Charles University in Prague
Faculty of Mathematics and Physics

DIPLOMA THESIS

Martin Kontsek

Počítačem usnadněné bydlení - Inteligentní vytápění

(Computer Assisted Living - Intelligent Heating Control)

Department of software engineering

Supervisor : Mgr. Jiří Iša, KTIML

Study programme : Information science

Study field : Database systems

I would like to thank my advisor Mgr. Jiří Iša for his helpful guidance, valuable and constructive advice and support in time of need.

I also thank Bc. Pavol Jusko for lending the needed hardware and expertise in connecting the make-shift heating system and Ing. Pavol Kontsek for assisting in the overall construction of the heating system as well as supplying additional electrical equipment.

I hereby declare that I have written the diploma thesis myself, using only cited sources. I agree with lending and distribution of the thesis.

Prague, 10th December 2009 Martin Kontsek

Contents

Název práce: Počítačem usnadněné bydlení
Autor: Martin Kontsek
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Jiří Iša, KTIML
E-mail vedoucího: jiri.isa@mff.cuni.cz

Abstrakt: Tématem práce je využití počítačových technologií pro usnadnění a vylepšení našeho života v domácnostech. Student zváží a porovná různé řídící postupy především s ohledem na dlouhodobé tepelné řízení budov. Vybrané postupy budou implementovány a jejich činnost konfrontována se v současnosti převládajícími reaktívnimi řídícími systémy a to ať už z pohledu výpočetní náročnosti, komfortu obyvatel budovy, nebo z pohledu nižší spotřeby energií potřebné k dosažení obdobných výsledků.

Klíčová slova: tepelné, řízení, PID, bydlení

Title: Computer assisted living
Author: Martin Kontsek
Department: Department of software engineering
Supervisor: Mgr. Jiří Iša, KTIML
Supervisor's e-mail address: jiri.isa@seznam.cz

Abstract: The goal of this work is to utilize computer technology for improvement and comfort of our lives in households. The student will consider and compare various control methods with respect to long term heating of buildings. Chosen methods will be implemented and their activity confronted with contemporary reactive control systems in comparison to computation complexity, home user comfort or from a energy-wise perspective needed to attain similar results.

Keywords: heat, control, PID, living

Chapter 1

Introduction

Since the discovery of fire, human kind has sought to heat their homes, shielding them from the harsh environment. One of the most important aspects of home heating has been the construction and maintenance of thermal insulation - the ability to contain a temperature gradient between the interior and exterior of buildings, respectively to keep a constant or user-controlled temperature of indoor areas regardless of outside conditions.

Lately, the cost of energy has gone up and with it the interest in advanced methods of heat regulation, namely better thermal insulation materials and heat regulation. Concerning heat regulation, one of the goals is to find an acceptable ratio between the economic and speed factor of regulation, in other words, to attain desired temperatures quickly and cost-effectively. Only in the recent decades has this been possible due to breakthroughs in heat sensor technology and computer networks.

Today, technology has advanced so far as to make these systems commercially available. However, there have been problems of achieving a stable temperature gradient, resulting in temperature oscillations which lower customer satisfaction as well as increase financial demand. These problems have various causes, such as insufficiencies in

1. Predictability of heat dissipation
2. Distribution of heat sensors
3. Amount of heat sensors
4. Distribution of heat from heat source

5. System programming and tuning

The purpose of this work is to address the first and fifth issue of such a system.

Distribution of heat sensors is a logistical problem and is unique for every heat area. Determining the best distribution is a matter of user preference as to wanting to know the temperature of a specific section. Having a sufficient amount of heat sensors is an economical issue. The ideal solution would be to have heat sensors in every point of the heating area while also having a function that would aggregate various temperatures into one meaningful value. In a typical home environment, having too many sensors would be costly and impractical. Distribution of heat from heat source should be understood as finding a way of getting hot air generated by the heat source to every corner of the heating area. Again, the ideal solution would be to have a very small heat source in every point of the heating area. This is clearly not feasible and probably not even possible without restricting movement of people inside the heated area. A real world example would be to have a fan above a water radiator blowing hot air throughout the heating area. All these issues are beyond the scope of this work and therefore will not be addressed.

Concerning the first issue, a heat model of a room is constructed and free variables describing the heat characteristics are found. This model is constructed quickly and reveals probable temperature variations that will appear in the near future. The fifth issue is system programming and tuning.

Imagine a house with such an intelligent heat control system installed. heat sensors are distributed and installed into different rooms, some are installed outside. Every room has a heat source (such as a water radiator) which is connected to a central server that can control the power input of these heat sources. A fireplace is located in the house, so is a small array of solar panels that use the sun's heat to warm the water inside them. Problems arise, for example, when the user wants to raise the temperature inside the house. During this process the outside temperature begins varying in such a manner, that the house overheats. Overheating does not necessarily have to result from outside temperatures, but also from a heat source of greater dimensions, which dissipates larger amounts of heat even after being turned off. The system recognizes this and cuts power to the heat sources, the temperature is decreased and before the system can recognize the decrease, the house is colder again and the situation repeats itself.

Although it is difficult to predict such behavior, an improvement may be achieved using a more innovative approach. There are four methods of control, which this work wishes to present:

1. ON/OFF control with temperature bounds
2. Proportional - Integral - Derivative (PID) control
3. Predictive control based on a linear heat model
4. Predictive control based on a non-linear heat model

PID control is quite an established theory, used heavily in control processes. It is simple, but robust and can be relied upon. PID has also been used in conjunction with neural networks for temperature control purposes [11]. ON/OFF control with temperature bounds is a method currently used that we will be working with. The principle idea is having a desired temperature and a upper and lower desired temperature bound. When the ambient temperature drops below the lower bound, heating ensues. When the ambient temperature reaches the upper bound, heating stops. The problem here is to choose the bounds in such a way as to not cause too much overheating (the bounds are too far apart) and also not having the heating turn off and on in rapid succession to avoid mechanical wear of used devices (the bounds are too close together).

Predictive control is a proposed paradigm for enhancing the control process. In this work we theoretically and experimentally present a few methods of this paradigm and compare it with other control methods. This type of control tries to predict the future temperature by finding parameters of a general heat model and then adjusts heating accordingly.

Chapter 2

From water heating to air heating

2.1 Motivation

The idea of heating a room filled with air is similar to heating a boiler filled with water. Although fluids and gases have different physical properties, principles behind their heating in a confined space are analogous.

Of course, faithful models of fluid and gas heating systems require powerful computer clusters and complex algorithms. Being constrained to less powerful systems, algorithm complexity and result accuracy must be lower.

Because of this lower accuracy, a model and simulation of a boiler heating water can be transformed into a model and simulation of a room with a heat source that heats air by adjusting parameters and simple algorithm tuning of the water boiler model. This model is quite comprehensible and modular, which means that additional parameters can be incorporated without rewriting the whole model.

2.2 Water heat model

There are two main equations that come into play in the water boiler model:

1. Calorimetric equation
2. Power-time heating equation

The Calorimetric equation[12] states:

$$Q = c \cdot m \cdot \Delta t \tag{2.1}$$

where

$$\Delta t = (t_1 - t_2) \tag{2.2}$$

c - specific heat capacity of water $[\frac{J}{kg \cdot K}]$

m - mass of water in the boiler [kg]

t_1 - temperature of hot water [K]

t_2 - temperature of cold water [K]

Q - amount of energy that is gained (or lost) by the change of water temperature [J]

The Power-time heating equation[7] states:

$$P = \frac{1}{\eta} \cdot \frac{c_{wh} \cdot m \cdot \Delta t}{\tau} \tag{2.3}$$

where

P - power output of a heat source [W]

τ - length of time that the heat coil is outputting power [h]

η - heating efficiency of the heat coil

Δt - temperature differential of water before and after heating [K]

c_{wh} - specific heat capacity of water $[\frac{W \cdot h}{kg \cdot K}]$

Variations of these two equations are enough to create a simple water boiler model.

A constant volume of water is contained inside the boiler by letting "cold" water flow in when "hot" water is being used. During this time, the water is being heated by a heat source such as a heat coil. The heat coil stops heating when the water gains a desired temperature, depending on the use of the boiler[1].

Let us establish other parameters of a water boiler, but before we do that, we will introduce the concept of the time interval. Since temperature measurements are not continuous, some time passes between measurements. This length of time shall be called a time interval. It has no determined length, however it must be non-zero and it shall be a number. The index of a time interval is i, where $i \in \mathbb{N}_0$. We will assume that a set of measurements[2] taken during time interval i shall be taken at start of the interval and will remain valid until the interval is at an end, after which another set of measurements are taken and so forth.

t_i - the temperature of cold water [K]

m_i - amount of displaced hot water, respectively the amount of cold water [kg]

t'_i - current water temperature [K]

Combining the two equations we get the simulated temperature equation

$$T_i = \frac{(m - m_i) \cdot t'_i + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} \tag{2.4}$$

where

T_i - The simulated temperature[3] [K]

Basically what we have here is a simplified water heat model that allows us to predict the water temperature quite accurately[7]. We will now address what are called free parameters.

[1] A typical home water boiler can have a desired temperature about 68° Celsius.

[2] All measurements required by the model, that is, every parameter with a lower case letter and an index i.

[3] Observe that this variable has a upper case letter. It can easily be seen that the simulated temperature refers to the temperature at the end of the time interval.

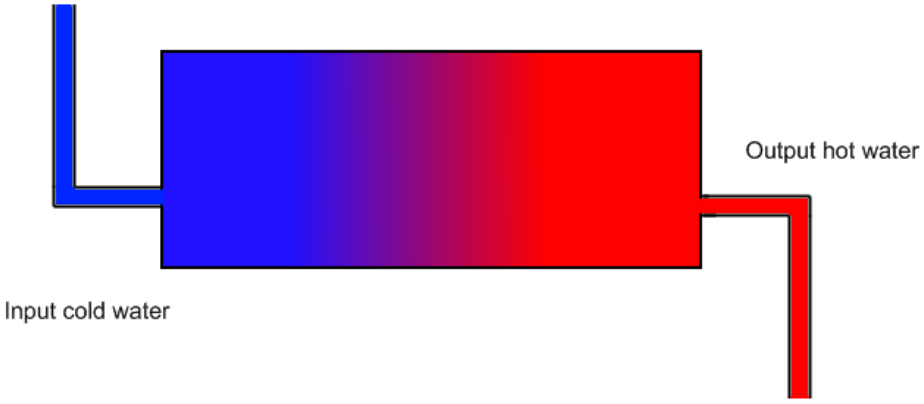

Figure 2.1: A simple water boiler diagram

2.2.1 Parameter constraints

We will now discuss the range of values that parameters may attain. Starting with temperature, the obvious constraint when using the Celsius scale is that the temperature shall not fall below -273.15. The upper bound is unrestricted.

Although water mass having a negative number value does not seem correct, an exception arises here. Taking into account a situation when "cold" water being used to refill the boiler has a higher temperature than water currently residing in the boiler. In this case the parameter m_i can actually be negative so as to additionally heat the boiler water. Power P_i is declared in Watts and cannot have negative values, the upper bound is unrestricted too. The specific heat capacity c_{wh} of water is a positive constant. Time τ_i is non-negative as well.

Heating efficiency $\eta_i \in \langle 0, 1 \rangle$. It differs with various types of fuels[7]. For example, electricity has a heating efficiency value of 0.98, wood has a value of 0.85 and charcoal a value of 0.78.

2.2.2 Free parameters and their localization

Among the parameters in the equation, three have been chosen to particularize the heat model in each time interval. These three are

- v_i - speed of water displacement
- t_i - temperature of mixed water
- η_i - heating efficiency

Notice that v_i has not been declared before, the reason being it is hidden inside the parameter m_i. There are a few issues associated with the heat model, one of them being, since the way the equation is written, the temperature drop caused by cold water happens instantly. This is not pleasant while trying to find free parameters of the model[4]. Because of that, the amount of displaced water is computed as

$$m_i = v_i \cdot \tau_i \tag{2.5}$$

How do we find this set of free parameters to faithfully represent a heat model in the current time interval? The method used here is a modification of the gradient descent algorithm. By retrieving power output, initial and final[5] temperatures during each time interval, we get the input[6] and output[7] parameters of the model.

The next issue is the number of free parameter triplets. In one scenario there can be so many triplets as there are time intervals. This has the benefit of dealing with different times of the day, when it is colder in the morning, warmer in the afternoon and colder again in the evening. Each day can be estimated as being basically the same at the day before regarding the heat model. Small fluctuations can be sorted out by rediscovering free parameters after each time interval is over. A lot of this depends on the length of time intervals.

Another scenario would be to assume that the heat model is homogeneous over time and there would be only one triplet describing it throughout the day. The free parameters could be rediscovered, for example, every time the temperature change is not in order with the model. In Chapter 4 we will review and evaluate experimental results using both scenarios.

[4]This will be explained later on.

[5]For example, at 09:00 room temperature was 21 °C and at 09:15 room temperature had risen to 22.3 °C. During this time power of the heat source was 180 W.

[6]Initial temperature, time duration, power output.

[7]Final temperature.

2.2.3 Gradient descent and AGDA

Gradient descent is an optimization algorithm designed to numerically find local minima of a function by taking proportional steps negative to the gradient, as opposed to gradient ascent, where steps proportional to the gradient are taken, resulting in the discovery of local maxima[6].

It is based on the observation that if the real-valued function $F(x)$ is defined and differentiable in a neighborhood of a point a, then $F(x)$ decreases fastest if one goes from point a in the direction of the negative gradient of F at $a - \nabla F(a)$. It follows that, if

$$b = a - \gamma \cdot \nabla F(a) \tag{2.6}$$

for $\gamma \in \mathbb{R}, \gamma > 0$, then $F(a) > F(b)$. With this observation in mind, one starts with a guess x_0 for a local minimum of F, and considers the sequence x_0, x_1, x_2, ... such that

$$x_{n+1} = x_n - \gamma_n \cdot \nabla F(x_n),\ n > 0 \tag{2.7}$$

We have

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots \tag{2.8}$$

so hopefully the sequence $\{x_n\}$ converges[8] to a local minimum, should it exist[6]. Note that the value of the step size γ is allowed to change at every iteration[6].

The gradient descent algorithm can take many iterations to converge towards a local minimum, if for example the starting point is in a "valley"-like curvature. Improvements can be made by finding the optimal γ per step used in methods based on conjugate gradient techniques[24].

Because of these weaknesses we use an acceleration of the gradient descent algorithm with backtracking for unconstrained optimization[8]. The idea is to modify the step length γ_k by means of a positive parameter Θ_k, in a multiplicative manner, in such a way to improve the behavior of the classical gradient algorithm. It has been shown that the resulting algorithm remains linear convergent, but the reduction in function value is significantly improved[8]. The algorithm starts with an initial point $x_0 \in dom(f)$ where

[8]Generally, this applies for infinitesimal γ_n.

$f : \mathbb{R}^n \to \mathbb{R}$ and generates a sequence of points according to the following iterative procedure:

$$x_{k+1} = x_k + t_k \cdot d_k \tag{2.9}$$

where

t_k - The step size

and

$$d_k = -g_k = -\nabla f(x_k) \tag{2.10}$$

Before we discuss the main algorithm, we will show an algorithm that is used to find the step length γ_k, which is called *line search with backtracking*. There are three steps in this procedure[8] and it considers the following scalars: $0 < \alpha < 0.5, 0 < \beta < 1$ and $s_k = \frac{-g_k^T \cdot d_k}{\|d_k\|^2}$

Step 1. Consider the descent direction d_k for f at point x_k. Set $t = s_k$.

Step 2. While $f(x_k + t \cdot d_k) > f(x_k) + \alpha \cdot t \cdot \nabla f(x_k)^T \cdot d_k$, set $t = t \cdot \beta$.

Step 3. Set $t_k = t$.

Variables in [8] use values $\alpha = 0.0001$ and $\beta = 0.8$, meaning that we accept a small portion of the decrease predicted by linear approximation of f at the current point. Observe that, if $d_k = -g_k$, then $s_k = 1$, which we implicitly use in this thesis.

Concerning the main algorithm, there are six steps[8].

Step 1. Consider a starting point $x_0 \in dom(f)$ and compute: $f_0 = f(x_0)$ and $g_0 = \nabla f(x_0)$. Set $k = 0$.

Step 2. Using the line search with backtracking procedure determine the step length t_k.

Step 3. Compute: $z = x_k - t_k \cdot g_k$, $g_z = \nabla f(z)$ and $y_k = g_z - g_k$.

Step 4. Compute: $a_k = t_k \cdot g_k^T \cdot g_k$, $b_k = -t_k \cdot y_k^T \cdot g_k$ and $\Theta_k = a_k/b_k$.

Step 5. Update the variables: $x_{k+1} = x_k - \Theta_k \cdot t_k \cdot g_k$ and compute f_{k+1} and g_{k+1}.

Step 6. Test a criterion for stopping the iterations. If the test is satisfied, then stop. Otherwise consider $k = k + 1$ and go to step 2.

The gradient descent algorithm can be immediately particularized from AGDA by skipping steps 3 and 4 and considering $\Theta_k = 1$ in step 5 where variables are updated[8].

Two stop criteria are used in this thesis. The first one being an iteration limit for k, the default value is 5000. The second one is $|\nabla f(x_k)| > 10^{-8}$. The test criterion is a logical AND, meaning that if one or both of the criteria are evaluated as true, the main algorithm ends.

2.3 Air heat model

In this section we will discuss the differences between water and air heating, as well as the heat function used in the thesis program.

2.3.1 Heat function and AGDA

The heat function has already been described here as the simulated temperature equation. Formally we define this function as

$$T(\eta_i, m_i, t_i) = \frac{(m - m_i) \cdot t_i' + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} \tag{2.11}$$

Function T is defined as $T : \mathbb{R}^3 \to \mathbb{R}$, the range of T is the Kelvin temperature scale. The domain has been discussed in section 2.2.1. An important aspect of the heat function with regards to AGDA is its differentiability. It is important when calculating the error functions E_i. The error function is defined as:

$$E_i = \frac{1}{2} \cdot (T(\eta_i, m_i, t_i) - S_i)^2 \tag{2.12}$$

where

S_i - The true ambient temperature measured at the end of time interval i.

Calculating partial differential equations for E_i we get:

$$\frac{\partial E_i}{\partial \eta_i} = (\frac{(m - m_i) \cdot t'_i + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} - S_i) \cdot \frac{P_i \cdot \tau_i}{m \cdot c_{wh}} \tag{2.13}$$

$$\frac{\partial E_i}{\partial m_i} = (\frac{(m - m_i) \cdot t'_i + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} - S_i) \cdot \frac{t_i - t'_i}{m} \tag{2.14}$$

$$\frac{\partial E_i}{\partial t_i} = (\frac{(m - m_i) \cdot t'_i + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} - S_i) \cdot \frac{m_i}{m} \tag{2.15}$$

Function T is not defined for conditions when $m = 0$ and $c_{wh} = 0$. Since air mass[9] and specific heat capacity are always greater than 0, these restrictions do not pose a problem. These equations are also used in the end condition in Step 6 of the main AGDA algorithm, where

$$|\nabla T(\eta_i, m_i, t_i)| = \sqrt[2]{(\frac{\partial E_i}{\partial \eta_i})^2 + (\frac{\partial E_i}{\partial m_i})^2 + (\frac{\partial E_i}{\partial t_i})^2} \tag{2.16}$$

2.3.2 Water - air model difference

The obvious difference between water and air is water being a liquid opposed to air being a gas. This has a few implications regarding the heat model.

Air mass is calculated by multiplying the volume of air with air density. The volume of air is usually computed as the size of a room, neglecting furniture volume. Air density is computed[9] as:

$$D = \frac{P_d}{R_d \cdot h} + \frac{P_v}{R_v \cdot h} \tag{2.17}$$

where

D - air density in $\frac{kg}{m^3}$

[9] Air mass m is a pre-defined constant and is not to be mistaken with the free parameter m_i.

h - temperature of air inside the room [K]

P_d - atmospheric[10] pressure of dry air [Pa] (default is $\approx$ 105035.82 Pa)

R_d - gas constant for dry air 287.05 $[\frac{J}{kg \cdot K}]$

P_v - pressure of water vapor [Pa]. This is calculated as:

$$P_v = E_s * RH \tag{2.18}$$

where

RH - relative humidity of air (range $\langle 0, 1 \rangle$)

E_s - saturation pressure of water vapor in mb which is calculated as:

$$E_s = c_0 * 10^{\frac{c_1 \cdot T_c}{c_2 + T_c}} \tag{2.19}$$

where

$c_0 = 6.1078$

$c_1 = 7.5$

$c_2 = 237.3$

T_c - input temperature $[^\circ C]$

R_v - gas constant for water vapor 461.495 $[\frac{J}{kg \cdot K}]$

Chapter 3

Power control

Generally, control theory concerns itself with the effect of behavior on dynamic systems. A typical concept is the feedback loop that controls the dynamic behavior of a system (see figure 3.1).

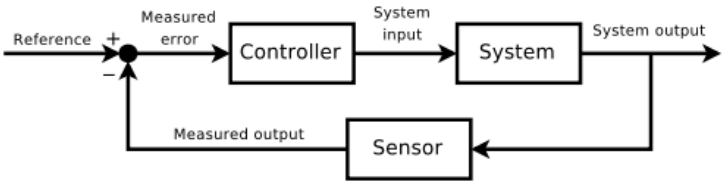

Figure 3.1: Feedback (closed) loop [13]

Applying this concept to our work, we see the "System" box as the heat source system, the "Sensor" as a thermometer, "Controller" as the controlling program and "Reference" as the desired temperature. The sensor detects the current room temperature and sends it to the controller. The controller receives the current room temperature and requests the desired temperature, which usually changes throughout the day. Then the controller decides, according to the measured error, if the heat source should be turned on or off and sends the corresponding signal to the heat source system.

There are two types of power control that are used in this work:

- Non-predictive control
- Predictive control

Non-predictive control means we do not consider future situations and react only to current inputs. Therefore no physical model is created and no heuristics are calculated. Examples of this are reactive on/off control, P, PI, PD and PID control.

Predictive control involves creating a model of future situations, possibly using accumulated data for model tuning, and influencing behavior of the controlled system accordingly. Predictive on/off control falls into this category.

Before we start with the simplest form of feed-forward loop control and that is the reactive on/off control, we will address the measured error.

3.1 Measured error

The measured error is a discrepancy between the desired temperature and the current room temperature. There are various ways to represent this relation, the most straight-forward example is the numerical difference between these two values. In this work, the measured error is computed as:

$$E(I) = \sum_{i \in I} E_i(t_d, t_c) \tag{3.1}$$

where

I - set of heating time intervals at the end of which temperature values were measured

$$E_i(t_d, t_c) = \frac{1}{2} \cdot (t_d - t_c)^2 + P(s_{i-1}, s_i) \tag{3.2}$$

where

t_d - desired temperature [°C]

t_c - current room temperature [°C]

P - penalty function defined as $P : X \to \mathbb{R}$, $X \in \{0, 1\}$

s_i - state of the heat source system at the i-th heating time interval

The penalty function P represents a penalty for changing the state of the heat source system. Here we define two states that a heat source system can be in. State "0" is the OFF state when no heat is generated. State "1" is the ON state when the heating is at its maximum power. Considering heat sources with variable power intake the number of states would be larger, however we will operate with only two states.

For the error function $E_i(t_d, t_c)$ to be consistent, the penalty value is equal to a temperature difference value in degrees Celsius. Calculating the value is a technical issue. The state of the heat source is usually changed by an electrical switch called the relay. The electrical life expectancy of general purpose and power relays is generally rated to be 100,000 operations minimum, while mechanical life expectancy may be one million, 10, or even 100 million operations[15]. Compared to the price of one relay, the penalty is generally very small and may be ommited. However, conditions may arise when the penalty value is an important factor. An isolated polar or space station using multiple power relays exposed to high risk/failure environments will may be of consideration.

3.2 Reactive on/off control with temperature bounds

Reactive on/off control means the controller sends a signal to the heat source system to begin heating if the measured error is larger than zero. This happens when the desired temperature is higher than the current room temperature.

With high sensor resolution[1], high sensor sensitivity[2], high sensor responsiveness[3] and low thermo-insulative values of walls of the room the rate of switches per a given time interval can be very high. Ignoring the bothersome sound of repeated relay switching, the sum of penalty function values over time could yield a large value. To circumvent this, a lower and/or upper temperature bound is suggested to overcome this issue.

An algorithm written in pseudo code using the lower and upper bound temperature is displayed in figure 3.2.

[1] Amount of numbers representing the fractional part of temperature.
[2] Capability of recognizing fluctuations of temperature.
[3] Amount of time needed to send temperature information to a receiving end.

```
above_state = LOW;

REPEAT

IF current_temperature >= desired_temperature + upper_bound
THEN
  set heat source to OFF state;
  above_state = HIGH;
ELSE
IF current_temperature < desired_temperature + upper_bound
AND above_state == LOW
THEN
  set heat source to ON state;
ELSE
IF current_temperature >= desired_temperature - upper_bound
AND above_state == HIGH
THEN
  set heat source to OFF state;
ELSE
  set heat source to ON state;
  above_state = LOW;

UNTIL end of control;
```

Figure 3.2: Reactive on/off control algorithm

Figure 3.3 is a graph of temperature values over time using this method in a software simulation. Figure 3.4 shows two different desired temperatures. Observe that only the lower bound is used. It can be seen that the complexity value is $O(1)$ for both memory and number of operations.

The orange line is the actual temperature progression, the green line is the desired temperature and the blue line symbolizes the outside temperature value. Usually the lower and upper bound are the same size, but it is not a rule. This type of control is most commonly used in households with thermostats. It has simple tuning and implementation since one needs only to set the desired temperature and temperature bounds. Low-end thermostats have often low sensitivity and resolution sensors, resulting in no

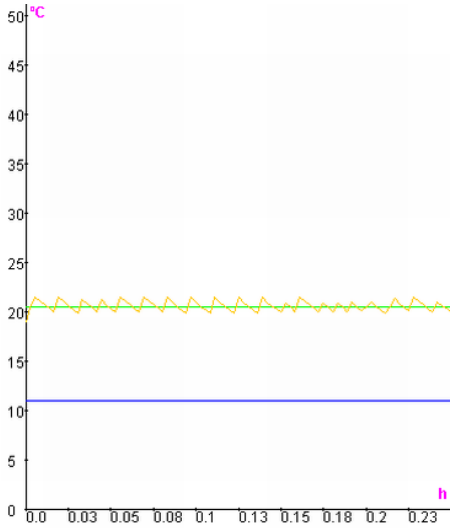

Figure 3.3: One desired temperature

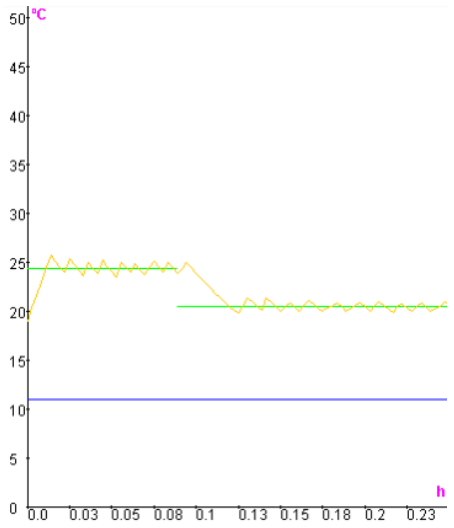

Figure 3.4: Two desired temperatures

need of temperature bounds.

3.3 PID control

A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. A PID controller attempts to correct the error between a measured process variable and a desired set point by calculating and then outputting a corrective action that can adjust the process accordingly and rapidly, to keep the error minimal[18].

The PID controller calculation involves three separate parameters; the proportional, the integral and derivative values. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element[18].

By tuning the three constants in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness

of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation. Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions[18].

We will define additional variables[19] that will be used later. They are

- Process variables
- Manipulated variables
- Controlled variables

Many external and internal conditions affect the performance of a process. These conditions may be expressed in terms of process variables such as temperature, pressure, flow, concentration, weight, level, etc. The process is usually controlled by measuring one of the variables that represent the state of the system and them by automatically adjusting one of the variables that determine the state of the system. Typically, the variable chosen to represent the state of the system is termed the "controlled variable", e.g. "CV" and the variable chosen to control the system's state is called the "manipulated variable"[19].We will also define the set point[18], e.g. "SP", as the desired temperature.

In regards to this work, the controlled variable is the current temperature of the room. The manipulated variable is the power value of the heat source system and the set point is the desired temperature.

If a controller starts from a stable state at zero error (PV = SP), then further changes by the controller will be in response to changes in other measured or unmeasured inputs to the process that impact on the process, and hence on the PV. Variables that impact on the process other than the MV are known as disturbances[18].

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable[18]. Hence:

$$MV(\tau) = P_{out} + I_{out} + D_{out} \tag{3.3}$$

3.3.1 Proportional term

The proportional term (sometimes called gain) makes a change to the output that is proportional to the current error value. The proportional

response can be adjusted by multiplying the error by a constant K_p, called the proportional gain. The proportional term is given by[18]:

$$P_{out} = K_p \cdot e(\tau) \tag{3.4}$$

where

P_{out} - Proportional term of output

K_p - Proportional gain, a tuning parameter

e - Error = SP - CV

τ - Time or instantaneous time[4]

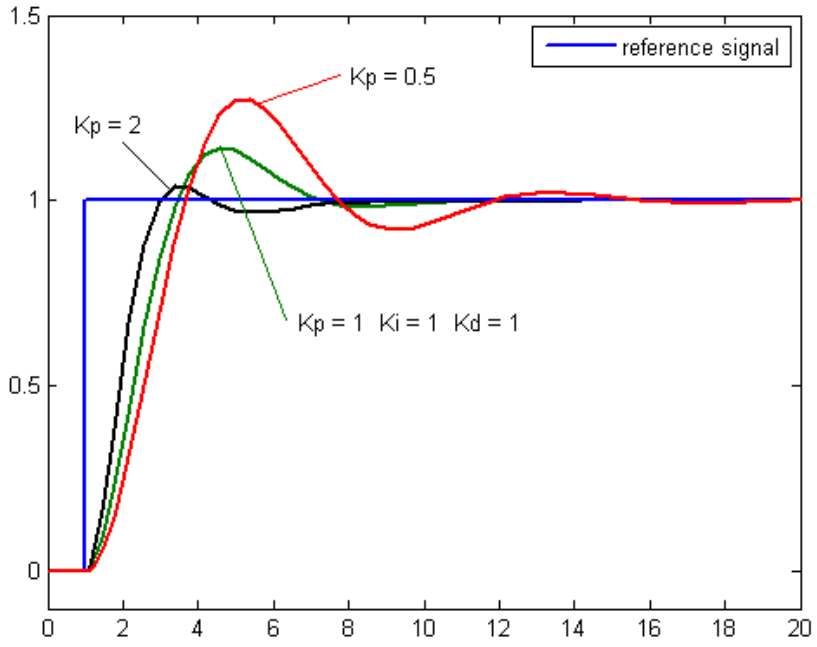

Figure 3.5: Plot of PV vs time, for three values of Kp (Ki and Kd held constant)[18]

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can

[4]The present time.

become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive or sensitive controller[18]. Using only the proportional term is commonly called proportional (P) control model[19].

3.3.2 Integral term

The integral term is also called reset mode, because after a load change it returns the controlled variable to set point and eliminates the offset which the plain proportional controller would leave. This mode has also been referred to as floating control, but it is most commonly called integral (I) control mode[19]. The mathematical expression of the integral-only controller is[18]:

$$I_{out} = K_i \cdot \int_0^t e(\tau)\, d\tau \tag{3.5}$$

where

I_{out} - Integral term of output

K_i - Integral gain, a tuning parameter

t - A dummy integration variable

The integral term[5] accelerates the movement of the process towards set point and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set point value[6][18].

3.3.3 Derivative term

Concerning the derivative term, it anticipates its future state and acts on that prediction. The derivative control mode becomes necessary as the

[5]When it is added to the proportional term.

[6]It can cross over the set point and then create a deviation in the other direction.

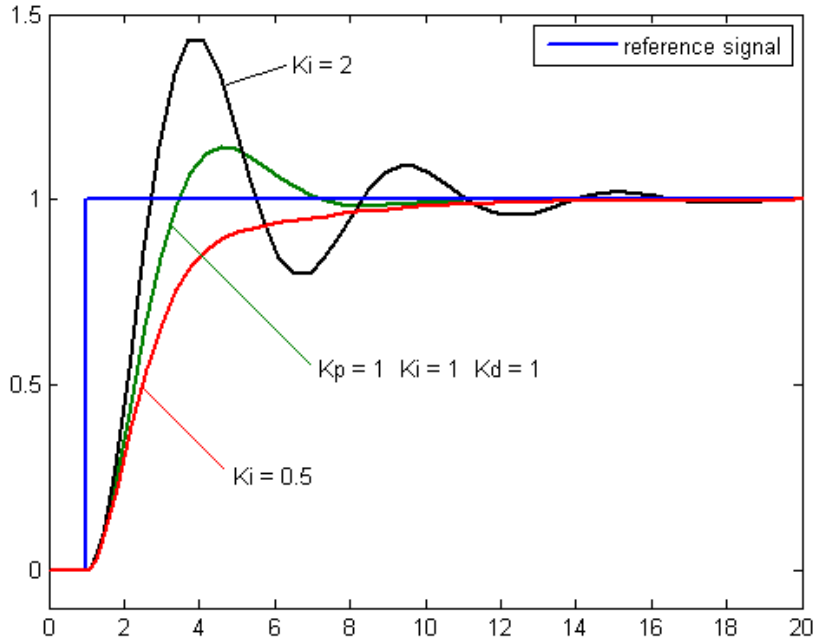

Figure 3.6: Plot of PV vs time, for three values of Ki (Kp and Kd held constant)[18]

size of processing equipment increases and, correspondingly, the mas and thermal inertia of such equipment. For large processes it is not good enough to respond to an error when it has already evolved, because the momentum of these large processes makes it very difficult to stop or reverse a trend once it has evolved[19]. The derivative term is given by[18]:

$$D_{out} = K_d \cdot \frac{\partial e}{\partial \tau}(\tau) \tag{3.6}$$

where

D_{out} - Derivative term of output

K_d - Derivative gain, a tuning parameter

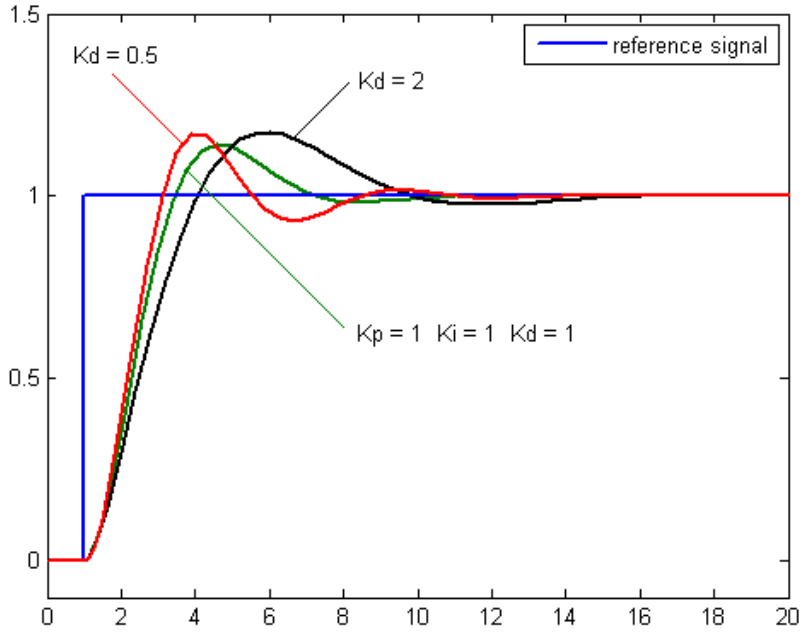

Figure 3.7: Plot of PV vs time, for three values of Kd (Kp and Ki held constant)[18]

The purpose of derivative control is to predict process errors before they have evolved and take corrective action in advance of that occurrence[19].

3.3.4 Calibration of variables

The proportional, integral and derivative gain variables need to be tuned to provide good control. There are a few methods of setting these parameters. A few methods are:

- Cohen - Coon using the reaction curve[20]
- Ziegler - Nicols using the reaction curve[14]
- Ziegler - Nicols using the oscillation method[21]
- Manual tuning

Cohen - Coon and Ziegler Nicols are basically table methods, e.g. P, I and D parameters are to be calculated according to table 3.3.4, table 3.3.4 and table 3.3.4.

Cohen and Coon carried out further studies to find controller settings which, based on the same model, lead to a weaker dependence on the ratio of delay to time constant. Their suggested controller settings are shown in table 3.3.4 [21].

-	K_p	T_r	T_d
P	$\frac{\upsilon_0}{K_0 \cdot \tau_0} \cdot (1 + \frac{\tau_0}{3 \cdot \upsilon_0})$	-	-
PI	$\frac{\upsilon_0}{K_0 \cdot \tau_0} \cdot (0.9 + \frac{\tau_0}{12 \cdot \upsilon_0})$	$\frac{\tau_0 \cdot (30 \cdot \upsilon_0 + 3 \cdot \tau_0)}{9 \cdot \upsilon_0 + 20 \cdot \tau_0}$	-
PID	$\frac{\upsilon_0}{K_0 \cdot \tau_0} \cdot (\frac{4}{3} + \frac{\tau_0}{4 \cdot \upsilon_0})$	$\frac{\tau_0 \cdot (32 \cdot \upsilon_0 + 6 \cdot \tau_0)}{13 \cdot \upsilon_0 + 8 \cdot \tau_0}$	$\frac{4 \cdot \tau_0 \cdot \upsilon_0}{11 \cdot \upsilon_0 + 2 \cdot \tau_0}$

Table 3.1: Cohen - Coon tuning using the reaction curve

Variable explanations:
T_r - an integral parameter. $T_r = \frac{K_p}{K_i}$

T_r - a derivative parameter. $T_d = \frac{K_d}{K_p}$

υ_0 - the maximum slope tangent time difference $(t_2 - t_1)$

K_0 - ratio between input value and plant output $(\frac{y_\infty - y_0}{u_\infty - u_0})$

τ_0 - maximum slope tangent and zero time difference $(t_1 - t_0)$

For a fuller understanding, figure 3.8 is provided.

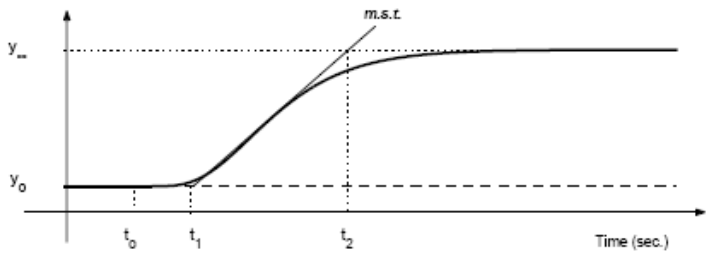

Figure 3.8: Plant step response [21]

The Ziegler - Nicols reaction curve tuning method is very sensitive to the ratio of delay to time constant [21].

-	K_p	T_r	T_d
P	$\frac{\upsilon_0}{K_0 \cdot \tau_0}$	-	-
PI	$\frac{0.9 \cdot \upsilon_0}{K_0 \cdot \tau_0}$	$3 \cdot \tau_0$	-
PID	$\frac{6 \cdot \upsilon_0}{5 \cdot K_0 \cdot \tau_0}$	$2 \cdot \tau_0$	$\frac{\tau_0}{2}$

Table 3.2: Ziegler - Nicols tuning using the reaction curve

The Ziegler - Nicols oscillation method is only valid for open-loop stable plants and is carried out through the following steps [21]:

1. Set the plant under proportional control with a very small gain.
2. Increase the gain until the loop starts oscillating[7].
3. Record the controller critical gain $K_p = K_c$ and the oscillation period of the controller output P_c.
4. Adjust the controller parameters according to table 3.3.4.

-	K_p	T_r	T_d
P	$\frac{K_c}{2}$	-	-
PI	$\frac{9 \cdot K_c}{20}$	$\frac{5 \cdot P_c}{6}$	-
PID	$\frac{3 \cdot K_c}{5}$	$\frac{P_c}{2}$	$\frac{P_c}{8}$

Table 3.3: Ziegler - Nicols tuning using the oscillation method

3.3.5 Use of PID in this work

Manual tuning requires a human to calibrate the PID variables in order to obtain a satisfactory controlled environment. There is no need for tuning algorithms, only an experienced person who has access to the plant. The

[7]Linear oscillation is required and it should be detected at controller output

method is quite fast to implement. A question arises whether it is better to tune variables by algorithm or by a human. There is a constant search for new tuning techniques promising better optimization and, depending on criteria, their is no "better" method per se. Considering the options, manual tuning was chosen for the nature of this work and its ease of implementation.

Heat control utilizes only two states - ON and OFF. Because of this, we decided to use a boundary value 0.5. All PID control values below 0.5 result in the heat control switching to OFF state and all values above and including 0.5 result in the heat control switching to ON state.

3.4 Predictive on/off control

This type of control is the innovation we are trying to bring into building heat control. We assume that by creating a numerical heat model of an enclosed space by means of studying previous heating data we will have a better understanding of how to manipulate states of a heating source system for an overall improvement. Improvement can be broken down into a three sections, namely:

- User satisfaction
- System strain
- Financial impact

User satisfaction deals with personal comfort after entering the controlled space. For example, let us assume that the current temperature is 21°C. If the user, before leaving, has set the desired temperature to be 25°C at 18:00 and comes back at 17:58, a controlling system utilizing only reactive control will still have the temperature set at 21°C and after reaching 18:00, it will have sent a signal to the heat source to begin heating. This may result, depending on the maximum power of the heat source, in oscillations and temperature delays, increasing discomfort.

System strain is also an issue when it comes to oscillations. As mentioned in section 3.1 too much relay switching can be deteriorative to a systems hardware. A large increase or decrease in temperature over a short span of time can cause fractures in material or moisture condensation since there is not enough time for temperature adaption and air exchange.

When the heat source is turned on, electrical energy is consumed, which increases financial impact on users. Although overheating is usually more of a problem, certain environments (greenhouses, for example) may need a stable temperature where fluctuations of warmth to either side are unwanted. Summing up the error function over time shows us a value which needs to be minimized.

Predictive control utilizes the air heat model in section 2.3. In order to be able to predict temperature changes previous heating data are needed to localize the model. Heating data contains four values that consist of one record:

- Temperature measurement
- Temperature change after first measurement
- Elapsed time between measurements
- Power value between measurements

This data can be taken from logs of an existing system or they can be constructed on-the-fly. Concerning the logs of an existing system, it can be seen from the consistence of the data record that it does not matter what type of control the system utilizes.

Constructing logs on-the-fly means we have to wait for temperature measurements to be taken. This can have negative side-effects which are discussed in section 4.3.

3.4.1 Goals and methods

Now that we have a way to predict future temperature values, the sole question of control arises. Ultimately, our goal is to reduce the sum of error function values over time. To do this, we select a time point in the future and try to find a series of points in time at which to change or retain the state of the heat source system. This can be done by creating a data structure that would contain all the possible series of states that the heat source, as time progresses, can attain. Such a structure would require large amounts of memory. However, constraining a few factors can greatly reduce the amount of needed memory.

We can constrain the scale of elapsed time between temperature measurements and the scale of power values. This is generally bounded by hardware

properties, so there is a fixed time length and there are two heat source system states - ON and OFF as mentioned in section 3.1. A logical outcome would be to create a tree or grid structure, where vertices would contain contemporary temperature and error values while edges would have direction and power values. We will now analyze the difference between these two data structures and come to a conclusion as to which is better suited for this work.

3.4.2 Tree vs. Grid data structure

Both trees and grids are a subset of a more general data structure, which are graphs. So let us begin with a definition of a graph. Then we will define a tree and describe its use as a proposed data structure for this work. Then we will do the same for a grid data structure. We will also describe possible changes that need to be done to the heat model for it to be used with the tree and grid data structure.

A graph[17] G is an ordered pair (V, E) where V is a non-empty set and E is a set of two-element subsets of set V. Elements of set V are called vertices of graph G and elements of set E edges of graph G. We will use modification of graph G and that is a directed graph G'; the difference being that E of graph G' contains ordered two-element subsets of V. We will call this ordering "direction".

A tree[17] T is a specific kind of graph G' in which every two vertices in V are connected by one and only one path[8]. In regard to implementation details, vertices defined as left nodes (left child of the node "above" it) will be those nodes that have edges with a power value equal zero going into them. Likewise, vertices defined as right nodes (right child of the node "above" it) are those node having edges with a non-zero power value. A root node is a node that has no edges going into it. A level is a set of nodes that have a path to root node of the same length. The n-th level is a set of node that have a path of length n to the root node.

Given a tree T and assuming its vertices and edges have values specified in section 3.4.1, figure 3.9 below displays such a graph. Usage of the tree T

[8] A path is a progression of non-repeating edges an vertices.

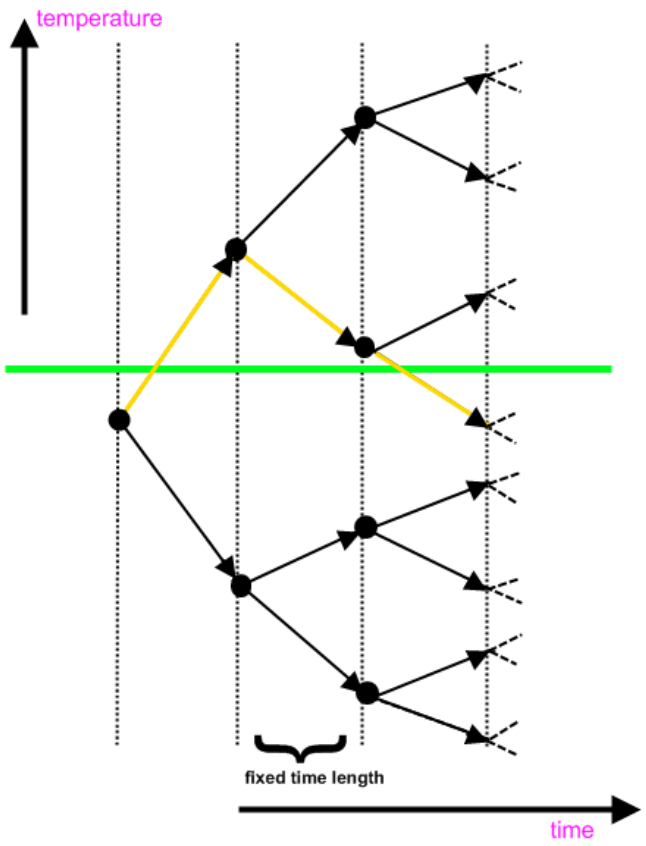

Figure 3.9: Tree structure in use

indicates the direction of edges which is always the general direction of time flow.

In figure 3.9 we see one possible path highlighted in a gold color. This path tries to minimize the error function over time.

The tree is recursively constructed by first creating the right node sub-tree and then the left node sub-tree. Every time a node is created, its parent error value is added to its calculated error value and saved as the new error value. Since by definition there is only one path from root node to every other node, the resulting error value describes the outcome of a progression of power values; in other words, control. To retrieve the power value progression, one needs only to follow edges leading to parent nodes, saving the

power values in a LIFO[9] until reaching the root node.

The memory complexity for a tree structure is $O(2^k - 1), k = \lceil \frac{p}{\tau} + 1 \rceil$ where p is the predicted time interval length and τ is the fixed time interval length between measurements. Retrieving the path has a complexity of $O(k)$ since while building the tree and creating a node having a time value equal or greater than the predicted time, we compare its error value with a previously referenced node[10] at the same level. If it has an error value that is lesser than that of the referenced node, we change the reference to the current node.

No additional changes in the heat model are required for use with the tree data structure.

A grid is a directed graph G'. Given a pair of vertices V_1 and V_2 from V there exists a path leading either from V_1 to V_2 or V_2 to V_1 exclusively. The definitions of left, right and root nodes and levels are the same as defined in the tree structure. The shape of a grid intuitively indicates a quadrangle, but the grid is computed only to half of a quadrangle, reminding a triangle. This is because when reaching the predicted time length, there is no need to construct node any further, hence the triangle.

As we see in figure 3.10 the grid is similar to the tree. The main advantage of the grid is that it needs less nodes. The reason for this we will show in an example. Assuming we have a tree of height three. We will traverse the tree to the right node of the root node's left child. We compare the temperature value of this node to the temperature value of the left node of the root node's right child. We see that the temperature value is the same, as are these two nodes (this is the red colored node in figure 3.10). Assuming the air model is linear, heating for a fixed time interval and not heating the next time interval is the same as not heating and then heating. Accuracy issues will be discussed in section 4.2.1.

There is one inconsistency, however, and that is the error value. When building the tree structure, the temperature of these two node is the same, but the error value differs. To circumvent this, a condition when creating such a "duplicate" node is tested. If we try to create a node on the same level that is already there[11] then we consider the error value. If the error value of the "new" node is lower, then this new error value is inserted into

[9] Last-In-First-Out stack.

[10] If no node is constructed yet, then this node shall be the first reference.

[11] Its temperature value equals the temperature value we want to insert.

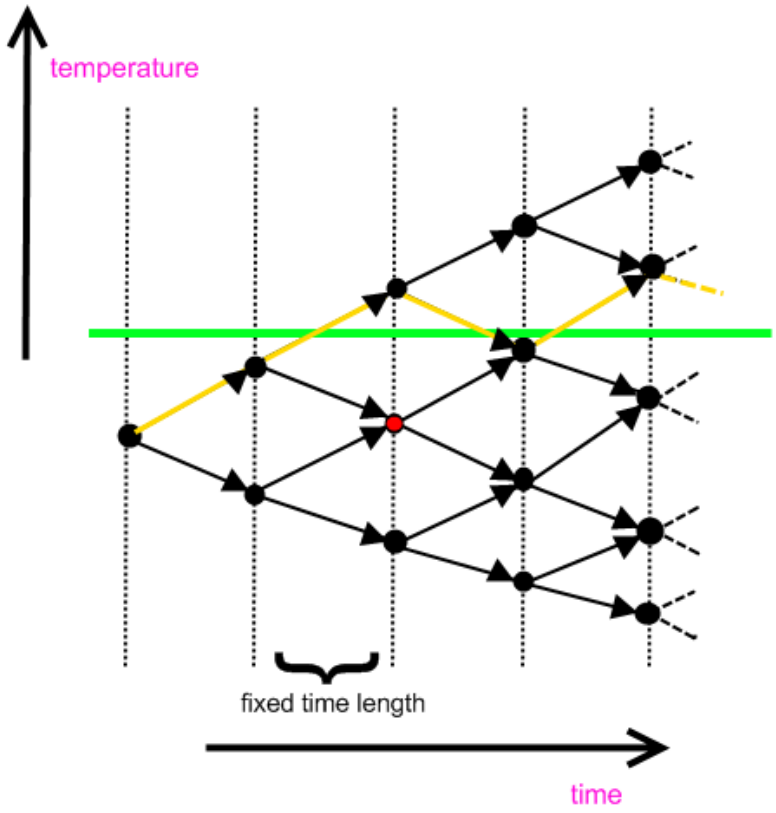

Figure 3.10: Grid structure in use

the existing node. A problem arises here, since each node has two parents. This is bypassed by defining a *preferred* parent, which is the one that the error value has propagated from. It can be seen here that the situation of reaching the predicted time length is analogous to finding the final node in the tree[12].

The memory complexity for a grid is $O(\frac{k\cdot(k+1)}{2}), k = \lceil \frac{p}{\tau} + 1 \rceil$. Retrieving the path has a complexity of $O(k)$ since path retrieval is analogous to tree path retrieval. The grid is constructed using a FIFO[13] and a BFS[14]

[12]We find a node with the smallest error value in the last level and retrieving power values by traversing the edges going against the direction of the current node into the node's preferred parent.

[13]First-In-First-Out stack.

[14]Breadth-First-Search.

technique. Assuming we have created nodes in the n-th level, we also have inserted them in FIFO. The pseudo code in figure 3.11 shows the creation of nodes in the (n+1)-th level. The algorithm in figure 3.11 repeats until the desired level is reached.

After analyzing both structures we come to a conclusion that the main difference is linearity, respectively non-linearity of the heat model. Although not immediately visible, it can be seen that by taking the recursive sequence of heat model values for the tree data structure neccessarily yields a non-linear heat model, where as the grid data structure yields a linear heat model. The following example demostrates this.

The heat model is defined as:

$$T(\eta_i, m_i, t_i) = \frac{(m - m_i) \cdot t'_i + m_i \cdot t_i}{m} + \frac{\eta_i \cdot P_i \cdot \tau_i}{m \cdot c_{wh}} \tag{3.7}$$

Since we repeatedly insert the room temperature value to retrieve a sequence of predicted temperature values depending on the state of heating, the only two variables we want to see are t'_i and P_i. We substitue all other variables to get the following equation:

$$T(\eta_i, m_i, t_i) = A \cdot t'_i + B + C \cdot P_i \tag{3.8}$$

where

A - $\frac{(m-m_i)}{m}$

B - $\frac{m_i \cdot t_i}{m}$

C - $\frac{\eta_i \cdot \tau_i}{m \cdot c_{wh}}$

In order for the model to be linear, a heating period followed by a non-heating period should be equivalent to a period of non-heating followed by a period of heating. Therefore:

$$A \cdot (A \cdot t'_i + B + C \cdot P_i) + B + C \cdot 0 = A \cdot (A \cdot t'_i + B + C \cdot 0) + B + C \cdot P_i \tag{3.9}$$

This can only be possible when A equals 1, which means we must set the free variable m_i equal to 0 and so B is also equal 0, which leaves us with only one free variable and that is η_i.

```
BEGIN
Take out the first node (N1) from FIFO;
Create N1's left (N1L) and right (N1R) child;
Insert N1L and N1R in a new FIFO;
Save a reference (RNL) from N1L;
Repeat
  Take out another node (N2);
  Create N2's left (N2L) and right (N2R) child;
  Save a reference (RN2R) from N2R;
  Compare error values of RN2R and RNL;
  Set dereferenced RNL to the node with the lowest error;
  Save a reference (RNL) from N2L;
  Insert N2L into the new FIFO;
Until last node in FIFO is reached;
Replace FIFO with new FIFO;
END.
```

Figure 3.11: Creating (n+1)-th level nodes

So although the grid data structure consumes less time and memory, the heat model is linear and is therefore believed to be less accurate.

3.5 Best-First search and other heuristics

Best-First search is an instance of the general Tree-Search or Graph-Search algorithm in which a node is selected for expansion based on an **evaluation** function $f(n)$. Traditionally, the node with the *lowest* evaluation is selected for expansion, because the evaluation measured the distance to the goal. There is a whole family of Best-First search algorithms with different evaluation functions. A key component of these algorithms is a **heuristic function**, denoted $h(n)$ [2]:

$h(n)$ = estimated cost of the cheapest path from node "n" to a goal node.

We will concentrate on **greedy best-first search**, with tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, is evaluated the nodes by using just the heuristic function: $f(n) = h(n)$[2].

The description of the greedy best-first search algorithm is as follows:

With regards to figure 3.12, the tree on the left has a black root node, the blue dashed line encompasses the OPEN set that contains node "B". Without loss of generality, let node "B" have the smallest error[15] of all the nodes in the OPEN set. Node "B" is colored green. On the right side of the arrow, node "B" is taken out of the OPEN set and children nodes of node "B" have been added to the OPEN set. This process continues on until a time limit is reached (implicitly 7.7 seconds in this project).

Concerning figure 3.13, the OPEN set contains all nodes that do not have children nodes. From the OPEN set only nodes that have the longest path to the root node are taken into account. Concretely, "A" and "B" are such nodes. Without loss of generality, let "A" have the smallest error of nodes "A" and "B". Therefore we take the path from node "A" to the root node (illustrated by green arrows), reverse it (as so the green arrows point from root to node "A") and we have gained a sequence of ON/OFF values.

Other heuristics such as dividing node error values by length from root to node were also tested. This was done mainly because nodes in the OPEN

[15]The smallest temperature error with regards to the quadratic error function, which is in this case the heuristic function of the best-first search algorithm.

set with the longest path to the root node were not always chosen because of similar error values with other nodes in the OPEN set that had considerably shorter paths and were obviously not suitable.

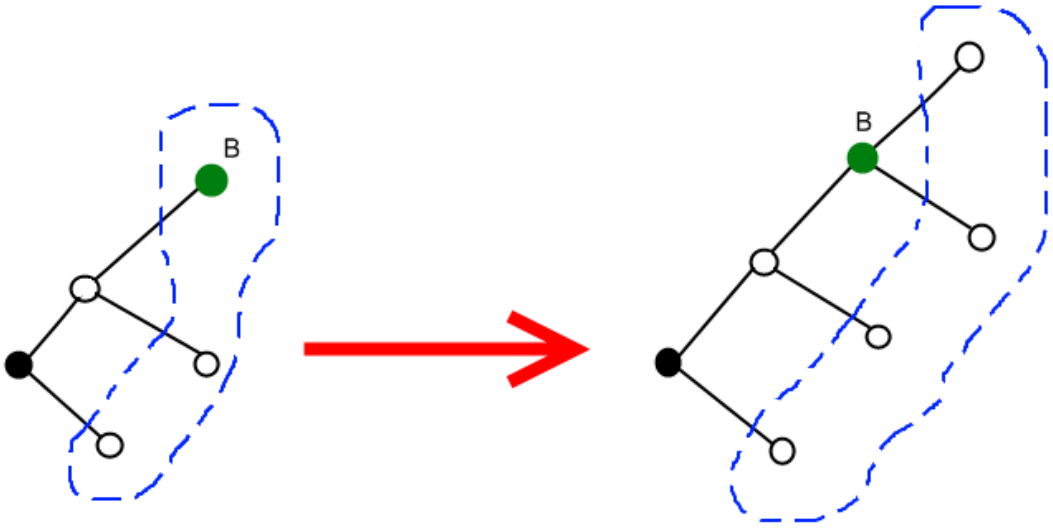

Figure 3.12: Our implementation of the best-first search algorithm.

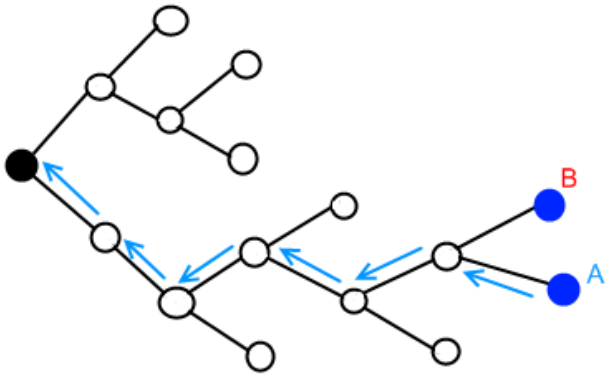

Figure 3.13: Tree data structure obtained by the best-first search algorithm.

Chapter 4

Experimental results

We conducted experiments to decide which method of control was the most suitable one. Four control methods were used:

- ON/OFF reactive
- PID with manual tuning
- Predictive heat model - grid data structure
- Predictive heat model - tree data structure

Each control method was utilized continuously for one day [1]. There were two rounds of experiments. The first round of experiments was conducted in August 2009 and the second round of experiments was conducted in October 2009 and November 2009.

We will also discuss software and hardware details of the experiments as well as system configuration and environmental parameters of days during the experiments.

4.1 Preceeding software simulations

Since the inception of this work, there have been several software programs used to create the project application and physical model. They are MATLAB, Scilab and finally the Java language used with NetBeans IDE.

[1] One day equals 22 hours; the remaining two hours are used for maintenance purposes.

The first model was a water boiler model and subsequently simulated usage of water written in MATLAB script. It utilized the ON/OFF reactive control method and neural network control, which was relatively successful. Afterwards there was an effort to change the water boiler model into an air heat model which would also operate in MATLAB and use Simulink as well for environmental simulation. However, since MATLAB and Simulink are expensive, a cheaper alternative was searched for, resulting in Scilab, since Scilab has built-in Simulink-like features and is open source software. After some research, Scilab was disregarded; the reason being no hardware support, lesser source code organization capabilities and GUI overhead[2].

Finally, the Java language was chosen for its familiarity, organizational and debugging features. Hardware interfaces were also easier to implement. The application documentation will be discussed in Appendix A.

4.2 Environment parameters and system configuration

At first a house was proposed for the experiment. The control application was to be interfaced with the house's heating system, thereby gaining control of the heating system. Unfortunately, use of house became unachievable when it was discovered that no such interface could be implemented and considering the season[3] it was not possible to heat the house. It was then we decided to use a wooden dog house for the experiment. Regarding the nature of this project, the actual structure to be heated was not constrained by many factors; it only needed to be sufficiently shielded from the outside weather. A make-shift heating device was constructed using these components:

- Three 100W light bulbs
- Thermal sensor DS18B20 with temperature resolution to one decimal floating point
- Atmel microchip

[2]At first what seemed a good idea, the Simulink-like features became more of an annoyance than help.

[3]It was late spring and the house owners were later reluctant to have the experiment conducted.

- Relay
- Additional cables and electromechanical parts

The inside of the dog house has about 0.16 m^3. Since the used light bulbs convert only about 5%, we can safely declare the heating system to have 300 W[4].

The location of the dog house was approximately N 40° 12' 54" and E 17° 23' 38" in an outdoors environment when the outside temperature oscillated around the usual temperature required in homes (between 19°C to 26.5°C) in August 2009. The average temperatures throughout the August 2009 experiments ranged from 19.0° to 26.5°. In October 2009, the experiment was relocated to a cellar for testing a temperature-stable environment as well as testing lower temperatures, since the average temperatures vary from 12.0° to 12.9°.

The application was developed on a computer with an Intel Core 2 Duo CPU clocked at 2.26 GHz with 2.96 GB RAM. The computer on which the application ran was an Intel Pentium M clocked at 1.73 GHz with 504 MB of RAM. The application-hardware interface consisted of a USB-Serial Port converter with the serial port configured at a baudrate of 19200 bits per second, 8 data bits, no parity and 1 stop bit. The application read temperatures from USB port using a Java library and wrote either "0" or "1" to the serial port, thus turning heating off or on.

A few application parameters used during experiments are listed here:

- room size: 0.16 m^3
- initial heat model parameters (η,v,t): 0.18, 0.12, 18.0
- maximum number of iterations during heat model discovery: 4000
- relative air humidity: 0.4

[4]A polystyrene wall taped with aluminium for light and heat reflection was taped to the entrance of the dog house as an additional measure.

4.2.1 Calculating data log size

Correct data log size was an issue that needed to be resolved[5]. An experiment was conducted to determine the optimal internal[6] data queue size[7].

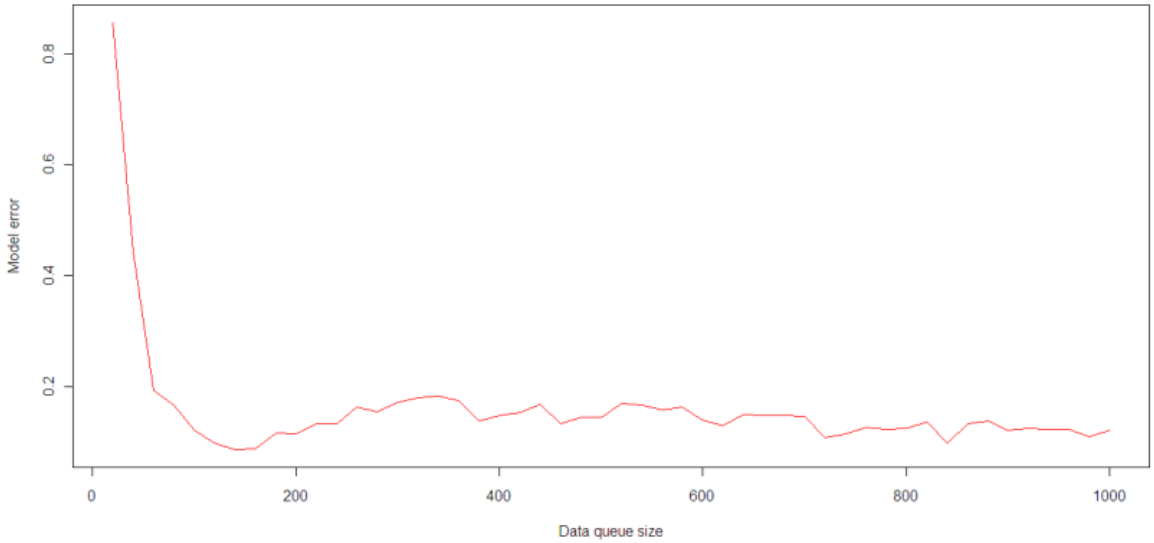

Figure 4.1: Model error plotted over the size of the applications internal data queue.

The model error was calculated by filling the application's internal data queue with existing heating data from a log file. The heat model was found based on this data. Then the heat model was used to calculate a temperature progression using the first data record temperature from the heating data log that followed the records used for finding the heat model. The temperature value was consequently inputted into the heat function, resulting in a 5 minute simulation. The error value is the absolute difference (linear error) between the last temperature progression element that were calculated by the heat function and the temperature from the data log after 5 minutes. This was done ten times for each data queue size, each n-th time loading

[5]A small data log may give an imprecise model, whereas a large data log may hinder performance, leading to prolonged model computation.

[6]Internal does not include the data written to a output file, but the heating data that is saved in the application for heat model discovery.

[7]The floating amount of temperature measurements; after a measurement is to be saved to the internal data queue, if the data queue is full, the oldest measurement value in the queue is removed and the new measurement value is inserted.

data records after skipping $(n-1) \cdot 20$ data records (first time loading data records from the 1st data record, second time from the 20th data record, third time from the 40th data record and so on). The resulting value was then averaged.

In figure 4.1 we see that the model error is high when the data queue size is small and stabilizes after about 200. However, it is not easy to choose the correct size of the internal data queue since the errors are higher for some values and lower for others, regardless of initial heat model parameter settings. This error checking method was used for a few log files, each one generating roughly the same model error graph like figure 4.1. For the application, an arbitrary value of 250 was chosen for the internal data queue. Although it may seem that values around 180 are better suited, running the tests a few times revealed that values around 250 are equally, if not better suited as the optimal data log size.

Another interesting comparison was made. While the overall error[8] value for the predictive three parameter (non-linear) model was about 7.95, for the predictive one parameter (linear) model the value was 7.65 which is smaller.

4.3 Results from experiments conducted in August 2009

The results are composed in the following fashion:

- Temperature graph
- Statistics table
- Further description

The X-axis of the temperature graph signifies the date and time, the Y-axis signifies the measured temperature. The green lines indicate desired temperatures throughout the day.

The table containing statistical data has four columns:

- Time of day - statistical information taken from that time interval.
- Heating time - the amount of time (in seconds) that the heating was turned on.

[8]Sum of all the mean error values over data queue size.

- Error - the error[9] value.
- Switching - the number of relay switches

The overall results are interesting in that we have discovered there is no "best" method. The non-linear model used in conjunction with the tree data structure proved to be very capable with regards to competition from the reactive and PID control methods. The linear model that was utilized by the grid data structure did not do so well and it can be assumed that it is inferior to the other control methods.

4.3.1 Reactive control method

.

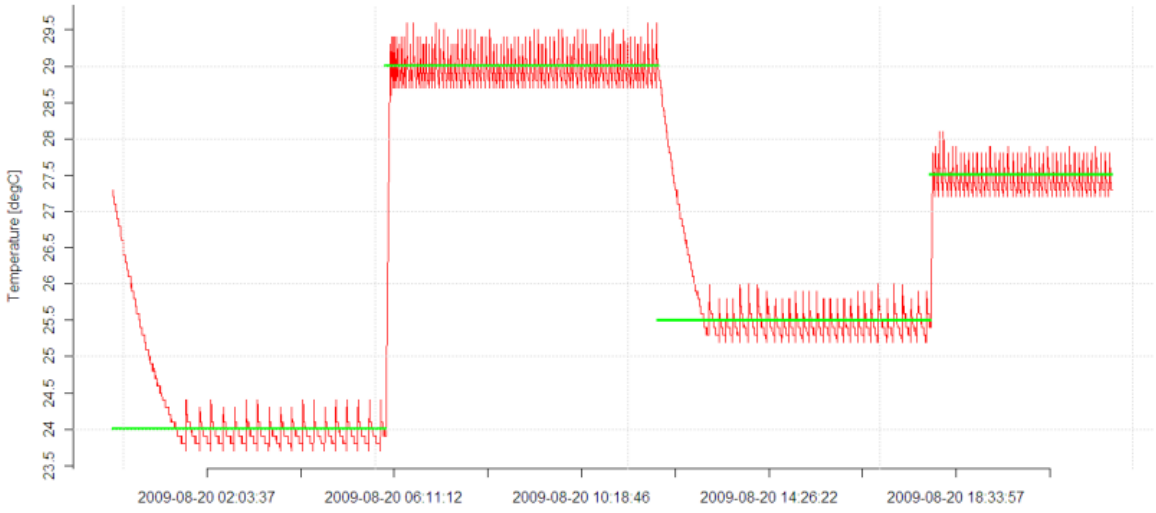

Figure 4.2: Reactive control method temperature graph.

The interpretation of figure 4.2 is quite straightforward. The distinct desired temperatures can be seen without much difficulty. The hysteresis was set to 0.3°C. Presumably had the hysteresis been smaller, the overall error would be lower, but the number of relay switches would have been higher as was the case with PID control. There were no surprising discoveries.

[9]The error was quadratic; it was calculated as the product of differences between the current desired temperature and the temperature after heating, divided by two. This means that the number of summands is dependent on the time between measurements and duration of the experiment.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	530	651.8	36
	06:00 - 11:59	2730	176.035	152
	12:00 - 17:59	940	578.54	60
	18:00 - 22:00	1440	31.835	84
$\sum$	22:00	5640	1438.21	332

Table 4.1: Statistics for reactive control method.

4.3.2 PID control method

.

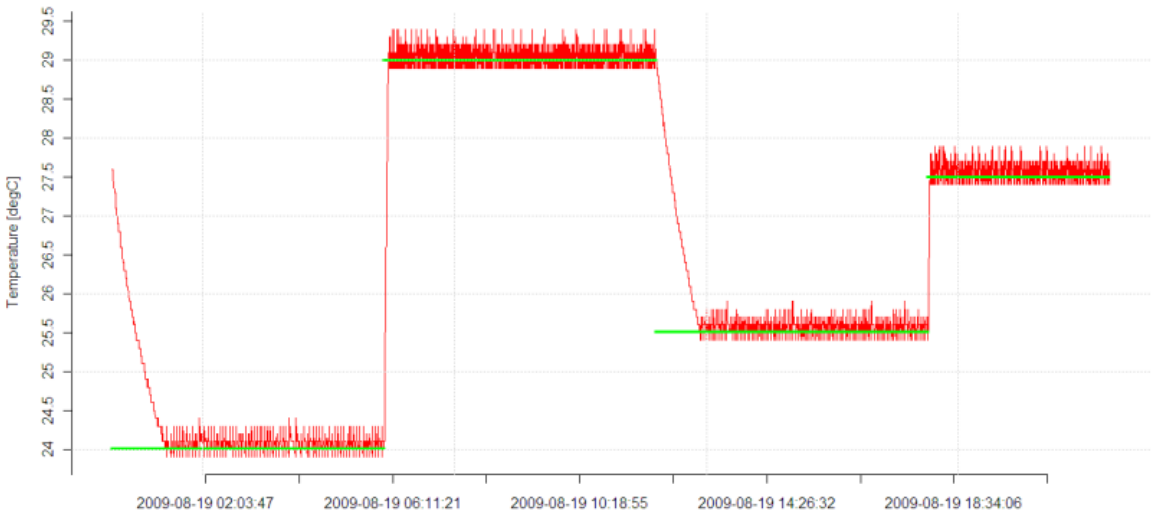

Figure 4.3: PID control method temperature graph.

Figure 4.3 shows us that this method of control has the smallest overall error. The PID tuning parameters were set like so:

- $K_c = 9$
- $T_i = 30$
- $T_d = 0.1$
- $CO_bias = 0$

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	670	645.63	130
	06:00 - 11:59	3010	158.125	496
	12:00 - 17:59	1040	562.18	204
	18:00 - 22:00	1500	21.635	248
$\sum$	22:00	6220	1387.57	1078

Table 4.2: Statistics for PID control method.

However, the number of relay switches as well as heating time was the highest. This may account for the decreased overall error value.

4.3.3 Predictive tree control method

.

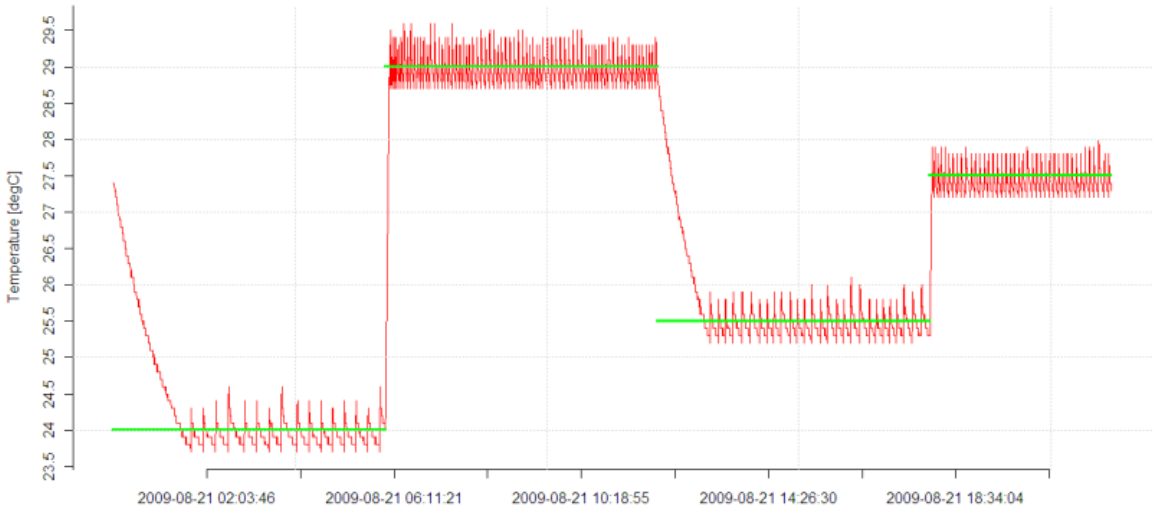

Figure 4.4: Predictive tree control method temperature graph.

The predictive method using the tree data structure and the non-linear model did quite well. It had a lower heating time (2.5% lower than reactive control), but a higher error.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	480	786.81	32
	06:00 - 11:59	2730	173.135	184
	12:00 - 17:59	890	547.165	56
	18:00 - 22:00	1400	34.415	84
$\sum$	22:00	5500	1541.525	356

Table 4.3: Statistics for predictive tree control method.

4.3.4 Predictive grid control method

.

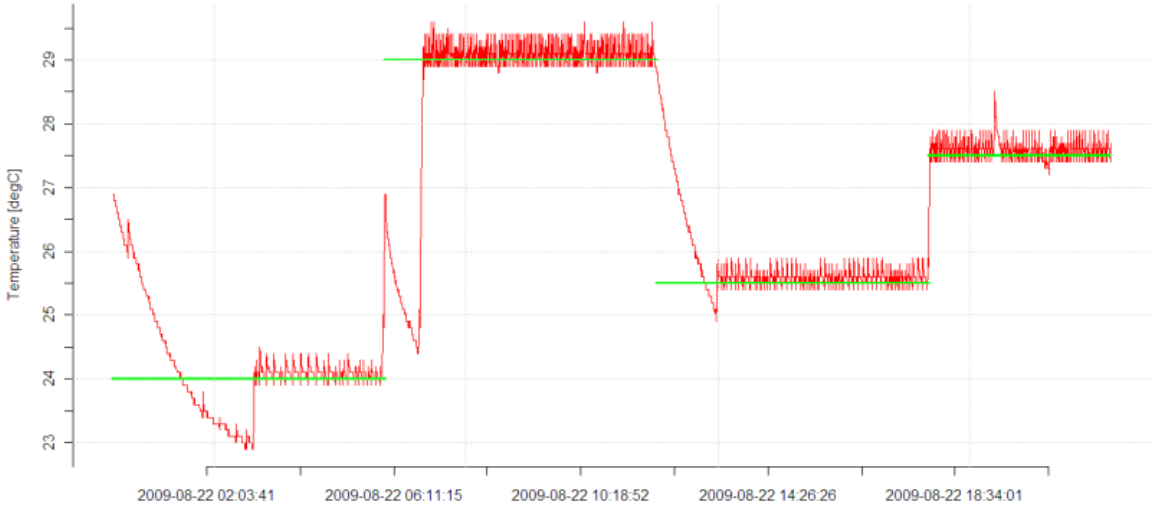

Figure 4.5: Predictive grid control method temperature graph.

The predictive method using the grid data structure and the linear model had the worst result. Although having the lowest heating time, we can see from the temperature graph that the linear model was often inaccurate.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	590	789.405	71
	06:00 - 11:59	2260	1944.405	259
	12:00 - 17:59	950	538.965	129
	18:00 - 22:00	1210	20.635	165
$\sum$	22:00	5010	3293.41	624

Table 4.4: Statistics for predictive grid control method.

4.4 Results in October 2009 November 2009

The second round of experiments were conducted in October 2009 and November 2009. They revealed weaknesses in the tree predictive methods that were thought to be comparable to other control methods. There were a few differences in the configuration of experiments:

- The dog house was moved to a cellar.
- Desired temperature were modified so as to not have the same desired temperature values in both rounds of experiments.

4.4.1 Reactive control method

.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	1200	3212.415	54
	06:00 - 11:59	3750	243.71	130
	12:00 - 17:59	1780	594.38	72
	18:00 - 22:00	3390	444.32	106
$\sum$	22:00	10120	4494.825	362

Table 4.5: Statistics for reactive control method.

To reflect the differences from the August 2009 experiments, figure 4.6 shows us that basically all values are higher. This is due to the fact that although the desired temperature values were on average lowered by 2.5°C,

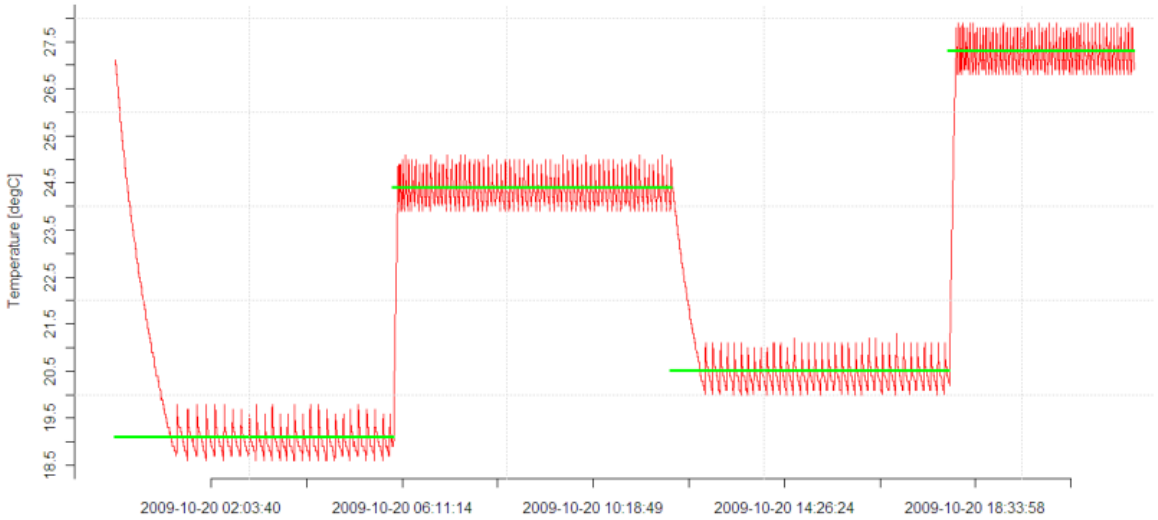

Figure 4.6: Reactive control method temperature graph.

the ambient temperature was lower by 7°C to 17°C. All in all, the reactive control method, although being simple, shows robustness, works fine and has the lowest error.

4.4.2 PID control method

.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	1440	3519.96	268
	06:00 - 11:59	5470	235.305	794
	12:00 - 17:59	2450	469.425	428
	18:00 - 22:00	5160	554.89	649
$\sum$	22:00	14520	4779.58	2139

Table 4.6: Statistics for PID control method.

Figure 4.7 shows us that this method of control did not have the smallest error as was the case in the first round of experiments. The PID tuning parameters were remained constant since the August 2009 experiments. The error value, as well as all other values, is higher than the reactive control

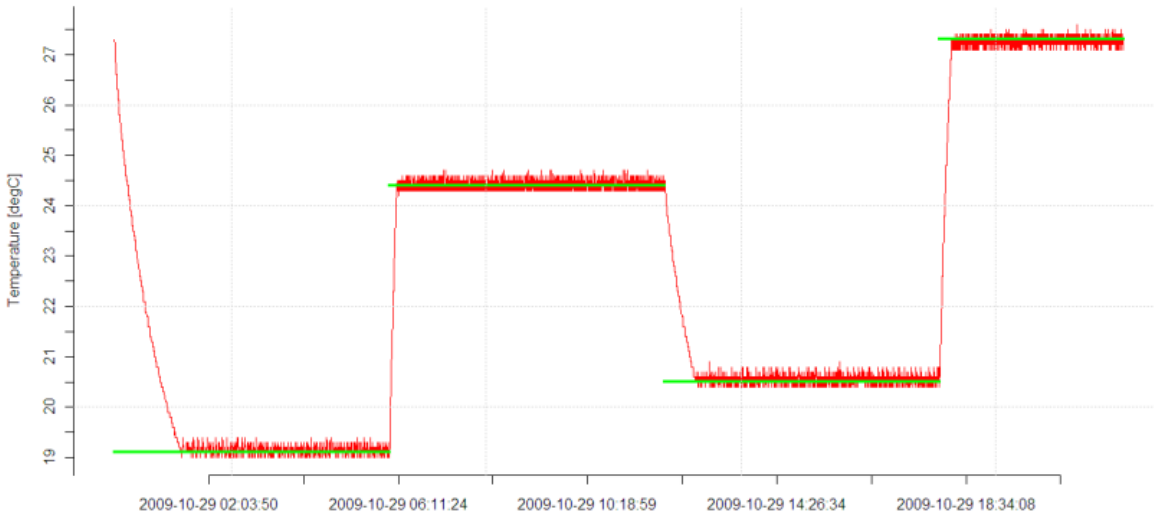

Figure 4.7: PID control method temperature graph.

method. This may be due to the PID tuning parameters, that remained the same. However, had a PID tuning variable method been implemented as specified in section 3.3.4, the results may have turned out in favor of the PID control method. However, to implement this method would take extensive system testing with different parameters and, due to this work's goal and time constraints, is not feasible.

4.4.3 Predictive tree control method

.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	2080	3149.785	259
	06:00 - 11:59	5440	2449.405	493
	12:00 - 17:59	2890	420.21	381
	18:00 - 22:00	4390	5078.63	217
$\sum$	22:00	14800	11098.03	1350

Table 4.7: Statistics for predictive tree control method.

The predictive method using the tree data structure failed to properly

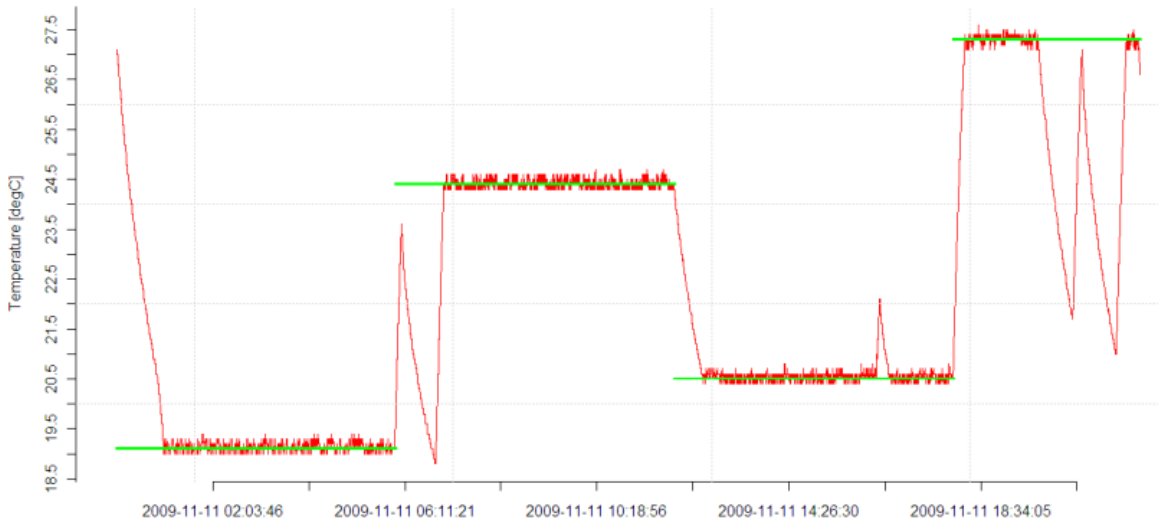

Figure 4.8: Predictive tree control method temperature graph.

sustain desired temperatures as seen in figure 4.8. Having tried various other methods like those mentioned in 3.5, we have yet to find a suitable variant of the predictive tree control method.

4.4.4 Predictive grid control method

.

-	Time of day	Heating time	Error	Switching
	00:00 - 05:59	1960	3173.78	232
	06:00 - 11:59	1690	59718.77	86
	12:00 - 17:59	2960	1013.565	344
	18:00 - 22:00	1790	43290.46	20
$\sum$	22:00	8400	107196.6	682

Table 4.8: Statistics for predictive grid control method.

The predictive method using the grid data structure and the linear model had still the worst results (figure 4.9).

Figure 4.9: Predictive grid control method temperature graph.

Chapter 5

Possible enhancements

There is quite a lot of potential in extending this work. First of all, more sensors could be incorporated and temperatures consequently somehow aggregated to better reflect the room temperature situation. It would definitely be better that having one sensor. When talking about multiple sensors, multiple rooms could also be a feature. Even though multiple instances of the application can of course regulate multiple rooms, the extension is meant as multiple rooms connected by "air ways" that could provide a more accurate simulation of a building.

There is no reason to limit the control system to heating. An array of air conditioners or perhaps freeze boxes could make use of this project, since setting the heat efficiency to a negative number would lead to a refrigeration model. Also, changing the specific heat capacity, the application could be used for its previous intention - boiler water heating.

Considering methods of control, there are myriads of possibilities ranging from neural networks, fuzzy logic systems, heat graphs[1] to improving the current control method like using fuzzy logic or neural network to find tuning parameters for PID control or the heat model.

[1]This method would use graph vertices as temperature values and oriented edges valued as length of time between measurements and power values. Although similar to the predictive graph being used in this work, this special graph would replace the whole heat model.

5.1 Hardware and heat model limitations

There are quite a few limitations regarding this project that result in inaccuracies and consequently sub-optimal control. The main limitation is the heat model. Far too many factors are present during heating to make a totally truthful physical model. The simulation of every air particle[2] inside the room, specific heat capacity of wall material, radiation of heat from the outside through walls, the function of air displacement and many more would be needed to faithfully represent the physical system accompanied by a much more power computer than we had at our disposal. Hardware reliability was also a problem; wires kept twisting and disconnecting, but that was a construction issue.

5.2 Future development

Although the application is not fully production-ready, plans of testing an inhabited house in winter are under way. If this project were to be given the green light, additional work would be required to interface the application with the server database which collects temperature data from sensors and another way to control the heating system since it is of a proprietary nature. The USB-Serial port would become obsolete. More importantly, multiple instances of the application would have to run simultaneously because there is more than one thermal sensor in the house.

The idea of prediction in control in general is not new and the basic principle of having a simpler predictive physical model for control purposes could be applied elsewhere as well, for example in motion control, the mentioned refrigeration system, etc.; the main advantage being faster computation and ease of correction in case the model becomes flawed.

[2]Molecules of gases, water vapor, dust.

Chapter 6

Conclusion

We have shown how we can take a heat model and use a variation of the gradient descent optimization algorithm to find the model's free parameters. In chapter 3 we discussed two control algorithms that are used in heating systems as well as the proposed algorithm that was assumed to surpass the previous control algorithms.

Experimental results in chapter 4 revealed that the proposed algorithm based on the non-linear model was capable of providing adequate control with respect to reactive and PID control algorithms, however only when ambient temperature was close to the desired temperature. Otherwise the non-linear model based algorithm fails to provide proper temperature control. The proposed algorithm based on the linear model was not so capable and will require further research and tuning if it is to be used.

The project was originally meant to function in a real building, however due to disagreements and technical problems this idea was scrapped. Algorithms that had been proposed to be implemented in an environment with higher stability and heat inertia were used in conditions with different characteristics, which could have had an impact on results. On the other hand, the newly proposed environment offered better control over experiments and their numerous, and even so prolonged, duration. Even though basic control methods provide acceptable efficiency in these simplified conditions, this work proposes and evaluates more complex approaches that achieve similar results. The question of benefits of these new approaches in a real and more challenging environment remains open.

Since the inception of this project, scientific areas such as artificial intelligence, software simulations of physical environments and control theory

were involved. Other areas include electrical engineering and software design patterns used in the application.

Appendix A

Calheating application documentation

Here we give the application documentation, which includes resource requirements, installation guide, formats as well as an overview of source code files.

A.1 Resource requirements

There are some requirements that need to be fulfilled in order to run the application. A computer with Java Runtime Environment version of least 1.6 plus some additional libraries that are also on CD. These libraries are:

- Swing Application Framework - appframework-1.0.3
- Swing Application Framework - swing-worker-1.1
- RXTX JAVA Communication API 2.1-7[23]

Basically, there are two modes the application has:

- Data generator simulation
- Real heating system control

For the simulation, one does not need more than the aforementioned requirements. For the application to be used with a real heating system, the application needs to be interfaced with the heating system via a comm

port. This comm port may be simulated using a USB-to-Serial Comm Port interface[1]. Concretely, the serial port has to have a baud rate of 19200 bits per second, carrying 8 data bits, 1 stop bit and no parity. This can be changed in the source code, but there is no guarantee of functionality.

The format of incoming and outgoing data shall be described in section A.4.4, the hardware resources used in the experiments are described in section 4.2.

There are no specific weather constraints, only those that apply to hardware[2]. However, considering the temperature that the experiments were conducted in, the application has not yet been tested under 18° C or over 80° C and therefore it is not known whether unexpected results may be attained.

Concerning the operating system, Windows XP has been used through out the application's development and testing, and although it is programmed in Java, there is no guarantee that it will function properly on another operating system.

A.2 Installation and usage

The application is installed by copying the "calheating" directory to a hard drive and executing the "calheating.bat" file contained inside the directory, after which the application frame appears as in figure A.1.

In the "File" menu, you may choose from the following items:

- Load configuration files
- Test - parameter comparison
- Exit

"Load configuration files" displays a load file dialog from which you can choose the configuration file for the data generator or the general configuration file depending on what kind of file type you select.

"Test - parameter comparison" compares the fidelity of the heat model with actual heating data. It is dependent on the "generate data" check box. If the check box is unchecked, the application will look for a heating data log having the file name "0.log". It will then extract a subset of heating data values plus 30 additional values for testing. After the heat model has been

[1] As it is used in our experiments.

[2] Extreme temperatures may damage thermal sensors or computers.

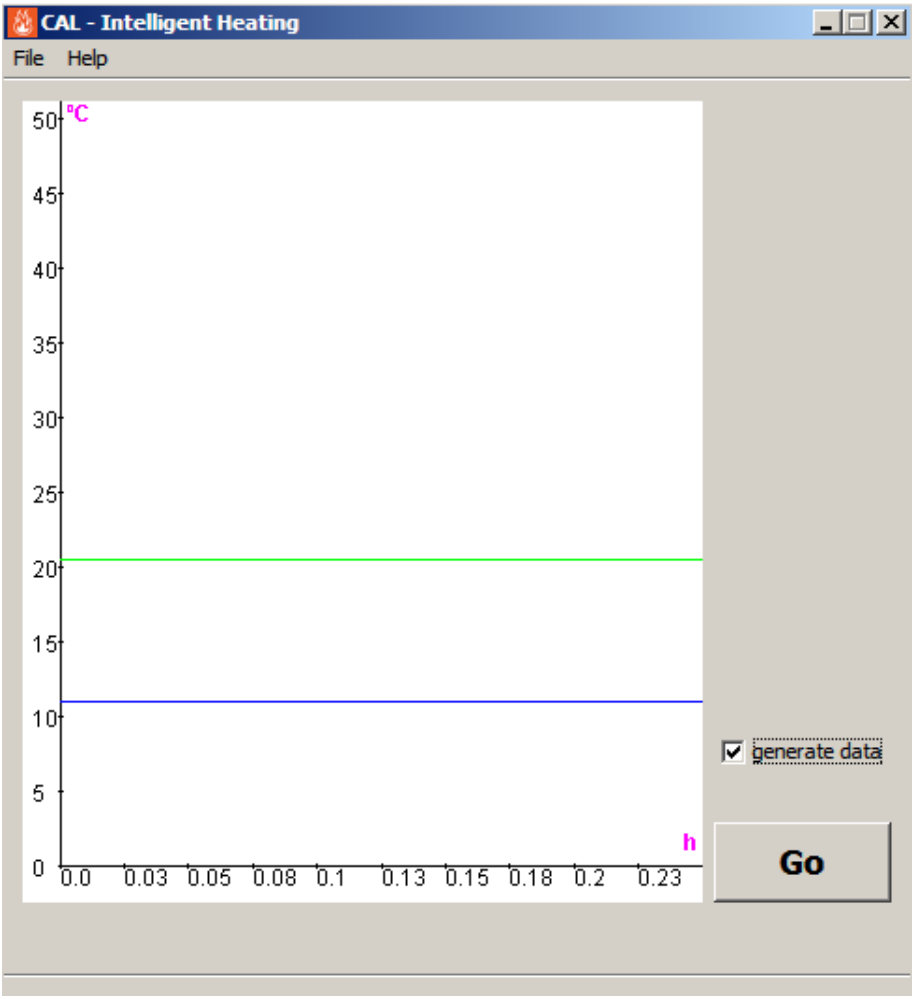

Figure A.1: Calheating application frame

discovered with the use of the subset of heating data, the model is tested against the 30 additional heating data values[3]. Model discovery is done 10 times for each subset and the resulting error value is the average error. The results are written to a file named "corrective_check.dat".If the check box

[3]The heat model will take temperature before heating, length of heating and if heating is to be turned on or not. Then the temperature after heating is calculated via the heat function. The resulting temperature is compared with the real temperature after heating a an error value is calculating by taking the absolute difference of the two temperature values.

is checked, then the data generator will randomly generate the three heat model parameters and with them will generate heating data the size of the maximum size that is set in the configuration file. Then a heat model will be discovered based on that data and will be tested against the "real" data generated from the randomly generated parameters.

The "Exit" menu item exits the application.

In the "Help" menu, you may choose the "About" menu item, which displays a frame containing information about the application.

The "Go" button operates depending on the "generate data" check box. If the check box is unchecked, the application tries to connect to the heating system via the USB-Serial port and tries to control the heating with the control method set in the configuration file. The "Go" button changes to the "Stop" button and until the "Stop" button is pressed. The application writes heating data records to a file named "0.log". When the number of data records reaches the amount set in the configuration file, the application create a file named "1.log" and the process repeats. It can be deducted that an internal counter is incremented each time the number of data records reaches the desired amount and the prefix of the file name is the value of the counter.

A.3 Source files

The applications source files are divided into one primary package "calheating" and three sub packages called "control", "io" and "model". We will briefly describe each Java source file. Each Java source file name is the same name that the class or interface contained in the file has. The Java documentation of the application is located on the attached CD.

calheating.CalheatingAboutBox.java

This file contains the dialog box class for displaying information about the application.

calheating.CalheatingApp.java

This file contains the main class of the application and the code for displaying the applications main frame. It also contains a utility function that reads values from configuration files as well as a debugging switch (boolean variable) for turning debugging prints on or off.

calheating.CalheatingView.java

This file holds the main frame view class of the application that is seen after starting the application. It reads a part of the general configuration file, namely the time factor and type of control method. It is also used to initialize the heat model and to start the heat control thread. It contains the menu of the application as well as all the code for displaying dialog boxes and testing the heat model.

calheating.LoadDialog.java

This file contains the dialog box class for choosing the general and data generator configuration files depending on what type of file is chosen.

calheating.control.Onoff.java

This file contains the reactive control method class as well as the predictive control methods along with the "ErrorNode" nested class that is used in predictive control. It also reads a part of the general configuration file. This class is designed from a singleton design pattern.

calheating.control.PID.java

This file holds the class containing the P, PD, PI and PID control methods, although only the PID control method is accessible through the general configuration file. This class is designed from a singleton design pattern.

calheating.control.TheGrid.java

This file contains the grid data structure, prediction methods as well as "ErrorNode" nested class that is used in prediction method.

calheating.control.TheTree.java

This file contains the tree data structure, prediction methods as well as "ErrorNode" nested class that is used in prediction method. The "ErrorNode" nested class is different than the equally named class in the "TheGrid" class, since it uses less variables and no methods except the constructor method.

calheating.io.DataIO.java

This file contains an interface for classes that connected to transferring data between the application and a heating system. The "OutsideManager" class and the "DataGenerator" class implement this interface.

calheating.io.OutsideManager.java

This file contains the class that connects to the make-shift heating system via the USB-Serial port and the "RXTXcomm.jar" library . It implements the "DataIO" interface. It also reads a part of the general configuration file. This class is designed from a singleton design pattern.

calheating.model.DataGenerator.java

This file contains a data generator that generates heating data from the heat model. It is used primarily for testing purposes since a real heating system can be slow to react. It reads the whole data generator configuration file. This class is a designed from a singleton design pattern.

calheating.model.DrawGraph.java

This file contains the panel that is used for graph plotting. This class extends the "JPanel" class. It contains methods for pixel-to-time conversion and vice-versa as well as all the parameters (color, margins, ...) for drawing the background of the graph.

calheating.model.FVector.java

Since the gradient descent method for discovery of the heat model searches for a vector of three values and since it is required to get the norm and prod-

uct of two of these vectors, this class has been created to make some of those calculations more comprehensible.

calheating.model.HeatModel.java

This file contains the heat model class and for example the heat function and the three parameters. It also has methods for returning the desired temperature and all the environmental parameters from the general configuration file. This class is designed from a singleton design pattern.

calheating.model.ModelDiscovery.java

This file contains the class used for discovering the three parameters of the heat model. It has methods for using the accelerated gradient descent algorithm as well as reading parts of the general configuration file. This class is designed from a singleton design pattern.

A.4 Formats

In this section we will describe the various text formats that are used in the application.

A.4.1 Configuration file format

Configuration files set various parameters of the application's methods. The format of a single parameter has the following template:

```
#<parameter_name>
//<comment>
<value>
```

It is easy to deduct that the "< parameter_name>" is the tag that the application identifies with one of its parameters. The "<comment >" tag is a comment used to clarify the parameter's purpose. The "< value>" tag is the parameters value.

The general and data generator configuration files (named "calheating.conf" and "datagen.conf" respectively) can be found on the attached

CD. There is a comment to every parameter and together with the parameter name the parameters purpose can be easily figured out.

A.4.2 Output log file format

This is the heating data file format. Every line of this file contains one heating data record that is structured the following way:

`<ts>;<temperature_bh>;<power>;<timelen>;<temperature_ah>\n`

where

`<ts>` - time stamp

`<temperature_bh>` - temperature before heating

`<power>` - power value (”0” or ”300”)

`<timelen>` - time length between measurements

`<temperature_ah>` - temperature after heating

The “<timestamp>“ tag is the current time in milliseconds returned by calling the Java System.currentTimeMillis() method. It returns the time difference measured in milliseconds between the current time and midnight, January 1^{st}, 1970.

The “<temperature_before_heating>“ tag is the temperature value at the “<timestamp>“ time.

The “<time_length>“ is the time length in seconds between temperature measurements.

The “<power>“ tag is the power value in Watts that the heating system heats the room. This value may be zero, when heating is unnecessary.

The “<temperature_after_heating>“ is the temperature value after the room has been heated[4], where the time is the sum of the “<timestamp>“ value and “<time_length>“[5].

[4]Or not heated, depending on the power value.

[5]The time length value has to be multiplied by 1000 in order to get the millisecond value.

To conclude the data record format, the “;“ character is a value separator and the “
n“ character is the data record separator which is the line feed character and is not displayed.

A.4.3 Correction check file format

This file is the output of heat model testing. It has the following format:

```
<data_size>;<error>\n
```

The “<data_size>“ tag shows how large the internal data queue was for discovering the heat model. The “<error>“ tag is the error value, which calculation is described in section 4.3.

Again, the “;“ character is a value separator and the “
n“ character is the data record separator which is the line feed character and is not displayed.

A.4.4 Input/Output data stream format

The input data stream from the thermal sensor had the temperature value enclosed in brackets[6]. The output data stream consisted of two values, which are “0“ for switching the relay contacts off and any other positive integer (integer “1“ was used) for switching the relay contacts on.

[6]The left bracket is the character “[“, the right bracket is the character “]“

Bibliography

[1] Barto A. G., Sutton R. S.: *Reinforcement Learning*, MIT Press, Cambridge, MA, 1998.

[2] Russel S. J., Norvig P.: *Artificial Intelligence: Modern Approach, Pages 94-96*, Prentice Hall, 1995

[3] Ghallab M., Nau D., Traverso P.: *Automated Planning: Theory and practice*, Morgan Kaufmann, 2004

[4] Friedewald M., Da Costa O., Punie Y., Petteri A. and Heinonen S.: *Perspectives of ambient intelligence in the home environment; Telematics and Informatics, Volume 22, Issue 3, Pages 221-238*, August 2005

[5] The Ambient Assisted Living Joint Programme
`https://www.aal-europe.eu`

[6] Gradient descent
`http://en.wikipedia.org/wiki/Gradient_descent`

[7] Water heating
`http://www.tzb-info.cz/t.py?t=16&i=97&h=38`

[8] Neculai A.: *An acceleration of gradient descent algorithm with backtracking for unconstrained optimization; Numer Algor 42, Pages 63–73*, January 2006

[9] Equations - Air Density and Density altitude
`http://wahiduddin.net/calc/density_altitude.htm`

[10] Raznjevič K.: *Handbook of Thermodynamic Tables and Charts*, McGraw-Hill, New York, 1976

[11] Le Ch., Baoming G., de Almeida A.: *Self-tuning PID Temperature Controller Based on Flexible Neural Network; Advances in Neural Networks – ISNN, Pages 138-147*, 2007

[12] Laider, Keith, J.: *The World of Physical Chemistry, Oxford University Press*, 1993

[13] Control Theory
`http://en.wikipedia.org/wiki/Control_theory`

[14] Cooper D.: *Practical Process Control: Proven Methods and Best Practices for Automatic PID Control*
`http://www.controlguru.com/pages/table.html`

[15] Relay Contact Life, pg. 2
`http://relays.tycoelectronics.com/appnotes/app_pdfs/13c3236.pdf`

[16] Heating data processing in "R"
`http://www.r-project.org/`

[17] Matoušek J., Nešetřil J.: *Kapitoly z diskrétní matematiky, Pages 95-96, Page 140*, Nakladatelství Karolinum, 2003

[18] PID controller
`http://en.wikipedia.org/wiki/PID_controller`

[19] Liptak B.: *Instrument Engineers' Handbook: Process Control, Pages 12-30*, Chilton Book Company, 1995

[20] Cohen-Coon PID tuning
`http://www.chem.mtu.edu/~tbco/cm416/cctune.html`

[21] Classical PID tuning
`http://csd.newcastle.edu.au/book_slds_download/Ch06c.pdf`

[22] Used icons
`http://freevectorgraphics.net`

[23] RXTX JAVA Communication API 2.1-7
`http://users.frii.com/jarvi/rxtx/download.html`

[24] J. R. Shewchuk: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, Pages 18-19, Pages 21-24*
http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

[25] Thesaurus.com
http://thesaurus.reference.com/