Minimum output is set to be 0V
Supportability	
+	The gateway will run on windows based OS (32/64bit) using C# and windows forms

7.4 Appendix 4: Fully dressed use case documents

7.4.1 Gateway

Use Case # 1 Gateway.

1	Use Case Name	Configuration
2	System/Scope	Gateway
3	Level	User Goal
4	Primary Actor	HD
5	Stake Holders	Control System
6	Preconditions	NA
7	Success Guarantee	Configuration created, opened and saved
8	Main success scenario	1 Open configuration 2 Edit configuration add sensors 3 Save configuration
9	Extensions	1A no file to open, create new configuration file 1B Error in file, create new configuration file 3A Save error, give message to user, retry?
10	Special Requirements	Want to use XML V1.0 based configuration files
11	Technology List	
12	Frequency of occurrence	Each time the user needs to change the configuration or the system will need the configuration data
13	MISC	The configuration will be broken up in one windows form for displaying and editing the configuration values and one class file for the configuration XML based functions. This file will also handle extensions 1A and 1B

Use case # 2 Gateway

1	Use Case Name	DisplayConfigData
2	System/Scope	Gateway
3	Level	User Goal
4	Primary Actor	LOG HD and Display, User (keyboard, mouse)
5	Stake Holders	Control System
6	Preconditions	
7	Success Guarantee	Configuration loaded, changed, and saved
8	Main success scenario	1 Open Configuration (USER) 2 Display configuration data 3 Edit Configuration data (USER) 4 Save Configuration data to HD
9	Extensions	4A Save new Configuration data Y/N/C 4A: Y save the information , exit form 4A: N do not save new information, exit form 4A: C Break operation and return to form
10	Special Requirements	Want to use XML based configuration files
11	Technology List	.NET V, XML Version 1.0
12	Frequency of occurrence	@ user request
13	MISC	The software will be de the GUI between the user and the configuration data

Use case #.3 Gateway

1	Use Case Name	LOG
2	System/Scope	Gateway
3	Level	User Goal
4	Primary Actor	Harddisc
5	Stake Holders	Control System
6	Preconditions	
7	Success Guarantee	Raw serial data received, parsed, saved and returned
8	Main success scenario	1 Raw serial data received 2 Split up raw serial data in messages 3 Parse the split data into sensor data [datetime;sensor1;sensor2; . .] 4 Save data
9	Extensions	1A no data received 1B Return empty string 3A return NAN if no value is found for that sensor 4A IO error, save error in error.log 4B retry
10	Special Requirements	
11	Technology List	.NET V, XML Version 1.0
12	Frequency of occurrence	@ program request
13	MISC	The LOG will be used by the other use cases in order to save error logs and, parse the raw serial data.

Use case #4 Gateway

1	Use Case Name	DisplaySerialData
2	System/Scope	Gateway
3	Level	User Goal
4	Primary Actor	ZigBee Coordinator[Serial port], timer
5	Stake Holders	Control System
6	Preconditions	The ZigBee network has been started and is up and running with all devices. The ZigBee SW is working correctly
7	Success Guarantee	Sensor values read, displayed and saved to file
8	Main success scenario	1 Get Configuration (serial port, and save file timer) 2 **User pressed start log button** 3 Disable Configuration button 4 Open COM port 5 Save file dialog, select file to save log 5 Start COM timer 5.1 Read existing serial data into memory, and text box 5.2 Sleep [100ms] 5.4 go to 3.1 6 Start save file timer 6.1 Send raw data to the log class, parsed data returned 6.2 Send Parsed data to parsed data text box 6.3 Save Parsed data, and raw data if checkbox is checked 6 .3 Sleep [Save File Timer] **7 Stop log button pressed** 7.1 Stop logging to file 7.2 Enable configuration button **8 User press configuration button** 8.1 Show DisplayConfigData form **9 User exit using Cross** 9.1 Hide application 9.2 Give notice of application still running **10 User exit using exit button** 10.1 Application stopped

Extensions	4A COM port error, give message box warning 4B Save error to error.log 4B Break saving operation 5 No file selected 5B Show message box warning 5C Break operation
Special Requirements	Want to use XML based configuration files
Technology List	ZigBee Pro Development Kit Gateway device – Coordinator COM/ Serial port/ or USB to Serial port
Frequency of occurrence	@ Program request (each sampling time)
MISC	

7.4.2 Predictor

Use case #1 Predictor

1	Use Case Name	Configuration (Predictor)
2	System/Scope	Predictor (Control System)
3	Level	User Goal
4	Primary Actor	OS
5	Stake Holders	Control System
6	Preconditions	NA
7	Success Guarantee	Configuration created, opened and saved
8	Main success scenario	*Model parameters (molar mass, gas constant etc.)* 1 Open configuration 2 Edit configuration 3 Recalculate parameters 4 Load default values 5 Save configuration *House parameters (heater effect, house area, volume ventilation, etc.)* 6 Open configuration 7 Edit configuration 8 Save configuration *Comfort Intervals (comfort temperature, low temperature)* 9 Open configuration 10 Set comfort temperature 11 Set low temperature 12 save new configurations

Extensions	*Model parameters (molar mass, gas constant etc.)* 1A no file to open, create new configuration file with default values 1B Error in file, create new configuration file 2A only numerical values and one separating sign allowed 3A Save error, save the error to the error log file *House parameters (heater effect, house area, volume ventilation, etc.)* 4A no file to open, create new configuration file with default values 4B Error in file, create new configuration file 5A only numerical values and one separating sign allowed 8A Save error, save the error to the error log file *Comfort Intervals (comfort temperature, low temperature)* 9A no file to open, create new configuration file with default values 9B Error in file, create new configuration file 10A only numerical values and one separating sign allowed 11A only numerical values and one separating sign allowed 12A Save error, save the error to the error log file
Special Requirements	Want to use XML V1.0 based configuration files
Technology List	
Frequency of occurrence	Each time the user needs to change the configuration or the system will need the configuration data
MISC	The configuration will be broken up into three windows forms for displaying and editing the configuration values and one class file for the configuration XML based functions. The configXML class will handle extensions 1A and 1B

Use case # 2 Predictor.

1	Use Case Name	Get sensor values
2	System/Scope	Predictor (Control System)
3	Level	User Goal
4	Primary Actor	OS
5	Stake Holders	Control System
6	Preconditions	Sensor values saved by the Gateway
7	Success Guarantee	Sensor values log file open and read to correct line
8	Main success scenario	1 Using log file (opening and closing at once reading is done) 2 Read to current line 3 Filter the data through a low pass filter 3 Convert the ADC values to temperature data
9	Extensions	1A no file to open, prompt user with file missing error message 1B Error in line, create new configuration file 2A End of file 2B Wait one sampling time 3A Non numerical values received 3B Save error log message 3C Jump to next line
10	Special Requirements	Want to use XML V1.0 based configuration files
11	Technology List	
12	Frequency of occurrence	Each time the user needs to change the configuration or the system will need the configuration data
13	MISC	The low pass filter function will be created in one class and read sensor values in another class

Use case # 3 Predictor

1	Use Case Name	Predictor
2	System/Scope	Predictor (Control system)
3	Level	User Goal
4	Primary Actor	Control System
5	Stake Holders	Control System
6	Preconditions	Sensor values read by Read sensor values Model, house and temperature data stored in XML file DAQ-6008 device on line
7	Success Guarantee	Learn function finished and regression models created
8	Main success scenario	*Learn part* 1 Read configuration 2 Run three sample times to stabilize Kalman gain 3 Set heaters to maximum 4 Comfort temperature reached 5 Save regression models *Prediction part* *6 Select OLS or K-OLS-SSM model* 7 Run heating time estimations 8 Get comfort intervals reference 9 Go to 6
9	Extensions	1A no file to open, prompt user with configuration file missing 1B Error in configuration, prompt user to create new file 4A Comfort temperature not reached within maximum time 4B Stop predictor and prompt user 5A File save error 5B prompt user and save error in error.log file
10	Special Requirements	Want to use XML V1.0 based configuration files
11	Technology List	DAQ-6008 USB device
12	Frequency of occurrence	Learn at fresh startup or user interaction Prediction each sampling time
13	MISC	

7.4.3 Control System

Use case #1 for control system

1	Use Case Name	Configuration
2	System/Scope	Control system
3	Level	User Goal
4	Primary Actor	Display
5	Stake Holders	Control System
6	Preconditions	
7	Success Guarantee	Configurations parameters opened, edited and saved.
8	Main success scenario	*Sensor values configuration* 1 Load current configuration from XML file 2 User button pressed load gateway configuration 3 Prompt user for path of gateway configuration 4 Get configuration from gateway config.xml file 5 Prompt user for path of sensor values log file 6 Read sensor.log path 7 Save new values to control system Config.XML file *Controller configuration* 8 Load current configuration from XML file 9 Edit configuration parameters set in text boxes 10 Calculate new SSM matrices using set parameters 11 Save button to save new configuration

9	Extensions	*Sensor values configuration* 1A File does not exist, Create new file 1B Error in file, Create new file 2B Prompt user with error message 4A No file selected 4B Prompt user with error message 6A Save error 6B Prompt user with error message *Controller configuration* 8A File does not exist, Create new file 8B Error in file, Create new file 6A Save error 6B Prompt user with error message
10	Special Requirements	Want to use XML V1.0 based configuration files
11	Technology List	
12	Frequency of occurrence	Each time the user needs to change the configuration or the system will need the configuration data.
13	MISC	This configuration also contains all parameters from the predictor configuration. Extensions 1A, B and 8A,B handled by the ConfigXML class In order to have changes made while running and save time used to open and read the XML file often the ConfigXML class should be set as static

Use case #2 for the control system

1	Use Case Name	Controller
2	System/Scope	Control system
3	Level	User Goal
4	Primary Actor	USB-6008 and Predictor
5	Stake Holders	Control System GUI
6	Preconditions	
7	Success Guarantee	Configurations parameters opened, control output calculated and sent to DAQ device.
8	Main success scenario	1 Load current configuration from XML file 2 Open DAQ-6008 device 3 Get current reference and sensor values from predictor 4 Get current controller from XML config file 5 Calculate control output 6 Send control output to DAQ-6008 device 7 Go to 3 (loop) 8 At control system GUI request stop control and close DAQ-6008

9	Extensions	1A File does not exist or file error, Prompt user for error 1B Break control 1C Open configuration 2A DAQ not connected, 2B Message user 2C Break control operation 4A Go to 1A 5A Error in Calculation 5B Save error message in error log file 6A DAQ-error 6B Send error message user 6C Break control operation
10	Special Requirements	Using XML V1.0 based configuration files
11	Technology List	DAQ-6008 USB Device Predictor
12	Frequency of occurrence	At control system request each sample time 24/7
13	MISC	All control system configuration set as a static value in the main form Control System GUI to speed up system since reading and parsing XML file takes too much time during loop.

Use case #3 for the control system

1	Use Case Name	Control System GUI
2	System/Scope	Control system
3	Level	User Goal
4	Primary Actor	Display
5	Stake Holders	
6	Preconditions	
7	Success Guarantee	Configurations parameters opened, edited and saved.
8	Main success scenario	1 Load current sensor configuration from XML file 2 Set up plots with correct sensors one series for each sensor and outside temperatures in one graph and inside temperatures in another graph. Separate the inside and outside sensors using the “out” keyword. 3 User button pressed start control 4 Start predictor 5 Start Controller 6 Plot Sensor values 7 Plot Controller values 8 Plot Predicted heating time 9 Go to 4 (loop)

9	Extensions	1A Handled by Config.XML class 4A Handled by predictor class 5A Handled by controller class 8A User interaction , stop control button pressed 8B Verify stop control using Y/N 8C Y- Break control operation 8E N – Continue operation
10	Special Requirements	Using XML V1.0 based configuration files
11	Technology List	Software – Gateway and Predictor USB -6008 device Display with minimum resolution 1024*768
12	Frequency of occurrence	24/7
13	MISC	

7.5 Appendix 5 – Source Code

7.5.1 Gateway Code Excerpts

7.5.1.1The Configuration Code

Setting the configuration data parameters

The XML serializer function works by enabling conversion of XML documents to common language runtime objects [1]. In order to use this function all the data needs to be collected in a class of objects. This class has been called *ConfigData* and can be seen in VScode 7-1

```
public class ConfigData
    {
        public Sensor[] Sensors;
        public Timers timer;
        public Serial serial;
```

VScode 7-1 The configdata class

From the VS output there is seen that the sensor is created as a sensor array since it will include several sensors, the serial and timer are singular properties. The next step is to create the sub element sensor with the configuration data selected for each sensor. This is done by setting the type, and sensor data as a struct. This can be seen in VScode 7-2.

```
 [XmlType(TypeName = "Sensor")]
    public struct Sensor
    {
        public string Macaddr;
        public string IO;
        public string Type;
        public string Location;
        public string Measureand;
        public string Range;
        public string Uncertanty;
        public string BatteryDate;
        public string MISC;
    }
```

VScode 7-2 Creating the XML nodes for the sensors

The timer and serial properties are set in same manner and for further information the code with notations can be found in Appendix.

Writing the XML data to file, *ConfigWriteData* method

The writing data method will use *FileStream* with parameter *FileMode.Create*, the file will be created on saving, and if the file previously exists it will be overwritten. The using statements are used to be sure the garbage handler will remove the created *FileStream* instance after use. The XML root is set before and the *XmlSerializer* method is used to create the XML file with the structure properties from the *ConfigData* class. This is seen in a code excerpt in VScode 7-3.

```
using (var fs = new FileStream(ConfigFileName, FileMode.Create))
                {
                    XmlRootAttribute root = new XmlRootAttribute("Config");
                    XmlSerializer xs = new XmlSerializer(typeof(ConfigData), root);
                    xs.Serialize(fs, Data);
                    fs.Close();}
```

VScode 7-3 XML write method

Reading the XML data, *ConfigReadData* method

It is also important to read the XML configuration data from within the program. The same file method is used as in the write function with different settings. The *XmlSerializer* function *deserialize* the XML document into data objects, based on the *ConfigData* class. The main parts of the read function can be seen in VScode 7-4

```
using (FileStream fs = new FileStream(ConfigFileName, FileMode.Open))
                {
                    XmlRootAttribute root = new XmlRootAttribute("Config");
                    XmlSerializer xs = new XmlSerializer(typeof(ConfigData), root);
                    Data = (ConfigData)xs.Deserialize(fs);
                    fs.Close();
                }
```

VScode 7-4 Reading from XML file

One important error handling is what will happen if the file contains faulty or missing data. This would create an file exception and the program would crash. This is handled by using a try and catch statement set around the function, and the catch will then create a new empty instance of the configuration XML file. This can be seen in VScode 7-5.

```
            catch(Exception e) Data = new ConfigData();
```

VScode 7-5 Creating a new file if missing or faulty

7.5.1.2 Code results

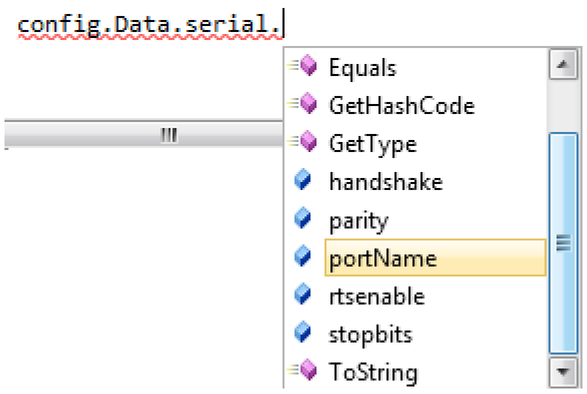
A new instance of the *config* class can be created and the *ConfigReadData* method can be run giving the program access to the XML data as objects. This is seen in VScode 7-6

```
config = new Config();

config.ConfigReadData("config.xml");
```

VScode 7-6 Running the read function

Then the serial port settings can be accessed as is seen in Figure 7-2



```
config.Data.serial.
```

Figure 7-2 Accessing the configuration data

When the correct settings are applied the new data can be saved using the write function, this can be seen in VScode 7-7.

```
config.Data.serial.portName = "COM1";
config.Data.timer.SaveTimer = 2000;

config.Data.Sensors[0].Macaddr = "0AAA";
config.Data.Sensors[0].IO = "00";


config.ConfigWriteData("config.xml");
```

VScode 7-7 Running the write function

When the code in VScode 7-7 is run the XML file data was created as can be seen in XMLscript 1.

```
<?xml version="1.0"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Sensors>
    <Sensor>
      <Macaddr>0AAA</Macaddr>
      <IO>00</IO>
    </Sensor>
  </Sensors>
  <timer>
    <SaveTimer>2000</SaveTimer>
  </timer>
  <serial>
    <portName>COM1</portName>
  </serial>
</Config>
```

XMLscript 1 Configuration result when using the write function

7.5.1.3Testing software and error handling

In the configuration use case the testing part is mainly to see how the created functions will react to false or bad data, either due to the file being tampered with or the configuration file being deleted to see that these eventualities are taken into account by the program and no crashes will occur. First the functions will be tested when the config.xml file has been removed.

Reading configuration file with missing file

If the *FileStream* function tries to read a file that is not there, an unhandled exception will be thrown resulting in software crash. This is fixed by the *try* and *catch* implemented in the read function. If a file is removed a new config.xml file will be created with the basic information given in the *ConfigData* class.

Reading configuration file with faulty data

If the *XmlSerializer* function reads data that is not of the type set in the *ConfigData* class there will be thrown an exception, and the same *catch* used for the missing file will create a new config.xml and overwrite the faulty data.

Error during saving of file

There might also be created an IO error when saving to file, if for instance another program is using the file at the exact same instance. These amongst other errors are handled by a try and catch statement around all methods that have the possibility to fail. All catch statements are included with a function to write the error to an errog.log file. The error.log file contains the time and date for the error, the type of error and a location notation on where the error occurs. This can be seen in VScode 7-8for the configuration write method.

```
using (System.IO.StreamWriter file = new System.IO.StreamWriter(@error.log, true))
                {
file.WriteLine(DateTime.Now.ToString()+e.Message + e.Source + "@ConfigWriteData");
               }
```

VScode 7-8 Error handling in configuration use case

7.5.1.4The DisplayConfigData code

The code is based on reading and saving the configuration data using the *config* class. In order to get the current available COM ports there is created a code for adding the computers available COM ports to the combo box. This can be seen in VScode 7-9.

```
foreach (string s in System.IO.Ports.SerialPort.GetPortNames())
            {
                comboBoxComPort.Items.Add(s);
            }
```

VScode 7-9 Adding the available COM ports to a combo box

Getting the other serial information is done using the enumerable lists contained in *System.IO.Ports.*

Two main methods are created to read and write the displayed configuration information, *DisplayConfigDat*a and *DisplaySaveConfig*, both will be explained in turn.

Reading the configuration data, the *DisplayConfigData()* method

In order to display the sensor information in the table[9] a *foreach* loop is used looping through all the sensor objects in Sensors. In order to keep the numbering "out of the user's hands" the first column is set as write protected and will only contain the sensor automatic counter. This can be seen in the VScode 7-10

```
foreach (Sensor sensor in config.Data.Sensors)
            {
                ConfigurationSensorTable.Rows.Add(i, sensor.Macaddr, sensor.IO, 
sensor.Type, sensor.Location, sensor.Measureand, sensor.Range, sensor.Uncertanty, 
sensor.BatteryDate, sensor.MISC);
                i++;
            }
```

VScode 7-10 reading the data to the table

The serial and timer configuration is simply read straight into their control containers as can be seen by the excerpts VScode 7-11

```
comboBoxComPort.Text = config.Data.serial.portName;
textboxSaveTimer.Text = Convert.ToString(config.Data.timer.SaveTimer);
```

VScode 7-11 reading configuration data into combo boxes

[9] Data grid view control is used as table

Saving the configuration information, The *DisplaySaveConfig()* method

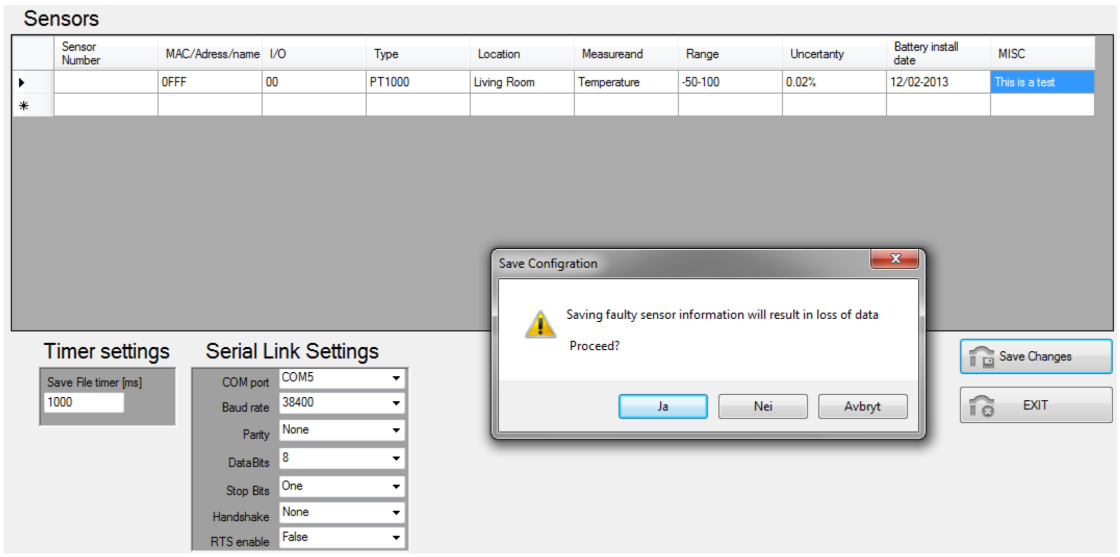
When saving the sensor information a switch case statement is used together with the *configWriteData* method. The switch case is used to give the user the ability to cancel saving changes in the standard windows setup yes is to save and exit, no is to exit without saving and cancel is break the saving operation and return to the form. The information in the table, combo boxes and text boxes are set as the data to their corresponding objects. An excerpt of this can be seen in VScode 7-12

```
//Sensors
config.Data.Sensors[i].Macaddr = (string)ConfigurationSensorTable[1, i].Value;
//timer
config.Data.timer.SaveTimer = Convert.ToInt32(textbox_saveTimer.Text);
//serial configuration
config.Data.serial.portName = comboBoxComPort.Text;
```

VScode 7-12 Excerpt of saving settings

7.5.1.5Testing and error handling

The *DisplayConfigData* is just a visual representation of the *ConfigData* class, and should be tested to work in the same way. There is however several conversions that should be tested and the testing will be done in the same manner. First the old configuration file is deleted, then the DisplayConfigData form is run and new sensor information is added. This can be seen in Figure 7-3.



Sensors

	Sensor Number	MAC/Adress/name	I/O	Type	Location	Measureand	Range	Uncertanty	Battery install date	MISC
▸		0FFF	00	PT1000	Living Room	Temperature	-50-100	0.02%	12/02-2013	This is a test
*										

Save Configration

Saving faulty sensor information will result in loss of data

Proceed?

Timer settings

Save File timer [ms]: 1000

Serial Link Settings

COM port	COM5
Baud rate	38400
Parity	None
DataBits	8
Stop Bits	One
Handshake	None
RTS enable	False

Figure 7-3 Saving configuration

Which results in the following config.xml file seen in XMLscript 2.

```xml
<?xml version="1.0"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Sensors>
    <Sensor>
      <Macaddr>0FFF</Macaddr>
      <IO>00</IO>
      <Type>PT1000</Type>
      <Location>Living Room</Location>
      <Measureand>Temperature</Measureand>
      <Range>-50-100</Range>
      <Uncertanty>0.02%</Uncertanty>
      <BatteryDate>12/02-2013</BatteryDate>
      <MISC>This is a test</MISC>
    </Sensor>
  </Sensors>
  <timer>
    <SaveTimer>1000</SaveTimer>
  </timer>
  <serial>
    <portName>COM5</portName>
    <baud>38400</baud>
    <parity>None</parity>
    <handshake>None</handshake>
    <databits>8</databits>
    <stopbits>One</stopbits>
    <rtsenable>false</rtsenable>
  </serial>
</Config>
```

XMLscript 2 new set information

This means that starting the program with an empty configuration file is working properly and the configuration parameters are saved correctly. In addition several other tests were performed with missing data and or changed data. These tests resulted in some extra security being added to the use case:

The combo box settings of the serial configuration are set as read only so the user only has the ability to select between the values available from the *system* namespaces.

The sample time text box should be set to only give the user the availability of entering integers between 1000 and 100000. The minimum length was set to be sure that the save file timer was not set at a 0ms interval making the program stall. The maximum length is just set to one hour in order to prevent data loss. In order to only allow integers the following code was added to a new *keypressed* , as seen in the VScode 7-1.

```
if (char.IsNumber(e.KeyChar) != true) e.Handled = true;
```

VScode 7-13 Only allow numbers

The maximum length was set as 7 digits[10] in the form design text box parameters; the minimum length is checked by the *textbox.length* property when saving. When it comes to the sensor information there is only one restriction. The length of the IO address should be 2 digits since the IO ports of the ZigBee devices are noted using two digits [15].

7.5.1.6The DisplaySerialdata Code

The display serial data form has three main methods, and two timer ticks. The methods created are the *DisplaySerialDataGetConfiguration* responsible for retrieving the configuration from the xml file, the *DisplaySerialDataLogStart* method responsible for starting the logging to file, and the *DisplaySerialDataLogStop* responsible for stopping the logging to file. The two timer ticks are the serialport timer responsible for reading the current bits available on the serial port, and the save file timer which is responsible for sending the data to the parser, displaying and saving the information.

***DisplaySerialDataGetConfiguration* method**
This method should be run when the start log button is pressed, getting the current configuration from the user. If there are errors in the configuration the saving is stopped. An excerpt of the method can be seen in VScode 7-14

```
SerialPort.PortName = config.Data.serial.portName;
                    SerialPort.BaudRate = config.Data.serial.baud;
                    SerialPort.DataBits = config.Data.serial.databits;
                    SerialPort.Parity = config.Data.serial.parity;
                    SerialPort.StopBits = config.Data.serial.stopbits;
                    SerialPort.Handshake = config.Data.serial.handshake;
                    SerialPort.RtsEnable = config.Data.serial.rtsenable;

                    SaveFileTimer.Interval = config.Data.timer.SaveTimer;
```

VScode 7-14 excerpt of Display serial data get configuration method.

The *DisplaySerialDataLogStart* method
This method should be run when the start logging button is pressed to prompt the user for a filename and location to save the log file. If the user does not select a file the start logging should be aborted, this is done by using the save file dialog ant the returned *DialogResult.OK* parameter. The configuration button is disabled during logging and the save log button is changed to a stop log button. An excerpt of the method can be seen in VScode 7-15

[10] 9999999ms = 2.8hours

```
  if (saveFileDialogParsedData.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                ParsedDataFilename = saveFileDialogParsedData.FileName;
                SerialTimer.Start();
                SaveFileTimer.Start();
                ButtonStartLog.Text = "Stop Log";
                ButtonDisplayConfiguration.Enabled = false;
                ButtonStartLog.Image = Resources.gateway_cross;
            }
```

VScode 7-15 of display serial data log start method

DisplaySerialDataLogStop

The display serial data log stop method is created to give the user the ability to stop saving and set new configuration parameters without exiting the program. The method closes the serial port and stops the timers. An excerpt of the method can be seen in VScode 7-16

```
MessageBox.Show("Data logging data stopped");
                    ButtonStartLog.Text = "Start Logging";
                    ButtonStartLog.Image = Resources.gateway_down;
                    firstTime = true;

                    SerialPort.Close();
                    SerialTimer.Stop();
                    SaveFileTimer.Stop();

                    ButtonDisplayConfiguration.Enabled = true;
```

VScode 7-16 Excerpt of the stop log method

In addition to this methods the code contained in the timer ticks contains the main functionality of this use case. The Serial timer use the above mentioned *serialport. ReadExisting* method and the return is passed to a text box containing the raw data. The save file timer make use of the LOG use case, and the raw text data.

7.5.1.7The LOG Code

The *splitmessage* algorithm

This algorithm is stable and should always return a complete message. The algorithm is reading to the end of the message by using the *IndexOF* method that returns the index of the selected character, >. Next the algorithm checks if the index of the start message is 0. If so the algorithm knows that a complete message is found. The Substring method reads the data between the start and end indexes and add it to a messages array. Then this part of the original message is removed, thus setting up the algorithm for splitting up the next message. If an end message sign is found but no start message sign, the message up to the end sign is just removed since this would indicate an incomplete message. This can be seen in VScode 7-17

```
while ((endpos = message.IndexOf('>')) != -1)
                    {
                        if (message.IndexOf('<') == 0)
                        {
                            //check for message length
                            value = message.Substring(1, endpos - 1);
                            Array.Resize(ref messages, messages.Length + 1);
```

```
                          messages[messages.Length - 1] = value;
                       }
                       message = message.Substring(endpos + 1);
                   }
               }
```

VScode 7-17 Splitting up the string message

The *LogParse* method

The log parse method takes the message array returned from the split algorithm and use the *foreach* loop counting through all the sensors in the configuration. The address length to the current sensor being checked is used to get the address of the sensor. Another *foreach* loop runs through all the messages in the array returned from the message splitter checking if the sensor address and the senor IO are found in the message array. If found the value is added to the logline with a semicolon as a separator. Several messages from the same sensor will as mentioned just be overwritten by the newest value, and the date and time is added to each new line. The NaN value is set to any sensors in the configuration that does not have any messages. An excerpt of the *LogParse* method can be seen in VScode 7-18

```
logline = DateTime.Now.ToString() + ";";
    foreach (Sensor sensor in sensors)
     {
       value = "NaN";
       addresslength = sensor.Macaddr.Length;
       foreach (string msg in message)
        {
          mac = msg.Substring(0, addresslength);
          IO = msg.Substring(addresslength, 2);
           if (mac == sensor.Macaddr && IO == sensor.IO)
           {
             value = msg.Substring(addresslength + 2);
           }
         } logline += value + ";";                    }
```

VScode 7-18 Parsing the data

The *logsave* method

The log save method is a straitgh forward file saving methos using the stremwriter function. An excerpt can be seen in VScode 7-19

```
using (System.IO.StreamWriter file = new System.IO.StreamWriter(@FileName, true))
                {
                    file.WriteLine(data);
                }
```

VScode 7-19 Saving the data

7.5.1.8Log Results

The next step is to test the entire code with the new use case. The new added code was tested by sending several known raw data messages, and studying the results. A typical message can be as follows

<0AAA00805><0CCC00706><0BBB00804>

Config date sets sensor 1 as 0AAA 00, sensor 2 as 0BBB 00, and sensor 3 as 0CCC 00

The raw data sent resulted in the following line added to the log file, correctly parsed

09.02.2013 22:32:21;805;804;706;

7.5.1.9LOG testing and error handling

The split message algorithm

The split message algorithm was tested with empty data , faulty data, no start sign and no end sign and everything was working as it should, returning only valid messages. One problem was found when the message was not anything, *null*, a null reference exception was thrown. This was taken care of by adding a try and catch statements around the parser, saving any eventual errors to a error log file with the time, date, type and location of the error. The message should however be set to an empty string in the main program at start up to avoid

The *logparse* method

When testing the *logparse* method an exception was thrown if error if the IO is not set in the code. This should be tested for in the configuration save and the following code is added to the Save Configuraiton method seen in VScode 7-20.

```
for (int j = 0; j < ConfigurationSensorTable.RowCount - 1; j++)
    {
      if ((string)ConfigurationSensorTable[2, j].Value == null)
      {
          IOmissing = true;
      }
    }
```

VScode 7-20 Added code to save configuration method

Setting a Boolean to true if there is not set a IO value and adding a if IOmissing true then break to the save configuration switch case.

During the testing of this method no errors occurred, but the StreamWriter method used has several exceptions including that if the filename is used by another program at saving time. In accordance with that the gateway should never stall a try and catch statement was also set around this method.

7.5.1.10 Space required for log file saving

The gateway will be running 24/7 so it is a good necessary to see what HD space is needed for saving the log files over several years. This calculation will be done with a large system in order to use the worst case scenario. There will be 50 sensors logging every minute for one year. Using the standard text file one character is the same as one byte, 8bit. Each sensor will have the maximum of 5 characters in parsed data mode. 4 will be the largest data value from the ADC, and one ; is used to separate the messages. In addition there is used 21characters for the date and time each minute message.

$$50sensors * 5bytes + 21byte = 271bytes\ pr\ minute \tag{7-3}$$

$$\frac{271bytes}{minute} * 60minutes * 24hours * 365days = 1.4GBpr\ year \tag{7-4}$$

1.4GB means that a typical 140GB HD will last for about 10 years using the worst case scenario. This means that disk space should not be an issue; a larger problem would be handling the 140GB text file and care should be used to split up the log data for instance each year, or when the ZigBee devices has a battery change. For this reason the code was changed in order to set the current year as part of the file name, such that one file will only contain the data for one specific year.

7.5.2 Predictor methods and algorithms

7.5.2.1Coding the configuration

The configuration code is based on the XML code from the gateway and the same principles are applied to the data grid view table as in section [7.5.1.1] and [2.3.3]. The code is for this reason not commented more on.

7.5.2.2Testing and error handling

The text boxes are only allowed to have one decimal sign and numbered keys in the input; this is done by restricting the key down event to these parameters. Since there is many text boxes there is also important to get the information of the current text box in focus. This can be seen in VScode 7-21.

```
if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar) && e.KeyChar != ',')
                e.Handled = true;

            //get the current name of the text box in focus
            TextBox txb = (TextBox)sender;

            // only allow one decimal point
            if (e.KeyChar == ',' && txb.Text.IndexOf(',') > -1)
                e.Handled = true;
        }
```

VScode 7-21 controlling the text input

One important note to this excerpt is that it is set to the Norwegian standard signs for comma and will need to be changed in order to function with other separating signs.

In addition all save parameters are made using a try and catch clause saving any error to a log file in the same principles as [7.5.1.3]. For more information on the code see APPENDIX.

7.5.2.3Coding the get sensor values

There are two main functions in the Read sensor values use case the *Filter.Cs* and the *SensorVal.Cs*, The *SensorVal.Cs* class works by reading the saved sensor information from the gateway from the last read line as specified. This class also converts the ADC sensor values from the gateway to temperature. The sensor values are returned as a List of doubles, and the time stamps are returned as a date time object. This can be seen in the Excerpt in VScode 7-22.

```
while ((data = sr.ReadLine()) != null)
                    {
                        if (i > LastLine)
                        {
                            LastLine = i;
                            StringArray = data.Split(';');

//time is returned as out in order to have the correct time stamps

                            time = Convert.ToDateTime(StringArray[0]);
//Convert the Values to Temperature values, -1 to remove date time stamp, and add
to list
                            for (int l = 1; l < StringArray.Length-1; l++)
                            {
                                ConvData =
Math.Round(((Convert.ToDouble(StringArray[l]) * 3 / 2048) - 0.5) * 100,2);
                                //filter the data through a low pass filter
                                ConvFiltData = filter.LowPassFilter(ConvData, l-1);
                                listSensVal.Add(ConvFiltData);
                            }
                            break;
                        }
                       i++;
                   }
```

VScode 7-22 Reading only the last line.

The low pass filter filtering the values can be seen in excerpt in

```
public double LowPassFilter(double SensorVal, int SensorNr)
{
            a = Ts / (Ts + Tf);
            yFiltered = (1 - a) * yk[SensorNr] + a * SensorVal;

            yk[SensorNr] = yFiltered;
            return yFiltered;
}
```

VScode 7-23 Low pass filter excerpt

7.5.2.4Coding the predictor use case

The predictor use case is made up of three main methods, creating and discretizing the State Space Matrices (SSM), estimating the disturbance state using the Kalman filter algorithm and the least squares algorithm. The state space matrices are made directly from the derivation of the model in [3.2.3], and the discretization of this model there is used a Zero Order Hold method. The function for discretizing the matrices based on the sample time from h, given from the gateway configuration file can be seen in VScode 7-24.

```
//Create the S matrix for discretization
Matrix S = (IA * h + (A * h * h) / (1 * 2) + (A * A * h * h * h) / (1 * 2 * 3) + (A * A * A
* h * h * h * h) / (1 * 2 * 3 * 4) + (A * A * A * A * h * h * h * h * h) / (1 * 2 * 3 * 4 *
5) + (A * A * A * A * A * h * h * h * h * h * h) / (1 * 2 * 3 * 4 * 5 * 6));

            // Calculate the discrete time matrices based on zero order hold
            Ad = IA + A * S;
            Bd = S * B;
```

VScode 7-24 discretizing the state space model

The Kalman filter algorithm is made using the Matrix class and the Kalman filter follows the algorithm defined in [3.2.3.1], the Kalman filter algorithm is seen in VScode 7-25.

```
            Matrix xhat = Ad * x + Bd * u;
            Matrix I = Matrix.IdentityMatrix(Ad.rows, Ad.cols);
            phat = Ad * phat * Matrix.Transpose(Ad) + Qv;
            K = phat * Matrix.Transpose(D) / (D * phat * Matrix.Transpose(D) + Rw);
            xbar = xhat + K * (Y - (D * xhat)[0, 0]);
            yhat = (D * xbar)[0, 0];
            phat = (I - K * D) * phat;

        }
```

VScode 7-25 *The Kalman Filter algorithm*

The least squares regression matrixes are calculated using the matrix functions and the known solution to the OLS matrix equation [24].

The least squares regression algorithm

```
for (int i = 1; i < dataPoints.Count; i++)
            {
                ti = ti + i;
                ti2 = ti2 + i * i;
                yi = yi + dataPoints[i];
                yiti = yiti + dataPoints[i] * i;
            }

            //Setup the regression Matrices
            Alpha[0, 0] = dataPoints.Count;
            Alpha[0, 1] = ti;
            Alpha[1, 0] = ti;
            Alpha[1, 1] = ti2;
            Beta[0, 0] = yi;
            Beta[0, 1] = yiti;
            Alpha = Alpha.Invert();
            Regression = Alpha*Matrix.Transpose(Beta);
```

System Learn function

The system learn function is based on applying the maximal power output to the heaters and log the data in the predictor until it reaches the set comfort temperature. After the system has been learned the predictor saves the regression models to the config.XML file and the controller takes over using the prediction times and the set comfort interval reference to keep the temperature at comfort level at the correct times. The system learn function can be seen in VScode 7-26.

```
public bool PredictorLearn(double Y, int counter)
        {
            int test = counter;
            if (counter ==1)
            {
                //Only calculate the discrete state space matrixes the first
time
                CalcualteSSM(273, 293);
                CalculateDiscreteSSM();

            }
            //run the Kalman filter
            KalmanFilter(Y, out yhat, out xbar);


            double setpoint =
ControlSystem.config.Data.Monday.ReferenceTemperature;

            if (Y < setpoint && counter>10) //allow some time for system to
stabilize
            {
                //Store the data in a matrix until the comfort temperature is
reached
                Dsaved.Add(xbar[1, 0]); // the disturbance vector
                Ysaved.Add(Y); // the Temperature data
                learn = true;

            }

            //Calculate the Disturbance vector using The least squares method
            if (Y >= setpoint) //need to set the comfort temperature as
reference
            {
                //Read all the current configuration in order to not overwrite
everything with blanks
                ControlSystem.config.ConfigReadData("config.xml");


                Matrix DisturbanceRegression = Matrix.ZeroMatrix(1, 2);
                Matrix yRegression = Matrix.ZeroMatrix(1, 2);

                DisturbanceRegression = LeastSquares(2, Dsaved);    //The
disturbance regression function
                yRegression = LeastSquares(2, Ysaved);              //The
Temperature regression function

                //Store the learned regression line in the XML file as learned
parameter under predictor
                ControlSystem.config.Data.controller.DpredictorAlpha =
DisturbanceRegression[0, 0];
                ControlSystem.config.Data.controller.DpredictorBeta =
DisturbanceRegression[1, 0];
                ControlSystem.config.Data.controller.YpredictorAlpha =
yRegression[0, 0];
                ControlSystem.config.Data.controller.YpredictorBeta =
yRegression[1, 0];

                ControlSystem.config.Data.controller.A11 = Ad[0, 0];
                ControlSystem.config.Data.controller.A12 = Ad[0, 1];
                ControlSystem.config.Data.controller.A21 = Ad[1, 0];
                ControlSystem.config.Data.controller.A22 = Ad[1, 1];

                ControlSystem.config.Data.controller.B11 = Bd[0, 0];
                ControlSystem.config.Data.controller.B21 = Bd[1, 0];

                //Write the new learned data
                ControlSystem.config.ConfigWriteData("config.xml");
                //Return false = Learning finished
               learn = false;
            }
```

VScode 7-26 The Learn function

7.5.2.5Additions to the Matrix library

In order to facilitate the Kalman filter calculations some additions were needed in the Matrix class, these additions are seen in VScode 7-27 through VScode 7-29

```
  private static Matrix Add(Matrix m1, double d)
        {
            Matrix r = new Matrix(m1.rows, m1.cols);
            for (int i = 0; i < r.rows; i++)
                for (int j = 0; j < r.cols; j++)
                    r[i, j] = m1[i, j] + d;
            return r;
        }
```

VScode 7-27 Adding double to a matrix

```
  private static Matrix Multiply(Matrix m, double n)                          //
Multiplication by constant n
        {
            Matrix r = new Matrix(m.rows, m.cols);
            for (int i = 0; i < m.rows; i++)
                for (int j = 0; j < m.cols; j++)
                    r[i, j] = m[i, j] * n;
            return r;
        }
```

VScode 7-28 Multiplying double to Matrix

```
private static Matrix Multiply(double n, Matrix m)                          //
Multiplication by constant n
        {
            Matrix r = new Matrix(m.rows, m.cols);
            for (int i = 0; i < m.rows; i++)
                for (int j = 0; j < m.cols; j++)
                    r[i, j] = m[i, j] * n;
            return r;
        }
```

VScode 7-29 Multiplying Matrix to double

7.5.2.6Testing and error handling

The read sensor values are then tested with the gathered experiment data, the sensor values are plotted to a Chart using the sensor values names in the configuration of the gateway as the different time series. In fig the inside temperatures time series are seen Figure 7-4.

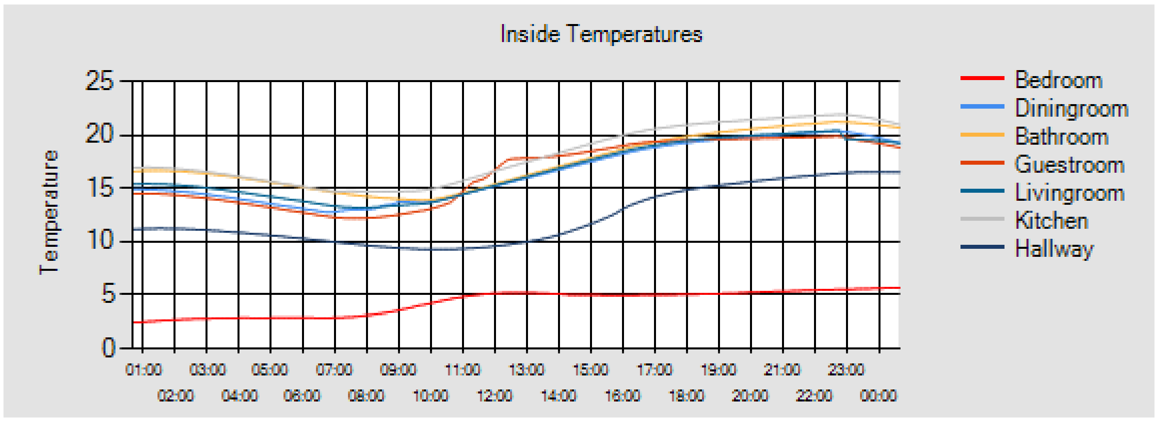


Figure 7-4 Testing the read sensor values use case

All methods that might crash have been set with a try and catch clause in the same manner as in the previous sections [2.3.5.3].

7.5.3 Controller methods and algorithms

7.5.3.1 The main configuration and tab controls

The Tab control is made by using inheritance. All other configuration forms are inherits the ConfigMainForm in the class setup. This can be seen for the house configuraiton in VScode 7-30

```
public partial class ConfigHouse : MainFormPage
```

VScode 7-30 Code excerpt of inheriting the main form

A panel is made to contain each of the configuration forms as seen in the house configuration example in VScode 7-31

```
this.pnl = panelConfigHouse;
```

VScode 7-31 Code for setting the panel of the House configuration form.

A tab control is then created in the main configuration form GUI where the different configuration pages are added as seen in VScode 7-32

```
tabControlConfiguration.TabPages.Add(new TabClass(new ConfigTemperatures()));
       tabControlConfiguration.TabPages.Add(new TabClass(new ConfigSensors()));
       tabControlConfiguration.TabPages.Add(new TabClass(new ConfigModel()));
       tabControlConfiguration.TabPages.Add(new TabClass(new ConfigHouse()));
       tabControlConfiguration.TabPages.Add(new TabClass(new ConfigControl()))
```

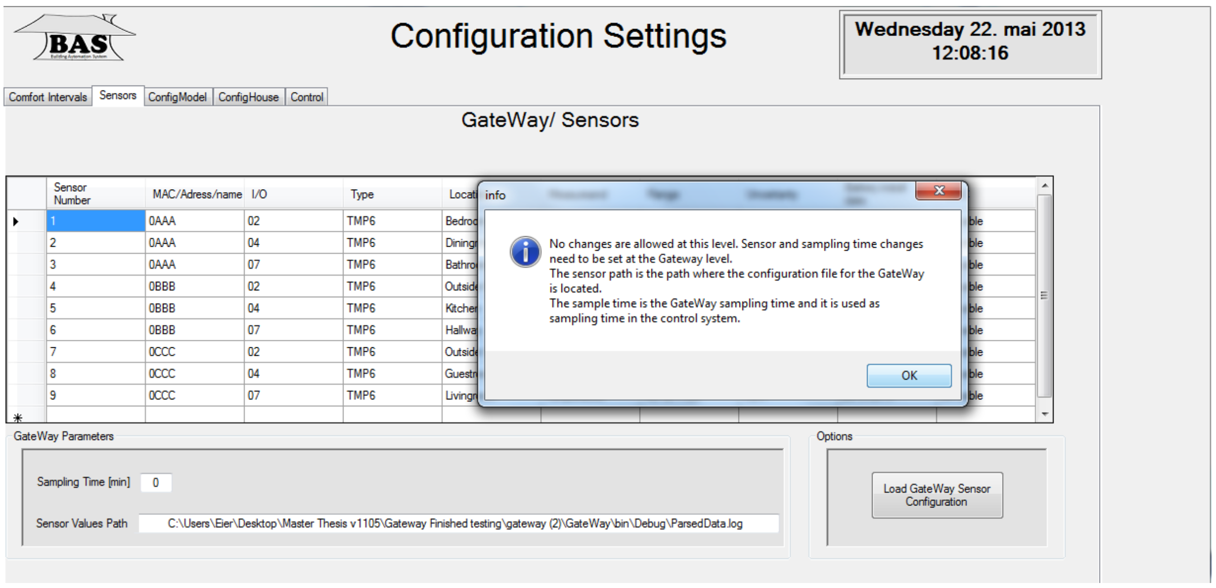
VScode 7-32 Adding all the configuration settings to the tab control in the main config form

The TabClass created to add the selected forms panel content can be seen in VScode 7-33

```
private Form frm;
        public TabClass(MainFormPage frm_content)
        {
            this.frm = frm_content;
            this.Controls.Add(frm_content.pnl);
            this.Text = frm_content.Text;
        }
```

VScode 7-33 Tab class excerpt

7.5.3.2Configuration help examples



Sensor Number	MAC/Adress/name	I/O	Type	Locati
1	0AAA	02	TMP6	Bedroo
2	0AAA	04	TMP6	Diningr
3	0AAA	07	TMP6	Bathro
4	0BBB	02	TMP6	Outside
5	0BBB	04	TMP6	Kitchen
6	0BBB	07	TMP6	Hallwa
7	0CCC	02	TMP6	Outside
8	0CCC	04	TMP6	Guestr
9	0CCC	07	TMP6	Livingr

No changes are allowed at this level. Sensor and sampling time changes need to be set at the Gateway level.
The sensor path is the path where the configuration file for the GateWay is located.
The sample time is the GateWay sampling time and it is used as sampling time in the control system.

Sampling Time [min] 0

Sensor Values Path C:\Users\Eier\Desktop\Master Thesis v1105\Gateway Finished testing\gateway (2)\GateWay\bin\Debug\ParsedData.log

Figure 7-5 Configuration information example

7.5.3.2.1 Configuration testing and error handling

The main parts of the configuration is the Config.XML file which has already been thoroughly tested. The error handling is done in the same manner as saving the date time and place of the error to a log file.

7.5.3.3Controller use case

The controller use case consist of a class for each of the controllers with functions for setting up the controllers in the LQR and MPC cases, and functions for calculating the control output in all cases.

In order to calculate the predicted references there was added a function to the Config.XML class this functions returns a reference vector based on the current date and time using the comfort intervals configuration and a switch case statement this is seen in VScode 7-34.

```
public Matrix GetComfortTemperature(DateTime starttime,int l)
        {
            Matrix temp = Matrix.ZeroMatrix(l, 1);
            DateTime time;
            int hour;
            for (int i = 0; i < l; i++)
            {
                time = starttime + TimeSpan.FromHours(i);
                 hour = time.Hour;

                switch (time.DayOfWeek)
                {
                    case DayOfWeek.Monday:
                        temp[i, 0] =
Convert.ToDouble(Data.Monday.ComfortIntervals[hour]);
```

```
                        break;

                           . . . .
            return temp;
```

VScode 7-34 Create reference vector

7.5.3.3.1 PID class

The D term in the PID controller only contributed noise to the system for this reason only a PI controller has been implemented.

The function for calcualting the PI controllers output can be seen in

```
public double PiController(double y, double r)
        {
            double e; // Error between Reference and Measurement
            double u; // Controller Output
            //PID Algoritm
            e = r - y;
            u = Kp * e + (Kp / Ti) * z;
            z = z + Ts * e;
```

VScode 7-35 Calculate PI controller output

7.5.3.3.2 LQR class

The LQR class is divided into two main functions the controllers setup and calculation of the steady state gains, i.e. solving the Riccati equation and calculating the controllers output

Controller setup

In order to give the controller integral action the eSSM models are needed calculated in a function called calculateESSM in the controller class and this can be seen in code excerpt VScode 7-36

```
//Get the sizes of the matrices for MIMO systems
            int nx = Ad.rows;
            int nu = Bd.cols;
            int ny = D.rows;


//Create the eSSM models
            At = Ad.AddRight(Matrix.ZeroMatrix(nx,
ny)).AddBelow(D.AddRight(Matrix.IdentityMatrix(ny, ny)));
            Bt = Bd.AddBelow(Matrix.ZeroMatrix(ny, nu));
            Dt = D.AddRight(Matrix.IdentityMatrix(ny, ny));
            Qt = Matrix.Transpose(Dt) * q * Dt;
```

VScode 7-36Calculate the eSSM models

The Riccati equation solver loop is run until the error between the new and the previous value is below 0.00000000001 the loop can be seen in VScode 7-37.

```
while (error > 1e-10 && iterations <= 10000)
            {
                f0 = (B_ * p0 * At) / (r + B_ * p0 * Bt);

                p1 = A_ * p0 * At + Qt - (A_ * p0 * Bt) * f0;

                f1 = (B_ * p1 * At) / (r + B_ * p1 * Bt);

                error = Matrix.MaxAbs(f1 - f0);

                p0 = p1;
                iterations++;
            }
```

VScode 7-37 Solving the Riccati equation and calculating the SS gain.

The controller gains are set for the scalar system as seen in VScode 7-38

```
ControlSystem.config.Data.controller.G1 = -f1[0, 0];
ControlSystem.config.Data.controller.G2 = -f1[0, 1];
```

VScode 7-38 Saving the scalar SS LQR gain

Calculate control output

The LQR control output for the scalar system is then simply calculated as seen in VScode 7-39.

```
dx = x - x_old;
u = (u_old + G1 * dx + G2 * (x_old - r));
```

VScode 7-39 Calculating the control output

7.5.3.3.3 MPC class

Since the MPC controller setup is complex the entire function is included with commentary, this is seen in VScode 7-40

Controller setup

```
        public void InitializeMPCControl(int L, double q, double r)
        {

         //Calculate the Extended State Space Matrixes
            essm.CalculateESSM(q_lqr, out At, out Bt, out Dt, out Qlqr);

            //Get the sizes of the eSSM matrices
            int nx = At.rows;
            int nu = Bt.cols;
            int ny = Dt.rows;
            int n = Dt.cols;


            // Calculate observability matrix
            O = Matrix.Parse(Dt.ToString());
            Matrix w = Matrix.Parse(Dt.ToString());

            for (int i = 2; i <= L; i++)
            {
                w *= At;
                O = O.AddBelow(w);
            }

            // Calculate extended observability matrix
            Matrix OB = O * Bt;

            // Calculate the lower block triangular Toeplitz matrix
            Matrix Ht = Matrix.ZeroMatrix(OB.rows,1);

            for (int i = 1; i < Ht.rows; i++)
            {
                Ht[i, 0] = OB[i-1,0];
            }
            Matrix HdL = Matrix.Parse(Ht.ToString());
            for (int i = 1; i < (L - 1); i++)
            {
                Matrix temp = Matrix.ZeroMatrix(Ht.rows, Ht.cols);
                for (int rows = i; rows < L; rows++)
                    for (int cols = 0; cols < Ht.cols; cols++)
                    {
                        temp[rows, cols] = Ht[rows - i, cols];
                    }
                HdL = HdL.AddRight(temp);

            }

            //Calculat the prediction model parameters
            F_L = Matrix.Parse(OB.ToString());
            F_L =F_L.AddRight(HdL);

            //Create the weighting matrixes as identity matrices
            Qt = Matrix.IdentityMatrix(L, L);
            Rt = Matrix.IdentityMatrix(L, L);

            //Multipling by weighting factors to create the weighting matrices
            Qt = Qt * q;
            Rt = Rt * r;

            //Create the Hessian Matrix
            H = Matrix.Transpose(F_L) * Qt * F_L+Rt;
```
VScode 7-40 Set up MPC controller matrices

The hessian matrix and the set up eSSM matrices is used as public values within the MPC class to avoid many out parameters in the methods.

Calculate control output

The MPC control output is calculated using the reference from the get air heater reference function contained in the Config.XML. Excerpts of the control output function is seen in VScode 7-41.

```
        //Create the state deviation parameter
        xt = Matrix.ZeroMatrix(2,1);
        xt[0, 0] = x - x_old;
        xt[1, 0] = x_old;


        //Create the future reference vectors
        Matrix pl = O * At * xt;

        //Get reference from config using the correct prediction horizon
        Matrix r1l = ControlSystem.config.GetAirHeaterReference(startReference, L);

        //Calculate future outputs
        Matrix f = Matrix.Transpose(F_L) * Qt * (pl - r1l);
        duf = -H.Invert()*f;
        //Only use 1.st output as control output
        u = u_old + duf[0, 0];
```

VScode 7-41 Calculate the MPC controller output

7.5.3.3.4 PID class

The PID controller needs no setup and is simply calculated as seen in

```
public double PiController(double y, double r)
        {
            double e; // Error between Reference and Measurement
            double u; // Controller Output
            //PID Algoritm
            e = r - y;
            u = Kp * e + (Kp / Ti) * z;
            z = z + Ts * e;
```

VScode 7-42 Calculate PID controller output

7.5.3.4 Control system GUI

The control system GUI is mainly graphical user interface and not much is needed discussed or explained on the code. The two functions creating the plots and time series is however worth a mention. The plot creation function can be seen in code excerpt in VScode 7-43

```
            foreach (Sensor sensor in config.Data.Sensors)
            {
                //Set up the two main graph series
                var seriesOutside = new Series();
                var seriesInside = new Series();

                //Add outside sensors to outside plot
                if (sensor.Location.Contains("Outside"))
                {
                    seriesOutside.Name = sensor.Location;
                    seriesOutside.ChartType = SeriesChartType.FastLine;
                    seriesOutside.XValueType = ChartValueType.Time;
                    chartOutsideTemperature.Series.Add(seriesOutside);

                }
                else
                //Add inside sensors to inside plot
                    if (!sensor.Location.Contains("Outside"))
                    {
                        seriesInside.Name = sensor.Location;
                        seriesInside.ChartType = SeriesChartType.FastLine;
                        seriesInside.XValueType = ChartValueType.Time;
                        chartInsideTemperature.Series.Add(seriesInside);
                    }
            //Set up axis
            chartOutsideTemperature.ChartAreas[0].AxisX.Title = "Time";
            chartInsideTemperature.ChartAreas[0].AxisY.Title = "Temperature";
            chartInsideTemperature.ChartAreas[0].AxisX.Title = "Time";
            chartInsideTemperature.ChartAreas[0].AxisY.Title = "Temperature";
```

VScode 7-43 Setting up graphs with correct sensory information

The plots were createt to display one day at a time, this is seen in VScode 7-44

```
////Kepp X axis displaying last 24 hours
            this.chartInsideTemperature.ChartAreas[0].AxisX.Minimum = (sensorTime -
TimeSpan.FromHours(24)).ToOADate();
            this.chartInsideTemperature.ChartAreas[0].AxisX.Maximum =
sensorTime.ToOADate();
```

VScode 7-44 Creating moving plots

7.5.3.4.1 Additions to the MATRIX class

The four additions needed in the matrix class is seen in functions from VScode 7-45 through VScode 7-48

```
public Matrix AddRight(Matrix m2)
        {
            if (rows != m2.rows)
            {
                throw new MException("Different rows!");
            }
            Matrix r = new Matrix(rows, cols + m2.cols);
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    r[i, j] = mat[i, j];
            for (int i = 0; i < m2.rows; i++)
                for (int j = 0; j < m2.cols; j++)
                    r[i, j + cols] = m2[i, j];
            return r;
```

VScode 7-45 Add one matrix to the right of another matrix

```
        public Matrix AddBelow(Matrix m2)
        {
            if (cols != m2.cols)
            {
                throw new MException("Different cols!");
            }
            Matrix r = new Matrix(rows+m2.rows, cols);
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    r[i, j] = mat[i, j];
            for (int i = 0; i < m2.rows; i++)
                for (int j = 0; j < m2.cols; j++)
                    r[i + rows, j] = m2[i, j];
            return r;
        }
```

VScode 7-46 Add one matrix below another matrix

```
        public Matrix SubMatrix(int rowfrom, int rows, int colfrom, int cols)
        {
            Matrix m = new Matrix(rows,cols);
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    m[i, j] = mat[rowfrom + i, colfrom + j];
            return m;
        }
```

VScode 7-47 Create a sub matrix from a matrix

```
public static double MaxAbs(Matrix m1)
        {
            double max = 0;
            for (int i = 0; i < m1.rows; i++)
                for (int j = 0; j < m1.cols; j++)
                {
                    if (Math.Abs(m1[i,j]) > max)
                        max = Math.Abs(m1[i,j]);
                }
            return max;
        }
```

VScode 7-48 Calculating the Absolute difference between two matrices

7.5.3.5 Testing and error handling

Testing the complete system is done in the Experiments part in paragraph [4.7]. Error handling is done in the same manner as in the gateway system paragraph [2.3.5.3]

7.6 Appendix 6: MATLAB scripts

7.6.1 The data processing script

```
clear all;
close all;
%-----------------SETTINGS-------------------------------------------------
%-------------------Set which data to plot
DorawPlot=true;         %true will plott the raw data
DoavgPlot=true;         %true will plot the average data
%DoavgPlot=false;
DoOutlierPlot=true;     %true will plot the outlier data
%DoOutlierPlot=false;
%-------------------Set interval for date time stamps on X axis
Xstamps=15;
%-------------------Set the Approved Standard deviation for outliers
StdFactor=2.0;

%----------import the data-------------------------------------------------
A = importdata('C:\Users\Stian skole\Desktop\Master Thesis v0904\Experiments\LOG
files\oneday.log')
%A = importdata('C:\Users\Stian skole\Desktop\Master Thesis v0904\Experiments\LOG
files\ParsedData2803-1826-all-power-off.log')
%A = importdata('C:\Users\Stian skole\Desktop\Master Thesis v0904\Experiments\LOG
files\ParsedData2903-1944-ss-ventsclosed.log')
data=A.data;
rowheaders=A.rowheaders;
interval=length(data)/Xstamps;

%------Create Titles and Axis values based on the rowheader---------------
%convert date time string into num vectors
[year,month,day,hour,minute,~] = datevec(rowheaders, 'dd.mm.yyyy HH:MM:SS')
Y=year(1);
m=month(1);
%get Month name from month name function
M=Monthname(m);
D=day(1);
%create the title based on the row headers data
TitleString = sprintf('Temperature readings started at %d %s %d',D,M,Y);
%remove year and seconds for beter plotting
Time=(datestr(rowheaders,'HH:MM'))
%if there is much data the month i also plotted on the X axis
if length(rowheaders)>470
    for k=1:length(rowheaders)
        D=day(k)
        H=hour(k)
        C=sprintf('D %d H %d',D,H)
        Time(k,1:length(C))=C;
        interval=10;
    end
end
%------create the x axis values based on the Time data---------------------
XaxisValues=Time(1:interval:length(Time),:);
count=length(XaxisValues)
XtickValues=1:length(data)/count:length(data);
%-----------Convert ADC data to temperature data--------------------------
VoltData=(data*(3/2048));
TempData=((VoltData-0.5)*100);
%----------Remove the Outliers---------------------------------------------
dataRemovedOutliers=OutlierRemover(TempData,StdFactor,DoOutlierPlot, XaxisValues,XtickValues,
TitleString);
%-----------Run the NaNremoval function.----------------------------------
[RemovedRows,NaNs,dataRemovedNaNs]=NaNremove(dataRemovedOutliers);

%----Smooth the data thorugh a low pass filter-----------------------------
dataFiltered = LPfilt(dataRemovedNaNs);

%----Split up TempData in inside and outside temperatures------------------
TempData=dataFiltered;
insideTemperatures=[TempData(:,1),TempData(:,2),TempData(:,3),TempData(:,5),TempData(:,6),Temp
Data(:,8),TempData(:,9)];
outsideTemperatures=[TempData(:,4),TempData(:,7)];

%-----------Plot the New TempData-----------------------------------------
figure('units','normalized','position',[.1 .1 .4 .4])
TitleProcessedPlot = sprintf('Processed data - %s',TitleString);
subplot(2,1,1)
plot(insideTemperatures)
%set date time on X axis
xmin=0;
xmax=length(data)
xlim([xmin xmax])
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)
set(gcf,'color','w')
%set(gcf, 'Position', [100 100 150 150])
```

```
title(TitleProcessedPlot)
legend('Bedroom','Diningroom','Bathroom','Guestroom','Livingroom','Kitchen','Hallway','EastOut
side','Location','EastOutside')
ylabel('Temperature [^oC]')
xlabel('Time');
subplot(2,1,2)
plot(outsideTemperatures)
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)
%set(gcf, 'Position', [100 100 150 150])
xlim([xmin xmax])
legend('Outside N', 'Outside S','Location','EastOutside')
ylabel('Temperature [^oC]')
xlabel('Time');

%-----plot Raw data---------------------------------------------------------
if DorawPlot
insideRawData=[data(:,1),data(:,2),data(:,3),data(:,5),data(:,6),data(:,8),data(:,9)]
outsideRawData=[data(:,4),data(:,7)]
TitleRawPlot = sprintf('Raw data - %s',TitleString);

figure('units','normalized','position',[.1 .1 .4 .4])

subplot(2,1,1)
plot(insideRawData)
%set date time on X axis
xmin=0;
xmax=length(data)
xlim([xmin xmax])
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)
set(gcf,'color','w')

title(TitleRawPlot)
legend('Bedroom','Diningroom','Bathroom','Guestroom','Livingroom','Kitchen','Hallway','EastOut
side','Location','EastOutside')
ylabel('ZigBee ADC values')
xlabel('Time');
subplot(2,1,2)
plot(outsideRawData)
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)%set(gcf, 'Position', [100 100 150 150])
xlim([xmin xmax])
legend('Outside N', 'Outside S','Location','EastOutside')
ylabel('ZigBee ADC values')
xlabel('Time');
end



%---------Plot averaged inside and outside data----------------------------
if DoavgPlot
insideAveragedTemperatures=mean(insideTemperatures,2);
outsideAveragedTemperatures=mean(outsideTemperatures,2);
TitleAvgPlot = sprintf('Averaged temperatures - %s',TitleString);

figure('units','normalized','position',[.1 .1 .4 .4])
subplot(2,1,1)
plot(insideAveragedTemperatures)
%set date time on X axis
xmin=0;
xmax=length(data)
xlim([xmin xmax])
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)
set(gcf,'color','w')

title(TitleAvgPlot)
legend('Inside Temperatures')
ylabel('Temperature [^oC]')
xlabel('Time');
subplot(2,1,2)
plot(outsideAveragedTemperatures)
set(gca,'XTick',XtickValues)
set(gca,'XTickLabel',XaxisValues)%set(gcf, 'Position', [100 100 150 150])
xlim([xmin xmax])
legend('Outside Temperatures')
ylabel('Temperature [^oC]')
xlabel('Time');
title('Averaged outside temperatures')

end
```

7.6.2 NaN removal function

```
function [ RemovedRows, NaNs, NonNanData] = NaNremove(data)
%NaN removal and interpolation of data
%   Data to be interpolated and NaN removed is input parameters
%   The outpout parameters is the "filtered data"

%first check if the first value is NaN if so remove until first not nan in
%all rows in order to be able to do interpolation

data;
%-------------Check forn NaNs in the first rows----------------------------
j=0;
if (find(isnan(data(1,:)))>0);
    j=1;
    while (find(isnan(data(j,:)))>0 & j<length(data(:,1)));
      j=j+1;
    end
    %remove all first rows with NaN data
    data=data(j:length(data(:,1)),:);
end

%-----------------Check for interpolatable NaNs------------------------------
if length(data(:,1))>2 %if not all values are removed

    for i=1:length(data(1,:))
    Non=data(:,i);
    NonNan(:,i)=interp1(find(~isnan(Non)),Non(~isnan(Non)),1:length(Non))';
    NaNs=length(find(isnan(data)));
    end

%----------------Check for NaNs at the end---------------------------------
  if (find(isnan(NonNan)) > 0 )

      [row,col,vals]=find(isnan(NonNan));
       EndNaNsRemoved=length(vals);
       row
        %Remove end rows with NaNs by using the lowest value left with NaNs
       NonNanData=NonNan(1:min(row)-1,:);



  %-------Print outputs------------------------------------------------------

  else

     NonNanData=NonNan;
 end
 RemovedRows=length(data)-length(NonNanData)+j;
else
    RemovedRows=length(data(:,1))+j;
    NonNanData=0;
    NaNs=0;
    disp('All data has been removed due to NaNs')
end


end
```

7.6.2.1Testing the NaN removal function

```
>> [RemovedRows,NaNs,NonNaNdata]=NaNremove(dataTest)

data =

   438    NaN    483    341    469    474    358    480    490
   438    475    483    341    469    474    358    480    490
   437    477    NaN    343    469    472    353    481    NaN
   439    478    NaN    335    472    477    355    483    491
   438    479    487    339    472    478    350    483    492
   438    479    489    342    NaN    479    351    485    492

RemovedRows = 2

NaNs = 4

NonNaNdata =

  438.0000  475.0000  483.0000  341.0000  469.0000  474.0000  358.0000  480.0000  490.0000
  437.0000  477.0000  484.3333  343.0000  469.0000  472.0000  353.0000  481.0000  490.5000
  439.0000  478.0000  485.6667  335.0000  472.0000  477.0000  355.0000  483.0000  491.0000
  438.0000  479.0000  487.0000  339.0000  472.0000  478.0000  350.0000  483.0000  492.0000
```

7.6.3 The Outlier Removal function

```
function [ dataRemovedOutliers ] = OutlierRemover(data, Factor,DoPlot, XaxisValues,
XtickValues, TitleString)
%This function finds and removes outliers based on the Factor value which
%is a factor of the standard deviation of the data set
%inputs data, StdFactos and Boolean value plot
Nydata=data;

%Create a matrix of mean values by
mu = mean(Nydata);
sigma = std(Nydata);
[n,p] = size(Nydata);
% Create a matrix of mean values by
% replicating the mu vector for n rows
MeanMat = repmat(mu,n,1);
% replicating the sigma vector for n rows
SigmaMat = repmat(sigma,n,1);
% Create a matrix of zeros and ones, where ones indicate
% the location of outliers
outliers = abs(Nydata - MeanMat) > Factor*SigmaMat

% Calculate the number of outliers
nout = sum(sum(outliers))
%Mark the outliers


for i=1:length(data(1,:))
i
    potential_outlier=outliers(:,i);
    X=1:length(Nydata(:,i));
    Y=Nydata(:,i);
  if DoPlot
    figure('units','normalized','position',[.1 .1 .4 .4])
    subplot(2,1,1)
    plot(Y,'b')
    xmin=0;
    xmax=length(data)
    xlim([xmin xmax])
    xlabel('Time [hours]')
    ylabel('Temperture (^{\circ}C)')
    set(gca,'XTick',XtickValues)
    set(gca,'XTickLabel',XaxisValues)
    set(gcf,'color','w')
    title(TitleString)

    hold
    scatter(X(potential_outlier),Y(potential_outlier), '*r','LineWidth',5)
    if i==1
        legend('Bedroom','Outliers');
    elseif i==2
        legend('Diningroom','Outliers');
        elseif i==3
        legend('Bathroom','Outliers');
    elseif i==4
        legend('Outside N','Outliers');
    elseif i==5
        legend('Guestroom','Outliers');
    elseif i==6
        legend('Livingroom','Outliers');
    elseif i==7
        legend('Outside S','Outliers');
    elseif i==8
        legend('Kitchen','Outliers');
    elseif i==9
        legend('Hallway','Outliers');


    hold
    end
  end

%interpolate the the outliers
    InterP(:,i)=interp1(find(~potential_outlier),Y(~potential_outlier),X)'

    if DoPlot
    subplot(2,1,2)
    xmax=length(data)
    xlim([xmin xmax])
    scatter(X(potential_outlier),Y(potential_outlier), '*r','LineWidth',5)
    set(gca,'XTick',XtickValues)
    set(gca,'XTickLabel',XaxisValues)
    hold
    plot(InterP(:,i),'b')
    xlim([xmin xmax])
    xlabel('Time')
    ylabel('Temperture (^{\circ}C)')
    title('Temperature data with marked removed outliers')
    set(gca,'XTick',XtickValues)
    set(gca,'XTickLabel',XaxisValues)
    if sum(potential_outlier)>0
    legend('Removed Outliers')
    end
    hold
    end
end
    dataRemovedOutliers=InterP;
end
```

7.6.4 The LP filter function

```
function [ y] = LPfilt( data )
%Smoothing function
%   Low pass filter

t=zeros(length(data),1);
for i=2:length(t)
    t(i)=t(i-1)+1;
end

ToS=zeros(length(data),9);
Ts=t(2)-t(1);
Tf=5*Ts;
a=Ts/(Ts+Tf);
OldToS=data(1,:);
ToS(1,:)=data(1,:);

for l=1:9
    for i=2:length(ToS)
    ToS(i,l)=(1-a)*OldToS(1,l)+a*data(i,l);
    OldToS(1,l)=ToS(i,l);
    end
end
y=ToS;
end
```

7.6.5 The month to name month function

```
function [ MonthName ] = MonthName( m )
%UNTITLED13 Summary of this function goes here
%   Detailed explanation goes here
for i=1:1
switch m
case 1
m='January';
break;
case 2
m='Febuary';
break;
case 3
m='March';
break;
case 4
m='April'
break;
case 5
m='May'
break;
case 6
m='June';
break;
case 7
m='July';
case 8
m='August';
break;
case 9
m='September';
break;
case 10
m='Oktober';
case 11
m='November'
break;
case 12
m='december';
break;
end
end
MonthName=m;
end
```

7.6.6 The control simulation function

```
% Simulating all controls with prediction horizon

clear all
%-----Continous State Space Model-----------------------------------------
Ac=-8.81*10^-4;
Bc=0.439246148*10^-5;
Dc=1;

%-----Discrete time model---------------------------------------------------
h=60; %sampling time of 6 minute
[Ad,Bd,Dd]=c2dm(Ac,Bc,Dc,zeros(1),h,'zoh');


% Simulation time horizon
dt=0.001; t=0:dt:24; t=t'; N=length(t);

%--------Optimal LQ parameters---------------------------------------------

q=1;
Rdu=0.001;
[G1,G2,At2,Bt2,Dt2]=dlqdu_pi(Ad,Bd,Dd,q,Rdu);  % MPC with infinite horizon

%--------PID parameters---------------------------------------------------
Kp=805;
Ti=10000;
Td=0;

%-------MPC--Prediction model matrices-------------------------------------
L=10; %Prediction Horizon
Q=10; R=0.001; %Weighting parameters

nx=size(Ad,1); nu=size(Bd,2); ny=size(Dd,1);
[HdL,OL,OLB]=ss2h(At2,Bt2,Dt2,zeros(ny,nu),L,0);
F_L=[OLB HdL];
Qt=q2qt(Q,L);
Rt=q2qt(R,L);

H=F_L'*Qt*F_L + Rt;
%--------------------------------------------------------------------------

%Steady state nominal values
us=258.7252;
xs=292.44; %
v=258.7;
r=292.44;

x_old=x;
y_old=ys;
u_old=us;
e_old=0;
z=xs(1);

%get refernce vector from comfort interval function
[WdRef,WeRef,t]=ComfortRef();
MPCref=WeRef';
N=length(t)

% Type of controller:1 PID controller, 2-(QP optimal controller),3 MPC
Cntrl=3;

L1=0;
for k=1:N-200

    %step change in disturbance outside temperature
    if k==1900
        v=250;
    end
    r=WeRef(k+L1);
    y=Dd*x;
    if Cntrl==1
    %PID control on velocity form
    e=r-y;
    g0=Kp;
    g1=-Kp*(1-(h/Ti));
    u=u_old+g0*e+g1*e_old;
    e_old=e;

    elseif Cntrl==2
    %LQ optimal control
     dx=x-x_old;
     u=u_old+G1*dx+G2*(y_old-r)
     %u=G1*x+G2*(y_old-r)

    elseif Cntrl==3
     %Unconstrained MPC control, constraints handled by if else
       r1L=MPCref(k+1+L1:k+L+L1);% adding L1 inboth referances. the
prediction time
       xt=[x-x_old;y_old];
       y_old=y;
       x_old=x;

       pl=OL*At2*xt;
       f=F_L'*Qt*(pl-r1L);
       duf=-inv(H)*f;
       u=u_old+duf(1:nu);
       u_old=u;

    end


     %setting oven constraints typical anti wind up mode
        if u>800
            u=800;
        end

        if u<0
            u=0;
        end
     x_old=x;
     u_old=u;
     y_old=y;



    % Save for plotting
    Y(k,1)=y;
    U(k,1)=u;
    R_pred(k,1)=r;
    R(k,1)=WeRef(k);

    %x=At*x+Bt*u;


    %send control signal to the nonlinear function model
    x=HouseMdl(h,v,x,u);

    %use temperature estimation

    L_est=TempEst(x,v,h,293,800)
    if L_est>L1
        L1=L_est
    end


        if WeRef(k+L1)>WeRef(k)
```

```
            r=WeRef(k+L1);
        end



end

%---------Plot results----------------------------------------------------
t=1:N-200;
figure(1)
plot(t/100,Y,'-r',t/100,R_pred,'-g',t/100,R,'-.')
legend('Output, y_k','Predicted Reference r_k_+_L','Reference, r_k')

figure(2)
subplot(211), plot(U)
title('Control, u');
subplot(212), plot(diff(U))
title('Control, du');
```

7.6.7 The PWM function

```
A=1;
t=0:0.001:1;
t=t(1:1000)
%Create carrier Sawtooth wave
a = 10*A;
T = 0.1;
c=a*mod(t,T);


t=t*6;
%setting ut the input length
m=1:1:length(t);
%Setting the input signal
u_max=1000; % maximum oven power
u=800;  %Control generated input signal
m(:)=u/u_max;

n=length(c);%Length of carrier sawtooth is stored to 'n'
for i=1:n%Comparing Message and Sawtooth amplitudes
if (m(i)>=c(i))
    pwm(i)=1;
else
    pwm(i)=0;
end
end

%subplot(2,1,2);
plot(t,pwm,'b','LineWidth',2);
axis([0 6 0 1.2])
title('PWM with 80% duty cycle');
xlabel('Time [Min]');
ylabel('1=On  ,  0=Off');
grid on;
```

7.7 Appendix 7: Expanded model

$$\frac{dT}{dt} = \widetilde{N}T\left(1 - \frac{p_i M_i}{\rho R T_i}\right) + \frac{\widetilde{N} p_i M_i (1 - x_i)(C_{pa} T_i + \widehat{H}_{dar}) + \frac{x_i}{M_{h_2o}}(\hat{C}_{pw} + H_{wlr} + H_{lh}) - \rho \widetilde{N}(1 - x)(C_{pa} T_i + \widehat{H}_{dar}) + \frac{x}{M_{h_2o}}(\hat{C}_{pw} + H_{wlr} + H_{lh}) + \dot{Q}_{supply} - UA(T - T_i)}{\rho V(\left(1 - \frac{P_i M_i x_i}{\rho_o R T_i}\right) * \hat{C}_{pa} + \frac{p_i M_i x_i}{\rho R T_i} * (\hat{C}_{pw}) - \frac{R}{M})}$$

Where $\widetilde{N}$ = N*V/3600, and T_i is inlet temperature ,Cp is heat capacity of dry air, Cpw is heat capacity of humid air, x is fraction of water in air the rest of the parameters are seen in Table 2-1. For more information the reader is advised to read the BAS master project.

The MATLAB model used for the derivation is

```
dx(1)=Nt*x1*(x2-(P*M_AH/(R*v)))/(x2)+(Vi*(P*M_AH/(R*v))*(Cpa*(v-273)+fr*(Cpw*(v-273)+H_wv_ref))-Vo*x2*(Cpa*(x1-273)+fr*(Cpw*(x1-273)+H_wv_ref))+(u-(UA*(x1-
v))))/(Vr*x2*Cp)+Md;
```

7.8 Appendix 8: Regression models for predictor

The regression parameters as seen in the configuration XML file

```xml
<DpredictorAlpha>1.1984954138447921</DpredictorAlpha>
<DpredictorBeta>0.0020909035984212621</DpredictorBeta>
<YpredictorAlpha>27.786336336336362</YpredictorAlpha>
<YpredictorBeta>0.3766537966537955</YpredictorBeta>
<A11>0.99203191483706066</A11>
<A12>0.19920212907348425</A12>
<A21>0</A21>
<A22>1</A22>
<B11>0.031808516293937</B11>
<B21>0</B21>
```

The Kalman filter disturbance model:

$$\text{Temperature} = A * \left[\frac{Temperature}{Dpredictor\ alpha + DpredictorBeta * i}\right] + B * Temperature$$

The OLS regression model:

Temperature =YpredictorAlpha+YpredictorBeta*i

Where i is the loop counter running each sampling time of 200ms.