Intelligent heating of a room

07-01-2016

Bart Wijlens
S1317733

Introduction

The heating of rooms on the UT is not done efficiently. The temperature is controlled by a central thermostat, if you want to change the temperature in an individual room you have to control the radiator manually. All the rooms are heated constantly because the thermostat is centralized, also when another temperature is needed or when nobody is present. The goal of this project is to make the heating of a room smarter. The research focuses primarily on the design of an algorithm that can predict how much time it will costs to heat the room . The results of this algorithm can be used to make the heating of the room smarter and more efficient. Examples for implementation are, changing properties of the room to make the heating more efficient, predict when to start and stop heating, change heating schedule to the weather forecast and more. This report shows the research done to make this self learning system work. The report is divided in three different parts:

Making the thermodynamic model

To test the self learning system a model of a room is needed. This part explains how a thermodynamic model of a room is designed. This model is later on implemented in Matlab.

Design self learning system in Matlab

The self learning system is designed and tested on the thermodynamic model in this part. All of this is done in Matlab.

Design self learning system in Python

Here the explanation and implementation of the self learning system in python/jython and openHAB is showed.

Inhoud

Work plan

The research consists of three steps. The design of a thermodynamic model of the room is the first step. Later on, this model will be used to test the self learning system. The second step is the design and implementation of the self learning system together with the thermodynamic model in Matlab. As a start an easy implementation is the first challenge, afterwards when there is time left, the self learning system will be made more complex. The whole system is first simulated in Matlab to test if it works correctly. Why choosing Matlab for this test? Matlab is relatively easy to use and has a lot of build in functionality to make simulations easier. When the system works correctly the last step has to be taken namely, implementation in openHAB. openHAB is primarily used to get the sensor information of the room and to act as interface. The self learning system and control of the room temperature are made in python/jython.

Making Thermodynamic model

Theory

Introduction

Making a thermodynamic model of a room is useful to analyse the heating. Where heat is "lost" (sink) and where heat is "produced" (source). For this project a simple model will be sufficient because it will only implement the most important sinks and sources. This model can be used to test the self learning system.

Used Method

There are a lot of different methods to make a thermodynamic model, one even more complex than the other. A simplified approach has been chosen in which the room is modelled as an electric circuit. This means that temperature is seen as voltage (effort) and heat flow as current(flow). Why choosing a simplified approach? First reason is to limit the time which is needed to make the model. To develop a complex model which is understandable as well as effective is time consuming. And besides, there is a higher risk of making errors. The second reason is that a complex model will not add that much to the results of this research. It would merely be a waste of time. Before starting with the thermodynamic model the equivalents for different electrical components are defined.

Radiator

The radiator can be seen as a current or voltage source depending on whether a thermostat is applied or not. When the temperature of the radiator is controlled it will give a constant temperature, so a constant effort. A source with a constant effort is a voltage source. Therefore a radiator with a thermostat is a Voltage Source. When the radiator has no temperature control, the temperature fully depends on the hot water flow. In this case it is modelled as a current source.

Walls and Windows

Walls and windows can store and loose heat, but also have a certain resistance when they transport heat from one room to another. The resistance of a wall, when heat is transported, can be modelled as a resistor. The formula is given below:

Resistor

$$R_{12} = \frac{\theta_1 - \theta_2}{q_{12}} \; with \; R_{12} = \frac{L}{A * \sigma}$$

$$L = length\ wall(m)$$

$$A = Area\ wall(m^2)$$

$$\sigma = Thermal\ Conductivity \left(\frac{W}{mK}\right)$$

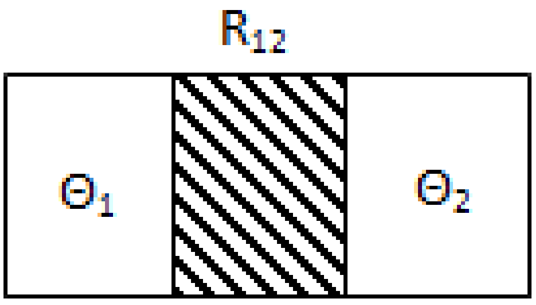


Figure 1:Model Thermodynamic Resistor

Figure 1 **Fout! Verwijzingsbron niet gevonden.**illustrates the situation, the white rectangles represent two different rooms with two different temperatures, the striped part represents a wall with resistance R_{12}.

The walls and windows can also store and loose heat. This is the same property as that of an electrical capacity, that is why this is modelled as an capacitor. The formula is given below:

Capacitor

$$q_1 = C_1 \frac{d\theta}{dt} \; with \; C_1 = m * c_p$$

$$m = mass\ object(kg)$$

$$c_p = Specific\ Heat(\frac{J}{kgK})$$

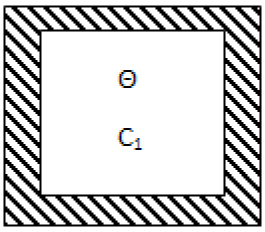


Figure 2: Model Thermodynamic Capacitor

The situation shown in Figure 2 the white rectangle is a room with temperature theta and capacity C_1(Capacity of the air). The Striped part represents a wall.

With this method every radiator is modelled as a source and all the walls are modelled as a combination of resistors and capacitors.

Walls

Each wall in the room has a resistance and a capacity. To make it easier to understand it is modelled as in Figure 3.

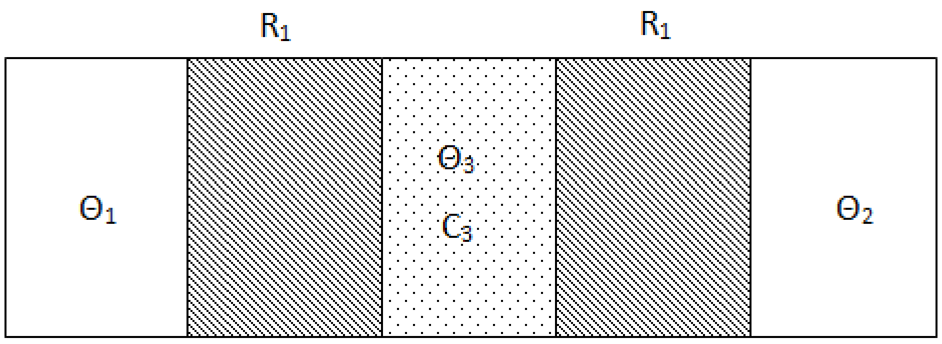


Figure 3: Model wall

This picture shows that the wall consists of two resistors in series with a capacitor connected in the middle, the equivalent circuit is given in Figure 4.

$$R1 = \frac{1}{2} R_{wall}$$

$$C_3 = C_{wall}$$

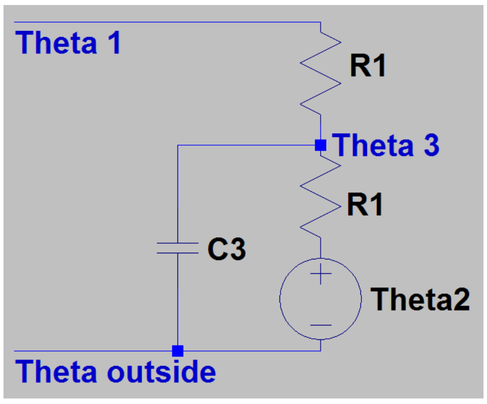


Figure 4:Themodynamic model wall

Radiators
The room is heated by two radiators with a thermostat so they can be modelled as a voltage source. The two sources are modelled as one whose value is the average of the two.

The doorway is difficult to model. It can be seen as a wall with a very low resistance (almost 0) and a capacity equal to the capacity of air. It can also be modelled as a voltage source, because it has a constant temperature controlled by the thermostat of the university. The temperature will be more or less constant because of the capacitance of the entire building including the air.

There is chosen to model the doorway as a Voltage source and it is added to the other two. Therefore in the model the room is heated by one source whose value is the average of the three separate sources.

With the individual parts of the room explained the thermodynamic model can be made.

Implementation

Introduction
The room that is going to be modelled is given in Figure 5. The orientations do not have to represent the real orientation of the room.

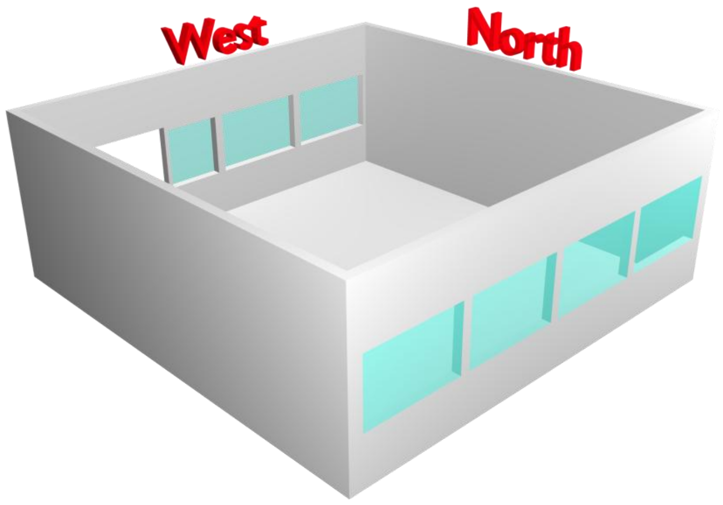


Figure 5: 3D Model Room

East Wall
The East wall: this wall consists of 2 radiators(white), glass(blue) and concrete(gray)(see Figure 6). The two types of surfaces should be modelled as two different walls. The two radiators are modelled as a voltage source with in series a resistance. This resistor has two reasons, first it represents the small resistance for transferring heat from the radiator to the air. The second reason is that it offers the possibility to make a bond graph later on, a voltage source without an internal resistance is not

realistic. It is assumed that the radiator has an unlimited heat flow.

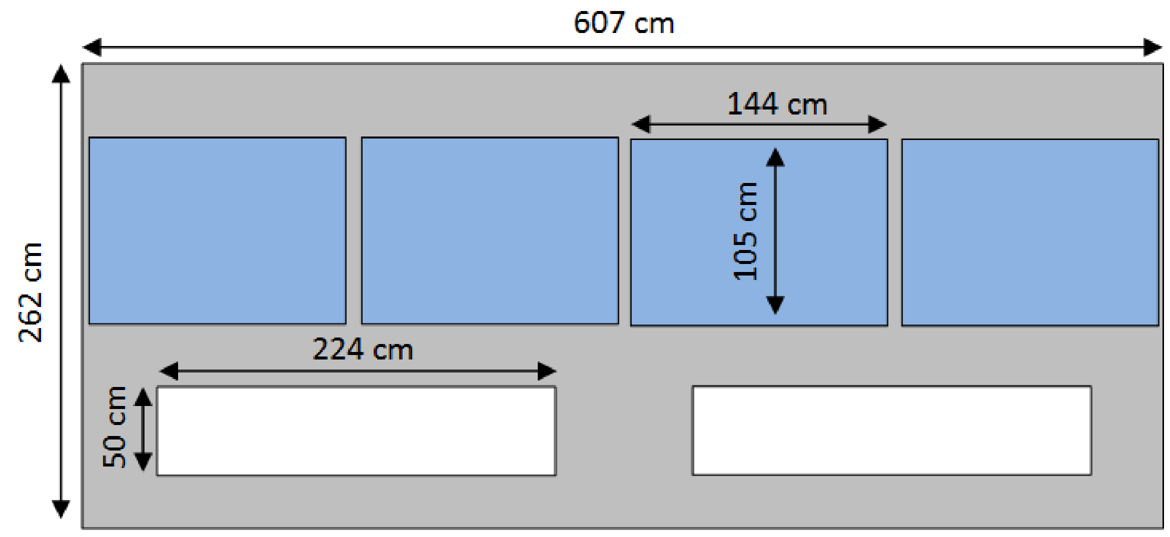
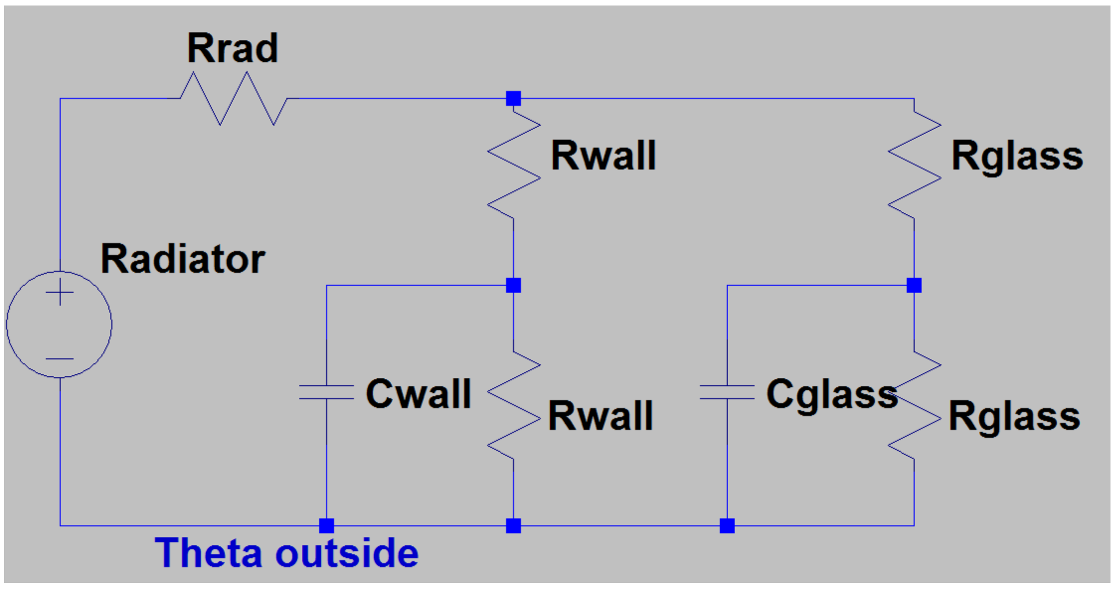


Figure 6: Drawing east wall

The circuit is given below



The actual values of the resistors and capacitors are calculated in the end.

West Wall

The second wall is the wall on the other side of the room. A drawing of this wall is given in Figure 7.

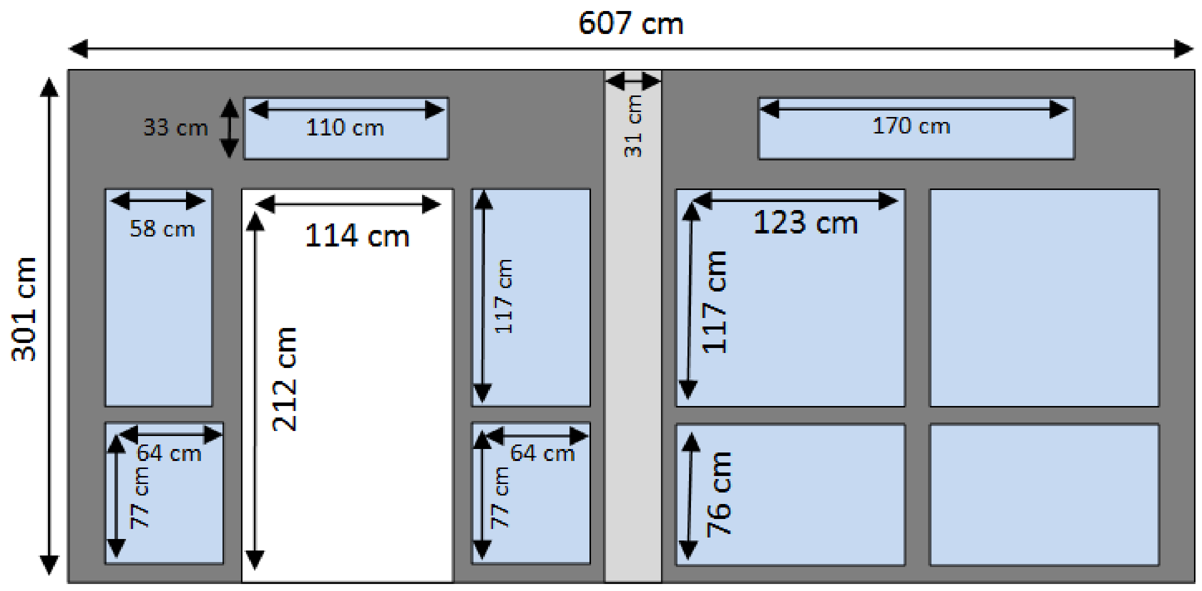


Figure 7: Drawing wall west

This wall consits of several windows(blue) placed in a metal wall dark gray) with one door (white). A concrete pillar can be found in the middle(ligth gray). The doorway has been described before so it is unnecessary to discuss it any further. The circuit is indentical to the east wall only in this case there is no source and the concrete is replaced by metal now.

North and West Wall

The North and West wall are the same, so they share the same circuit. A drawing of the walls is given in Figure 8.

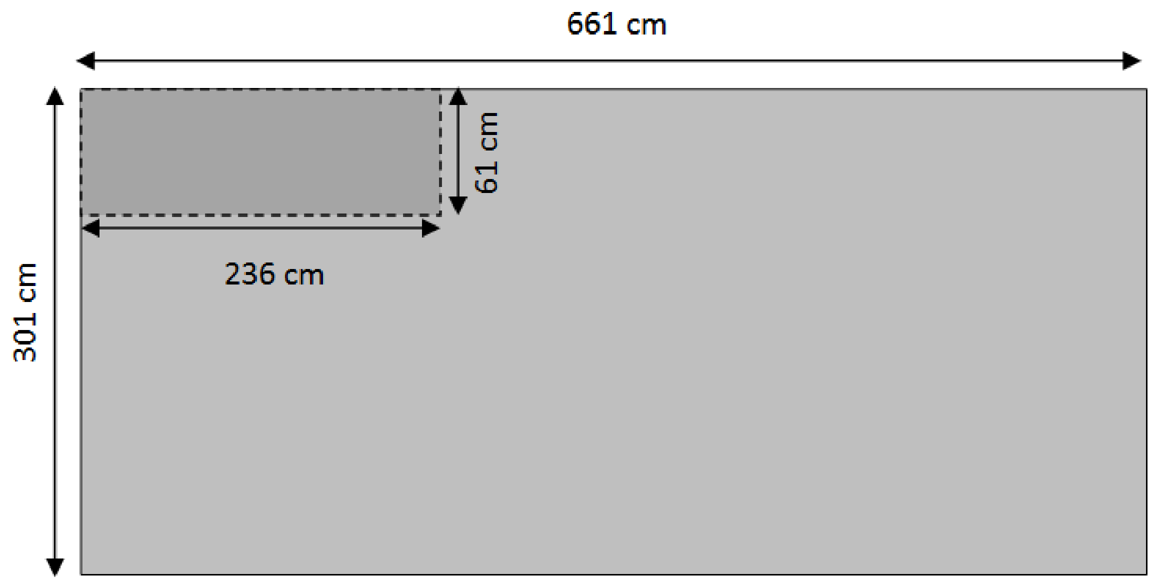
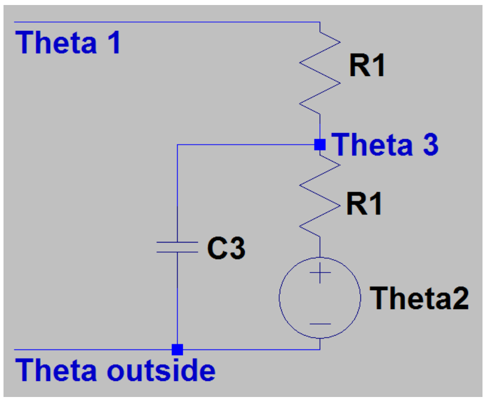


Figure 8: Drawing north/west wall

The circuit for this wall is



This is the standard circuit for a wall.

Floor and Roof

The floor and roof both consist of concrete. The dimensions can be found in Figure 9.

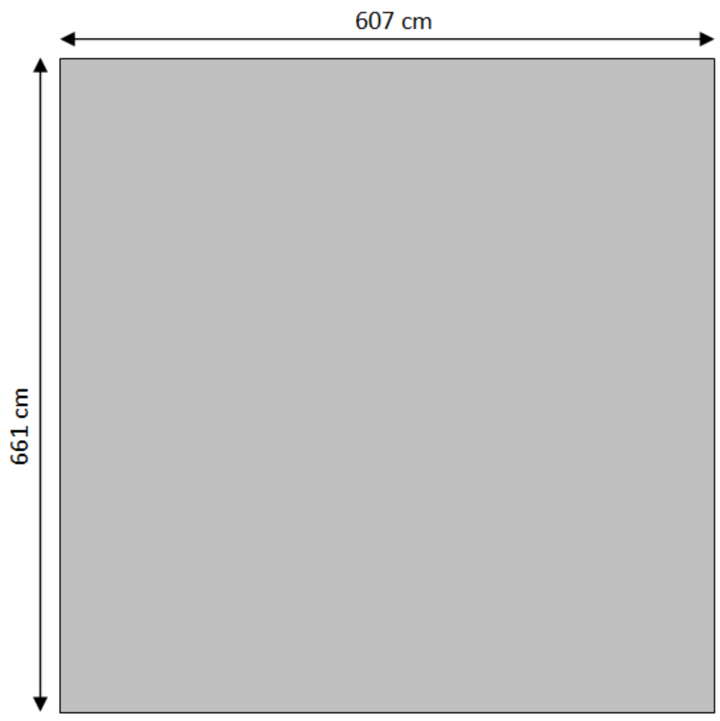
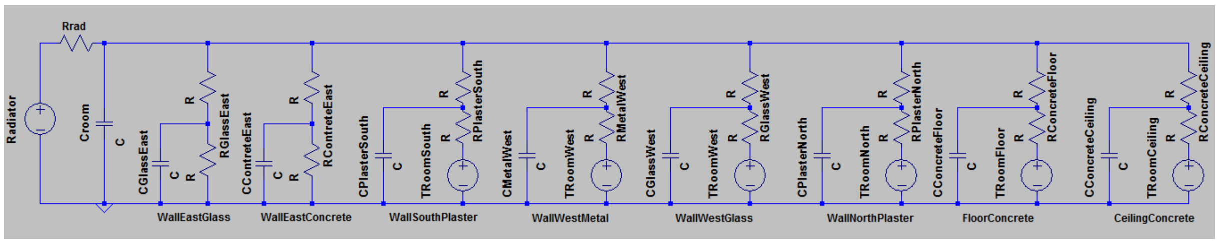


Figure 9: Drawing roof/floor

The thermodynamic representation is the same as for a wall.

Resulting circuit

When all the circuits are put together it results in the following schematic.



Rrad
Radiator
Croom
C
CGlassEast
C
R
R
RGlassEast
CContreteEast
C
R
R
RContreteEast
CPlasterSouth
C
TRoomSouth
R
R
RPlasterSouth
CMetalWest
C
TRoomWest
R
R
RMetalWest
CGlassWest
C
TRoomWest
R
R
RGlassWest
CPlasterNorth
C
TRoomNorth
R
R
RPlasterNorth
CConcreteFloor
C
TRoomFloor
R
R
RConcreteFloor
CConcreteCeiling
C
TRoomCeiling
R
R
RConcreteCeiling
WallEastGlass
WallEastConcrete
WallSouthPlaster
WallWestMetal
WallWestGlass
WallNorthPlaster
FloorConcrete
CeilingConcrete

Making the State Space model

To simulate the circuit above in matlab it needs to be converted to a state space diagram, the convertion is done in two steps. First the circuit is rewritten to a bond graph, this bond graph is then converted to a state space. The conversion can also be done directly but doing it this way , the chance of making errrors is smaller.

Bond graphs are a different way of displaying efforts and flows. It consists of nodes and bonds. Nodes are points where the efforts(voltage) or the flows(current) are the same. The nodes are displayed with zeroes for equal effort, and ones for equal flows. The bonds are the lines that connect the different nodes, the direction the bond points, is the direction in which the power is positive.

To make a Bond graph and calculate the state space for the full circuit would take a lot of time and it is unnecessary. The circuit determined above shows that the same parts can be found multiple times. The circuit consist of: 1 source + room, 2 walls to outdoor and 6 walls to other rooms.

In Figure 10 the different parts are shown. The parts that occur multiple times all share the same voltage so they can all be placed after each other. The final bond graph consists of one or more of the three different parts shown below.

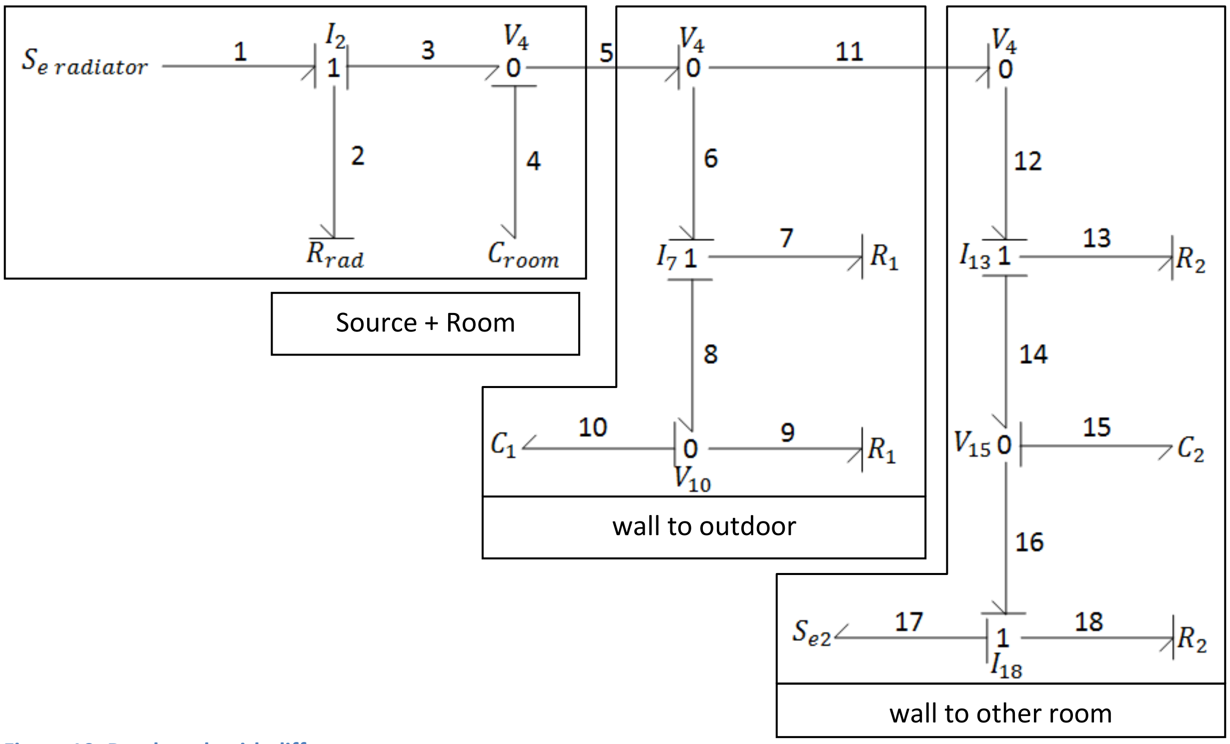


Figure 10: Bondgraph with different parts

The bond graph in Figure 10 is used to make the state space. The influence of the different parts of the Bond Graph can be determined from this state space. This information is used for the determination of the full state space see(Appendix 1: Calculation state space). The matrix on the next page is the final state space diagram.

$$\begin{bmatrix} \dot{V}_{room} \\ \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \dot{V}_4 \\ \dot{V}_5 \\ \dot{V}_6 \\ \dot{V}_7 \\ \dot{V}_8 \end{bmatrix} = \begin{bmatrix} -\frac{1}{C_{room}}(\frac{1}{R_{rad}}+\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4}+\frac{1}{R_5}+\frac{1}{R_6}+\frac{1}{R_7}+\frac{1}{R_8}) & \frac{1}{R_1C_{room}} & \frac{1}{R_2C_{room}} & \frac{1}{R_3C_{room}} & \frac{1}{R_4C_{room}} & \frac{1}{R_5C_{room}} & \frac{1}{R_6C_{room}} & \frac{1}{R_7C_{room}} & \frac{1}{R_8C_{room}} \\ \frac{1}{R_1C_1} & -\frac{2}{R_1C_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_2C_2} & 0 & -\frac{2}{R_2C_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_3C_3} & 0 & 0 & -\frac{2}{R_3C_3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_4C_4} & 0 & 0 & 0 & -\frac{2}{R_4C_4} & 0 & 0 & 0 & 0 \\ \frac{1}{R_5C_5} & 0 & 0 & 0 & 0 & -\frac{2}{R_5C_5} & 0 & 0 & 0 \\ \frac{1}{R_6C_6} & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_6C_6} & 0 & 0 \\ \frac{1}{R_7C_7} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_7C_7} & 0 \\ \frac{1}{R_8C_8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_8C_8} \end{bmatrix} \begin{bmatrix} V_{room} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{bmatrix} \begin{bmatrix} \frac{1}{R_{rad}C_{room}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{R_3C_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{R_4C_4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{R_5C_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{R_6C_6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{R_7C_7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{R_8C_8} \end{bmatrix} \begin{bmatrix} S_e \\ S_{e3} \\ S_{e4} \\ S_{e5} \\ S_{e6} \\ S_{e7} \\ S_{e8} \end{bmatrix}$$

$$tempRoom = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{room} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{bmatrix}$$

The state space is known but without component values it is not very useful. The components are calculated in Appendix 2: Calculate Values for components. Keep in mind that the results give only an indication of the of real values. The table below shows the calculated values.

Capacitor	Value	Resistor	Value
C_{room}	146257.65	R_{rad}(Wall East Glass)	Very low resistance
C_1(Wall East Glass)	73237,22	R_1(Wall East Glass)	0.058
C_2(Wall East Wall)	Insulated, low capacity	R_2(Wall East Wall)	Insulated, High Resistance
C_3(Wall West Metal)	Insulated, low capacity	R_3(Wall West Metal)	Insulated, High Resistance
C_4(Wall West Glass)	28871	R_4(Wall West Glass)	0.00029
C_5(Wall North Plaster)	525800	R_5(Wall North Plaster)	0.00214
C_6(Wall South Plaster)	525800	R_6(Wall South Plaster)	0.00214
C_7(Roof Concrete)	6778560	R_7(Roof Concrete)	0.0274
C_8(Floor Concrete)	6778560	R_8(Floor Concrete)	0.0274

Adding Variables to state space

The room has different properties that can be on or off. These properties will have an influence on the state space made of the room. For this project only a few of these properties are taken into account. In this report these properties will be referred to as states.

The following properties (states) are taken into account:

The Shutters

The windows in the room have shutters, when these are closed the resistance value of the windows will increase.

Incoming Sunlight

Incoming sunlight will cause the room to heat up faster so it is implemented in the heating-source of the room. It will increase the heating temperature of the source by a certain amount of degrees. In order to simplify the model, this amount is constant. In real life it will constantly chance as a result of different circumstances like the time of day, season, intensity of sunlight etcetera.

Door to entry

The door will act as a variable resistance between the room and the entry. When the door is open the resistance value will be low, when it is closed the value will be high.

People in room

People in the room will act as a heating source. The value so this source is equal to their body temperature, 37 degrees Celcius. The source is nonexistent when nobody is present.

Implementation in Matlab

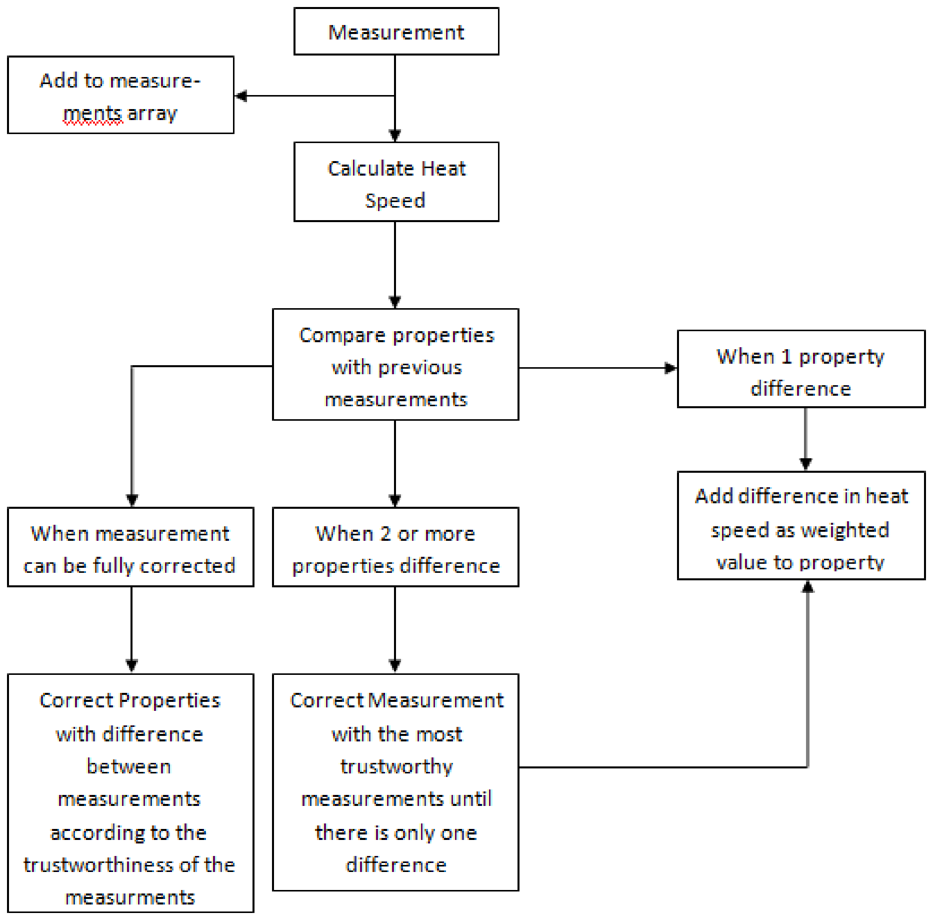
Design self learning System

The design of the self learning system has gone through several stages, three to be precise. A redesign of the system is made whenever a problem occurs. What is the reason these problems could arise? Most of the problems are caused by design mistakes, certain situations have been overlooked which provided wrong results. These different designs will be discussed shortly and at the end a final design can be found. The goal of the system is to predict the time which is needed to heat the room for its different states. These states are unknown to the system, it only gets the measurement results of the sensors placed in the room.

First Version

Working Principle

The system takes measurements with a certain interval. Each measurement consists of all the states of the room and its temperatures. When the next measurement is done, the heating speed (derivative of the temperature over time) for the previous measurement is determined. The next step is comparing the measurement to the previous measurements. First the system determines how the states of the room have changed in between the measurements; next it calculates the heating speed difference between the measurements. Afterwards it tries to relate the state differences to the difference in heating speed. The influence of each state is saved. When the influences of all the states are known to the system it can predict how much time it will cost to heat the room. See block diagram for the working of the system


Measurement
Add to measure-ments array
Calculate Heat Speed
Compare properties with previous measurements
When 1 property difference
Add difference in heat speed as weighted value to property
When measurement can be fully corrected
When 2 or more properties difference
Correct Properties with difference between measurements according to the trustworthiness of the measurments
Correct Measurement with the most trustworthy measurements until there is only one difference

Problem
The main problem of this system is, it does not cover the room very well. The first problem is caused by the states. Some of the states depend on other states. For example, the influence of the incoming sunlight which depends on the state of the shutters. The sunlight will influence the temperature differently depending on whether the shutters are open or closed. The system is not designed to handle this problem. A possible fix would be to set the relation between the states by hand. This is not implemented because it would interfere with the previous set goal; the states must be unknown. The second problem is that the influencing temperature compared to the room temperature is not taken into account. When the difference between those two is larger the influence will increase. This is something that is not incorporated in the system and therefore yielded incorrect results.

Taken to next version
Although the system did not work properly, it had some ideas build in that could be used for a next design. The idea of states showed to be useful for representing influences on the room. Also the way measurements are saved and the heating speed is used in the next version.

Second version

Working Principle
The second and third system have a lot in common. For that reason the explanation is not very detailed. Unlike the first system which focussed on the change of states, this system is build around linearly approaching the influences of the temperatures. It tries to find a linear relation between the heating speed and the temperature difference between the room and the influencing temperature. Each of these approaches is done for a certain set of states(properties of the room with a Boolean value), in order to get rid of the dependency problem that showed up in the previous version. The system makes a linear approach by determining a slope-, offset- and exponent value. A full explanation of how the second system works is given in the next section because the second and third system have a lot in common.

Problem
Because of a wrong approach of the problem, calculation mistakes were made. To explain the problems, it is assumed that the measured signal consists of a combination of three influencing temperatures.

$$Heating\ Speed = a_1(dT_1 - offset_1)^{b_1} + a_2(dT_2 - offset_2)^{b_2} + a_3(dT_3 - offset_3)^{b_3}$$

The problems start with the calculation of the offsets. The offset is determined by searching for a measurement which has a heating speed of zero. Offset$_X$ is equal to dT$_X$ is assumed by the system. This approach is not always valid however, for a heating speed of zero the offsets does not necessarily have to be equal to dT. Which means that some of the results will be correct and others might be wrong.

The next mistake is the calculation of the slope. The system makes the following calculation

$$a_x = \frac{Heating\ Speed}{(dT_x - offset_x)^{b_x}}$$

The only way this calculation gives a correct result is when the influence of two of the three temperatures is zero.

$$a_1 = a_1 + \frac{a_2(dT_2 - offset_2)^{b_2}}{(dT_1 - offset_1)^{b_1}} + \frac{a_3(dT_3 - offset_3)^{b_3}}{(dT_1 - offset_1)^{b_1}}$$

Taken to next version

The full system is used for the final version, only the part for the linear approach is replaced.

Final Version

Working Principle

The final version is build around the same principles as the previous one, linear approximation. The system tries to approximate the influence of temperatures on the room with a linear relation. Keep in mind that the implementation is simplified. In the section "improvements" some ideas can be found to improve the system. Due to time- and technical problems these have not been implemented.

Working of system

The system works with two types of variables. The first type are states. States are properties of the room with a Boolean value, which means that they can be turned on or off. For example: a door or a window that is open or closed. Temperatures are the second type of variable. Temperatures are normal variables that represent the temperatures in the room. All the temperatures are relative to the outdoor temperature. Below, a list of temperatures and states of the room is given.

The room used for testing has the following states:

- ➔ Door room open
- ➔ Shutter window closed
- ➔ People in room
- ➔ Incoming sunlight

The temperatures that will influence the room temperature are

- ➔ Entry Temperature
- ➔ Outdoor Temperature
- ➔ Radiator temperature

The system works as follows. First it measures all the temperatures and states of the room, next the heating speed for the previous measurement is determined. The heating speed is the difference between the current room temperature and the room temperature for the previous measurement divided by the time difference and it is given in Celcius/s. Determination of the linear relation between the temperatures and the corresponding heating speed is the next step. The temperatures are given relative to the room temperatures. The results of this approximation are stored in a matrix. It is important to realise that the system doesn’t know what the states are and what kind of influence they will have, it must approximate the influence of these states by itself.

Linear Approximation

The measured heating speed of the room is approximated by a sum of linear relations. Each of these relations represents the influence of one temperature on the total heating speed.

$$Heating\ speed\ total = a_{temp1} * \left(dT_{temp1}\right)^{b_{temp1}} + a_{temp2} * \left(dT_{temp2}\right)^{b_{temp2}} + a_{temp3} * \left(dT_{temp3}\right)^{b_{temp3}}$$

For which dT is given by:

$$dT = Temperature - Room\ Temperature$$

It is assumed that the linear approximation doesn't have offsets on the x -or y- axis. When the system is "fully covered" this is an accurate approximation. "Fully covered" means that there are no unknown influences on the room. To be more specific, there are no unknown heat sources and no unknown changes to resistance or capacitance values. If unknown influences are present the result will get an offset in the x and/or y direction. The offsets are not implemented because they could not be determined (see section Improvements)

The linear approximation algorithm tries to approach the influences of each of the individual temperatures linearly.

$$Heating\ Speed = a * (dT)^b$$

$$dT = Temperature - Room\ Temperature$$

The linear approximation works in four steps. It starts with sorting out all the measurements with the same set of states, to ensure that only the measurements with the same set of states are used for calculation. Secondly, the system filters all the measurements for which only one dT is unequal to zero. With these measurements the slope "a" can be calculated.

Side note

The results can be improved by subtracting measurements from each other in such a way that only one of the dTs is unequal to zero. This is not implemented because of the technical difficulty (See improvements).

The third step; an approximation of the slope is done by calculating the slope for all the measurements for which only one dT is unequal to zero and then taking the average. It is repeated for different values of b(exponent), resulting in a list of slopes for different b's. The fourth and last step is the determination of the exponent and is done by comparing the measurement points with all the different slopes. Best representation is the slope that agrees most with the measurements. This process is repeated for all the different state combinations.

Side note

Because of the chosen simplification the system needs some time before it has data to calculate the slopes with.

Implementing results

With the influence of temperatures known the second part of the self learning system can be used. Keeping the room at temperature is the responsibility of this part. This means heating the room in the morning when people arrive and stop heating in the evening when the people have left. With the influences of the temperatures on the room and the states of the room, the system can approximate the best way to heat the room. It can determine for which set of states it will heat most efficient and what is the most effective temperature for the radiator.

Matlab Implementation

Make the simulation of State Space

To simulate the room in matlab, a script is created. The simulation works as follows.

First the script ‘StartSimulation.m’ is ran. It will initialise all start variables for the simulation. After that the script ‘SimulateModel.m’ will be called. This script makes a simulation of the change of states that occur over time, next it will run ‘RunSimulationRoom.m’ every minute. "RunSimulationRoom.m" simulates the room temperature for one minute implementing all the factors that will influence this temperature. It consists of the state space calculated before. As initial values for the states variables, the value of the states of the previous measurement are used. All the matlab scripts can be found in the appendix.

Implement Self Learning System

The matlab implementation consists of a few different scripts. To start with ‘SaveMeasurement.m’ ‘SimulateModel.m’ will call this script every minute. The ‘SaveMeasurement.m’ script saves the current values for the states and temperatures in an array and it also calculates the heating speed for the previous measurement. The array is 1000 entries long which means that 1000 measurements can be done. After the measurement is saved the script ‘ProcessMeasurement.m’ is called. It will analyse all the measurements and determines the linear relation. For full explanation of the code, see the matlab files in the appendix.

Test Simulation State Space in Matlab

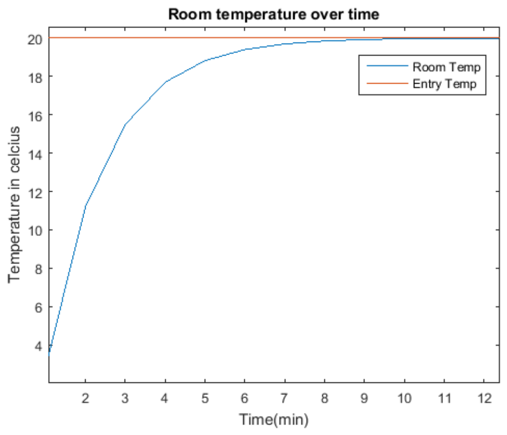
Test Self Learning system Matlab(Influence Entry)

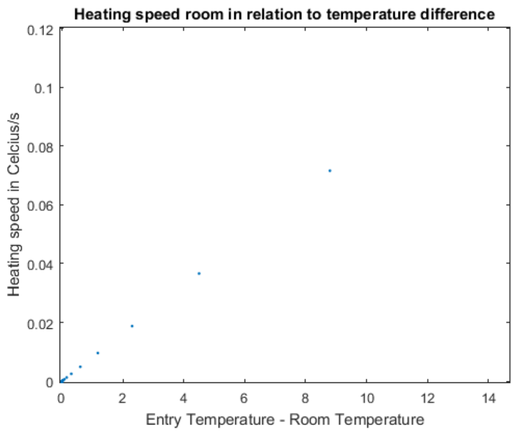
The self learning system is tested in two different ways. The first tests will be done by only varying one temperature and see if the system calculates the right values for the heating speed. Next all the temperatures will vary, it is checked how well the system approaches the heating speeds in this case. In some cases a state value is given, this is the sum of all the different influencing states.

- ➔ Door room open(value 1)
- ➔ Shutter window closed(value 2)
- ➔ People in room(value 4)
- ➔ Incoming sunlight(value 8)

So state 3 means 1 + 2, in this case the door is open and the shutters are closed. The results of the measurements are given below

First Tests





The left graph shows the room temperature over time. The right graph shows the heating speed influence of the entry temperature. The results of the self learning system show the following relation to the heating speed of the entry temperature:

$$Heating\ Speed\ = 0.0081(dT)$$

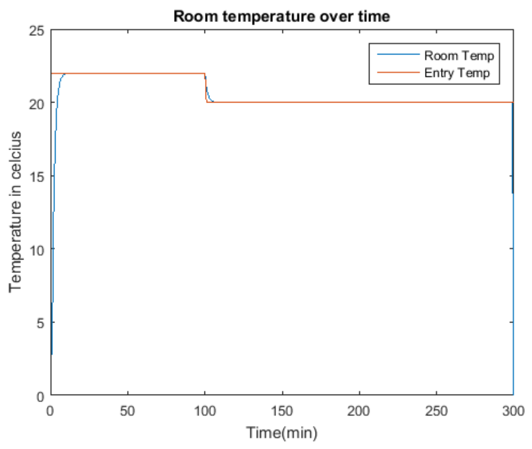
$$dT\ = Entry\ Temperature - Room\ Temperature$$

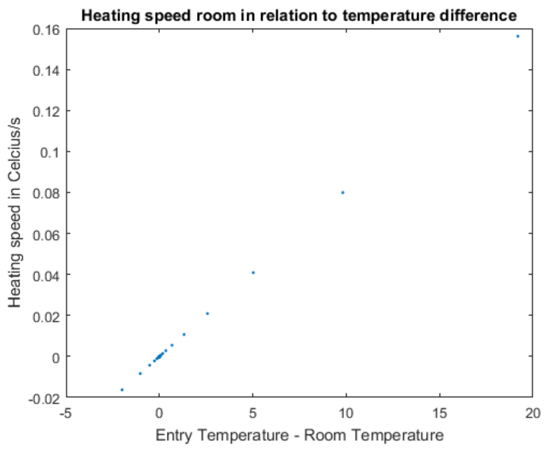
The values are compared to the linear part of the graph, the results are given below:

	Self Learning System	From Graph	Error
Slope(State = 0)	0.0081	0.0083	2.5 %

The error values are calculated by dividing the value from the graph by the value from the self learning system. Keep in mind that the value from the graph is determined by hand, so it includes a reading error. This experiment is repeated with a few different temperatures of the entry temperature

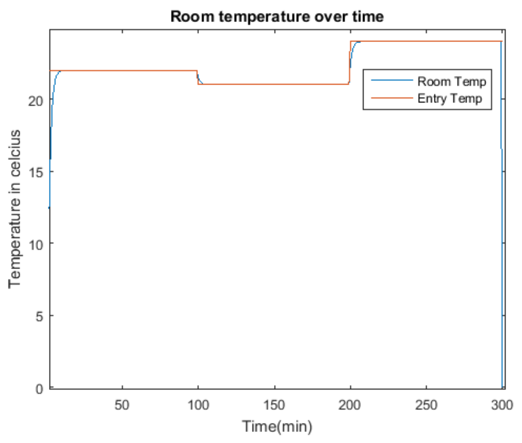
Second Test

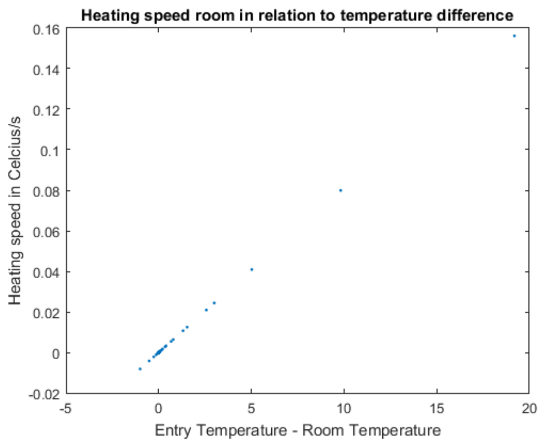




	Self Learning System	From Graph	Error
Slope(State = 0)	0.0081	0.0083	2.5 %

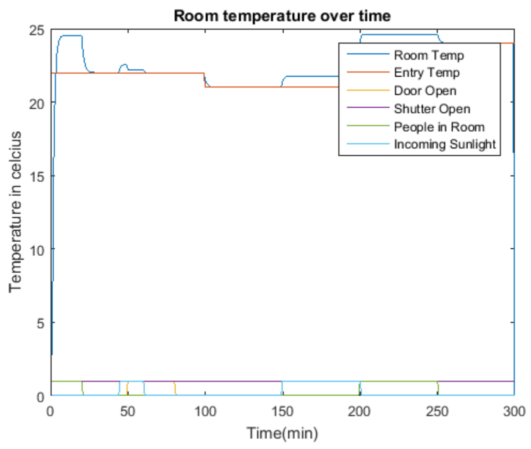
Third Test

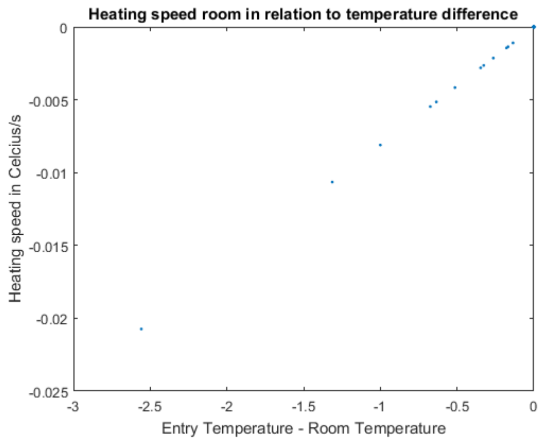


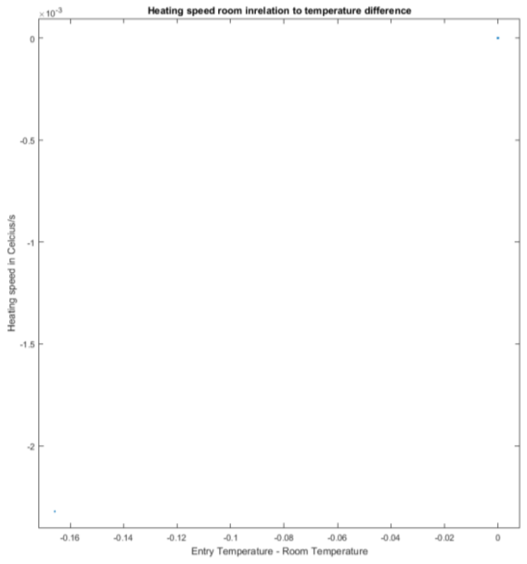


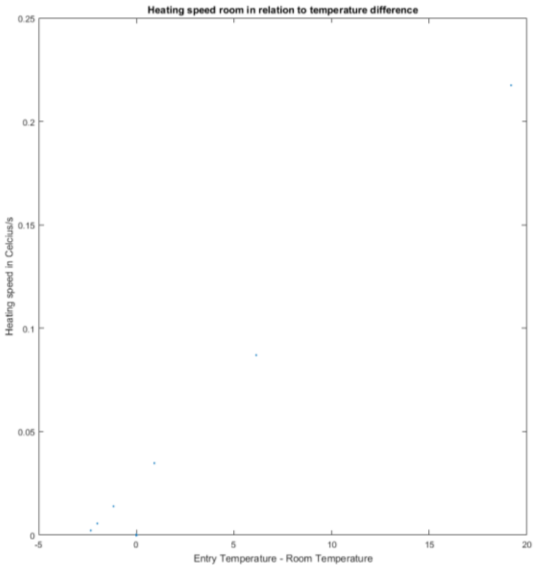
	Self Learning System	From Graph	Error
Slope(State = 0)	0.0081	0.0083	2.5 %

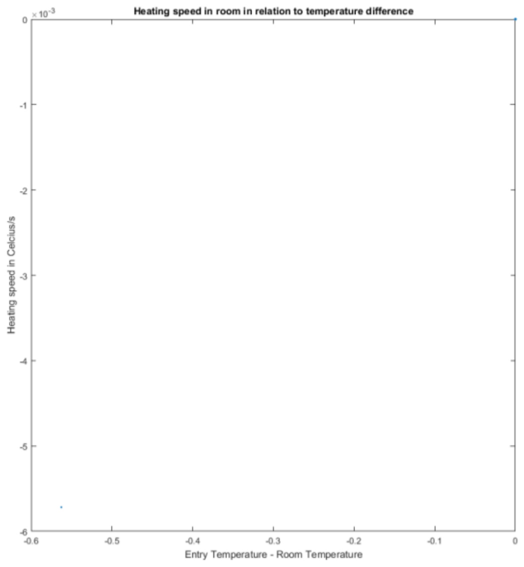
Last Test

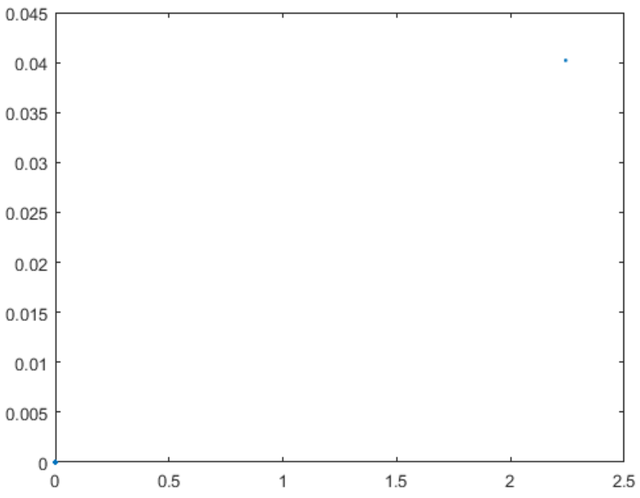












	Self Learning System	From Graph	Error
Slope(state = 2)	0,0081	0.0083	2.5 %
Slope(state = 3)	0,0140	0.015	7.1 %

Slope(state = 6)	0,0403(Root Function)	0.01(Linear function)	n.v.t
Slope(state = 11)	0,0102	0.0105	2.9 %
Slope(state = 15)	0,018	0.018	0 %

Discussion results

The system works very well when only one temperature is changed (with errors about 3%). The results show that the slopes for state 0 and state 2 are the same, this is correct since the shutter has no effect on the influence of the entry. The slopes for state 2 and state 3 are different, this is also correct because the door does change the influence of the entry temperature. The problems occurs when an offset to the room heating is introduced. For these measurements the offset is caused by people in the room(state 6). The measurements show that the result is a straight line with an offset, the system thinks the line should go through (0,0) resulting in wrong determination of the relation. The people act as an extra heating source causing the room to heat up or cool down. This additional source is not covered by one of the three influencing temperatures, so the system cannot adapt. Introducing an offset to the linear approximation will partly solve this problem. Unfortunately the shift depends on the temperature of the room. A possible solution for this problem can be found in the section improvements. The results of state 11 and state 15 should also have an offset, but this does not show because the result is based on only one measurement.

Test Self Learning system Matlab(Influence all Temperatures)

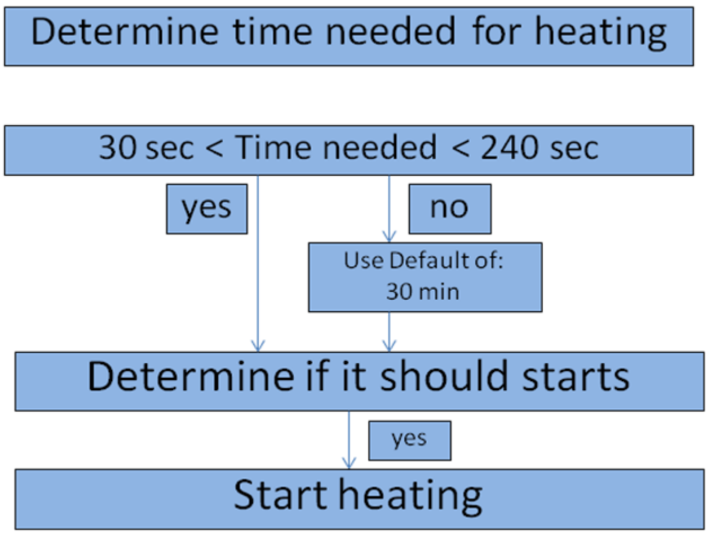
In the second set of tests the behaviour of the system is tested in case a number of temperatures change. This test is much harder to make because it must have points where two of the three temperatures are 0. No simulation has been found with more than one changing temperature that differs from the previous tests. So there are no test results with more than one changing temperature. In the section improvements a test can be found with more than one temperature changing, this is done with an improved system however.

Working with OpenHAB

Write Self learning system in Python/Jython

The system made in matlab is converted to python, the simulation part is removed in this process. After the conversion the system is tested with the simulation results of the thermodynamic model as an input. If the results of the system in python are the same as the results of the system in matlab it works correctly.

Next is looked at the implementation in openhab. The self learning system is controlled by openhab. Every time a “state” or “temperature”(except room temperature itself) changes the self learning system is triggered. Another part of the openHAB implementation is using the measured data. This part works as follows. First a calculation is made how much time it will cost to heat the room to a certain goal temperature. This prediction is checked to find out if it is realistic. If not, the default value of 30 minutes is used. After the system has determined the time needed to heat the room, it calculates when it should start in order to reach the goal temperature in time. When the system heats the room the radiator is set to 25 degrees, for cooling down it is set to 14 degrees. The goal time is guiding, which means when the goal time is reached the radiator is set to the goal temperature no matter if the goal temperature is reached or not. See block diagram below.


Determine time needed for heating
30 sec < Time needed < 240 sec
yes
no
Use Default of: 30 min
Determine if it should starts
yes
Start heating

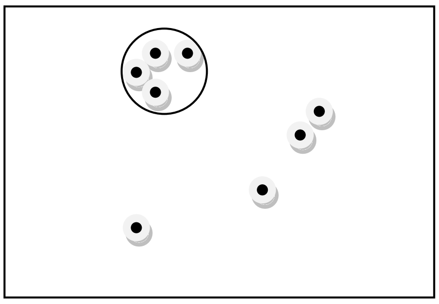
Test full system

The full system has not been tested. The simulations show that improvements to the system are necessary before a real life implementation can be realised. Some of the improvements are build in matlab but there was not enough time to implement them in openHAB.

Improvements

Filter results to get the most common value

A filter can be added to filter out the measurements that result in incorrect slopes . The filter will select the results with the most measurement points within a certain error range (see picture below) and it will remove results that do not match with other measurements. In this way wrong results will have less to no influence of the final result.



Subtract measurements to get better results

Working principle

Only using the results for which the dT is zero, makes doing a linear approach, very time consuming. It is not very common that two of the three temperatures are equal to the room temperature. To increase the number of measurement points with two of the three dT are zero, the measurements are subtracted from each other.

Measurement 1

$$HS_1 = a(dT_1)^b + a_2(dT_2)^{b_2} + a_3(dT_3)^{b_3}$$

Measurement 2

$$HS_2 = a(dT_4)^b + a_2(dT_5)^{b_2} + a_3(dT_6)^{b_3}$$

Subtracted

$$HS_1 - HS_2 = a(dT_1^b - dT_4^b) + a_2(dT_2^{b_2} - dT_5^{b_2}) + a_3(dT_3^{b_3} - dT_6^{b_3})$$

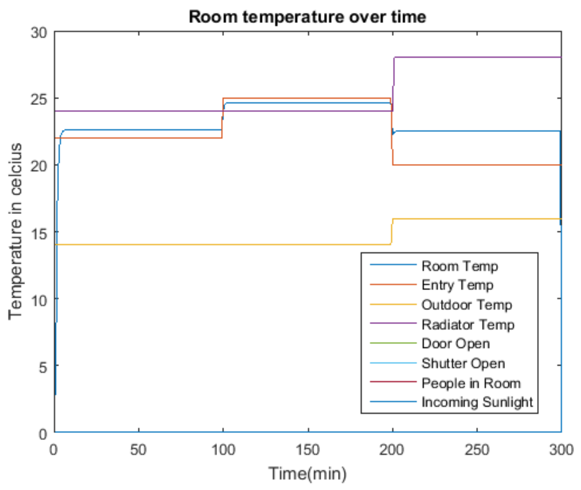
If two of the tree subtractions result in zero, the slope can be determined. Multiple measurements can be used to get to this result. The most difficult part is determining which measurements must be subtracted from each other.

Where is this implemented in the current system?

The system will start with the determination of the exponent values by using measurements which have two dTs of zero. When no measurements with this property are found the default value of 1(normal line) is used. During the next step the system will use the explained improvement to determine the slopes (a values).

Small test

The improvement described above has been build in Matlab, the code can be found in the GetCorrectedMeas.m. To test the improvement a small simulation is made. The results for the influence of the radiator are given below



	Self Learning System	Previous measurement	Error
Slope(Normal system)	???	0.0081	???
Slope(Improved system)	0.0081	0.0081	0 %

The results are promising. The normal system cannot make a linear approach for any of the three influencing temperatures. The improved version can determine the influence of the radiator. More tests have to be done to see how well the system works but the results are promising.

Implement x and y offsets

In the beginning, x and y offsets were implemented, but this caused so many errors that they were removed later on. To take care of influences that are not covered by the measurements the offsets are useful. However, a good way has not been found to implement this into the system. The implementation of offsets makes the determination of slopes harder.

Side node:

After some tests a possible solution has been found. The unknown influences will all result in an offset in the room temperature. This is why the offset will be the same for all three temperatures. When the three dTs are all equal to each other and the heating speed is zero, the offset is equal to dT. This can be proven with the following equation

$$HS = a(dT_1 - offset)^b + a_2(dT_2 - offset)^{b_2} + a_3(dT_3 - offset)^{b_3}$$

$$dT_1 = dT_2 = dT_3 = dT$$

$$HS = 0$$

$$0 = a(dT - offset)^b + a_2(dT - offset)^{b_2} + a_3(dT - offset)^{b_3}$$

When dT is bigger than zero it means that the heating speed must also be bigger than zero. When dT is smaller than zero, the heating speed must be smaller than zero. The only solution for this equation is then dT = offset.

$$0 = a(offset - offset)^b + a_2(offset - offset)^{b_2} + a_3(offset - offset)^{b_3}$$

$$0 = a(0)^b + a_2(0)^{b_2} + a_3(0)^{b_3}$$

The offset depends on the temperature difference between the room and the source that is causing the offset. This problem can be solved by approaching the offset linear in the same way as has been done for the influencing temperatures. To do the linear approach, the value of the temperature causing the offset has to be known. This value can be determined by searching for a measurement which has an offset of zero. An offset of zero means that the room temperature is equal to the temperature of the source that causes the offset.

Discussion

The tests done with matlab are showing promising results for a real life implementation, however there is one complication. The real life implementation cannot be tested before the improvements have been added. At this moment the only measurement values used are the ones for which two of the three temperatures are zero. This is a scenario that almost never happens. The "Subtract measurements from each other to get better results" improvement would greatly increase the number of measurements that can be used. If this is not applied, it will take a very long time before some results can be determined. A second problem showed up during the simulations. For influences on the room that are no heating source the system works fine, it determines the slope very well. but when an influence has the properties of a heating source the problems start to occur. If the heating source would add a standard amount of degrees this problem could be solved with a simple offset. Unfortunately this is a scenario that almost never happens, most of the time the heating source averages out with the other heating sources in the room. In the section improvements a possible solution for this problem has been given. Since the improvement is not applied the only solution for now is to implement heating sources as temperatures and influences on resistance values or capacitance values as states.

Conclusion

The main structure of the system stands for this moment, although it still needs a lot of improvement before it is fully functional. For influences on the room that are no heating source it works fine and determines the slope very well with small errors about 3%. Influences that have properties of a heating source should, at this moment, be implemented as temperature not as state. Some improvements to the system could be added but these are not/not well tested.

Appendix

Appendix 1: Calculation state space

From this bond graph the state space equations are determined

$$\frac{dV_4}{dt} = \frac{I_4}{C_{room}}$$

$$I_4 = I_3 - I_5$$

$$I_3 = I_2$$

$$I_2 = \frac{V_2}{R_{rad}}$$

$$V_2 = S_e - V_4$$

$$I_2 = \frac{S_e - V_4}{R_{rad}}$$

$$I_3 = \frac{S_e - V_4}{R_{rad}}$$

$$I_5 = I_6 + I_{11}$$

$$I_6 = I_7$$

$$I_7 = \frac{V_7}{R_1}$$

$$V_7 = V_4 - V_{10}$$

$$I_7 = \frac{V_4 - V_{10}}{R_1}$$

$$I_6 = \frac{V_4 - V_{10}}{R_1}$$

$$I_{11} = I_{12}$$

$$I_{12} = I_{13}$$

$$I_{13} = \frac{V_{13}}{R_2}$$

$$V_{13} = V_4 - V_{15}$$

$$I_{13} = \frac{V_4 - V_{15}}{R_2}$$

$$I_{11} = \frac{V_4 - V_{15}}{R_2}$$

$$I_5 = \frac{V_4 - V_{10}}{R_1} + \frac{V_4 - V_{15}}{R_2}$$

$$I_4 = \frac{S_e - V_4}{R_{rad}} - \frac{V_4 - V_{10}}{R_1} - \frac{V_4 - V_{15}}{R_2}$$

$$\frac{dV_4}{dt} = \frac{S_e - V_4}{R_{rad}C_{room}} - \frac{V_4 - V_{10}}{R_1 C_{room}} - \frac{V_4 - V_{15}}{R_2 C_{room}}$$

$$\frac{dV_4}{dt} = S_e \frac{1}{R_{rad}C_{room}} - \frac{V_4}{C_{room}}\left(\frac{1}{R_{rad}} + \frac{1}{R_1} + \frac{1}{R_2}\right) + V_{10}\frac{1}{C_{room}} + V_{15}\frac{1}{C_{room}}$$

$$\frac{dV_{10}}{dt} = \frac{I_{10}}{C_1}$$

$$I_{10} = I_8 - I_9$$

$$I_8 = I_7$$

$$I_7 = \frac{V_7}{R_1}$$

$$V_7 = V_4 - V_{10}$$

$$I_7 = \frac{V_4 - V_{10}}{R_1}$$

$$I_8 = \frac{V_4 - V_{10}}{R_1}$$

$$I_9 = \frac{V_{10}}{R_1}$$

$$I_{10} = \frac{V_4 - V_{10}}{R_1} - \frac{V_{10}}{R_1}$$

$$I_{10} = \frac{V_4 - 2V_{10}}{R_1}$$

$$\frac{dV_{10}}{dt} = \frac{V_4 - 2V_{10}}{R_1 C_1}$$

$$\frac{dV_{15}}{dt} = \frac{I_{15}}{C_2}$$

$$I_{15} = I_{13} - I_{16}$$

$$I_{13} = \frac{V_{13}}{R_2}$$

$$V_{13} = V_4 - V_{15}$$

$$I_{13} = \frac{V_4 - V_{15}}{R_2}$$

$$I_{16} = I_{18}$$

$$I_{18} = \frac{V_{18}}{R_2}$$

$$V_{18} = V_{15} - S_{e2}$$

$$I_{18} = \frac{V_{15} - S_{e2}}{R_2}$$

$$I_{16} = \frac{V_{15} - S_{e2}}{R_2}$$

$$I_{15} = \frac{V_4 - V_{15}}{R_2} - \frac{V_{15} - S_{e2}}{R_2}$$

$$I_{15} = \frac{V_4 - 2V_{15} + S_{e2}}{R_2}$$

$$\frac{dV_{15}}{dt} = \frac{V_4 - 2V_{15} + S_{e2}}{R_2 C_2}$$

The resulting state space is

$$\begin{bmatrix} \dot{V_4} \\ \dot{V_{10}} \\ \dot{V_{15}} \end{bmatrix} = \begin{bmatrix} -\frac{1}{C_{room}}\left(\frac{1}{R_{rad}} + \frac{1}{R_1} + \frac{1}{R_2}\right) & \frac{1}{R_1 C_{room}} & \frac{1}{R_2 C_{room}} \\ \frac{1}{R_1 C_1} & -\frac{2}{R_1 C_1} & 0 \\ \frac{1}{R_2 C_2} & 0 & -\frac{2}{R_2 C_2} \end{bmatrix} \begin{bmatrix} V_4 \\ V_{10} \\ V_{15} \end{bmatrix} + \begin{bmatrix} \frac{1}{R_{rad} C_{room}} & 0 \\ 0 & 0 \\ 0 & \frac{1}{R_2 C_2} \end{bmatrix} \begin{bmatrix} S_e \\ S_{e2} \end{bmatrix}$$

$$tempRoom = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_4 \\ V_{10} \\ V_{15} \end{bmatrix}$$

In the same way the other states can be added

$$\begin{bmatrix} \dot{V}_{room} \\ \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \dot{V}_4 \\ \dot{V}_5 \\ \dot{V}_6 \\ \dot{V}_7 \\ \dot{V}_8 \end{bmatrix} = \begin{bmatrix} -\frac{1}{C_{room}}\left(\frac{1}{R_{rad}} + \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} + \frac{1}{R_6} + \frac{1}{R_7} + \frac{1}{R_8}\right) & \frac{1}{R_1 C_{room}} & \frac{1}{R_2 C_{room}} & \frac{1}{R_3 C_{room}} & \frac{1}{R_4 C_{room}} & \frac{1}{R_5 C_{room}} & \frac{1}{R_6 C_{room}} & \frac{1}{R_7 C_{room}} & \frac{1}{R_8 C_{room}} \\ \frac{1}{R_1 C_1} & -\frac{2}{R_1 C_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_2 C_2} & 0 & -\frac{2}{R_2 C_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_3 C_3} & 0 & 0 & -\frac{2}{R_3 C_3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{R_4 C_4} & 0 & 0 & 0 & -\frac{2}{R_4 C_4} & 0 & 0 & 0 & 0 \\ \frac{1}{R_5 C_5} & 0 & 0 & 0 & 0 & -\frac{2}{R_5 C_5} & 0 & 0 & 0 \\ \frac{1}{R_6 C_6} & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_6 C_6} & 0 & 0 \\ \frac{1}{R_7 C_7} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_7 C_7} & 0 \\ \frac{1}{R_8 C_8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{2}{R_8 C_8} \end{bmatrix} \begin{bmatrix} V_{room} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{bmatrix} \begin{bmatrix} \frac{1}{R_{rad} C_{room}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{R_3 C_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{R_4 C_4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{R_5 C_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{R_6 C_6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{R_7 C_7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{R_8 C_8} \end{bmatrix} \begin{bmatrix} S_e \\ S_{e3} \\ S_{e4} \\ S_{e5} \\ S_{e6} \\ S_{e7} \\ S_{e8} \end{bmatrix}$$

$$tempRoom = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{room} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{bmatrix}$$

Appendix 2: Calculate Values for components

R$_{rad}$

Low Resistance, value is guessed at 0.01, will be determined with simulation

C$_{room}$

Material: air at 20˚C

$C_{room} = m_{room} * c_{p\ air}$

$m_{room} = l * w * h * \rho = 6{,}07\ m * 6{,}61\ m * 3.01\ m * 1{,}205\ kg/m^3 = 145{,}53\ kg$

$c_{p\ air} = 1005 \frac{j}{kgK}$

$C_{room} = 145{,}53\ kg * 1005 \frac{j}{kgK} = 146257.65 \frac{J}{K}$

Wall East

Insulated Glass(C_1 and R_1)

Consists of two glass plates of 3 mm thick with an argon layer of 1.4 mm thick in between

C_1

$C_1 = C_{glass} * 2 + C_{argon}$

$C_{glass} = m_{glass} * c_{p\ glass}$

$m_{glass} = l * w * h * \rho = (1{,}05\ m * 1{,}44\ m * 0.003\ m) * 4 * 2400 \frac{kg}{m^3} = 43{,}55\ kg$

$c_{p\ glass} = 840 \frac{j}{kgK}$

$C_{glass} = 43{,}55\ \ kg * 840 \frac{j}{kgK} = 36582 \frac{J}{K}$

$C_{argon} = m_{argon} * c_{p\ argon}$

$m_{argon} = l * w * h * \rho = (1{,}05\ m * 1{,}44\ m * 0.014\ m) * 4 * 1{,}69 \frac{kg}{m^3} = 0.14\ kg$

$c_{p\ argon} = 523 \frac{J}{kgK}$

$C_{argon} = 0.14\ \ kg * 523 \frac{J}{kgK} = 73{,}22 \frac{J}{K}$

$C_1 = 36582 \frac{J}{K} * 2 + 73{,}22 \frac{J}{K} = 73237{,}22 \frac{J}{K}$

R_1

$R_1 = 0{,}35 \frac{m^2K}{W} / ((1{,}05\ m * 1{,}44\ m) * 4) = 0.058 \frac{\mathrm{K}}{\mathrm{W}}$

Isolated Wall(C_2 and R_2)

The outside wall is around 15 cm thick. Consists of two layers with in between an isolation layer

C2

$\mathrm{C_{polyurethane\ foam}} = \mathrm{m_{polyurethane\ foam}} * \mathrm{c_{p\ polyurethane\ foam}}$

$$\mathrm{m_{polyurethane\ foam}} = \mathrm{l * w * h} * \rho = \big((6{,}07\ \mathrm{m} * 6{,}61\ \mathrm{m}) - (1{,}05\ \mathrm{m} * 1{,}44\ \mathrm{m} * 4)\big) ** 1100 \frac{\mathrm{kg}}{\mathrm{m^3}} \\ = 43{,}55\ \mathrm{kg}$$

$$c_{p\ \mathrm{polyurethane\ foam}} = 1760\,\frac{\mathrm{j}}{\mathrm{kg}}$$

$$C_{\mathrm{polyurethane\ foam}} = 43{,}55\ \mathrm{kg} * 840\,\frac{\mathrm{j}}{\mathrm{kg}} = 36582\ \mathrm{J}$$

$$R_2 = 0{,}25\,\frac{\mathrm{W}}{\mathrm{m^2K}} * (6{,}07\ \mathrm{m} * 6{,}61\ \mathrm{m}) - (1{,}05\ \mathrm{m} * 1{,}44\ \mathrm{m} * 4) = 8.5\,\frac{\mathrm{K}}{\mathrm{W}}$$

http://www.firebid.umd.edu/material-database.php

This is an approximation, an Isolated wall has a high resistance and low capacity.

Appendix 3: Matlab Scripts

StartSimulation.m

```
%*******************************************************
%   Initialize simulation, this script starts the simulation
%*******************************************************
    clear;

%*******************************************************
%   Variables for storing measurement
%*******************************************************
    Measurements = zeros(1000,9);
    MeasurementPointer = 0;

%*******************************************************
%   Set Room temperature
%*******************************************************
    RoomTemp = 10;

%*******************************************************
%   Set temperatures of the walls
%*******************************************************
    WallEntryTemp = RoomTemp;
    WallOutdoorTemp = RoomTemp;

%*******************************************************
%   Initialize variables, are overwritten by SimulateModel.m
%*******************************************************
    HeatingSpeed = 0;
    RadiatorTemp = 0;
    EntryTemp = 0;
    OutdoorTemp = 0;
    DoorOpen = false;
    ShutterOpen = false;
    PeopleInRoom = false;
    IncomingSunlight = false;

%*******************************************************
%   Set Measurment interval in seconds
%*******************************************************
    LengthMeasurement = 60;

%*******************************************************
%   Number of States that influence the Room
%*******************************************************
    NumberOfStates = 4;

%*******************************************************
%   Matrix will store the influece of each different state combination
%*******************************************************
    RelationMatrix = zeros(16,9);

%*******************************************************
%   Start with simulation of the model
%*******************************************************
run('SimulateModel.m');
```

RunSimulationRoom.m

```
%********************************************************
%   Simulate State Space
%********************************************************

%********************************************************
%   Set Simulation time variable
%********************************************************
    t = 0:0.1:LengthMeasurement;

%********************************************************
%   Set Values Capacitors
%********************************************************
    Croom =  146257.65;
    C1 = 73237.22;
    C2 =  0.0001;
    C3 =  28871;
    C4 =  525800;
    C5 =  525800;
    C6 =  6778560;
    C7 =  6778560;
    C8 =  0.0001;

%********************************************************
%   Set Values resistors
%********************************************************
    Rrad = .001;
    R1 = 0.058 / 2 + (1 - ShutterOpen) * 0.058;
    R2 = 100;
    R3 = 0.00029 / 2 + (1 - DoorOpen) * 0.003;
    R4 = 0.00214 / 2;
    R5 = 0.00214 / 2;
    R6 = 0.0274 / 2;
    R7 = 0.0274 / 2;
    R8 = 100;

%********************************************************
%   Set Values Sources
%********************************************************
    Se = ((RadiatorTemp - OutdoorTemp) + DoorOpen * (EntryTemp -
OutdoorTemp) + PeopleInRoom * (35.5 - OutdoorTemp) + IncomingSunlight * (25
- OutdoorTemp)) / (1 + DoorOpen + PeopleInRoom + IncomingSunlight);
    Se3 = EntryTemp - OutdoorTemp;
    Se4 = EntryTemp - OutdoorTemp;
    Se5 = EntryTemp - OutdoorTemp;
    Se6 = EntryTemp - OutdoorTemp;
    Se7 = EntryTemp - OutdoorTemp;
    Se8 = EntryTemp - OutdoorTemp;

%********************************************************
%   Make A Matrix
%********************************************************
    A = zeros(9,9);
    A(1,1) = -(1/Croom) * ((1/Rrad) +
(1/R1)+(1/R2)+(1/R3)+(1/R4)+(1/R5)+(1/R6)+(1/R7)+(1/R8));
    A(1,2) = 1/(R1 * Croom);
    A(1,3) = 1/(R2 * Croom);
    A(1,4) = 1/(R3 * Croom);
    A(1,5) = 1/(R4 * Croom);
    A(1,6) = 1/(R5 * Croom);
    A(1,7) = 1/(R6 * Croom);
    A(1,8) = 1/(R7 * Croom);
    A(1,9) = 1/(R8 * Croom);

    A(2,1) = 1/(R1 * C1);
    A(2,2) = -2/(R1 * C1);

    A(3,1) = 1/(R2 * C2);
    A(3,3) = -2/(R2 * C2);

    A(4,1) = 1/(R3 * C3);
    A(4,4) = -2/(R3 * C3);

    A(5,1) = 1/(R4 * C4);
    A(5,5) = -2/(R4 * C4);

    A(6,1) = 1/(R5 * C5);
    A(6,6) = -2/(R5 * C5);

    A(7,1) = 1/(R6 * C6);
    A(7,7) = -2/(R6 * C6);

    A(8,1) = 1/(R7 * C7);
    A(8,8) = -2/(R7 * C7);

    A(9,1) = 1/(R8 * C8);
    A(9,9) = -2/(R8 * C8);

%*******************************************************
%   Make B Matrix
%*******************************************************
    B = zeros(9,7);
    B(1,1) = 1/(Rrad * Croom);
    B(4,2) = 1/(R3 * C3);
    B(5,3) = 1/(R4 * C4);
    B(6,4) = 1/(R5 * C5);
    B(7,5) = 1/(R6 * C6);
    B(8,6) = 1/(R7 * C7);
    B(9,7) = 1/(R8 * C8);

%*******************************************************
%   Make C Matrix
%*******************************************************
    C = zeros(1,9);
    C(1,1) = 1;

%*******************************************************
%   Make D Matrix
%*******************************************************
    D = 0;

%*******************************************************
%   Set initial values for the states, are the state values at the end of
%   the previous simulation
%*******************************************************
    x0 = [RoomTemp  - OutdoorTemp,WallOutdoorTemp -
OutdoorTemp,WallOutdoorTemp - OutdoorTemp,WallEntryTemp -
OutdoorTemp,WallEntryTemp - OutdoorTemp,WallEntryTemp -
OutdoorTemp,WallEntryTemp - OutdoorTemp,WallEntryTemp -
OutdoorTemp,WallEntryTemp - OutdoorTemp];

%********************************************************
%   Set input array for simulation
%********************************************************
    u = zeros(length(t),7);
    u(1:length(t),1)  = Se;
    u(1:length(t),2)  = Se3;
    u(1:length(t),3)  = Se4;
    u(1:length(t),4)  = Se5;
    u(1:length(t),5)  = Se6;
    u(1:length(t),6)  = Se7;
    u(1:length(t),7)  = Se8;

%********************************************************
%   Create and Simulate State space for Room Temperature
%********************************************************
    sys = ss(A,B,C,D);
    y = lsim(sys,u,t,x0);

%********************************************************
%   Create and Simulate State space for Wall outdoor temperature
%********************************************************
    Cmat2 = zeros(1,9);
    Cmat2(1,2)  = 1;

    sys = ss(A,B,Cmat2,D);
    y2 = lsim(sys,u,t,x0);

%********************************************************
%   Create and Simulate State space for wall Entry temperature
%********************************************************
    Cmat3 = zeros(1,9);
    Cmat3(1,4)  = 1;

    sys = ss(A,B,Cmat3,D);
    y3 = lsim(sys,u,t,x0);

%********************************************************
%   Assign output values of State Space to output variables
%********************************************************
    RoomTemp        = y(length(t))   + OutdoorTemp;
    WallOutdoorTemp = y2(length(t))  + OutdoorTemp;
    WallEntryTemp   = y3(length(t))  + OutdoorTemp;
```

FindHeatingTime.m

```
%*********************************************************************
%   Determine Time needed for the set of the state
%*********************************************************************
    function y =
FindHeatingTime(Measurement,NumberOfStates,RelationMatrix,RoomGoalTemp)
    %*********************************************************************
    %   Initialize variables
    %*********************************************************************
    tempvar = Type(Measurement,NumberOfStates) + 1;
    HeatingTime = 0;

    %*********************************************************************
    %   Integrate temperature to get time needed to heat the room
    %*********************************************************************
    if (RelationMatrix(tempvar,4) ~= 3)
        for Temp = Measurement(5):0.001:RoomGoalTemp
            TotalHeatingSpeed = 0;
            TotalHeatingSpeed = TotalHeatingSpeed +
GetHeatingSpeedEntry(Measurement,RelationMatrix,tempvar,Temp);
            TotalHeatingSpeed = TotalHeatingSpeed +
GetHeatingSpeedRadiator(Measurement,RelationMatrix,tempvar,Temp);
            TotalHeatingSpeed = TotalHeatingSpeed +
GetHeatingSpeedOutdoor(Measurement,RelationMatrix,tempvar,Temp);
            HeatingTime = HeatingTime + ((RoomGoalTemp -
Temp)/TotalHeatingSpeed) * 0.001;
        end
        y = HeatingTime;
    else
        y = 60 * 30;
    end
    end

%*********************************************************************
%   Determine the heating speed influence of Entry for a certain set of
state
%*********************************************************************
    function a =
GetHeatingSpeedEntry(Measurement,RelationMatrix,Type,RoomTemp)
    HeatingSpeed = 0;
    switch RelationMatrix(Type,1)
        case 1
            HeatingSpeed = HeatingSpeed + (Measurement(7) - RoomTemp -
RelationMatrix(Type,3)) * RelationMatrix(Type,2);
        case 2
            if (Measurement(7) - RoomTemp - RelationMatrix(Type,3) > 0)
                HeatingSpeed = HeatingSpeed + (Measurement(7) - RoomTemp -
RelationMatrix(Type,3))^2 * RelationMatrix(Type,2);
            else
                HeatingSpeed = HeatingSpeed + -(Measurement(7) - RoomTemp -
RelationMatrix(Type,3))^2 * RelationMatrix(Type,2);
            end
        case 0.5
            if (Measurement(7) - RoomTemp - RelationMatrix(Type,3) > 0)
                HeatingSpeed = HeatingSpeed + abs(Measurement(7) - RoomTemp
- RelationMatrix(Type,3))^.5 * RelationMatrix(Type,2);
            else
                HeatingSpeed = HeatingSpeed + -abs(Measurement(7) -
RoomTemp - RelationMatrix(Type,3))^.5 * RelationMatrix(Type,2);
            end
    end
    a = HeatingSpeed;
    end
%*********************************************************************
%   Determine the heating speed influence of Radiator for a certain set of
state
%*********************************************************************
    function b =
GetHeatingSpeedRadiator(Measurement,RelationMatrix,Type,RoomTemp)
    HeatingSpeed = 0;
    switch RelationMatrix(Type,4)
        case 0
            HeatingSpeed = HeatingSpeed + (Measurement(6) - RoomTemp -
RelationMatrix(Type,6)) * RelationMatrix(Type,5);
        case 1
            if (Measurement(6) - RoomTemp - RelationMatrix(Type,6) > 0)
                HeatingSpeed = HeatingSpeed + (Measurement(6) - RoomTemp -
RelationMatrix(Type,6))^2 * RelationMatrix(Type,5);
            else
                HeatingSpeed = HeatingSpeed + -(Measurement(6) - RoomTemp -
RelationMatrix(Type,6))^2 * RelationMatrix(Type,5);
            end
        case 2
            if (Measurement(6) - RoomTemp - RelationMatrix(Type,6) > 0)
                HeatingSpeed = HeatingSpeed + abs(Measurement(6) - RoomTemp
- RelationMatrix(Type,6))^.5 * RelationMatrix(Type,5);
            else
                HeatingSpeed = HeatingSpeed + -abs(Measurement(6) -
RoomTemp - RelationMatrix(Type,6))^.5 * RelationMatrix(Type,5);
            end
    end
    b = HeatingSpeed;
    end
%*********************************************************************
%   Determine the heating speed influence of Outdoor temperature for a
certain set of state
%*********************************************************************
    function c =
GetHeatingSpeedOutdoor(Measurement,RelationMatrix,Type,RoomTemp)
    HeatingSpeed = 0;
    switch RelationMatrix(Type,7)
        case 0
            HeatingSpeed = HeatingSpeed + (Measurement(8) - RoomTemp -
RelationMatrix(Type,9)) * RelationMatrix(Type,8);
        case 1
            if (Measurement(8) -RoomTemp - RelationMatrix(Type,9) > 0)
                HeatingSpeed = HeatingSpeed + (Measurement(8) - RoomTemp -
RelationMatrix(Type,9))^2 * RelationMatrix(Type,8);
            else
                HeatingSpeed = HeatingSpeed + -(Measurement(8) - RoomTemp -
RelationMatrix(Type,9))^2 * RelationMatrix(Type,8);
            end
        case 2
            if (Measurement(8) - RoomTemp - RelationMatrix(Type,9)> 0)
                HeatingSpeed = HeatingSpeed + abs(Measurement(8) - RoomTemp
- RelationMatrix(Type,9))^.5 * RelationMatrix(Type,8);
            else
                HeatingSpeed = HeatingSpeed + -abs(Measurement(8) -
RoomTemp - RelationMatrix(Type,9))^.5 * RelationMatrix(Type,8);
            end
    end
    c = HeatingSpeed;
```

```
end
```

SaveMeasurement.m

```
%*****************************************************************
%   Save measurement in matrix
%*****************************************************************
    disp('Save Measurement...');

%*******************************************************************
%   Determine Heating Speed of Measurement
%*******************************************************************
    if (MeasurementPointer > 0)
        Measurements(MeasurementPointer,9) = (RoomTemp - 
Measurements(MeasurementPointer,5))/LengthMeasurement;
    end

%*******************************************************************
%   Set Pointer to correct location
%*******************************************************************
    if (MeasurementPointer >= 1000)
       circshift(Measurements,-1);
    else
       MeasurementPointer = MeasurementPointer + 1;
    end

%*******************************************************************
%   Save measurement
%*******************************************************************
    Measurements(MeasurementPointer,1) = DoorOpen;
    Measurements(MeasurementPointer,2) = ShutterOpen;
    Measurements(MeasurementPointer,3) = PeopleInRoom;
    Measurements(MeasurementPointer,4) = IncomingSunlight;
    Measurements(MeasurementPointer,5) = RoomTemp;
    Measurements(MeasurementPointer,6) = RadiatorTemp;
    Measurements(MeasurementPointer,7) = EntryTemp;
    Measurements(MeasurementPointer,8) = OutdoorTemp;
    Measurements(MeasurementPointer,9) = 0;

%*******************************************************************
%   Process Measurement
%*******************************************************************
   if(MeasurementPointer > 1)
        run('ProcessMeasurement.m');
   end
disp('Measurement saved');
```

Type.m

```
%**********************************************************
%   Determine which set of states the measurement has (from 0 - 15)
%**********************************************************
    function y = Type(Measurement,NumberOfStates)
        tempvar = 0;
        for m = 1:1:NumberOfStates
            if (Measurement(m) == 1)
                tempvar = tempvar + 2^(m - 1);
            end
        y = tempvar;
        end
    end
```

GetRelation.m

```
function a =
GetRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasurement
)
%*******************************************************************
%   Determine Properties when linear
%*******************************************************************
    EntryLinear     =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,7,1);
    RadiatorLinear  =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,6,1);
    OutdoorLinear   =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,8,1);
%*******************************************************************
%   Determine Properties when Exponential
%*******************************************************************
     EntryExponential    =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,7,2);
     RadiatorExponential =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,6,2);
     OutdoorExponential  =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,8,2);
%*******************************************************************
%   Determine Properties when Root
%*******************************************************************
     EntryRoot    =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,7,0.5);
     RadiatorRoot =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,6,0.5);
     OutdoorRoot  =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,8,0.5);
%*******************************************************************
%   Determine Which relation each component has
%*******************************************************************
    EntryRelation =
GetEntryRelation(EntryLinear,EntryExponential,EntryRoot);
    RadiatorRelation =
GetRadiatorRelation(RadiatorLinear,RadiatorExponential,RadiatorRoot);
    OutdoorRelation =
GetOutdoorRelation(OutdoorLinear,OutdoorExponential,OutdoorRoot);

%*******************************************************************
%   Output Matrix
%*******************************************************************
    a = [EntryRelation,RadiatorRelation,OutdoorRelation];
end

function q =
GetPointsRelation(Measurements,MeasurementPointer,NumberOfStates,StateMeasu
rement,Temperature,RelationPower)
%*******************************************************************
%   Set Variables
```

```
%*******************************************************************
    Total = 0;
    Average = 0;
    AveragePoints = 0;
    Items = 0;
    Points = 0;
    PointsMeas = 0;
    ItemsOffset = 0;
    total_measurements = 0;

%*******************************************************************
%   Display debug
%*******************************************************************
    disp('-------------------------------------');
    disp(StateMeasurement);
    disp(total_measurements);

%*******************************************************************
%   Average of normalization of all the measurements
%*******************************************************************
    for k = 1:1:MeasurementPointer
    %*******************************************************************
    %   Check state of measurement
    %*******************************************************************
        if (Type(Measurements(k,1:9),NumberOfStates) == StateMeasurement)

%*******************************************************************
        %   Check if 2 of three dTs are 0

%*******************************************************************
            if (UsefullMeasurement(Measurements(k,1:9),Temperature) == 1)

%*******************************************************************
            %   Check if signs of heating speed and dT are the same

%*******************************************************************
                if (Measurements(k,9) > 0.001)
                    if ((Measurements(k,Temperature) - Measurements(k,5)) >
0)
                        Total = Total + (Measurements(k,9) /
((abs(Measurements(k,Temperature) - Measurements(k,5)))^RelationPower));
                        Items = Items + 1;
                    end
                elseif (Measurements(k,9) < -0.001)
                    if ((Measurements(k,Temperature) - Measurements(k,5)) <
0)
                        Total = Total + (Measurements(k,9) / (-
((abs(Measurements(k,Temperature) - Measurements(k,5)))^RelationPower)));
                        Items = Items + 1;
                    end
                end
            end
        end
    end
    %*******************************************************************
    %   if there were 1 or more usefull measurements the average slope is
    %   calculated
    %*******************************************************************
    if (Items > 0)
        Average = Total / Items;
```

```
%*****************************************************************
        %   Check how Close Measurement is to the Average

%*****************************************************************
        for l = 1:1:MeasurementPointer
            if (Type(Measurements(l,1:9),NumberOfStates) ==
StateMeasurement)
                if (UsefullMeasurement(Measurements(l,1:9),Temperature) ==
1)

%*****************************************************************
                        %   Check if signs correspond

%*****************************************************************
                        if (Measurements(l,9) > 0.001)
                            if ((Measurements(l,Temperature) -
Measurements(l,5)) > 0)
                                PointCorrected = (Measurements(l,9) /
((abs(Measurements(l,Temperature) - Measurements(l,5)))^RelationPower));
                                PointsMeas = (1 - abs((PointCorrected -
Average))) * 100;
                                if (PointsMeas < 0)
                                    PointsMeas = 0;
                                end
                                Points = Points + PointsMeas;
                                PointsMeas = 0;
                            end
                        elseif (Measurements(l,9) < -0.001)
                            if ((Measurements(l,Temperature) -
Measurements(l,5)) < 0)
                                PointCorrected = (Measurements(l,9) / (-
((abs(Measurements(l,Temperature) - Measurements(l,5)))^RelationPower)));
                                PointsMeas = (1 - abs((PointCorrected -
Average))) * 100;
                                if (PointsMeas < 0)
                                    PointsMeas = 0;
                                end
                                Points = Points + PointsMeas;
                                PointsMeas = 0;
                            end
                        end
                end
            end
        end
        AveragePoints = Points / Items;

%*****************************************************************
        %   Output Value

%*****************************************************************
        q = [AveragePoints,Average,0];
    else

%*****************************************************************
        %   Output Value no Items available

%*****************************************************************
        q = [0,0,0];
    end
end
```

```
function k = GetEntryRelation(EntryLinear,EntryExponential,EntryRoot)
%*******************************************************************
%   Initialize variables
%*******************************************************************
EntryRelation = 0;
EntryFactor = 0;

%*******************************************************************
%   Test which of the entries has the highest number of points
%*******************************************************************
    if (EntryRoot(1) > EntryExponential(1))
        if (EntryRoot(1) > EntryLinear(1))
            EntryRelation = 0.5;
            EntryFactor = EntryRoot(2);
            EntryOffset = EntryRoot(3);
        else
            EntryRelation = 1;
            EntryFactor = EntryLinear(2);
            EntryOffset = EntryLinear(3);
        end
    else
        if (EntryExponential(1) > EntryLinear(1))
            EntryRelation = 2;
            EntryFactor = EntryExponential(2);
            EntryOffset = EntryExponential(3);
        else
            EntryRelation = 1;
            EntryFactor = EntryLinear(2);
            EntryOffset = EntryLinear(3);
        end
    end
k = [EntryRelation,EntryFactor,EntryOffset];
end
function l =
GetRadiatorRelation(RadiatorLinear,RadiatorExponential,RadiatorRoot)
RadiatorRelation = 0;
RadiatorFactor = 0;

    if (RadiatorRoot(1) > RadiatorExponential(1))
        if (RadiatorRoot(1) > RadiatorLinear(1))
            RadiatorRelation = 0.5;
            RadiatorFactor = RadiatorRoot(2);
            RadiatorOffset = RadiatorRoot(3);
        else
            RadiatorRelation = 1;
            RadiatorFactor = RadiatorLinear(2);
            RadiatorOffset = RadiatorLinear(3);
        end
    else
        if (RadiatorExponential(1) > RadiatorLinear(1))
            RadiatorRelation = 2;
            RadiatorFactor = RadiatorExponential(2);
            RadiatorOffset = RadiatorExponential(3);
        else
            RadiatorRelation = 1;
            RadiatorFactor = RadiatorLinear(2);
            RadiatorOffset = RadiatorLinear(3);
        end
    end
l = [RadiatorRelation,RadiatorFactor,RadiatorOffset];
```

```
end
function m =
GetOutdoorRelation(OutdoorLinear,OutdoorExponential,OutdoorRoot)
OutdoorRelation = 0;
OutdoorFactor = 0;

    if (OutdoorRoot(1) > OutdoorExponential(1))
        if (OutdoorRoot(1) > OutdoorLinear(1))
            OutdoorRelation = 0.5;
            OutdoorFactor = OutdoorRoot(2);
            OutdoorOffset = OutdoorRoot(3);
        else
            OutdoorRelation = 1;
            OutdoorFactor = OutdoorLinear(2);
            OutdoorOffset = OutdoorLinear(3);
        end
    else
        if (OutdoorExponential(1) > OutdoorLinear(1))
            OutdoorRelation = 2;
            OutdoorFactor = OutdoorExponential(2);
            OutdoorOffset = OutdoorExponential(3);
        else
            OutdoorRelation = 1;
            OutdoorFactor = OutdoorLinear(2);
            OutdoorOffset = OutdoorLinear(3);
        end
    end
m = [OutdoorRelation,OutdoorFactor,OutdoorOffset];
end
```

UsefullMeasurement.m

```
%*******************************************************************
%   Test if measurement has 2 or more dTs that equal 0
%*******************************************************************
function y = UsefullMeasurement(Measurement,temperature)
    RadiatorTempZero = 0;
    EntryTempZero = 0;
    OutdoorTempZero = 0;

    if (abs(Measurement(6) - Measurement(5)) <= 0.001)
        RadiatorTempZero = 1;
    end
    if (abs(Measurement(7) - Measurement(5)) <= 0.001)
        EntryTempZero = 1;
    end
    if (abs(Measurement(8) - Measurement(5)) <= 0.001)
        OutdoorTempZero = 1;
    end

    if (temperature == 6)
        if (EntryTempZero == 1 && OutdoorTempZero == 1)
            y = 1;
        else
            y = 0;
        end
    elseif (temperature == 7)
        if (RadiatorTempZero == 1 && OutdoorTempZero == 1)
            y = 1;
        else
            y = 0;
        end
    elseif (temperature == 8)
        if (RadiatorTempZero == 1 && EntryTempZero == 1)
            y = 1;
        else
            y = 0;
        end
    else
        y = 0;
    end
end
```

SimulateModel.m

```
%********************************************************
%   Start with simulation of the model
%********************************************************

%********************************************************
%   Number of timesteps that are simulated
%********************************************************
    NumberOfMinutes = 300;

%********************************************************
%   Initialise Output Graphs
%********************************************************
    RoomTempGraph = zeros(NumberOfMinutes,1);
    RoomWallEntryTempGraph = zeros(NumberOfMinutes,1);
    RoomWallOutdoorTempGraph = zeros(NumberOfMinutes,1);

%********************************************************
%   Initialise input graphs
%********************************************************
    EntryTempGraph = zeros(NumberOfMinutes,1);
        %EntryTempGraph(1:NumberOfMinutes) = 20;
        EntryTempGraph(1:NumberOfMinutes / 3) = 22;
        EntryTempGraph(NumberOfMinutes / 3:2 * NumberOfMinutes / 3) = 25;
        EntryTempGraph(2 * NumberOfMinutes / 3:NumberOfMinutes) = 20;
    OutdoorTempGraph = zeros(NumberOfMinutes,1);
        OutdoorTempGraph(1:NumberOfMinutes) = 20;
        OutdoorTempGraph(1:2 * NumberOfMinutes / 3) = 14;
        OutdoorTempGraph(2 * NumberOfMinutes / 3:NumberOfMinutes) = 16;
    PeopleInRoomGraph = zeros(NumberOfMinutes,1);
        %PeopleInRoomGraph(1:250) = 1;
        %PeopleInRoomGraph(200:250) = 1;
    IncomingSunlightGraph = zeros(NumberOfMinutes,1);
        %IncomingSunlightGraph(45:60) = 1;
        %IncomingSunlightGraph(150:200) = 1;
    ShutterOpenGraph = zeros(NumberOfMinutes,1);
        %ShutterOpenGraph(75:85) = 1;
        %ShutterOpenGraph(1:NumberOfMinutes) = 1;
    DoorOpenGraph = zeros(NumberOfMinutes,1);
        %DoorOpenGraph(1:200) = 1;
        %DoorOpenGraph(50:80) = 1;
        %DoorOpenGraph(200:250) = 1;
    RadiatorTempGraph = zeros(NumberOfMinutes,1);
        RadiatorTempGraph(1:NumberOfMinutes) = 20;
        RadiatorTempGraph(2 * NumberOfMinutes / 3:NumberOfMinutes) = 28;
        RadiatorTempGraph(1:2 * NumberOfMinutes / 3) = 24;
    HeatingSpeedGraph = zeros(NumberOfMinutes,1);

%********************************************************
%   Simulate Model
%********************************************************
for i = 1:1:NumberOfMinutes - 1
    %********************************************************
    %   Set all the states and temperatures
    %********************************************************
        WallEntryTemp = EntryTemp;
        WallOutdoorTemp = OutdoorTemp;
        run('RunSimulationRoom.m');
        RoomTempGraph(i) = RoomTemp;
        RoomWallEntryTempGraph(i) = WallEntryTemp;
        RoomWallOutdoorTempGraph(i) = WallOutdoorTemp;
        RadiatorTemp = RadiatorTempGraph(i + 1);
        EntryTemp = EntryTempGraph(i + 1);
        OutdoorTemp = OutdoorTempGraph(i + 1);
        DoorOpen = DoorOpenGraph(i + 1);
        ShutterOpen = ShutterOpenGraph(i + 1);
        PeopleInRoom = PeopleInRoomGraph(i + 1);
        IncomingSunlight = IncomingSunlightGraph(i + 1);
        run('SaveMeasurement');
end
%******************************************************************
%   Calculate the relation matrix
%******************************************************************
    run('ProcessMeasurement.m')

%******************************************************************
%   <Improvement> to increase the number of usefull measurements
%******************************************************************
    matrix =
GetCorrectedMeas(Measurements,MeasurementPointer,RelationMatrix,NumberOfSta
tes);

%*******************************************************
%   Plot the simulation
%*******************************************************
    plot(RoomTempGraph); hold on;
    plot(EntryTempGraph); hold on;
    plot(OutdoorTempGraph); hold on;
    plot(RadiatorTempGraph); hold on;
    plot(DoorOpenGraph); hold on;
    plot(ShutterOpenGraph); hold on;
    plot(PeopleInRoomGraph); hold on;
    plot(IncomingSunlightGraph);
    %legend('Room Temp','Entry Temp')
    xlabel('Time(min)')
    ylabel('Temperature in celcius')
    title('Room temperature over time')
    legend('Room Temp','Entry Temp','Outdoor Temp','Radiator Temp','Door
Open','Shutter Open','People in Room','Incoming Sunlight')
    %legend('Room Temp','Entry Temp','Door Open','Shutter Open','People in
Room','Incoming Sunlight')
    %plot(Measurements(:,7) - Measurements(:,5),Measurements(:,9));
```

GetCorrectedMeas.m

```
%*****************************************************************
%   Subtract measurements to get more useful results
%*****************************************************************
function a =
GetCorrectedMeas(Measurements,MeasurementPointer,RelationMatrix,NumberOfSta
tes)
%*****************************************************************
%   Initialize variables
%*****************************************************************
    MatrixCorrectedMeas = zeros(1000,9);
    WritePosition = 1;
    Measurement = zeros(1,9);

%*****************************************************************
%   Start Subtraction
%*****************************************************************
for l = 1:1:2^NumberOfStates
    StateMeasurement = l - 1;
    %*****************************************************************
    %   Display Debug
    %*****************************************************************
    disp('--------------------------------------------')
    disp(StateMeasurement)
    disp('--------------------------------------------')

    %*****************************************************************
    %   determine first measurement k1
    %*****************************************************************
    for k1 = 1:1:MeasurementPointer
        if (Type(Measurements(k1,1:9),NumberOfStates) == StateMeasurement)

%*****************************************************************
            %   determine second measurement k2

%*****************************************************************
            for k2 = 1:1:MeasurementPointer
                if (Type(Measurements(k2,1:9),NumberOfStates) ==
StateMeasurement)
                    if (k1 ~= k2)
                        tempvar = 0;
                        Measurement(1:5) = Measurements(k1,1:5);
                        Measurement(6) = (Measurements(k1,6) -
Measurements(k1,5))^RelationMatrix(StateMeasurement + 1,1) -
(Measurements(k2,6) - Measurements(k2,5))^RelationMatrix(StateMeasurement +
1,1);
                        Measurement(7) = (Measurements(k1,7) -
Measurements(k1,5))^RelationMatrix(StateMeasurement + 1,4) -
(Measurements(k2,7) - Measurements(k2,5))^RelationMatrix(StateMeasurement +
1,4);
                        Measurement(8) = (Measurements(k1,8) -
Measurements(k1,5))^RelationMatrix(StateMeasurement + 1,7) -
(Measurements(k2,8) - Measurements(k2,5))^RelationMatrix(StateMeasurement +
1,7);
                        Measurement(9) = (Measurements(k1,9) -
Measurements(k2,9));
                        if (abs(Measurement(6)) < 0.001)
                            tempvar = tempvar + 1;
                        end
                        if (abs(Measurement(7)) < 0.001)
                            tempvar = tempvar + 1;
                        end
                        if (abs(Measurement(8)) < 0.001)
                            tempvar = tempvar + 1;
                        end
                        if (tempvar >= 2)
                            if (abs(Measurement(9)) > 0.0001)
                                MatrixCorrectedMeas(WritePosition, 1:9) =
Measurement(1:9);

                                WritePosition = WritePosition + 1;
                                disp('New Measurement!!');
                            end
                        else

%*******************************************************************
                            %   If 2 measurements where not usefull try
                            %   three k3

%*******************************************************************
                            for k3 = 1:1:MeasurementPointer
                                if
(Type(Measurements(k3,1:9),NumberOfStates) == StateMeasurement)
                                    if (k2 ~= k3 && k1 ~= k3)
                                        tempvar = 0;
                                        Measurement(6) =    Measurement(6)
- (Measurements(k3,6) - Measurements(k3,5))^RelationMatrix(StateMeasurement
+ 1,1);
                                        Measurement(7) =    Measurement(7)
- (Measurements(k3,7) - Measurements(k3,5))^RelationMatrix(StateMeasurement
+ 1,4);
                                        Measurement(8) =    Measurement(8)
- (Measurements(k3,8) - Measurements(k3,5))^RelationMatrix(StateMeasurement
+ 1,7);
                                        Measurement(9) = (Measurement(9) -
Measurements(k3,9));
                                        if (abs(Measurement(6)) < 0.001)
                                            tempvar = tempvar + 1;
                                        end
                                        if (abs(Measurement(7)) < 0.001)
                                            tempvar = tempvar + 1;
                                        end
                                        if (abs(Measurement(8)) < 0.001)
                                            tempvar = tempvar + 1;
                                        end
                                        if (tempvar >= 2)
                                            if (abs(Measurement(9)) >
0.0001)

MatrixCorrectedMeas(WritePosition,1:9) = Measurement(1:9);
                                                WritePosition =
WritePosition + 1;

                                                disp('New Measurement!!');
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
```

```
    end
end
a = MatrixCorrectedMeas;
end
```

ProcessMeasurement.m

```
%*****************************************************************
%   Fill in the Relation matrix
%*****************************************************************
    for l = 1:1:2^NumberOfStates
    RelationMatrix(l,1:9) = GetRelation(Measurements,MeasurementPointer -
1,NumberOfStates,l - 1);
    end
```

Appendix 4: Python / Jython scripts

```
import time



class SelfLearningSystem(Rule):

  def __init__(self):

    self.logger = oh.getLogger("TestLoger")

    self.room_temp = get_room_temperature()

    self.radiator_temp = get_radiator_temperature()

    self.entry_temp = get_entry_temperature()

    self.outdoor_temp = get_outdoor_temperature()

    self.door_open = get_door_open()

    self.shutter_open = get_shutter_open()

    self.people_in_room = get_people_in_room()

    self.incoming_sunlight = get_incoming_sunlight()

    self.number_of_states = 4

    self.goal_temp = get_goal_temperature()

    self.previous_goal_temp = get_goal_temperature()

    self.goal_time = get_goal_time()

    self.time_needed = 0;

    self.relation_matrix = [

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

           ]
```

```
        self.measurements = [
                    [
                        self.door_open, self.shutter_open, self.people_in_room, self.incoming_sunlight,
                        self.room_temp, self.radiator_temp, self.entry_temp, self.outdoor_temp, 0
                    ]
                    ]
        self.state_room = [
                        self.door_open, self.shutter_open, self.people_in_room, self.incoming_sunlight,
                        self.room_temp, self.radiator_temp, self.entry_temp, self.outdoor_temp
                  ]
        self.time_last_meas = time.clock()


    def save_measurement(self, length_measurement):
        # ****************************************************
        #   Save the measurement in list
        # ****************************************************


        # ****************************************************************
        #   Determine Heating Speed of Measurement
        # ****************************************************************
        if len(self.measurements) >= 1:
            if length_measurement != 0:
                self.measurements[(len(self.measurements) - 1)][8] = (self.room_temp - 
self.measurements[(len(self.measurements)) - 1][4]) / length_measurement
            else:
                self.measurements[(len(self.measurements) - 1)][8] = 0


        # ****************************************************************
        #   Set Pointer to correct location
        # ******************************************************************
        if len(self.measurements) >= 1000:
            del self.measurements[0]

        # ******************************************************************
        #   Save measurement
        # ******************************************************************
        measurement = [self.door_open, self.shutter_open, self.people_in_room,
self.incoming_sunlight, self.room_temp, self.radiator_temp, self.entry_temp, self.outdoor_temp, 0]
        self.measurements.append(measurement)

    def get_relation(self, target_state):
        # ******************************************************************
        #    Determine Properties when linear
        # ******************************************************************
        entry_linear = self.get_points_relation(target_state, 6, 1)
        radiator_linear = self.get_points_relation(target_state, 5, 1)
        outdoor_linear = self.get_points_relation(target_state, 7, 1)
        # ******************************************************************
        #    Determine Properties when Exponential
        # ******************************************************************
        entry_exponential = self.get_points_relation(target_state, 6, 2)
        radiator_exponential = self.get_points_relation(target_state, 5, 2)
        outdoor_exponential = self.get_points_relation(target_state, 7, 2)
        # ******************************************************************
        #   Determine Properties when Root
        # ******************************************************************
        entry_root = self.get_points_relation(target_state, 6, 0.5)

        radiator_root = self.get_points_relation(target_state, 5, 0.5)

        outdoor_root = self.get_points_relation(target_state, 7, 0.5)

        # ****************************************************************

        #   Determine Which relation each component has

        # ****************************************************************

        entry_relation = get_best_relation(entry_linear, entry_exponential, entry_root)

        radiator_relation = get_best_relation(radiator_linear, radiator_exponential, radiator_root)

        outdoor_relation = get_best_relation(outdoor_linear, outdoor_exponential, outdoor_root)

        # ****************************************************************

        #   Output Matrix

        # ****************************************************************

        if (

            entry_linear + entry_exponential + entry_root != 0 and

            radiator_linear + radiator_exponential + radiator_root != 0 and

            outdoor_linear + outdoor_exponential + outdoor_root != 0

           ):

            return entry_relation + radiator_relation + outdoor_relation + [1]

        else:

            return entry_relation + radiator_relation + outdoor_relation + [0]


    def process_measurement(self):

            for state in range(0, 2**self.number_of_states):

                relation = self.get_relation(state)

                self.relation_matrix[state] = relation


    def get_points_relation(self, target_state, target_temperature, relation_power):
```

```
# *****************************************************************

#    Set Variables

# *****************************************************************

total = 0

average_points = 0

offset = 0

items_offset = 0

items = 0

points = 0

points_meas = 0

# *****************************************************************

#    Calculate Offset value

# *****************************************************************

for k in range(0, len(self.measurements) - 1):

    if self.get_state(self.measurements[k]) == target_state:

        if abs(self.measurements[k][8]) <= 0.0001:

            offset += self.measurements[k][target_temperature] - self.measurements[k + 1][4]

            items_offset += 1

if items_offset > 0:

    offset_average = offset / items_offset

else:

    offset_average = 0

# *****************************************************************

#    Average of normalization of all the measurements

# *****************************************************************

for k in range(0, len(self.measurements)):

    if self.get_state(self.measurements[k]) == target_state:
            if abs(self.measurements[k][8]) > 0.0001:

                if (self.measurements[k][target_temperature] - self.measurements[k + 1][4] -
offset_average) > 0:

                    temp_var = abs(self.measurements[k][target_temperature] -

                                   self.measurements[k + 1][4] - offset_average)

                    total += self.measurements[k][8] / (temp_var ** relation_power)

                    items += 1

                elif (self.measurements[k][target_temperature] - self.measurements[k + 1][4] -
offset_average) < 0:

                    temp_var = abs(self.measurements[k][target_temperature] -

                                   self.measurements[k + 1][4] - offset_average)

                    total += (self.measurements[k][8] / (-(temp_var ** relation_power)))

                    items += 1

        if items > 0:

            average = total / items

            # *******************************************************************

            #    Check how Close Measurement is to the Average

            # *******************************************************************

            for k in range(0, len(self.measurements)):

                if self.get_state(self.measurements[k]) == target_state:

                    if abs(self.measurements[k][8]) > 0.0001:

                        if (self.measurements[k][target_temperature] - self.measurements[k + 1][4] -
offset_average) != 0:

                            temp_var = abs(self.measurements[k][target_temperature] -

                                           self.measurements[k + 1][4] - offset_average)

                            point_corrected = (self.measurements[k][8]) / (temp_var ** relation_power)

                            points_meas = (1 - abs((point_corrected - average) / 2)) * 100

                        if points_meas < 0:

                            points_meas = 0
                    points += points_meas

                    points_meas = 0

                average_points = points / items

            # *****************************************************************

            #   Output Value

            # *****************************************************************

            return [average_points, average, offset_average]

        else:

            # *****************************************************************

            #   Output Value no Items available

            # *****************************************************************

            return [0, 0, offset_average]


def get_state(self, measurement):

        state = 0

        for m in range(0, self.number_of_states):

            if measurement[m] == 1:

                state += 2 ** m

        return state


def find_heating_time(self, initial_values, room_goal_temp):

        heating_time = 0

        state = self.get_state(initial_values)

        if self.relation_matrix[state][9] == 1:

            for temp in [float(j) / 1000 for j in range(int(initial_values[4] * 1000), int(1000 *
room_goal_temp), 1)]:

                total_heating_speed = 0

                total_heating_speed += self.get_heating_speed(initial_values, temp, 0)
                total_heating_speed += self.get_heating_speed(initial_values, temp, 1)
                total_heating_speed += self.get_heating_speed(initial_values, temp, 2)
                if total_heating_speed != 0:
                    heating_time += ((room_goal_temp - temp)/total_heating_speed) * 0.001
                else:
                    heating_time = 0
            return heating_time
        else:
            return -1


def get_heating_speed(self, measurement, current_temp, type):
        heating_speed = 0
        state = self.get_state(measurement)
        if type == 0:
            target_temp = 6
        elif type == 1:
            target_temp = 5
        else:
            target_temp = 7
        value = measurement[target_temp] - current_temp
        if self.relation_matrix[state][type * 3] == 0:
            heating_speed += (value - self.relation_matrix[state][type * 3 + 2]) * \
                        self.relation_matrix[state][type * 3 + 1]
        elif self.relation_matrix[state][type * 3] == 1:
                if value - self.relation_matrix[state][type * 3 + 2] > 0:
                    heating_speed += (value - self.relation_matrix[state][type * 3 + 2])**2 * \
                                self.relation_matrix[state][type * 3 + 1]
                else:
                    heating_speed -= (value - self.relation_matrix[state][type * 3 + 2])**2 * \
                            self.relation_matrix[state][type * 3 + 1]
            else:
                if value - self.relation_matrix[state][type * 3 + 2] > 0:
                    heating_speed += abs(value - self.relation_matrix[state][type * 3 + 2])**.5 * \
                            self.relation_matrix[state][type * 3 + 1]
                else:
                    heating_speed -= -abs(value - self.relation_matrix[state][type * 3 + 2])**.5 * \
                            self.relation_matrix[state][type * 3 + 1]
            return heating_speed

        def getEventTrigger(self):
            return [
                ChangedEventTrigger("TEMP_ENTRY", None, None),
                ChangedEventTrigger("TEMP_RADIATOR", None, None),
                ChangedEventTrigger("TEMP_OUTDOOR", None, None),
                ChangedEventTrigger("DOOR_OPEN", None, None),
                ChangedEventTrigger("SHUTTER_OPEN", None, None),
                ChangedEventTrigger("INCOMING_SUNLIGHT", None, None),
                ChangedEventTrigger("PEOPLE_IN_ROOM", None, None),
                TimerTrigger("*/5 * * * * ?")
            ]

        def execute(self, event):
            if event.triggerType == TriggerType.TIMER:
                self.goal_temp = get_goal_temperature()
                self.goal_time = get_goal_time()
                time_difference = ((60 * 24) + (self.goal_time - (time.gmtime().tm_hour * 60 +
            time.gmtime().tm_min + 60))) % (60 * 24)
                # When goal temperature is 0.5 degrees higher than current room temperature
                if self.goal_temp - get_room_temperature() > 0.5:
                self.state_room = [get_door_open(), get_shutter_open(), get_people_in_room(),
            get_incoming_sunlight(), get_room_temperature(), 25, get_entry_temperature(),
            get_outdoor_temperature()]
                self.time_needed = self.find_heating_time(self.state_room, self.goal_temp)
                # Check if the time needed is realistic
                if 30 <= self.time_needed <= 240:
                    # Check if the system should already start with heating
                    if time_difference <= self.time_needed / 60:
                        # Starting is needed
                        set_radiator_temperature(25)
                        self.previous_goal_temp = self.goal_temp
                        self.logger.info("Heating...")
                        self.logger.info("Got:" + str(time_difference - self.time_needed / 60) + " Minutes")
                    else:
                        # Still got time for heating
                        self.logger.info("Passive...")
                        self.logger.info("Still got: " + str(time_difference - self.time_needed / 60) + "minutes")
                        self.logger.info("Time Needed to heat from " + str(get_room_temperature()) + " to " +
            str(self.goal_temp) + " is " + str(self.time_needed) + " seconds")
                        set_radiator_temperature(self.previous_goal_temp)
                # If time needed is not realistic use default value of 30 minutes
                else:
                    # Check if the system should already start with heating
                    if time_difference < 30:
                        # Starting is needed
                        set_radiator_temperature(25)
                        self.logger.info("Heating...")
                        self.previous_goal_temp = self.goal_temp
                        self.logger.info("Got:" + str(time_difference) + " Minutes")
                    else:
                        # Still got time for heating
                        self.logger.info("Passive...")
                        self.logger.info("Still Got:" + str(time_difference - 30) + " Minutes")
                        set_radiator_temperature(self.previous_goal_temp)
                # When goal temperature is 0.5 degrees lower than current room temperature
                elif self.goal_temp - get_room_temperature() < -0.5:
                self.state_room = [get_door_open(), get_shutter_open(), get_people_in_room(),
            get_incoming_sunlight(), get_room_temperature(), 14, get_entry_temperature(),
            get_outdoor_temperature()]
                self.time_needed = abs(self.find_heating_time(self.state_room, self.goal_temp))
                if 30 <= self.time_needed <= 240:
                    if time_difference <= self.time_needed / 60:
                        set_radiator_temperature(14)
                        self.logger.info("Cooling doing...")
                        self.previous_goal_temp = self.goal_temp
                        self.logger.info("Got:" + str(time_difference - self.time_needed / 60) + " Minutes")
                    else:
                        self.logger.info("Passive...")
                        self.logger.info("Still got: " + str(time_difference - self.time_needed / 60) + "minutes")
                        self.logger.info("Time Needed to cool down from " + str(get_room_temperature()) + "
            to " + str(self.goal_temp) + " is " + str(self.time_needed) + " seconds")
                        set_radiator_temperature(self.previous_goal_temp)
                else:
```

```
            if time_difference < 30:
                set_radiator_temperature(14)
                self.logger.info("Cooling Down...")
                self.previous_goal_temp = self.goal_temp
                self.logger.info("Got:" + str(time_difference) + "minutes")
            else:
                self.logger.info("Passive...")
                self.logger.info("Still Got:" + str(time_difference - 30) + "minutes")
                set_radiator_temperature(self.previous_goal_temp)
        # When goal temperature is reached
        else:
            self.logger.info("Passive...")
            self.logger.info("Goal Temperature is reached")
            self.logger.info("Room Temp: " + str(get_room_temperature()))
            self.logger.info("Goal Temp: " + str(self.goal_temp))
            set_radiator_temperature(self.goal_temp)


        #self.logger.info(str(self.time_needed))
        #self.logger.info(str(time.gmtime().tm_year))
        #self.logger.info(str(self.goal_time))
    else:
        self.logger.info("Event trigger {}", event)
        self.logger.info("Type of event is {} {}", event.triggerType, type(event.triggerType))
        self.logger.info("Is it a change? {}", event.triggerType == TriggerType.CHANGE)
        self.room_temp = get_room_temperature()
        self.entry_temp = get_entry_temperature()
        self.outdoor_temp = get_outdoor_temperature()
```

```
            self.radiator_temp = get_radiator_temperature()

            self.door_open = get_door_open()

            self.shutter_open = get_shutter_open()

            self.people_in_room = get_people_in_room()

            self.incoming_sunlight = get_incoming_sunlight()

            self.save_measurement(time.clock() - self.time_last_meas)

            self.process_measurement()

            self.time_last_meas = time.clock()

            self.logger.info(str(len(self.measurements)))



def getRules():

    return RuleSet([

        SelfLearningSystem()

    ])



def get_room_temperature():

    item = ItemRegistry.getItem("TEMPERATURE").state

    if str(item) != "Uninitialized":

        return float(str(item))

    else:

        return 0



def get_goal_temperature():

    item = ItemRegistry.getItem("TEMP_GOAL").state
    if str(item) != "Uninitialized":
        return float(str(item))
    else:
        return 0



def get_goal_time():
    time_min = 0
    item = ItemRegistry.getItem("TIME_GOAL_HOUR").state
    if str(item) != "Uninitialized":
        time_min = int(str(item)) * 60
    item = ItemRegistry.getItem("TIME_GOAL_MIN").state
    if str(item) != "Uninitialized":
        time_min += int(str(item))
    return time_min




def get_entry_temperature():
    item = ItemRegistry.getItem("TEMP_ENTRY").state
    if str(item) != "Uninitialized":
        return float(str(item))
    else:
        return 0



def get_radiator_temperature():
    item = ItemRegistry.getItem("TEMP_RADIATOR").state
    if str(item) != "Uninitialized":
        return float(str(item))
    else:
        return 0


def set_radiator_temperature(temperature):
    oh.postUpdate("TEMP_RADIATOR", str(temperature))


def get_outdoor_temperature():
    item = ItemRegistry.getItem("TEMP_OUTDOOR").state
    if str(item) != "Uninitialized":
        return float(str(item))
    else:
        return 0


def get_door_open():
    item = ItemRegistry.getItem("DOOR_OPEN").state
    if str(item) != "Uninitialized":
        if item == OnOffType.ON:
            return 1
        else:
            return 0
    else:
        return 0



def get_shutter_open():
    item = ItemRegistry.getItem("SHUTTER_OPEN").state
    if str(item) != "Uninitialized":
        if item == OnOffType.ON:
            return 1
        else:
            return 0
    else:
        return 0



def get_incoming_sunlight():
    item = ItemRegistry.getItem("INCOMING_SUNLIGHT").state
    if str(item) != "Uninitialized":
        if item == OnOffType.ON:
            return 1
        else:
            return 0
    else:
        return 0



def get_people_in_room():
    item = ItemRegistry.getItem("PEOPLE_IN_ROOM").state
    if str(item) != "Uninitialized":
        if item == OnOffType.ON:
            return 1
        else:
            return 0
    else:
        return 0



def get_best_relation(relation_1, relation_2, relation_3):
    if relation_1[0] > relation_2[0]:
        if relation_1[0] > relation_3[0]:
            return [relation_1[0], relation_1[1], relation_1[2]]
        else:
            return [relation_3[0], relation_3[1], relation_3[2]]
    elif relation_2[0] > relation_3[0]:
        return [relation_2[0], relation_2[1], relation_2[2]]
    else:
        return [relation_3[0], relation_3[1], relation_3[2]]
```

Sources

http://lpsa.swarthmore.edu/Systems/Thermal/SysThermalModel.html

https://github.com/openhab/openhab/wiki