Control of HVAC testbeds: remote access and software tools for cooperative research

FERRAN MONTES TORN

Master's Degree Project
Stockholm, Sweden June 2013

XR-EE-RT 2013:014
## Contents

1 Introduction 9
   1.1 Motivations ................................................. 9
   1.2 What is necessary to be developed ....................... 9
   1.3 Literature review ........................................ 10
      1.3.1 General methodologies used in HVAC systems ....... 11
   1.4 Statement of contributions .............................. 14
   1.5 Structure of the manuscript ............................. 14

2 Description of the KTH testbed 15
   2.1 Peculiarities of KTH HVAC Testbed ...................... 15
      2.1.1 KTH testbed: Description of our testbed .......... 15
      2.1.2 Soft PLCs and the default controller ............... 20
      2.1.3 Wireless Sensor Network (WSN) ....................... 20
      2.1.4 People counter ........................................ 23
      2.1.5 Programmable Logic Controller (PLC) ............... 23
   2.2 Mathematical modeling ................................. 23

3 Tool description 27
   3.1 Class diagram ............................................. 27
   3.2 Controller classes ....................................... 29
   3.3 Communication classes .................................. 35
   3.4 General classes ......................................... 37
   3.5 Controllers .............................................. 40
      3.5.1 MPC .................................................. 40
      3.5.2 PID .................................................. 42
      3.5.3 LQR .................................................. 44
   3.6 Implementation on the KTH testbed ..................... 45

4 Manual 47

5 Experimental results 51

6 Conclusions 53
   6.1 Issues and further developments ....................... 53

References 54
List of Tables

2.1 Summary of the mote of the WSN (T, H, C, L stand for temperature, humidity, CO$_2$ and light respectively) ........................................ 22
3.1 Methods included in the Controller class .................................... 29
3.2 Methods included in the MPController class ................................. 30
3.3 Methods included in the MPControllerLinear class ....................... 31
3.4 Methods included in the MPControllerQuadratic class ................... 31
3.5 Methods included in the LQRController class ............................... 32
3.6 Methods included in the PIDController class ............................... 32
3.7 Methods included in the ControllerInterface class ....................... 33
3.8 Methods included in the KTHConnection class ............................. 35
3.9 Methods included in the TCPConnection class ............................. 36
3.10 Methods included in the Experiment class .................................. 37
3.11 General options ........................................................................ 38
3.12 Variable options ....................................................................... 38
3.13 Comfort options ........................................................................ 38
3.14 Controller options ..................................................................... 39
3.15 Connection options .................................................................... 39
3.16 Experiment options .................................................................... 39
List of Figures

2.1 Kungliga Tekniska Högskolan - Royal Institute of Technology (KTH) Q building 15
2.2 Map of the second floor ......................................................... 16
2.3 Scheme of the whole HVAC system ............................................ 17
2.4 The temperature of the hot water that is carried from the central system to the radiators depends on the outside temperature conditions .......................... 17
2.5 The radiator, the air inlet and outlet and the air conditioning outlet of the water tank lab (WTL) .................................................... 18
2.6 Scheme of the air inlet of the WTL ............................................... 19
2.7 Scheme of the air conditioning system of the WTL ......................... 19
2.8 SCADA interface of the system that provides the fresh air to the venting and cooling system ......................................................... 20
2.9 Tmote Sky ............................................................................. 21
2.10 Typical star network topology, it is used also in our network .............. 21
2.11 Map of the sensors deployment ................................................ 22
2.12 The people counting devices: in order left to right the IRC3030 and the IRC3000 23

3.1 UML diagram with overall class structure of the library implemented ...... 28
3.2 MPC diagram .......................................................................... 40
3.3 MPC diagram .......................................................................... 41
3.4 PID block diagram with back-calculation anti-windup. When the computed output is above the saturation limit, the difference between the computed output and the saturation level is integrated in the next iteration ................. 43
3.5 This figure illustrates the consequences of a windup situation. The output of the PID and input of the plant has been cut to fit in the saturation limits . . 44
3.6 LQR block diagram .................................................................... 44
3.7 The KTH testbed server manages the testbed control decisions which can come from an external user through the control library or from a simple PI controller located in a PLC next to the server ............................... 45
3.8 Communications flow diagram ................................................... 46

4.1 Block diagram of the experiment proposed describing how to use the implemented library ................................................................. 48

5.1 Room temperature measurements and supply air temperature or temperature set point ............................................................... 51
5.2 Actions taken from the optimization problem. After half an hour, the bounds for the temperature change and the system stops to heat the room and starts to cool it .................................................... 52
Abstract

This thesis describes an object oriented programming software that implements a set of automatic controllers for the management-over-the-cloud of Heating, Venting and Air Conditioning (HVAC) systems. More precisely, the toolbox provides general Model Predictive Controls (MPCs), Proportional, Integrative and Derivatives (PIDs), and Linear Quadratic Regulators (LQRs) objects for automatically and remotely actuate any HVAC system over the Internet. The aim is to endow users with some libraries that enable the fast and easy prototyping of control strategies, and thus to boost research on the management of variables such as temperatures, humidity and CO$_2$ levels in buildings.

The library has been then exploited to manage a laboratory testbed located at the KTH in Stockholm, and has been shown to provide the research community an instrument for sharing research results and efforts.
Chapter 1

Introduction

1.1 Motivations

It is a well known fact that the worldwide population tends to increase and so it does the worldwide energy demand. In addition, we have seen how the energetic demand in developing and heavily populated countries like China, India or Brazil is increasing year after year while inevitably the natural resources around us decrease. So in order to avoid an unsustainable energetic situation, the necessity to optimize the usage of available resources has become a great concern and many research work has been done in that direction. More precisely, if we take a look to the energetic demands of a developed country, we see that the use of Heating, Venting and Air Conditioning (HVAC) systems in office buildings for comfort purposes represents a high percentage of the overall demand so trying to find a way to control this kind of systems in an efficient way has become more and more important.

The increased computer capabilities experienced in the last years has introduced some more options to control HVAC systems in an efficient way. So when talking about the main motivations for working in these topic it has to be mentioned the high energetic impact of HVAC systems and the possibility to try new control algorithms. Many other researchers are working on increasing the efficiency of HVAC control systems. The main aim of this thesis is then to provide them a library that allows the fast implementation, combination and modification of control algorithms. This report thus explains the design and the usage of this library.

1.2 What is necessary to be developed

In the last years computers have experienced a huge increase in the computational power, a decrease in their size and an extraordinary decrease in their costs. This boosted the development of new control techniques and new possibilities on how to monitor and interact with physical variables in a controlled environment. In particular, HVAC control algorithms now may require a big amount of computational capabilities, since they can be run in almost any computer available. Practitioners are thus enabled to choose from several different control strategies, from the classical ones to the most advanced Model Predictive ones.

The availability of all these options then is creating the necessity for a tool that enables to easily switch from one controller to another, with as less effort as possible. Fast changing the control algorithm in fact enables fast prototyping and testing of several control algorithms combinations.

Our vision is thus to exploit the increased capacity of both computational and networking capabilities to allow the controller designers to remotely and fast do their job. We thus want to allow various different practitioners (or researchers) to overcome physical barriers, and thus to cooperate easily from different research institutions. KTH is thus opening its
control testbed to the research community, hoping to provide a common controllers testing environment where different control schemes can be switched and combined online and efforts-less.

1.3 Literature review

This section describes a list of research projects focusing on building control technologies. The aim is thus to describe what kind of approaches have been taken up to now to manage HVAC systems, and thus to understand what kind of needs the developed library must cope.

A common approach in various building control projects is to exploit WSNs to acquire the data and to detect user activity. Importantly, there is a strong requirement of not resorting to any advanced sensing technology that is expensive, generating privacy concerns or requiring changes in user behavior. E.g., cameras, Radio Frequency Identification (RFID) tags, or wearable sensors. Instead, simple, wireless, binary sensors are favored since they are cheap, easy to retrofit in existing buildings, require minimal maintenance and supervision, and do not have to be worn or carried. In a schematic way, the most used hardware tools are:

- temperature, humidity, light and CO$_2$ sensors (wired, embedded on a mote or soldered on a mote's board);
- acoustic sensors (also microphones);
- Passive Infrared (PIR) sensors (for motion detection or people counting);
- switch-door sensors (magnetic);
- cameras;
- RFID tags.

Simple sensors are used in many energy intelligent buildings in the interest of activity recognition. For instance, PIR based sensors are often used (especially with lighting system) for occupancy detection. The sensors are connected directly to local lighting fixtures. These PIR sensors are also simple movement sensors and often cannot actually determine if the room is occupied or not. E.g., if the persons stand still, they will fail in detecting the occupancy. PIR sensors are used for instance in [1], where Padmanabh et al. investigate the joint use of microphones and PIR sensors for inferring the scheduling of conference rooms. An other project where PIR sensors have been used is the AIM Project [2], where authors used sensors to get some physical parameters, like temperature and light, as well as PIR to infer user presence in each room of a house.

Indoor activity recognition is in general implemented to provide inputs to a control strategy that aims for energy savings in buildings. To this aim, other sensors than PIRs can be used: in Greener Building [3] authors perform indoor activity recognition using simple infrared, pressure and acoustic sensors. An other strategy is to use door sensors. For instance [4] uses them in conjunction with PIR sensors to automatically turn off the HVAC system when the occupants are sleeping or away from home. Also in [5] Agarwal et al. chose to use a combination of a magnetic reed switch door sensor and a PIR sensor module to build their building occupancy model. An even more complete testbed has been constructed in ARIMA [6]. Here, to gather data related to total building occupancy, wireless sensors are installed in a three-story building in eastern Ontario (Canada) comprising laboratories and 81 individual work spaces. Contact closure sensors are placed on various doors, PIR motion sensors are placed in the main corridor on each floor, and a carbon-dioxide sensor is positioned in a circulation area. In addition, the authors collect data on the number of people who log in to the network on each day. This thus gives possibility to the managers of the building to be aware of the air quality and to have CO$_2$ levels indications.
Other approaches are based on several simultaneous Sensor Networks. E.g., in [7] there are 3 independent complex sensor networks: one, Labview-based, to acquire the indoor data (temperature at different height, humidity and CO$_2$), one to acquire the outdoor environmental data (temperature, humidity, lighting, acoustic and motion), and one to data log the system.

Information on the state of the considered system can of course be gathered also using measurement technology that directly infer the behavior of the occupants. An example is iDorm [8], where pressure pads are used to measure whether the user is sitting or lying on the bed as well as sitting on the desk chair. At the same time, a custom code that publishes the activity on the IP network senses computer-related activities of the user. The testbed thus measures several activities, like whether the occupant is running the computer's audio entertainment system or the video one. Other approaches are also to use entry-exit logs of the building security systems [9], or active badges, cameras, and vision algorithms. E.g., Erickson et al. propose a wireless network of cameras to determine real-time occupancy across a larger area in a building, [10, 11]. In [12] and [13], instead, the occupants should be equipped with sensor badges, with which it is possible to achieve relatively accurate localization using, for example, RFID tags.

Similarly, in SPOTLIGHT [14], the authors present a prototype system that can monitor energy consumption by individuals using a proximity sensor, while the building used in [15] is featured with an ultrasonic location system that is a 3D location system based on a principle of triangulation and relies on multiple ultrasonic receivers embedded in the ceiling and measures time-of-flight to them. The location system provides three-dimensional tracking solution.

In conclusion, in building energy and user comfort management area, WSNs can play an important role by continuously and seamlessly monitoring the building energy use, which lays the foundation of energy efficiency in buildings. The sensor network provides basic tools for gathering the information on user behavior and its interaction with appliances from the home environment. Sensor Networks can also provide a mechanism for user identification, so that different profiles can be created for the different users living in the same apartment / house.

We finally remark that, in contrast with other smart home applications such as medical monitoring and security system, applications focusing on energy conservation can tolerate a small loss in accuracy in favor of cost and ease of use. Specially in building automation, occupants prefer to spend a little bit more but do not have to suffer to adapt to a new technology. Therefore, an energy intelligent building might not require cameras or wearable tags that may be considered intrusive to the user. Nevertheless, wireless sensor networks are today considered the most promising and flexible technologies for creating low-cost and easy-to-deploy sensor networks in scenarios like those considered by energy intelligent buildings.

1.3.1 General methodologies used in HVAC systems

There is an abundant literature on different approaches for the control of HVAC systems and for the treatment of the relative information.

Management of information on occupancy patterns as already said, one of the most influencing parameter in the management of HVAC systems is occupancy and occupant behavior.

Real-time occupancy is usually detected by means of inference techniques, where the information comes from sensor data [2, 3, 4, 5, 6, 1, 8]. It is in general also possible to exploit Bayesian inference techniques, usually with the aim of predicting the behaviors of the occupants.

In AIM system [2], Barbata et al. build user profiles by using a learning algorithm that extracts characteristics from the user habits in the form of probability distributions. A sensor network continuously collects information about users presence/absence in each
In the house in a given monitoring period. At the end of this monitoring time the cross-correlation between each couple of 24 hour data presence patterns is computed for each room of the house in order to cluster similar daily profiles.

In OBSERVE [10, 11], Erickson et al. construct a multivariate Gaussian model, a Markov Chain model, and an agent-based model for predicting user mobility patterns in buildings by using Gaussian and agent based models. The authors use a wireless cameras network for gathering traces of human mobility patterns in buildings. With this data and knowledge of the building floor plan, the authors create two prediction models for describing occupancy and movement behavior. The first model comprises of fitting a Multivariate Gaussian distribution to the sensed data and using it to predict mobility patterns for the environment in which the data is collected. The second model is an Agents Based Model (ABM) that can be used for simulating mobility patterns for developing HVAC control strategies for buildings that lack an occupancy sensing infrastructure. While the Markov Chain is used to model the temporal dynamics of the occupancy in a building.

In [16] authors propose a general method to predict the possible inhabitant service requests for each hour in energy consumption of a 24-hour anticipative time period. The idea is to exploit Bayesian networks to predict the user’s behavior. In [17] authors adopt neural networks modeling the occupants behavior, and in cascade to this they create a system able to control temperature, light, ventilation and water heating.

In [18] authors propose a belief network for occupancy detection within buildings. The authors use multiple sensory input to probabilistically infer occupancy. By evaluating multiple sensory inputs, they determine the probability that a particular area is occupied. In each office, PIR and telephone on/off hook sensors are used to determine if rooms are in occupied states. The authors use Markov chains to model the occupied state of individual rooms, where the transition matrix probabilities are calculated by examining the distribution of the sojourn times of the observed states.

All these works focus on the creation of occupancy models, that are then exploited for control purposes. There are indeed several manuscripts reporting usages of these models. E.g., in [19] daily occupancy profiles of occupancy are used in conjunction with a simple PID controller that works only on the heating accordingly to the profile, trying to set the indoor temperature to a certain set-point. In [4] the HVAC system is turned on or off when the occupants are away or asleep. This smart thermostat uses a Hidden Markov Chain (HMC) model to estimate the probability that the home is in one of the states away, active or sleep with transition every 5 minutes.

A lot of literature considers also model predictive control strategies applied to occupancy models. For example, in [7] Dong et al. use a Gaussian Mixture Model to categorize the changes of a selected feature. These are observations for an Hidden Markov Model that estimates the number of occupants. To estimate the duration of the occupants in a certain area it has been used a Semi Markov Model based on the pattern of the CO2 acoustic, motion and lightning changes. All these information are given to a Non-Linear MPC that, solved by dynamic programming, gives the optimal control profile to use.
Management of weather forecasts predictions  we then notice that an other big issue in smart HVAC control is how to manage predictions of weather forecasts. In general, predictive strategies (in the sense that account for weather predictions and their uncertainty) turn out to be more efficient and promising compared to the conventional, non predictive strategies in thermal control of buildings [20],[21],[22],[23],[24],[25].

In [23] authors have developed both certainty-equivalence controllers using weather predictions and a controller based on stochastic dynamic programming for a solar domestic hot water system. The strategies are based on probability distributions derived from the weather data. The simulation results have shown that the predictive control strategies can achieve a lower energy cost compared to a non-predictive strategy.

In [25] the use of a short-term weather predictor based on the real weather data in the control of active and passive building thermal storage inventory is explored. The predicted variables include ambient air temperature, relative humidity, global solar radiation, and solar radiation. A receding horizon policy is applied, i.e., an optimization is computed over a finite planning horizon and only the first action is executed. At the next time step the optimization is repeated over a shifted prediction horizon. It has been shown that the electrical energy savings relative to conventional building control can be significant.

A predictive control strategy using a forecasting model of outdoor air temperature has been tailored in [22] to account for intermittently heated Radiant Floor Heating (RFH) systems. The control action here consists in deciding when to supply the heat to the floor. In the conventional intermittent control technique the decision is based on the past experience. The experimental results show that use of the predictive control strategy could save between 10% and 12% of the total energy consumption during the cold winter months compared to the existing conventional control strategy.

Other MPC techniques focus instead in the manipulation of passive thermal storage systems. E.g., [24] exploits predicted future disturbances while maintaining comfort bounds for the room temperature. Both conventional, non-predictive strategies and the predictive control strategies are then assessed using a performance bound as a benchmark. The performance bound is an ideal controller, i.e., no mismatch between the controlled process model and the real plant and perfectly known disturbances. In general predictive controllers outperform the non-predictive ones because the room temperature can be kept within its comfort bounds with minimum energy, i.e., low cost energy sources are exploited as much as possible. This project considers both high-demand energy sources (e.g., chillers, gas boilers, conventional radiators) and low-demand energy methods (e.g., blind operation and evaporate cooling) for heating and cooling.

This is also the practice employed in standards 382/1 [26] and 380/4 [27] of Schweizerischer Ingenieur und Architekten Verein (SIA 2). Low-demand energy sources make use of the thermal storage capacity of the building and thus are slow and heavily dependent on weather conditions. Hence the model predictive control should fit very well: if predictions of the future system evolution can be computed, low cost energy sources can be used for controlling the building and meeting the occupants requirements. The aim is to avoid the conventional expensive energy sources as much as possible in favor of the low cost ones.

In this context it is important to consider also the uncertainty of the predictions. [28, 29, 30, 31] have all considered also these uncertainties, even if in different ways. In [28], authors incorporate a stochastic occupancy model within the control loop. [29] instead proposes a stochastic predictive building temperature regulator where weather and load disturbances are modeled as Gaussian processes. [30] also uses a stochastic MPC and weather predictions. Firstly it solves a non-convex optimization problem and then it applies a disturbances feedback. [31] finally considers stochastic approach on the uncertainty of the forecast disturbances (the outside temperature, the occupancy and the solar radiation) solving the problem considering a scenarios based approach and a statistical learning procedure to learn these statistics from real and local data.
Effect of automatic blinds and lighting control  also these systems have an important
effect on heating and cooling requirements. A noticeable paper in this subject is [21], where
the authors investigate the reduction in annual primary energy requirements for indoor
climate control achieved in Rome by applying automated lighting control.

Remarks  the current academic trend is to build testbeds and perform experiments on
it, to validate the simulated results. The most famous are the one from the ETH Zurich
(Switzerland) [32], and the one of the University of California, Berkeley, [33]. Confirming a
trend followed also in other automatic control frameworks, both these groups are applying
Model Predictive Control strategies.

We nonetheless notice that the testbed considered in the current thesis has its own
peculiarities, not only from topological points of view (i.e., the map of the building) but also
in the structure of the HVAC system (i.e., the actuation system) and, even more importantly,
in the climatic environment where the testbed lies. Thus some care must be placed in
comparing and assessing the architectural choices made by the designers of these testbeds.

1.4 Statement of contributions

Despite the fact that there exists a proliferation of projects focusing on the control of HVAC
systems, there exist no general tools that abstract the necessities and options available for
the control of these systems. The main contribution of this thesis is thus to describe these
abstractions, and then provide a software implementing the related interfaces.

The developed library implements then general objects with common interfaces for each
type of controller, making this the concept of "controller" abstract. This means that despite
the fact that each controller shares the same class structure, different objects can behave in
different ways. We notice that this choice makes easy the continuous improvement of the
library, i.e., the addition of more control options and the management of more situations.
The vision is thus to offer to the research community not only a useful software that stan-
dardizes and make prototyping easier and faster, but also a tool that can evolve in time and
adapt to novel necessities.

1.5 Structure of the manuscript

The first chapters of this report are dedicated to explain the HVAC technology. Section 2
offers a general view of the HVAC technology deployed in the testbed and a mathematical
introduction to the modeling of the KTH testbed, which has been the referring framework
and inspiration of this work. Then section 3 follows with a detailed explanation of the
produced software, among with a brief description of the theories of the implemented con-
trollers. The last sections, 4 and 5 respectively, propose some experiments, with the aim of
instructing the reader on how the tool respectively. Finally we draw some conclusions and
remark plausible further developments in section 6.
Chapter 2

Description of the KTH testbed

2.1 Peculiarities of KTH HVAC Testbed

2.1.1 KTH testbed: Description of our testbed

The testbed is implemented in the same building where is located the Automatic Control Laboratory of the KTH (Stockholm), more precisely on the second floor of KTH Q building.

As shown on map of the floor in Figure (2.2) there is a main corridor where all the rooms are facing (four laboratories, one conference hall, one storage room and one study room). Our attention is primarily concentrated on the room A:225 (the water tank lab), to date the only room with the permission to be controlled.

All the rooms of the second floor of the KTH Q building, except storage room and PCB Lab, are equipped with a HDH sensor on the wall surface to detect temperature and CO₂ level; so we can get temperature and CO₂ of the rooms A:213, A:225, A:235 and A:230 (see Figure 2.2 about better information on the location of these rooms). The thermal level and the air quality of these rooms can be controlled by the venting, cooling and heating actuators. However the KTH automation control group has actually the permission to work only on a limited area of the second floor; this is due not to affect the wellness of the people that spend several time in that area. The sub-area that can be controlled is the room...
A:225; more precisely this room is a laboratory, i.e. the WTL. The WTL is equipped with a large sensing system composed by:

- a WSN, that takes temperature, humidity, light and CO$_2$ values
- a motion detection sensor
- an occupancy sensor (actually it is a people counter).

Aside from this there is also an "external" PLC that allows the user to control the actuation signals switching the control mode from the internal mode (that executes the default control of the section 2.1.2) to the external mode, i.e. the one decided by the user.

Since the final scope of the project is to control some climate feature of the indoor environment, we are interested on how the climate system works. Basically there are three main source of climate control: ventilation system, heating system and the cooling system.

**The heating**

The heating system is based on commonly used radiators. In the WTL there are four radiators connected in parallel. As all common radiators, they are heated up by a hot water transition though their circuits. This water is heated thanks to a *district heating* system, of course provided by an external company. This water heats a secondary circuit. The water in the secondary circuit then is send in all the building with the aid of pumps and, of course, it is carried to the WTL radiators also. The temperature of the this water may be modified and it depends on the outside temperature conditions. However this temperature is set by the Akademiska Hus and it can not be modified at will. The default temperatures are shown in the Figure 2.4. Before the first radiator there is a valve whose opening percentages may be set by the user through a SCADA web interface. The system is built in a way to "help" the user: setting a certain percentage of actuation it tries to provide that percentage of the total amount of the available heating power.

---

1The district heating (less commonly called teleheating) is a system for distributing heat generated in a centralized location for residential and commercial heating requirements such as space heating and water heating. The heat is often obtained from a cogeneration plant burning fossil fuels but increasingly biomass, although heat-only boiler stations, geothermal heating and central solar heating are also used, as well as nuclear power. This system is important because it allows to do the heating conversion far from the city walls but above all it uses low quality heating sources, renewable energy sources.
Figure 2.3: Scheme of the whole HVAC system

Figure 2.4: The temperature of the hot water that is carried from the central system to the radiators depends on the outside temperature conditions
The venting and the cooling

The Q building is equipped with three separated ventilation units for fresh air supply. The fresh air flow for areas with special applications (like laboratories or conference halls) is regulated by Demand Controlled Ventilation (DCV) whilst, for floors housing office areas, there is a constant fresh air flow; in both these two cases venting is provided only in the day time in a specific time slot (from 07:00 to 16:00).

The venting and the cooling system are actually strictly connected in our particular case. In fact for the air system exists only one circuit. This circuit provide to the venting and cooling system fresh air at a temperature between 20 and 21 degrees at the same pressure in all the duct. This is due to a system composed by heat exchanger, pumps and a heating/cooling system.

The ventilation system is composed by two parts: one for getting fresh air and one for taking away the exhausted air. Fresh air is supplied in the duct by one of the ventilation units, while the damper regulates the airflow that comes into the room. The air outlet is just a hole in the wall, where is placed a tube, with a damper inside, to allow the air to flow through. This exhaust air flows into a duct. Then this air is carried to the heater exchanger. In this way the system is helping itself giving a “free” pre-heating to the fresh air that is coming inside. In the WTL the ventilation duct is split in two equal branches, both branches have two ventilation output and two air conditioning output.

The cooling system is based on air conditioning units that work on induction principle; a schematic drawing is shown in Figure 2.7. The primary air, supplied by one of the ventilation units, is injected to a plenum\(^2\). The plenum is equipped with nozzles of various sizes, small pipes from which the air can be discharged. Due to the high pressure in the plenum, the air comes out through the nozzles at a high velocity and creates a zone with lower pressure, since an increase in the velocity produces a decrease of the pressure. This depression causes

---

\(^2\)A plenum is a housing where it is created and stored air with a greater pressure than the atmospheric one.
2.1 Peculiarities of KTH HVAC Testbed

The room air to be sucked up through a cooling heat exchanger, which consists of a coil where chilled water flows. So, the sucked room air is cooled by this heat exchanger, mixed with the primary air and discharged into the room from the sides of the device.

The AC unit can also be used to heat the room: the only difference from the cooling process is that the water that is flowing in the coil is hot. Of course the temperature on the cooling coil is set by the Akademiska Hus and it cannot be changed at our will. As for the radiator, also this temperature depends on the outside temperature. During our experiments it was around 16°C.

To set the cooling actuation there is a valve before the water chilled circuit whose opening percentage can be set by the user from the SCADA web interface.

Seeing how is built the system, we can understand that for the cooling we need that the venting system is active. If the system is running only in a venting way, fresh air is coming from the cooling unit nozzles also (around the 30% of the total amount of available air flow). Viceversa when we want to run the system in the cooling mode we need an air flow to chill. So we need that the venting system be active but, however, in the room is flowing in the same time a air flow at around 20°C from the venting nozzles.

**Effects of the ventilation on the cooling/heating** The ventilation could affect heating and cooling. Figure 2.8 shows the entire air process in the schematic version that the user will find on the SCADA web interface. The fresh air from outside is imported through a valve (ST201), then it is filtered. After that it is processed by an efficient heater exchanger which exploits the heat of exhaust air flow. As shown in the Figure 2.8 the imported fresh
air is warmed up to 19.4°C. Then a pump (TF001) pushes the warmed air to the heating and cooling system sequentially; due to this the temperature of the fresh air that flows through the ducts is around 20°C in each room. Therefore the temperature of fresh air discharged into the rooms is always around 20°C. So, when the temperature of the room is below 20°C the air from the ventilation helps the heating system to increase the indoor temperature, otherwise if the temperature is greater than 20°C, above 23°C, the ventilation system helps to make the room temperature lower.

![Figure 2.8: SCADA interface of the system that provides the fresh air to the the ventilation and cooling system](image)

### 2.1.2 Soft PLCs and the default controller

In Q building, the deployed ventilation units, and cooling/heating processes are connected to three Soft PLCs. One Soft PLC is placed on the second floor which directly connected to the ventilation unit and related sensor and actuators of the second and third floor. The other two Soft PLCs are placed on eighth floor. Both of them can control the ventilation, the heating and the cooling systems from fourth to eighth floor.

Soft PLC is basically a software package which emulates the functionality of a standard PLC inside a PC. The product that is used is Fidelix Soft PLC. It has internet access and is able to communicate as an OPC client to an OPC server.

The Soft PLCs can be manually controlled over the SCADA system and its data are saved on the SCADA system server for archiving purposes. The Soft PLCs exploits the default controllers, designed by the Akademiska Hus, and they are programmed to deliver a set-point temperature of 21°C (with ±1°C dead band) and a CO₂ level below 850ppm defined by KTH Environmental and Building Department.

### 2.1.3 WSN

To gather further information about the indoor environment it has been implemented a WSN using sensor nodes, such as Tmote Sky. Tmote Sky includes a number of on-board sensor such as light, temperature and humidity sensor. In addition of that, other external sensors may be connected to the mote using the ADC channel on the 16-pin Tmote Sky.
2.1 Peculiarities of KTH HVAC Testbed

expansion area. In particular we soldered on the board of some motes a CO₂ sensor and a better temperature sensor (to avoid alteration in the measured temperatures by heat the microprocessor and to be able to reach places that wasn’t accessible in other ways). Currently in the WTL there are 10 motes, 4 motes are in the rooms beside, 1 is in the corridor and one is outside. All of them follow star network communication typology to send data to root mote which is marked as black in the Figure 2.11. In this application has been used as routing-layer protocol a Collection Tree Protocol (CTP). According to this some number of nodes in the network advertise themselves as tree roots. Nodes form a set of routing trees to these roots. CTP is address-free, i.e. a node doesn’t send a packet to a particular root; a node implicitly chooses his root selecting a next hop nodes generate routes to root using a routing gradient. A scheme is shown in Figure 2.10 The root mote is connected to PC and forwards these sensing data to database through serial forwarder every 30 seconds. Here below there is the Table (2.1) that summarizes the features of the motes we have in our network:
Figure 2.11: Map of the sensors deployment

<table>
<thead>
<tr>
<th>Mote Id</th>
<th>Spot</th>
<th>T</th>
<th>H</th>
<th>C</th>
<th>L</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>WTL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Root</td>
</tr>
<tr>
<td>1001</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Exhaust air outlet</td>
</tr>
<tr>
<td>1002</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Environment</td>
</tr>
<tr>
<td>1003</td>
<td>Corridor</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Corridor</td>
</tr>
<tr>
<td>1004</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Fresh air inlet</td>
</tr>
<tr>
<td>1005</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Room wall temperature</td>
</tr>
<tr>
<td>1006</td>
<td>Outside</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Outdoor wall temperature</td>
</tr>
<tr>
<td>1007</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Room wall temperature</td>
</tr>
<tr>
<td>1008</td>
<td>WTL</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>Surface of radiator hot water inlet</td>
</tr>
<tr>
<td>1009</td>
<td>WTL</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>Surface of radiator hot water outlet</td>
</tr>
<tr>
<td>1011</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Room wall temperature</td>
</tr>
<tr>
<td>1012</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Air conditioning outlet</td>
</tr>
<tr>
<td>1020</td>
<td>WTL</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>Environment</td>
</tr>
<tr>
<td>1035</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Environment</td>
</tr>
<tr>
<td>1036</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Room wall temperature</td>
</tr>
<tr>
<td>1037</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Ceiling</td>
</tr>
<tr>
<td>1038</td>
<td>WTL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Floor</td>
</tr>
<tr>
<td>1039</td>
<td>3rd Floor</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Floor</td>
</tr>
<tr>
<td>1109</td>
<td>PCB Lab</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Beside room environment</td>
</tr>
<tr>
<td>1110</td>
<td>PCB Lab</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Beside room environment</td>
</tr>
<tr>
<td>1111</td>
<td>PCB Lab</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Beside room environment</td>
</tr>
<tr>
<td>1213</td>
<td>Storage room</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>Beside room environment</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of the mote of the WSN (T, H, C, L stand for temperature, humidity, CO₂ and light respectively)
2.2 Mathematical modeling

2.1.4 People counter

To have the number of the people that are actually in the room it has been mounted a people counter over the entrance of the WTL. The people counter in our testbed is a thermal based people counter of IRISYS. It is composed by two modules: the dual view IP and the node IP master. The node IP master IRC3000 is a people counting devices with the imaging optics, sensor, signal processing and interfacing electronics all contained within a moulded plastic housing. The unit is used in a downward looking manner, as the unit functions optically recognize the heat emitted by people passing underneath as infrared radiation, collected through a germanium lens with a 60° field of view. The sensing area is a square on the floor whose width is approximately equal to the mounting height. The dual view IP IRC3030 multiplexes both the thermal people count data, the stored count data, diagnostic information and the video stream into a single IP data stream. The IP capability allows remote viewing, configuration and data collection over IP infrastructure, either over an in-store LAN, or for worldwide viewing from remote locations over the internet. With this functionality the occupancy data catch from these sensors are directly sent to the KTH network.

![Image](image.png)

Figure 2.12: The people counting devices: in order left to right the IRC3030 and the IRC3000

2.1.5 PLC

Besides the soft PLC there is an “external” PLC that allows us to communicate with our system (catch data and actuate). This PLC is the Fidelis FX-2025A. It is a freely programmable web server based on industrial PC with Windows CE operating system. The user interface of the FX-2025A is based on user-friendly html-files created in FideGraph editor. As the soft PLC, it has access to all the data from the central system (the one build up by Akademiska Hus) and can send command to the actuation devices. Moreover it is also connected to dedicated temperature, CO₂ and binary motion sensors, located in the WTL but also in other rooms of the second floor.

2.2 Mathematical modeling

This section introduces the mathematical models describing the dynamics of the CO₂ concentration and the room temperature of a laboratory in KTH. This room is in fact used as a testbed, as described in http://hvac.ee.kth.se/. In order to decrease the computational effort required, the models are based on a simplified physical model, that can be used for the simulation of whole buildings, both for cooling and heating purposes. The model is based on the following assumptions:
no infiltrations are considered, so that the inlet airflow in the room equals the outlet airflow.

- the thermal effects of the vapor production are neglected.

Remark: this library uses linear equivalent formulations of nonlinear CO$_2$ concentration models. This avoids intractable nonlinear dynamics that would complicate the control of the testbed with model-based controllers such as MPC strategies. The nonlinear room thermal model derived in [31].

Room temperature model: The heat flow due to ventilation can be expressed as

$$\dot{m}_{\text{vent}}c_{pa}\Delta T_{\text{vent}} = \dot{m}_{\text{vent}}c_{pa}(\Delta T_h - \Delta T_c) = c_{pa}(u_h - u_c)$$ (2.1)

where $\dot{m}_{\text{vent}}$ is the ventilation mass flow, expressed in kg/s, and where $c_{pa}$ is the specific heat of the dry air, measured in J/KgC. The nonnegative variables $\Delta T_h$ and $\Delta T_c$ represent the temperature difference through the heating and cooling coils respectively. Hence the inputs $u_h$ and $u_c$ multiplied by $c_{pa}$ are the heating and cooling power respectively. Hence, the room temperature dynamics can be described by a discretized Linear Time Invariant (LTI) system as

$$x(k + 1) = Ax(k) + Bu(k) + Eu(k)$$

$$y(k) = Cx(k)$$ (2.2)

where

- $x$ represents the Model state;
- $A$ represents the Dynamics Room matrix;
- $x$ represents the Initial conditions;
- $B$ represents the Dynamics Actuator matrix;
- $u$ represents the Input values;
- $E$ represents the Dynamics Disturbance matrix;
- $u$ represents the Disturbance values;
- $C$ represents the Dynamics Output matrix;
- $y$ represents the Output values;

The initial conditions $x(k)$ are an array with the inner and outer measured temperature values of all the walls:

- Room temperature
- Wall 1 temperature, output (outdoors)
- Wall 1 temperature, input
- Wall 2 temperature, output
- Wall 2 temperature, input
- Wall 3 temperature, output
- Wall 3 temperature, input
- Wall 4 temperature, output
• Wall 4 temperature, input
• Wall 5 temperature, output
• Wall 5 temperature, input
• Wall 6 temperature, output
• Wall 6 temperature, input

The actuator vector \( u(k) = (u_h(k), u_c(k), \Delta T_{rad}(k)) \) contains the optimal values for the thermal power to be applied to the room, while the disturbance vector \( w(k) = (T_{amb}(k), I(k), N_{people}(k)) \) contains the random disturbances at time \( k \). The output \( y(k) \) is the room temperature at time \( k \) while \( A, B, E \) are appropriate matrices.

So, for a prediction horizon \( N \) it holds that
\[
x(1) = Ax(0) + Bu(0) + E(0) \\
x(2) = Ax(1) + Bu(1) + E(1) \\
x(3) = Ax(2) + Bu(2) + E(2) \\
\vdots \\
x(N) = Ax(N - 1) + Bu(N - 1) + E(N - 1)
\]

(2.3)

Since only the initial state \( x(0) \) is available, only the state in \( k = 1 \) can be computed directly. The other ones are thus computed using a batch approach by expressing the state in function of the known state \( x(0) \). In general, each state can be computed this way with the expression
\[
x(k) = A^k x(0) + \sum_{i=0}^{k-1} (A^{k-1-i} Bu(i)) + \sum_{i=0}^{k-1} (A^{k-1-i} Ew(i))
\]

(2.4)

It is useful to express the previously computed states over the prediction horizon in a matrix form:
\[
X = S_x x(0) + S_u U + S_w W
\]

(2.5)

where
\[
S_x = \begin{bmatrix}
A \\
A^2 \\
\vdots \\
A^N
\end{bmatrix}
\]

(2.6)

\[
S_u = \begin{bmatrix}
B & 0 & 0 & \ldots & 0 \\
AB & B & 0 & \ldots & 0 \\
A^2B & AB & B & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
A^NB & A^{N-1}B & A^{N-2}B & \ldots & A^{N-N}B
\end{bmatrix}
\]

It is useful to express the previously computed states over the prediction horizon in a matrix form:
\[
X = S_x x(0) + S_u U + S_w W
\]

(2.5)

where
\[
S_x = \begin{bmatrix}
A \\
A^2 \\
\vdots \\
A^N
\end{bmatrix}
\]

(2.6)

\[
S_u = \begin{bmatrix}
B & 0 & 0 & \ldots & 0 \\
AB & B & 0 & \ldots & 0 \\
A^2B & AB & B & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
A^NB & A^{N-1}B & A^{N-2}B & \ldots & A^{N-N}B
\end{bmatrix}
\]

In conclusion, the output of the thermal model could be expressed mathematically as in 2.7.
\[
U = C_{pa} \cdot u_h + C_{pa} \cdot u_c + \Delta T_{rad} \cdot A_{rad} \cdot h_{rad}[W]
\]

(2.7)

where \( U \) is the total power, \( C_{pa} \cdot u_h \) is the heating power, \( C_{pa} \cdot u_c \) is the cooling power and \( \Delta T_{rad} \cdot A_{rad} \cdot h_{rad} \) is the thermal radiator power.
**Room CO₂ model:** the dynamics of the CO₂ concentration in the room can be modeled with

\[ V \frac{dC_{CO_2}}{dt} = (\dot{m}_{vent} C_{CO_2,i} + g_{CO_2} N_{people}) - \dot{m}_{vent} C_{CO_2} \]  

(2.8)

In order to linearize the CO₂ concentration dynamics, the nonlinear term \( \dot{m}_{vent} (C_{CO_2} - C_{CO_2,i}) \) is replaced with \( u_{CO_2} \), where \( C_{CO_2,i} \) is a constant and \( (C_{CO_2} - C_{CO_2,i}) \) is a nonnegative variable. However, to meet physical bounds on the control input in the original nonlinear model, the following constraint on the input \( u_{CO_2} \) in the linear formulation must be satisfied at each time step \( k \):

\[ \dot{m}_{vent}^{\min} (C_{CO_2} - C_{CO_2,i}) \leq u_{CO_2} \leq \dot{m}_{vent}^{\max} (C_{CO_2} - C_{CO_2,i}) \]  

(2.9)

Hence, the CO₂ concentration dynamics can be described by the discrete Linear Time Invariant (LTI)

\[ x_{CO_2}(k+1) = ax_{CO_2}(k) + bu_{CO_2}(k) + ew_{CO_2}(k) \]

\[ y_{CO_2}(k) = x_{CO_2}(k) \]  

(2.10)

where \( a \) represents an attenuation factor, \( x_{CO_2} \) is the model state, \( b \) captures the dynamics of the actuator, \( u_{CO_2} \) represents the value of the actuator, and \( ew_{CO_2} \) represents the disturbance.

As in the temperature model, a batch approach is used to compute the matrix of the computed states over the prediction horizon. The only difference is here that for each step the CO₂ model considers one state, one actuator and one disturbance (the room occupancy level).

Summarizing, the CO₂ dynamics could be expressed mathematically as

\[ U_{CO_2} = \dot{m}_{vent} \cdot (C_{CO_2} - C_{CO_2,i}) \ [ppm]. \]  

(2.11)
Chapter 3

Tool description

The main motivation behind this software tool is to allow a user to remotely control one or more variables with one or more control algorithms in different HVAC systems and to easily switch between controllers. To fulfill this idea, the software has been designed with the following characteristics:

- flexibility: the user can design experiments using any number of controllers and combine them as desired. From the user point of view, each controller is an instance of the desired controller class which implements the basic features of MPC, PID or LQR control algorithms. The tool allows the user to treat each controller as an independent object, so that different control schemes can be easily compared;

- scalability: the class implements only a proper subset of the vast set of potential control algorithms. The code must thus allow future development, easy implementation and integration of novel controllers. Importantly, the code must be forward compatible. This has been achieved by endowing the user with the ability to create new controllers by extending old classes. More precisely, by creating a new class that inherits from the Controller class (so that common and basic functions and properties will be available) and that implements some basic functions defined in an opportune interface.

We remark that, to fulfill these requirements, the programming language has to be selected carefully. Our choice was to use Matlab® since:

- it allows Object Oriented Programming, essential to implement abstractions of the controllers;

- it allows users to connect remotely to the testbed and to have a platform-independent code that can be executed in virtually every computer. Matlab is in fact an interpreted programming language that is supported by many computer architectures;

- in the automatic control research community, Matlab is de facto a standard. It is in fact widely used and documented. This makes easier to share, maintain and improve the code. Additionally it includes some predefined control functions and it is aligned with our aim of providing fast prototyping instruments.

3.1 Class diagram

Figure 3.1 shows the Universal Modeling Language (UML) structure of the implemented library. In what follows we provide a description of each one of the classes presented in the above diagram. Note that the user only has access to the methods defined or implemented in the classes LQController, MPCControllerLinear, MPCControllerQuadratic and PIDController. All the properties defined in these classes are accessible via opportune getter/setter functions.
Figure 3.1: UML diagram with overall class structure of the library implemented
3.2 Controller classes

**Controller:** root class for all the controllers. It defines some general features common to any of the controllers such as the error handling or the conversion from the computed output values to the final actuation percentages. The user has no direct access to that class. The description of the methods included in this class is in Table 3.1.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller( )</td>
<td>tController</td>
<td>void</td>
<td>Constructor for the general controller</td>
</tr>
<tr>
<td>GetErrorNumber( )</td>
<td>iNumber</td>
<td>void</td>
<td>Gets the number of errors in the error report</td>
</tr>
<tr>
<td>GetErrorReport( )</td>
<td>-</td>
<td>cell</td>
<td>Gets the error report with all the errors caught</td>
</tr>
<tr>
<td>AddError( )</td>
<td>string</td>
<td>void</td>
<td>Adds an error to the error report</td>
</tr>
<tr>
<td>SupplyAirTemperature</td>
<td>fRequiredSup</td>
<td>float</td>
<td>Computes the cooling valve opening percentage from the processed supply</td>
</tr>
<tr>
<td>ToCoolingPercentage( )</td>
<td>plyAirTemper</td>
<td></td>
<td>air temperature value, the fresh inlet temperature and the opening</td>
</tr>
<tr>
<td></td>
<td>ture, fAirIn</td>
<td></td>
<td>venting valve</td>
</tr>
<tr>
<td></td>
<td>letTemperature,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fVentingPer-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>centage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MassFlowToVentingPercentage( )</td>
<td>fMassFlow,</td>
<td>float</td>
<td>Computes the venting valve opening percentage from the processed air</td>
</tr>
<tr>
<td></td>
<td>strKindOfDuct</td>
<td></td>
<td>mass flow of the duct indicated in the strKindOfDuct parameter</td>
</tr>
</tbody>
</table>

Table 3.1: Methods included in the Controller class
**MPController**: Root class for the MPC controllers. It defines some general features common to any MPC controller such as the mathematical model. The description of the methods included in this class is in Table 3.2.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPController( )</td>
<td>-</td>
<td>MPController</td>
<td>Base class for the MPC controllers. Child of Controller, defines properties and methods common for Linear and Quadratic MPCs</td>
</tr>
<tr>
<td>ModelInit( )</td>
<td>-</td>
<td>void</td>
<td>Initializes all the matrices used in the system state model and the cell arrays dedicated to save the generated data</td>
</tr>
<tr>
<td>ModelDiscretize( )</td>
<td>-</td>
<td>void</td>
<td>Discretize the system state model with the selected sampling time.</td>
</tr>
<tr>
<td>ModelEstimator( )</td>
<td>-</td>
<td>void</td>
<td>Estimator for the system state model</td>
</tr>
<tr>
<td>ModelInterpolateForecasts( )</td>
<td>aW, aTimes, aaTemp, aaRad, fOccupancy</td>
<td>void</td>
<td>Adapts the hourly forecasts to the sampling time by interpolation</td>
</tr>
<tr>
<td>ModelGenerateConstantsBatch( )</td>
<td>-</td>
<td>void</td>
<td>Fills the matrices created in the ModelInit() function using the batch approach explained in section(reference)</td>
</tr>
<tr>
<td>ConstraintsSetInequality( )</td>
<td>-</td>
<td>void</td>
<td>Creates the constraint matrices with the inequality formulation explained in section(reference)</td>
</tr>
<tr>
<td>ConstraintsSetEquality( )</td>
<td>-</td>
<td>void</td>
<td>Creates the constraint matrices with the equality formulation explained in section(reference)</td>
</tr>
<tr>
<td>SolverSetCostFunction( )</td>
<td>-</td>
<td>void</td>
<td>Abstract method. Creates the cost function for the optimization problem</td>
</tr>
<tr>
<td>SolverSetSettings( )</td>
<td>-</td>
<td>void</td>
<td>Abstract method. Creates the settings for the solver</td>
</tr>
<tr>
<td>SolverSolveProblem( )</td>
<td>-</td>
<td>void</td>
<td>Abstract method. Solves the optimization problem</td>
</tr>
</tbody>
</table>

Table 3.2: Methods included in the MPController class
3.2 Controller classes

**MPControllerLinear**: Inherits from MPController and creates an instance of a MPC with linear cost function. Implements the functions defined in the ControllerInterface. The description of the methods included in this class is in Table 3.3.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPControllerLinear( )</td>
<td>tControlSettings, objConnection</td>
<td>MPController Linear</td>
<td>Constructor for the linear MPC. The first input parameter is a struct with the MPC operation settings and the second one an instance of the KHTConnection class.</td>
</tr>
</tbody>
</table>

Table 3.3: Methods included in the MPControllerLinear class

**MPControllerQuadratic**: Inherits from MPController and creates an instance of a MPC with quadratic cost function. Implements the functions defined in the ControllerInterface. The description of the methods included in this class is in Table 3.4.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPControllerQuadratic( )</td>
<td>tControlSettings, objConnection</td>
<td>MPController Quadratic</td>
<td>MPControllerQuadratic Constructor for the quadratic MPC. The first input parameter is a struct with the MPC operation settings and the second one an instance of the KHTConnection class.</td>
</tr>
</tbody>
</table>

Table 3.4: Methods included in the MPControllerQuadratic class
**LQController:** Creates an instance of the Linear Quadratic Regulator. Implements the functions defined in the ControllerInterface. The description of the methods included in this class is in Table 3.5.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQController()</td>
<td>tControlSettings,</td>
<td>LQController</td>
<td>Constructor for the LQR. The first input parameter is a struct with the LQR operation settings and the second one an instance of the KHTConnection class.</td>
</tr>
<tr>
<td></td>
<td>objConnection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Methods included in the LQController class

**PIDController:** Creates an instance of the Proportional Integrative Derivative controller. Implements the functions defined in the ControllerInterface. The description of the methods included in this class is in Table 3.6.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIDController()</td>
<td>tControlSettings,</td>
<td>PIDController</td>
<td>Constructor for the quadratic PID. The first input parameter is a struct with the PID operation settings and the second one an instance of the KHTConnection class.</td>
</tr>
<tr>
<td></td>
<td>objConnection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6: Methods included in the PIDController class
Controller Interface: User interface which defines the accessible methods to the user. It is intended that the user executes this methods in order in each iteration. Each controller implement this functions in a different way calling the methods owned by each controller and explained the the beginning of this section. The description of the methods included in this class is in Table 3.7.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcquireForecasts( )</td>
<td>void</td>
<td>void</td>
<td>Acquires the forecasts over the prediction horizon. Only used in MP controllers, otherwise returns an error.</td>
</tr>
<tr>
<td>AcquireMeasurements()</td>
<td>void</td>
<td>void</td>
<td>Gets the measurements needed to start the control loop from the sensors and signal identifications provided in the configuration file.</td>
</tr>
<tr>
<td>ComputeControlInputs()</td>
<td>void</td>
<td>void</td>
<td>Computes the control inputs for the plant following a given control scheme and generates an output according to it.</td>
</tr>
<tr>
<td>ComputeAction( )</td>
<td>void</td>
<td>void</td>
<td>Computes the control inputs for the plant following a given control scheme and generates an output according to it.</td>
</tr>
</tbody>
</table>

Table 3.7: Methods included in the ControllerInterface class

The methods in the table Table 3.7 are defined as abstract and so they can be implemented in different ways in each one of the controllers. For each controller the detailed description of these methods is the following:

- **AcquireForecasts:**
  - MPC: For the temperature model this method acquires the temperature, radiation and occupancy forecasts over the prediction horizon. For the CO2 model, it acquires occupancy forecast over the prediction horizon and generates the vector \( w(i) \) with as many entries as actuators.
  - PID: Returns an error.
  - LQ: Returns an error.

- **AcquireMeasurements:**
  - MPC: This method fills the initial conditions vector \( (x_o) \) with the temperature measurements from the plant. If the identification is set as 'NoValue', that measurement is omitted and estimated after the measuring loop ends.
  - PID: Gets the measurement of the controlled variable from the plant.
  - LQ: Gets the measurement of the system states from the plant.

- **ComputeControlInputs:**
  - MPC: Fills the dynamic matrix, the input matrix and the disturbance matrix over the prediction horizon and the matrices with the configured constraints depending on the controlled variable. Also creates the cost function either linear or quadratic and solve the optimization problem with the solver selected in the configuration file. From the vector returned by the solver, only the set of solutions corresponding to the first time step (prediction horizon = 1) is selected for actuating the plant.
- PID: Generates the error comparing the set point and the measured value and computes the action to be taken in order to correct the error.
- LQ: Computes the pole placement in the system model and computes the gain matrix $K$.

**ComputeAction:**
- For all the controllers this function computes the actuation value (percentage of valve opening) from the solution obtained by the solver and sends this value to the actuators.
3.3 Communication classes

**KTHConnection**: Creates an instance of an object able to handle the requests to and the data received from the KTH testbed server. This class is the only which knows the internal functionality of the server and the object created is one of the parameters to the constructor call in each controller. This way it is assured that all the controllers are using the same connection to the server. It has no direct access to the TCP/IP socket and so it inherits from the TCPConnection class to access the network. The description of the methods included in this class is in Table 3.8.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KTHConnection()</td>
<td>achUser, achPassword</td>
<td>KTHConnection</td>
<td>Constructor for the KTHConnection class</td>
</tr>
<tr>
<td>ConnectToHVAC()</td>
<td>void</td>
<td>void</td>
<td>Connects to the KTH HVAC systems with the credentials set in the constructor. It also creates the object which manage the TCP socket</td>
</tr>
<tr>
<td>GetDataMote()</td>
<td>achDeviceID, achVariable, achSignalID</td>
<td>void</td>
<td>Get data from the mote indicated by achDeviceID and achSignalID related to the sensed variable achVariable. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>SetDataPLC()</td>
<td>achSignalID, achLocation, fAction</td>
<td>void</td>
<td>Sends the computed data to the actuators in the KTH test bed. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>GetDataPLC()</td>
<td>achDeviceID, achVariable, achSignalID</td>
<td>void</td>
<td>Get data from the PLC indicated by achDeviceID and achSignalID related to the sensed variable achVariable. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>GetWeatherForecast()</td>
<td>iFrom, iTo</td>
<td>void</td>
<td>Gets the weather forecasts in hours from the hour indicated in iFrom to the hour indicated in iTo. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>GetWeatherNow()</td>
<td>void</td>
<td>void</td>
<td>Gets the weather forecasts of the current hour. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>GetDataPCounter()</td>
<td>void</td>
<td>void</td>
<td>Gets the current room occupancy. Saves the results in tMessage.afData</td>
</tr>
<tr>
<td>ComputeHash()</td>
<td>iLen, ahcHexMessage</td>
<td>string</td>
<td>Computes the hash of 512 bytes of the password to access the HVAC system</td>
</tr>
</tbody>
</table>

Table 3.8: Methods included in the KTHConnection class
**TCPConnection:** Root communication class which handles the TCP bi-directional connection to the server. It is the only one with access to the TCP/IP socket. The user has no direct access to that class. The description of the methods included in this class is in Table 3.9.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPConnection( )</td>
<td>varargin</td>
<td>void</td>
<td>Constructor for the TCPConnection class. The variable input arguments are the destination IP address in string format, the destination port number in integer format, the destination buffer size in integer format and the fixed message length in integer format</td>
</tr>
<tr>
<td>SendData( )</td>
<td>void</td>
<td>void</td>
<td>Sends through the TCP socket tMessage.iSize bytes of data in tMessage.afData</td>
</tr>
<tr>
<td>ReceiveData( )</td>
<td>void</td>
<td>void</td>
<td>Receives through the TCP socket tMessage.iSize bytes and saves it in tMessage.afData</td>
</tr>
<tr>
<td>CloseConnection( )</td>
<td>void</td>
<td>void</td>
<td>Closes the TCP socket</td>
</tr>
</tbody>
</table>

Table 3.9: Methods included in the TCPConnection class
### 3.4 General classes

**Experiment:** Common public class with properties and method related to the experiment running such as the experiment time or the methods to save the generated data, to plot the generated data or to control the iterations done as well as the post-processing methods when needed. The description of the methods included in this class is in Table 3.10.

<table>
<thead>
<tr>
<th>method</th>
<th>inputs</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment( )</td>
<td>tExperiment</td>
<td>Experiment</td>
<td>Constructor for the experiment class. The input parameters in the struct tExperiment are the experiment time and the actuating time in hours</td>
</tr>
<tr>
<td>SetFutureJulianDate()</td>
<td>fTimeInHours, chTimeOfWhat</td>
<td>void</td>
<td>Converts the either the actuating time (chTimeOfWhat='A') or the experiment time (chTimeOfWhat='E') from Gregorian to Julian date and sets the corresponding global variable</td>
</tr>
<tr>
<td>CheckCurrentAndFutureJulianDates( )</td>
<td>chTimeOfWhat</td>
<td>boolean</td>
<td>Check if either the actuating time (chTimeOfWhat = 'A') or the experiment time (chTimeOfWhat = 'E') has expired</td>
</tr>
<tr>
<td>Plot( )</td>
<td>varargin</td>
<td>void</td>
<td>Plots the data in the controller objects included in the varargin input</td>
</tr>
<tr>
<td>SaveData( )</td>
<td>varargin</td>
<td>void</td>
<td>Save the data generated by the experiment in Matlab format (.mat)</td>
</tr>
<tr>
<td>WriteLogging( )</td>
<td>varargin</td>
<td>void</td>
<td>Writes into the log file the data for the current iteration included in the varargin input argument</td>
</tr>
<tr>
<td>PostProcessing( )</td>
<td>tCO2Data, tTemperatureData</td>
<td>fPostTsa, fPostTrad, fPostmVentCO2</td>
<td>Computes the post-processing for the experiment stated in section Table 4</td>
</tr>
</tbody>
</table>

**Table 3.10:** Methods included in the Experiment class

**Configuration options**

As has been mentioned before, this library wants to provide general controllers to the user. However, there are multiple different configuration and formulations not only for the controllers but also for the way that the user designs the experiment. The code has been designed to include several options in order to give flexibility to the user and also to allow an easy implementation of some other options and formulations. To allow a better understanding to the user of the available options they has been split in several groups.
### General options
Options related to settings common to function of all controllers.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSamplingTimeInSeconds</td>
<td>int</td>
<td>Sampling time of the model, in seconds</td>
</tr>
<tr>
<td>achSolver</td>
<td>string</td>
<td>Solver for the optimization problem</td>
</tr>
<tr>
<td>achConstraintsType</td>
<td>string</td>
<td>Set to inequality' for inequality constraints or 'equality' to include also the equality constraints of the states</td>
</tr>
<tr>
<td>iPredictionHorizon</td>
<td>int</td>
<td>Prediction horizon for the MPC controller</td>
</tr>
<tr>
<td>iControlHorizon</td>
<td>int</td>
<td>Controller horizon for the MPC controller</td>
</tr>
</tbody>
</table>

Table 3.11: General options

### Variable options
Options related to the variables bounds in order to set the constraints for the optimization problem.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThermalPowerMax</td>
<td>int</td>
<td>Maximum thermal power allowed in [W]</td>
</tr>
<tr>
<td>ThermalPowerMin</td>
<td>int</td>
<td>Minimum thermal power allowed in [W]</td>
</tr>
<tr>
<td>HeatingPowerMax</td>
<td>int</td>
<td>Maximum heating power allowed in [W]</td>
</tr>
<tr>
<td>HeatingPowerMin</td>
<td>int</td>
<td>Minimum heating power allowed in [W]</td>
</tr>
<tr>
<td>CoolingPowerMax</td>
<td>int</td>
<td>Maximum cooling power allowed in [W]</td>
</tr>
<tr>
<td>CoolingPowerMin</td>
<td>int</td>
<td>Minimum cooling power allowed in [W]</td>
</tr>
<tr>
<td>Cpa</td>
<td>int</td>
<td>Dry air specific heat in [J/(Kg*°C)]</td>
</tr>
<tr>
<td>Arad</td>
<td>int</td>
<td>Surface area of the radiator in [m²]</td>
</tr>
<tr>
<td>Hrad</td>
<td>int</td>
<td>Radiator heat transfer coefficient in [W/m²K]</td>
</tr>
<tr>
<td>CO2Max</td>
<td>int</td>
<td>Maximum value for the CO2 in [ppm]</td>
</tr>
<tr>
<td>CO2Min</td>
<td>int</td>
<td>Minimum value for the CO2 in [ppm]</td>
</tr>
<tr>
<td>InitialCO2</td>
<td>int</td>
<td>Initial value for the CO2 in [ppm]</td>
</tr>
<tr>
<td>MVentMin</td>
<td>int</td>
<td>Minimum value for mass air flow in [kg/s]</td>
</tr>
<tr>
<td>MVentMax</td>
<td>int</td>
<td>Maximum value for mass air flow in [kg/s]</td>
</tr>
</tbody>
</table>

Table 3.12: Variable options

### Comfort options
Options related to the environmental limits.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemperatureDayMax</td>
<td>float</td>
<td>Maximum temperature by day, in [°C]</td>
</tr>
<tr>
<td>TemperatureDayMin</td>
<td>float</td>
<td>Minimum temperature by day, in [°C]</td>
</tr>
<tr>
<td>TemperatureNightMax</td>
<td>float</td>
<td>Maximum temperature by night, in [°C]</td>
</tr>
<tr>
<td>TemperatureNightMin</td>
<td>float</td>
<td>Minimum temperature by night, in [°C]</td>
</tr>
<tr>
<td>NightStartAt</td>
<td>int</td>
<td>Set the threshold hour in which the comfort levels change from day to night</td>
</tr>
<tr>
<td>CO2Max</td>
<td>int</td>
<td>Maximum CO2 level for the model state in [ppm]</td>
</tr>
<tr>
<td>CO2Min</td>
<td>int</td>
<td>Minimum CO2 level for the model state in [ppm]</td>
</tr>
</tbody>
</table>

Table 3.13: Comfort options
3.4 General classes

Controller options
Options related to the source of the measurements and the actuators.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ControlledID</td>
<td>string</td>
<td>Identification for the controller. This variable is only used for the log file</td>
</tr>
<tr>
<td>ControlledVariable</td>
<td>Variables</td>
<td>Array of data type Variables defining which physical variable is going to be controlled</td>
</tr>
<tr>
<td>DeviceID</td>
<td>DeviceID</td>
<td>Array of data type DeviceID defining which device is going to be read</td>
</tr>
<tr>
<td>SensorSignalID</td>
<td>SensorSignalID</td>
<td>Array of data type SensorSignalID defining which signal is going to be read</td>
</tr>
<tr>
<td>ActuatorSignalID</td>
<td>ActuatorSignalID</td>
<td>Array of data type ActuatorSignalID defining which actuator is going to be actuated</td>
</tr>
</tbody>
</table>

Table 3.14: Controller options

Note that the options for each controller described in Table 3.14 are generic for all the controllers and involve parameters related with the general behavior of the controller. However, each controller has its own well known tuning parameters (i.e. constants P, I, D in the PID controller which have not been described here but can be set in the main script since all them has been defined in the main class of each controller as public attributes.

Connection options
Options related to the network connection with the test bed.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>achUser</td>
<td>string</td>
<td>User name for connecting to the HVAC system</td>
</tr>
<tr>
<td>achPassword</td>
<td>string</td>
<td>Password for the connecting to the HVAC system</td>
</tr>
</tbody>
</table>

Table 3.15: Connection options

Experiment options
Options related to the experiment like the time limit for the experiment, the actions and the sleep mode.

<table>
<thead>
<tr>
<th>option</th>
<th>input</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fExperimentTimeInHours</td>
<td>float</td>
<td>Experiment time for the current experiment, in hours</td>
</tr>
<tr>
<td>fActuatingTimeInHours</td>
<td>float</td>
<td>Actuating time for the current experiment, in hours</td>
</tr>
<tr>
<td>achFolder</td>
<td>string</td>
<td>Base folder for saving the log and the generated data</td>
</tr>
<tr>
<td>fSleepTimeInSeconds</td>
<td>float</td>
<td>Sleeping time between actions, in hours</td>
</tr>
</tbody>
</table>

Table 3.16: Experiment options
3.5 Controllers

3.5.1 MPC

Model Predictive Control is based in solving a constrained finite time optimal control problem at each iteration as it is shown in the diagram below:

![MPC Diagram](image)

For each iteration, the input for the MPC are the measurements of the model states from the plant. In general these inputs can be estimated, in case that the user has no enough sensing units to perform direct observations, or in case these sources of information are temporarily not reliable enough. Model predictive controllers rely on dynamic models of the plant. This gives the possibility of predicting, also taking into account forecasts of the controlled variables, the next states of the system given the current time instant $k$.

The capability of predicting the future status of a system gives, once one knows possible constraints in the inputs, in the outputs and in the status of the system, the possibility of forming an optimization problem that, once solved, returns the inputs (a control strategy) that minimize the given cost while (if possible) satisfying the posed constraints. In MPC paradigms, every time the optimization problem is solved only the first step of the computed control strategy is actually used. Indeed after applying the first sample of the control signal, the plant state is sampled again and the calculations are repeated, starting now from the novel current state. This yields a new control action and thus a novel predicted state path.

We now explain this strategy in deeper details. At each sampling time, starting from the current state, an open-loop optimal control problem is solved over a finite horizon (see the top diagram in Figure 3.3). The computed optimal input signal is then applied to the process only during the following sampling interval $[t, t+1]$. At the next time step $t+1$ a new optimal control problem based on new measurements of the state is solved over a shifted horizon (bottom diagram in Figure 3.3). The prediction horizon is being shifted forward in each iteration and for this reason MPC is also called *receding horizon control*. The main advantage of MPC is the fact that it allows the current behavior of the system to be optimized, while keeping future temporal behavior also into account.

If we consider the discrete-time linear time-invariant system defined in equation 2.2, MPC strategies approach such a constrained regulation problem assuming that a full measurement or estimate of the state $x(t)$ is available at the current time $t$. Then, the general finite time optimal control problem is stated in equation 3.1:

$$
J^*_0(x(t)) = \min U_0(x_N) + \sum_{k=0}^{N-1} q(x_k, u_k)
$$

subject to:

$$
x_{k+1} = Ax_k + Bu_k + Eu_k, k = 0, \ldots, N - 1
$$

$$
x_0 = x(t)
$$

(3.1)
where $U_0 = u_0, \ldots, u_{N-1}$ and the first element $u_0$ is applied to the system in each iteration.

The MPC algorithm implemented in the implemented software includes the following options and combinations:

**Linear cost functions**, that minimize an expression of the form

$$J^* = \sum_{k=0}^{NNu} u_k.$$  \hspace{1cm} (3.2)

Here $N$ is the prediction horizon and $Nu$ is the number of actuators in the plant model.

**Quadratic cost functions**, that minimize an expression of the form

$$J^* = J^*_{linear} + J^*_{quadratic}$$  \hspace{1cm} (3.3)

$$J^*_{linear} = \sum_{k=0}^{NNu} u_k$$  \hspace{1cm} (3.4)

$$J^*_{quadratic} = \begin{bmatrix}
  u_1u_1 & u_1u_2 & \ldots & u_1u_{NNu} \\
  u_2u_1 & u_2u_2 & \ldots & u_2u_{NNu} \\
  \vdots & \vdots & \ddots & \vdots \\
  u_{NNu}u_1 & u_{NNu}u_2 & \ldots & u_{NNu}u_{NNu}
\end{bmatrix}$$  \hspace{1cm} (3.5)

Where $N$ is the prediction horizon and $Nu$ is the number of actuators in the plant model.

**Inequality constraints**, bounds for the actuation values ($U$) and for the output ($Y$) and expressed as

- Constraints on the outputs
  $$Y_{min} \leq Y \leq Y_{max}$$  \hspace{1cm} (3.6)

  that can be expressed as
  $$Y_{min} \leq CS_x(0) + CS_xU + CS_yW \leq Y_{max}$$  \hspace{1cm} (3.7)
• constraints on the inputs

\[ u_{\text{min}}^k \leq u_k \leq u_{\text{max}}^k \]

\[ \text{mvent}_i \min [x(k) - \text{CO}_2i] \leq u_k \leq \text{mvent}_i \max [x(k) - \text{CO}_2i] \] (3.8)

Note that the last constraint bounds the air mass flow and only applies for the \text{CO}_2 model.

Inequality and equality constraints, expressed as

\[ x_{k+1} = Ax_k + Bu_k + Ew_k, k = 0, ..., N - 1 \] (3.9)

**Deterministic MPC**, a flag that lets the MPC use only a point forecast of the weather, solar radiation and room occupancy;

**Stochastic MPC**, that lets the MPC exploit not only the point forecasts of the weather, solar radiation and room occupancies, but also some statistics of the forecasts errors.

**Remark 1. We use CPLEX as the selected solver.** IBM ILOG CPLEX Optimization Studio (often informally referred to simply as CPLEX) is an optimization software package from IBM released in 1988. Despite the fact that it was originally designed to implement the simplex method in C, today this software supports also other types of mathematical optimization problems, and it offers interfaces other than just C.

The CPLEX Optimizer solves integer programming problems, large linear programming problems using either primal or dual variants of the simplex method or barrier interior point methods, convex and non-convex quadratic programming problems, and convex quadratically constrained problems (solved via second-order cone programming, or SOCP).

For each one of the constraints presented before, CPLEX needs two expressions, one for the maximum value and another one for the minimum. In addition, when using CPLEX as the solver for MPC optimization problems, it is required to express the inequalities with the operator 'less or equal', so the first expression is positive and the second one is negative since the inequality needs to be turned up. Equation 3.10 is an example of how to express constraints when using CPLEX.

\[ \begin{bmatrix} [u_1 & u_2 & u_3] \\ [-u_1 & -u_2 & -u_3] \end{bmatrix} \leq \begin{bmatrix} Y_{\text{max}} - CS_zx(0) - CS_w \leq \begin{bmatrix} u_1^{\text{max}} \\ u_2^{\text{max}} \\ u_3^{\text{max}} \end{bmatrix} \leq \begin{bmatrix} Y_{\text{min}} + CS_zx(0) + CS_w \leq \begin{bmatrix} u_1^{\text{min}} \\ u_2^{\text{min}} \\ u_3^{\text{min}} \end{bmatrix} \end{bmatrix} \] (3.10)

### 3.5.2 PID

A Proportional, Integrative and Derivative (PID) controller is a generic control loop feedback mechanism based in the minimization of an error value which is the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the plant control input. This input is then the sum of three separate terms: proportional, integrative, derivative.
3.5 Controllers

![PID block diagram with back-calculation anti-windup. When the computed output is above the saturation limit, the difference between the computed output and the saturation level is integrated in the next iteration.](image)

- **Proportional term**
  The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant $P$, called the proportional gain constant. 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 become unstable (see the section on loop tuning). In contrast, a small gain results in a small output response to a large input error, and to a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. The proportional term is given by:
  \[ P_{\text{out}} = P e(t) \]  
  \[ P \] 

- **Integrative term**
  The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain ($I$) and added to the controller output. The integral term accelerates the movement of the process towards the set point and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the set point value. The integral term is given by:
  \[ I_{\text{out}} = I \int_0^t e(\tau) \, d\tau \]  
  \[ I \] 

- **Derivative term**
  The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain $D$. The derivative action predicts system behavior and thus improves settling time and stability of the system. Derivative action, however, has a very high inherent sensitivity to measurement noise. If this noise is severe enough, the derivative action will be erratic and actually will degrade the control performance. Large, sudden changes in the measured error (which typically occur when the set point is changed) cause a sudden, large control action stemming from the derivative term. The derivative term is given by:
  \[ D_{\text{out}} = D \frac{d}{dt} e(t) \]  
  \[ D \] 

Summarizing, the mathematical output of a PID is the sum of the three terms:

\[ y = P + I \int e(k) \cdot dt + D \frac{de(k)}{dt} \]  

\[ y \]
In a situation with a large change in the set point, the integral term can accumulate a significant error during the rise (when the change is positive), leading to an overshooting situation and to a saturation of the system. The error continues to increase as this accumulated error is unbounded (windup).

Figure 3.5: This figure illustrates the consequences of a windup situation. The output of the PID and input of the plant has been cut to fit in the saturation limits.

In order to avoid this situation, an anti-windup method should be implemented. A first approach could consist in limiting the output value of the PID to the saturation limits of the plant but this could lead to an incorrect process control. Instead, the PID implementation in this library uses an anti-windup method consisting in back-calculation of the integrative part as it is shown in Figure 3.4.

When the output saturates, the integral term in the controller is recomputed so that its new value gives an output at the saturation limit. Then the system has an extra feedback path that is generated as the difference between the output of the controller and the saturation limit. The signal is zero when there is no saturation. Thus, it will not have any effect on the normal operation when the actuator does not saturate. When the actuator saturates, the signal is different from zero and is added to the integrative term to be computed in the next iteration.

3.5.3 LQR

The Linear Quadratic Regulator (LQR) uses a mathematical algorithm that minimizes a cost function with weighting factors. The cost function is often defined as a sum of the deviations of key measurements from their desired values $(x - x_d)$ which means that this algorithm finds those controller settings that minimize the undesired deviations.

If we consider the discrete-time linear time-invariant system dynamics defined in equation (2.2), the LQR implemented in this library minimizes the discrete quadratic cost function in equation (3.15)

$$J(u) = \sum_{n=0}^{\infty} (x^T Q x + u^T R u + 2 x^T N u)$$

with state-feedback law $u[n] = -K x[n]$ by solving the associated Riccati equation:

$$A^T S + SA - (SB + N) R^{-1} (B^T S + N^T) + Q = 0$$

and calculating the state-feedback gain $K$ from the Riccati equation:

$$K = R^{-1} (B^T S + N^T)$$
3.6 Implementation on the KTH testbed

This section describes how to transfer and receive data from the server. We start with a quick review of the server architecture, described in some details in Figure (3.7).

![Figure 3.7](image)

**Figure 3.7:** The KTH testbed server manages the testbed control decisions which can come from an external user through the control library or from a simple PI controller located in a PLC next to the server.

The KTH testbed server, called in Figure (3.7) PC-HVAC, is located in the same lab room used as a testbed. The server is in charge of managing the decision control action inputs and also of collecting the measurements received from the sensors. It has two Network Interface Controllers (NIC) which are the input and output gates for the applications running in the server:

- **NIC 1:** gives access to the KTH network and through it to the Internet. Any user which wants to control the testbed from any place inside or outside KTH are establishing connection through this NIC.

- **NIC 2:** gives access to the Akademiska Hus controller located in a PLC in the same lab as the server and, through it, to the testbed itself. This PLC acts as default controller and is in charge of controlling the testbed when no actions from an external user are received. This controller implement a simple PI algorithm based on 'if' conditions and its only duty is to maintain the environmental variables bounded within given minimum and maximum values. It is controlled from a SCADA application running inside the controller and the user can access to it with a web browser via its IP address.

The control of the testbed can be switched from the PLC to an external user. Note that, independent on who is actually controlling the testbed, the control signals will go through this PLC, since it is located between the server and the testbed.

The sensors deployed in the room are motes powered by batteries which are sending data continuously to the server through a base mote connected to a USB port in the server. Additionally, the PLC has its own sensors deployed in the room (used in the default mode to perform the original PI loop). The data collected by both the USB base mote and the PLC and also the data from the weather forecasts is sent to a Labview application running in the server and saved into a database. The user can access to the data saved in the database via a web server application able to generate the corresponding SQL requests by filling a web form.

Figure 2.11 shows how the motes and the sensors are deployed in the testbed room. To have a measurement for each one of the states included in the room model defined in section 2, some motes needed to be placed in each one of the room walls (both inside and outside of the room, including the floor and the ceiling). Nonetheless, as mentioned in 2, if a sensor is missing in some of the walls, the measurement for this state can be estimated.

The communications with the server of all the instantiated controllers are centralized in one communication object of the class 'KTHConnection'. Objects of this class pass to the constructors of each controller a parameter that induce the server and the controller to share a common TCP/IP interface. This object is the only interface with the KTH server. It is
thus the unique one implementing the queries needed to send and fetch data from the server via the web. It inherits from the basic and more common TCP class 'TCP Connection' which is the only one using TCP sockets. The communication flow diagram is shown in Figure 3.8.

![Figure 3.8: Communications flow diagram.](image)

The communication starts and ends in one of the methods defined in the KTHConnection class. This creates the corresponding query and sends it to the server via the methods in the TCPConnection class, which in its turn is completely transparent to the user. The information is sent in a message with fixed size.

**Peculiarities**

- **Security.** This tool grants access to the server only to the desired users. The mechanism nonetheless relies on the KTH firewalls. Our approach consists on assigning statically one port to access the server and actuate the controlled variables, and dedicate instead other doors to read the data sent by the server. One of the main aims of this toolbox is to allow external collaboration for controlling the testbed. One user can operate the actuators but more users can read the measurements.

- **Software: Labview.** All the data from the sensors and to the actuators goes through the server implemented in Labview. At the beginning the server is in an idle state, listening a given TCP port and waiting for an incoming TCP connection. Once a connection is established the client sends user credentials and the server grants or denies the access to the server. If the access is granted, the client is able to send requests to the server in a given format, and in this way it receives the desired data from the sensors and actuates the specified devices.
This section proposes and details how to perform an experiment on the testbed. The aim is thus to help a better understanding for the user through examples.

The experiment proposes to use two linear MPCs. We remark that the aim the designed experiments is to show how to control the environmental variables in the testbed and to be a guide, and not on how to perform this task in an energy efficient way.

**Definition of the experiment:** the experiment exploits two MPCs which iteratively compute the optimal thermal power (MPC 1) and the optimal CO$_2$ value (MPC 2) using the functions defined in 3.1.

Note that the measurements are done inside the MPC blocks. However, the measurement block has been left above the MPC blocks for better understanding. After computing the optimal values, a post-processing function is called in order to obtain the adequate values for the actuators.

Two MPCs are defined as follows:

- **MPC1**
  - Controlled variable: Temperature;
  - Computes the optimal heating power, cooling power and thermal power.

- **MPC2**
  - Controlled variable: CO2;
  - Computes the optimal CO2 level.

The objective of the experiment is to maintain the temperature and CO$_2$ levels in the KTH testbed between the predefined comfort levels while minimizing the power energy used. The first step is then to define opportune bounds for both the actuating value and the comfort levels:

**Actuation bounds:**

\[ 0W \leq HeatingPower \leq 6000W \]
\[ 0W \leq CoolingPower \leq 6000W \]
\[ 0W \leq ThermalPower \leq 8000W \]
\[ 0ppm \leq CO2 \leq 500ppm \] (4.1)

**Comfort bounds:**

\[ 16C \leq RoomTemperature(Nooccupancy) \leq 20C \]
\[ 20C \leq RoomTemperature(Occupancy) \leq 25C \]
\[ 0ppm \leq CO2 \leq 850ppm \] (4.2)
The time of the experiment is one hour an after half an hour the temperature bounds change from 16-20 C to 20-23 C in order to check how the system handles the new situation. In a real implementation this change in the temperature bounds could model the occupancy timetable of the room and decrease the comfort bounds when the room is supposed to be empty in order to achieve less energy consumption.

So, the steps to start the experiment are the following:

1. Fill the Configuration file:
   - Controller parameters:
     - \( \text{Samplingtime} = 10 \text{minutes} \)
     - \( \text{Actuatingtime} = \text{Samplingtime} \)
     - \( \text{Sleepingtime} = \text{Actuatingtime}/2 \)
   - DeviceID: \([1 \times \text{Number of states (13)}]\) array with one of the values of the enumerator Controller.DeviceID in each position.
   - SignalID: \([1 \times \text{Number of states (13)}]\) array with one of the values of the enumerator Controller.SignalID in each position.
   - ActuatorID: \([1 \times \text{Number of actuators (3)}]\) array with one of the values of the enumerator Controller.ActuatorID in each position.

   Remark: it may happen that some of the states correspond to a non trusted sensor, or no sensor at all. In this case they have to be estimated, or that there are not enough actuator values for each actuator. In these cases, the corresponding position in DeviceID, SignalID and ActuatorID can be set with the void value 'NoValue' in order fill the required array length.

2. Create instances from the classes KTHConnection, Experiment and for the needed controllers.

3. MPC1 computes the optimal values for the heating, cooling and thermal power by executing the following functions in order from the Controller Interface:
   - \( \text{MPC1.AcquireForecast()} \)
   - \( \text{MPC1.AcquireMeasurements()} \)
   - \( \text{MPC1.ComputeControlInputs()} \)
4. MPC2 computes the optimal value for the CO₂ by repeating the previous step:
   - MPC2.AcquireForecast()
   - MPC2.AcquireMeasurements()
   - MPC2.ComputeControlInputs()

5. After retrieving the optimal CO₂ level from MPC2, the optimal level of the ventilation mass flow can be computed from the second equation in 4.3:
   \[
   \dot{m}_{\text{vent}} = \frac{U_{CO_2}}{C_{CO_2}} \left( \frac{Kg}{s} \right)
   \] (4.3)

6. Depending on if the room needs to be cooled or heated, the supply air temperature \(T_{sa}\) is then computed from one of these expressions:
   \[
   T_{sa} = \frac{u_h}{\dot{m}_{\text{vent}}} + T_{room} [C]
   \]
   \[
   T_{sa} = \frac{u_c}{\dot{m}_{\text{vent}}} + T_{room} [C]
   \] (4.4)

7. The supply air temperature obtained in the previous step has to be bounded according to the comfort bounds. In case it exceeds either the higher or the lower bound, then \(T_{sa}\) is equal to the bound and the ventilation mass flow is recomputed according to one of these expressions depending on the bounded variable:
   \[
   \dot{m}_{\text{vent}} = \frac{u_h}{T_{sa} - T_{room}} \left[ \frac{Kg}{s} \right]
   \]
   \[
   \dot{m}_{\text{vent}} = \frac{u_c}{T_{sa} - T_{room}} \left[ \frac{Kg}{s} \right]
   \] (4.5)

8. Finally, the desired temperature in the radiator can be computed as
   \[
   T_{rad} = \Delta T_{rad} + T_{room} [C]
   \] (4.6)

9. Once we have the set point values for the air temperature \((T_{sa})\), for the radiator temperature \((T_{rad})\) and for the ventilation mass flow \((\dot{m}_{\text{vent}})\), the array ‘afNewAction’ of each controller is filled as \(MPC1.\text{afNewAction} = [T_{sa}, T_{rad}, 0]\), \(MPC2.\text{afNewAction} = \dot{m}_{\text{vent}}\). Note that the last action of MPC1 is zero because there is none actuator assigned to that position, so the system is not going to read that action value.

10. Finally the computed output values can be send to the actuators by calling the function ‘ComputeAction’ for each controller:
    - MPC1.ComputeAction()
    - MPC2.ComputeAction()

This function transforms Celsius degrees into a cooling duct valve opening percentage. Moreover it transforms a ppm number to a venting duct valve opening percentage. Once the valve percentages are computed, they are sent to the actuators.

11. This process is repeated every Actuation Time minutes until the Experiment time expires. While the program is running, the measured, computed and actuated data is being saved in both array variables and in a plain text log file. Once the program has finished, the arrays are saved in Matlab extension files in the folder specified in the Configuration file and with the experiment date as a file name.
Chapter 5

Experimental results

Several experiments have been performed to test the stability and robustness of the library during long periods of time. However, the results have been conditioned by the fact that in the summer period the heating system is turned off. In any case the expected results are that the optimization values given by the MPC solvers maintain both the comforts levels and the actuated values within the predefined bounds stated in chapter 4.

We now explain the obtained results. For all the figures the abscissas represent time, and more precisely the steps of the actuation samples (which in this case is equivalent to the sampling time). In this particular case we plot an experiment lasted for 1 hour (meaning that, since the actuation time is 10 minutes, there are 6 time steps).

Figure 5.1 depicts the variations of the measured temperature and CO$_2$ and the post-processed actions or set points applied in each case. The top plot shows how the Supply Air Temperature ($T_{sa}$, green line) calculated from equation (4.4) after the post-processing changes after half an hour from the comfort bounds 20 $\rightarrow$ 23 $^\circ$C to the comfort bounds 16 $\rightarrow$ 20 $^\circ$C. As shown in the figure, in both cases the value for $T_{sa}$ is bounded to the maximum temperature. When $T_{sa}$ changes, the system actuates the AC system and the room temperature starts changing, tending to meet $T_{sa}$.

The bottom plot of Figure 5.1 instead shows that the controllers compute that no action should be performed to correct the CO$_2$ concentration. This is due to the fact that, during the experiment time, the values of CO$_2$ remain constant and within the comfort bounds. Indeed during this test the room was completely empty, letting the CO$_2$ levels remain constant.

Figure 5.2 then plots the decisions taken by the optimization problem in equation (3.2), which can be either heating ($u_h$, blue line) or cooling ($u_c$, green line). While the temperature is within the comfort bounds the solution of the optimization problem tries to heat the system. As mentioned before, nothing happens: the radiators were in fact not working at the time the experiments were made. However, when the comfort bounds changes to the lower levels the temperature measurement becomes higher than the novel limits (20 $^\circ$C). As expected, the system then starts to actuate the AC subsystem, and the temperature of the room starts diminishing.
Figure 5.1: Room temperature measurements and supply air temperature or temperature set point.

Figure 5.2: Actions taken from the optimization problem. After half an hour, the bounds for the temperature change and the system stops to heat the room and starts to cool it.
Chapter 6

Conclusions

The main aim of this thesis was to design and implement a software library in Matlab to help the fast prototyping of remote controllers, able to fit in any situation which involves automatic control, not only for HVAC systems but also for other environments. However, while developing the library, we realized that this task would be too generic and beyond the possibilities for a MS thesis project. There are in fact too many variables, mathematical formulations and options to perform a good generalization that is perfectly suitable for every possible controller.

As a consequence, during the development many parameters and methods tailored for HVAC systems had to be added to the main library structure. This eventually made the offered library a tool that is mainly fitted for the remote control of the KTH HVAC testbed. Thus, even if the produced software achieves the main objective of providing a tool enabling users from different institutions to test control strategies under the same environment and share the results in an easy way, the contributions are not as generic as desired at the beginning of the work.

6.1 Issues and further developments

Being the testbed still in its development, the offered library has not been tested exhaustively. Thus a future development direction is to test the software under different conditions (specially during winter time) and perform the necessary adjustments.

From software engineering points of view, we highlight some additional improvements that would make this tool more efficient, secure, useful and easier to use:

- **improve security**: since the communication between the client and the testbed is done via the Internet, the server must have at least one port open to allow external connections and this could be a threat for the security of the system. In order to avoid that malicious users can interact with the server some improvements in the security should be taken into account. For now, the security measures adopted are the HASH encryption of the password and the firewall rules set in the KTH routers.

- **adaptability to other testbeds**: the software has been designed so that other different institutions can make other researchers access to their testbeds. This can be done creating a class that is equivalent to 'KTHConnection', such as 'InstitutionConnection'. This should implement the specific requests needed to fetch information from the institution testbed. Since this class is the unique one knowing what are the expected requests and replies to and from the testbed server, none other changes should be required to add option to connect to any other testbed. We notice that, to let users from other institutions change the names used to identify each sensor and actuator, this kind of settings are collected in an externalized configuration file.
• **graphical User Interface:** to reach as many researchers as possible, it may be useful to create a user-friendly interface that allows users without programming skills to implement controllers. However, we notice that this would be a difficult task: the code has in fact been designed in a way that users need only to modify the main script and the configuration files to use the library.

• **supporting different estimators:** the testbed exploits raw information coming from the various sensors deployed in the various rooms. Up to now no signal processing techniques are applied, and this is from statistical perspectives for sure not the optimal choice. An extension is thus to implement libraries that allow the fast prototyping of signal processing techniques, similarly to what has been done here.

• **support more numerical optimization solvers:** the solver used in this library is CPLEX. Despite being certainly a very fast and reliable option, it is however a proprietary software from IBM, and users may do not want to rely on proprietary software for their experiments. This case can then be addressed by adapting the methods 'SolverSetSettings()', 'SolverSetCostFunction()', 'SolverSolveProblem()'.


