UNIVERSITY IN BELGRADE
SCHOOL OF ELECTRICAL ENGINEERING

MINIMIZATION OF ELECTRICITY CONSUMPTION IN A CENTRALIZED VENTILATION SYSTEM

Master's thesis

Mentor:

Dr Dragan Olćan, professor

Candidate:

Nikolina Bunijevac
2022/3199

Belgrade, September 2024.

ACKNOWLEDGEMENT

This master's thesis is the result of my work experience at the European Organization for Nuclear Research (CERN) up to this date. With the support of my colleagues in Cooling and Ventilation group at CERN, it was possible to gather necessary documentation, learn details about plant operation and, ultimately, contribute to the initial steps of optimizing electricity consumption for the ventilation system that operates at CERN. I would especially like to express my gratitude to my supervisor Diogo Monteiro for initial training, guidelines and support throughout the project.

CONTENT

1. INTRODUCTION

HVAC (heating, ventilation and air conditioning) systems are systems that maintain required air conditions in indoor spaces, in terms of temperature, humidity and air quality. They are an integral part of many residential and industrial buildings, some of which require very specific conditions for operation. Since the operation of these systems involves movements and heating/cooling of large volumes of air, the energy consumption is high - these systems contribute to around 38% of residential building consumption [9]. Besides residential buildings, electricity consumption of HVAC systems is a concern in industrial sites as well. One example is cooling and ventilation systems that maintain air quality for accelerator machinery and experimental equipment at the European Organization for Nuclear Research (CERN). During the experiments run, the electricity consumption dedicated to cooling and ventilation makes 12% (75GWh) [13] of total electricity consumed by the Large Hadron Collider (LHC), CERN's flagship accelerator. High consumption raises financial and environmental concerns. Therefore, many efforts have been put into optimization of electricity consumption of HVAC devices in recent decades. Different approaches have been taken, from increasing insulation quality to more complex methods, like optimization of system controls. Most HVAC systems are controlled with standard PID (proportional–integral–derivative) controllers. They are stable and reliable, but rather simple and not optimal in terms of energy efficiency. The control signals are calculated based on system deviation from defined required references. They do not consider future conditions, nor do they make predictions, while this additional information could be beneficial in optimizing controls for future settings and achieving energy savings. Unlike them, one of the advanced control methods that incorporates future predictions into control strategy is model predictive control (MPC). MPC consists of two crucial parts: a digital model of system and optimization algorithm. A digital representation of system is developed and used to make predictions in future, e.g., in HVAC case, based on weather forecasts and given set of controls. Using this information, the model is exploited to test different control strategies. Based on predicted results, optimization function is calculated. After sufficient number of iterations, optimization algorithm can choose the optimal set of controls. The optimal set of controls is the one that minimizes electricity consumption during certain defined time period in future, while keeping the system parameters (e.g., air temperature in the monitored area) within required constraints. MPC is expected to operate the plant in the most optimal way possible, but its installation does not come without difficulties. The complexity of this approach lies in model development, whose precision highly influences resulting performance. However, the potential savings are a driving factor for putting effort inro further research in this field.

This thesis will provide an overview of basic model predictive controller implementation for one HVAC plant at the CERN and compare the electricity consumption of digital system run by both MPC and PID regulation strategies. Both are coded in the programming language Python. Additionally, critique and plans for future improvements are provided.

In Chapter 2, the importance of HVAC systems in maintaining desirable indoor air conditions is discussed, along with their substantial electricity consumption, providing motivation for advance controls application both at CERN and outside.

Chapter 3 introduces the model predictive control (MPC) method.

Chapter 4 describes the HVAC system and its modeling. This chapter explains the design and functionality of the existent HVAC system at CERN. Additionally, it gives details of the process of a digital system representation.

In Chapter 5, the optimization problem within MPC framework is formulated. Optimization function is described, as well as optimization space with its variables (types, ranges), and additional constraints. The overview of genetic algorithm is given, a search heuristic inspired by the process of natural selection.

Chapter 6 presents and compares the results obtained from the standard regulation strategy and advanced MPC on ten chosen points taken from real operating conditions, on the time period of two hours. Three cases are shown and discussed in more detail.

Finally, Chapter 7 summarizes the work conducted in this research, outlining the key findings and results. It provides conclusions drawn from the study and suggests potential areas for future research to further enhance the efficiency and effectiveness of predictive control strategies in HVAC systems.

2. PROBLEM OVERVIEW

2.1. Global energy consumption and impact of HVAC systems

Despite the urgent need to address climate change, global energy use and CO2 emissions continue to increase [10]. This trend is driven by population growth, rising wealth, and improving living standards worldwide. Although improvements in efficiency have somewhat reduced these impacts, allowing wealth to grow faster than energy use, energy consumption and emissions have still risen at similar rates. As a result, progress toward reducing carbon emissions has been limited over the past two decades [9].

HVAC (heating, ventilation, and air conditioning) systems are among the largest contributors to global energy consumption and carbon emissions. HVAC systems are the crucial elements to provide comfort in residential and public buildings, as well as for the operation of the many industrial complexes. Air conditioning systems maintain required air conditions in indoor spaces, in terms of temperature, humidity and air quality. The impact of HVAC systems to the environment is only expected to rise in future [12].

The building sector it total take 40% share of total global primary energy use and about 30% of global greenhouse gas emissions [17], while HVAC systems contribute to around 38% of residential building consumption [9]. Therefore, improving HVAC systems efficiency plays a crucial role in energy and emissions reduction.

2.2. Efforts of CERN and Cooling and Ventilation group towards higher energy efficiency

In response to increasing awareness regarding growing environmental consciousness, together with energy rising costs, many governments, companies and organizations have been taking actions to reduce their energy footprint, including Organization for Nuclear Research (CERN). In a report published in 2021, CERN management set a goal "to establish itself as the model for transparent and environmentally responsible research organizations," which includes implementing "actions and technologies aiming at energy saving and reuse" [6].

Among many initiatives, Cooling and Ventilation group (CV), part of engineering department (EN), focuses on improving efficiency of cooling and ventilation systems. Cooling and ventilation systems maintain air quality for accelerator machinery and experiments equipment, but very energy intensive. During the experiments run, the electricity consumption dedicated to cooling and ventilation makes 12% (75GWh) [13] of total electricity consumed by the Large Hadron Collider (LHC), CERN's flagship accelerator.

Various actions have been taken to reduce energy consumption, from thoughtful mechanical design considerations, improved maintenance and operation practices, as well as innovation in automation and control strategies. A key approach in innovation of control strategies in CV is the "controls optimization method," which will be discussed in the following section.

2.2.1. Controls optimization

Controls optimization method concerns operating HVAC plant in a more energy-efficient way by using available actuators in calculated manner, while not compromising on performance. This

technique primarily involves changes on software level, while achieving significant savings, making this method a very cost-effective measure [13]. The illustration of controls optimization method pipeline is shown in Figure 1, where four main stages are depicted: digital twin, controller design, simulation and analysis, and deployment.

Traditionally, controller parameters are defined by experience and physical knowledge of plants and fine-tuned during deployment process. However, controls optimization focuses more on controller design side in virtual environment, where simulations and analysis of different controller designs are possible using created digital twin model. A digital twin is a virtual replica of a plant, typically designed using detailed technical documentation, in this thesis referred as well as digital model. Creating reliable models is challenging since many necessary information might be unavailable or uncertain. Overcoming this issue is possible with the help of historical data with model fitting techniques, if they are at disposal.

When it comes to controller design, the most common approach for plant control strategy is a classical PID (proportion-integrative-derivative) controller. Recently, more advanced techniques have been considered, such as model predictive controller (MPC), in details described in chapter 3. The outcome of controller design stage is a set of several proposed control strategies. The next step Simulation & Analysis concerns testing proposed control strategies with the help of Digital Twin. The result of this phase is the most optimal controller design, that is deployed in the final step, in the real plant.

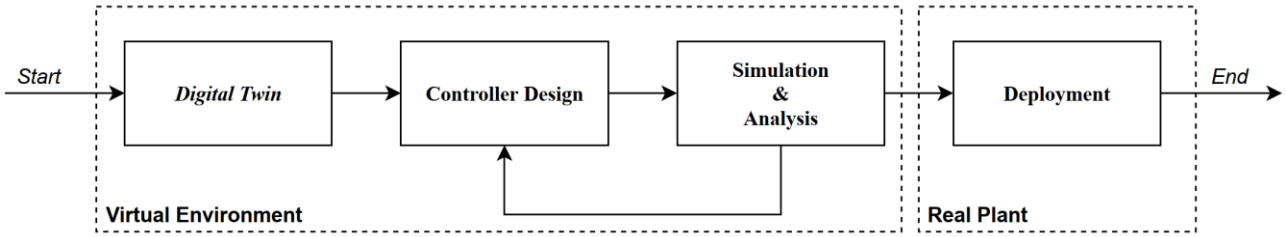

Figure 1: Controls optimization method [13]

The controls optimization technique has been successfully applied to several CV plant projects at CERN, achieving energy savings of up to 75%. More information regarding past, current and future projects can be found in [13]. Regarding HVAC systems, there are several ongoing projects, described in the following paragraphs.

i) Controls optimization for HVAC systems

One of the critical questions in controls optimization method is choice of model. Three major modeling techniques are white-box, grey-box and black-box modeling approaches [12]. White-box methods, also referred to as physical-based or engineering methods, utilize physical principles to calculate thermal and energy behaviors. This method requires a lot of effort in terms of expert knowledge, and usually results in computationally heavy models. Black-box models do not consider physical relations but rely solely on historically measured or generated data to identify underlining system processes using machine learning and statistical methods. The focus on this approach has been increasing with improvements in technology and data-mining techniques, especially because of its simplicity and flexibility. Even though easily scalable on other similar installations, attention should be paid to proper validation of results, since the physical sense of data is lost. Therefore, the biggest effort falls into data collection and pre-processing phases, where physical knowledge is exploited, such as the size of air-conditioned room. These models are not computationally heavy. Grey-box

approaches combine first-principal physics and data-driven strategies. They utilize simplified physics-based models and enable faster calculations.

Regarding controller design, as mentioned in the previous chapter, classical controls such as PID are widely used but often do not guarantee optimal energy performance. Therefore, advanced control techniques have been analyzed, and most commonly, model predictive control (MPC). Instead of relying only on current and past system behavior and deviation from given setpoints like classical controls, this technique uses system model to predict future system states and deviations. Using this information as input for optimization algorithm within MPC framework, it is possible to obtain set of controls that optimizes electricity consumption while making sure that system does not compromise on performance in term of regulated variables tracking. Research has shown that MPC approach in HVAC can reach savings of around 20% [18]. CERN Cooling and Ventilation group recently collaborated with academia on a project where white-box model was developed and deployed in one HVAC within MPC framework [8]. The results on energy savings are still not available.

This thesis aims to explore the benefits of black-box models, motivated by their simplicity and scalability, making MPC application easier in future projects. It presents a checkpoint in current project within CV group, that aims to apply MPC with black-box model on one HVAC plant at CERN site. Details on MPC framework, modeling, optimization problem and obtain results are described in the following chapters.

3. MODEL PREDICTIVE CONTROL

Model-based predictive control (MPC) is an advanced control technique that uses a digital system model to predict its future behavior [16]. By solving an optimization problem, which may include constraints, MPC implicitly determines the control law. This approach shifts the primary focus of controller design towards the accurate modeling of the process. Digital models are usually widely available across various engineering fields, making the MPC implementation easier. The second part, optimization with system constraints, is formulated explicitly, in such a way that the physical understanding of system parameters is preserved, making controller tuning more intuitive. Another important feature of MPC is that it can control systems that are challenging or impossible to manage with conventional feedback controller (for example, due to the complexity of constraints). That is why MPC is currently widely implemented in industrial applications.

In this chapter, a brief overview of model predictive control is given, starting with introduction, where key differences between standard control loop and MPC loop are described. Next, the most common and promising applications are mentioned. Finally, a brief description of controller design is given at the end.

3.1. Introduction

In technical system automation, feedback controllers (or closed-loop controllers) adjust a manipulated variable u by comparing a reference signal r with a measured output y, based on the error $e = r - y$. There are several categories of classical controllers, based on the working principle. The description and comparison of classical and predictive controller categories follows, with PID (proportional–integral–derivative) controller and MPC (model predictive controller) as category representatives, respectfully.

3.1.1. Classical controllers - PID

Classical controllers are types of controllers that focus on responding to past and current system behavior. The best-known classical controller is the PID controller. It is most widely used in industrial applications due to its simplicity and effectiveness [1]. However, finding proper parameterization is not a simple task, even though the several setup rules are known and defined. This specially stands for nonlinear or time-variant systems. Therefore, even though suitable for the most industrial application, PIDs are not an easy implementable solution for a certain number of applications, such as in case of many constraints in the process.

A schematic representation of a classical feedback control loop is shown in Figure 2. Signals on the schematics are reference signal r, error signal e, control (manipulated variable) u, system output y, and external inputs for the system z.

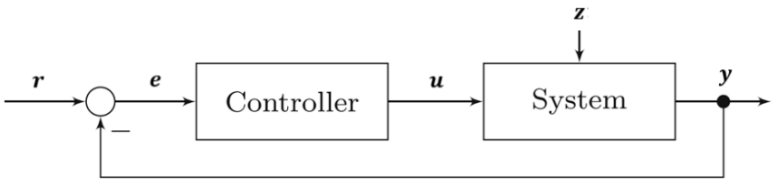

Figure 2: Block diagram of a classical feedback control loop (e.g. PID control) [16]

3.1.2. Predictive controllers – MPC

Predictive controllers use a system model to produce predictions of future behavior. Based on this information, it is possible to anticipate and correct a deviation before it occurs. One example of predictive controller is so called model predictive controller (MPC). Based on the possibility to predict future system states, it produces the optimal set of controls for a certain time in future, known as prediction horizon.

Two main parts of the MPC are the model and the optimizer, as shown in Figure 3. As in every optimization problem, the cost function needs to be defined. In control system applications, it usually includes system error, but it can consider electricity consumption as well, financial cost etc. The cost function is calculated based control inputs, system state and predictions given by the model.

The manipulated variable in this optimization problem is the set of system commands. In each iteration of the optimization algorithm, optimizer produces the current set of commands and sends them to the model. The model produces future outputs, sends them back to optimizer, allowing the algorithm to evaluate the performance of the current commands through cost function and produce a new set of controls for the next iteration. This is repeated until the optimal solution is found, or when a stop criterion is reached (e.g., number of iterations).

The constant interaction between prediction and optimization is what distinguishes MPC from standard control logic, where only precomputed control strategies are applied. The second advantage is the fact that hard systems constraints can easily be implemented in the optimizer and included in the search for the optimal controls, which is not the case for the standard controllers. Standard controllers often implement ad hoc constraint management and therefore define set points sufficiently far from constraints, leading to the suboptimal plant operation. On the other hand, because the constraints are directly and explicitly included in the controller design (later shown in equations (4) and (5)), set point and plant operation with MPC are optimal [11].

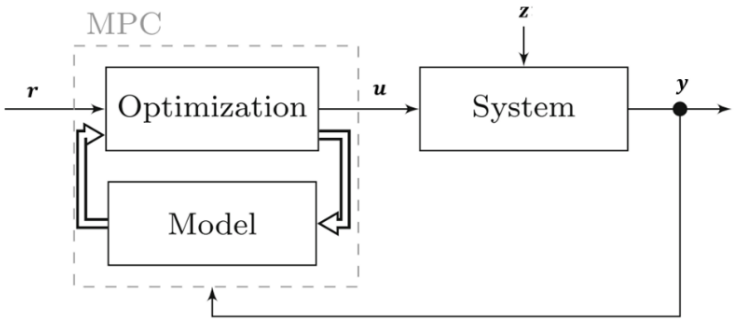

Figure 3: Simplified block diagram of an MPC-based control loop [15]

3.2. Applications

Because of its ability to easily include constraints and intuitive formulation of control law as an optimization problem, MPC has found its application in many fields of engineering. The most prevalent industries are process industry, power electronics, manufacturing, building climate and energy [16]**.** Regarding building climate and energy, the application of MPC ranges from HVAC system control to entire system grids for energy management. Following global concerns regarding energy savings, research in this field has been trending in the last decade.

3.3. Basic MPC problem formulation

MPC solutions are applicable to many different processes, with various constraints and optimization requirements. When it comes to implementation, the biggest effort comes from the proper model creation. These models are usually already developed and available, depending on the industry domain, which makes the controller design process easier. Depending on the system complexity, the model can range from simple step function to more neural network. The choice of optimization algorithm as well follows the nature and complexity of the problem. However, the basic mathematical MPC formulation is general and covers all cases and is presented in equations (1) - (6).

In the basic MPC problem formulation, the model *f* depends on the current state *x[i]* and the system controls *u[i]*, in time *i* (3). With these inputs, the model predicts the state in the next point in time *x[i+1]*. The state constraints are defined $x[i] \in \boldsymbol{X}$ (4), as well as input constraints $u[i] \in \boldsymbol{U}$ (5), and initialization *x[0]=x[k]* (6) (consisting of measurements or estimations). The optimization variable for the time *k* $U[k]=U=\{u[0],\ldots, u[N-1]\}$ (2) is a set of system controls for the prediction horizon, size *N*. The optimization requires the definition of the cost function *J(x[i],u[i])* (1), summarized on prediction horizon. This leads to a basic MPC problem formulation:

$$\min_{U} \sum_{i=0}^{N-1} J(x[i], u[i]) \tag{1}$$

$$s.t. \;\; U = \{u[0], \ldots, u[N-1]\} \tag{2}$$

$$x[i+1] = f(x[i], u[i]) \tag{3}$$

$$x[i] \in \boldsymbol{X} \tag{4}$$

$$u[i] \in \boldsymbol{U} \tag{5}$$

$$x[0] = x[k] \tag{6}$$

3.3.1. MPC for HVAC - implementation

HVAC systems consist of several components where thermodynamic processes occur. These processes are nonlinear – and one possibility to model such processes is with neural networks. The approach in this research is to use physics-informed neural networks, which is detailed described in Chapter 4 System modeling.

The nature and complexity of optimization problem formulation limits the choice of optimization algorithm. Since the optimization function that estimates electricity consumption (later described in 5.2.3) is nonlinear, the chosen solver is genetic algorithm. Because of simplicity in its implementation, it has been widely used for optimal controls in energy systems [3]. For more elaborate description of optimization problem, refer to Chapter 5 Optimization problem.

4. SYSTEM MODELING

This chapter begins with the physical system description. Details of system modeling follow, along with describing simplified model. Next, an explanation of the chosen structure and neural networks architecture are given. Lastly, details of training and validation processes are provided.

4.1. System description

The modelled system is the HVAC system in one of the buildings at CERN. The purpose of this air handling unit is to control temperature and humidity in the main hall where machinery is held. The system is chosen for this research since it is complete – consists of all main HVAC components – and because both temperature and humidity regulation exist.

4.1.1. System components

The system is equipped with a mixing chamber, which is an area where outside air (fresh air) and return air from hall are mixed. The amount of air coming from these two sources is regulated by damper openings: fresh air damper opening and return air damper opening. The next component is a water heating coil, which increases the air temperature via heat exchange process between air on one side (primary side) and hot water running through pipes on the other side (secondary side). The process is controlled with valve opening on the secondary side which dictates the water flow. The following component is water cooling coil, which serves to lower down air temperature in case of temperature regulation, or to remove water from the air by condensation process in case of humidity regulation. This component works on similar principles as the previous one and is controlled by the valve opening. The next one is humidifier, serving to add moisture (increase humidity) and is controlled by the command that determines the flow of air steam. Electric heater follows, with purpose of additional heating power, and controlled via electric power modulation. The last one is the variable speed fan. Besides these main components, the system is equipped with measuring stations, filters and other additional components. Front view of the system can be seen in Figure 4.

The temperature and the humidity in the main hall are the controlled variables. The test hall is supplied by air from the air handling unit. The test hall temperature and humidity are also affected by the heat load produced by machinery and people, and by the outside air temperature. The outside air temperature effect is possible due to the dissipation through walls and door openings. There is an additional component in this hall – extraction fan with extraction air damper. The extraction air damper opening controls the extraction air flow rate, and balance building pressure. The extraction fan speed is constant.

Figure 4: HVAC system in building 311

i) Simplified model

In this research, simplified plant design was modelled. First, there are five main components: mixing chamber, cooling coil, heater, fan and the zone (Figure 5). The control annotations can be found in Table 1 (for details, see in 4.1.2.i)). Additionally, only temperature regulation is observed.

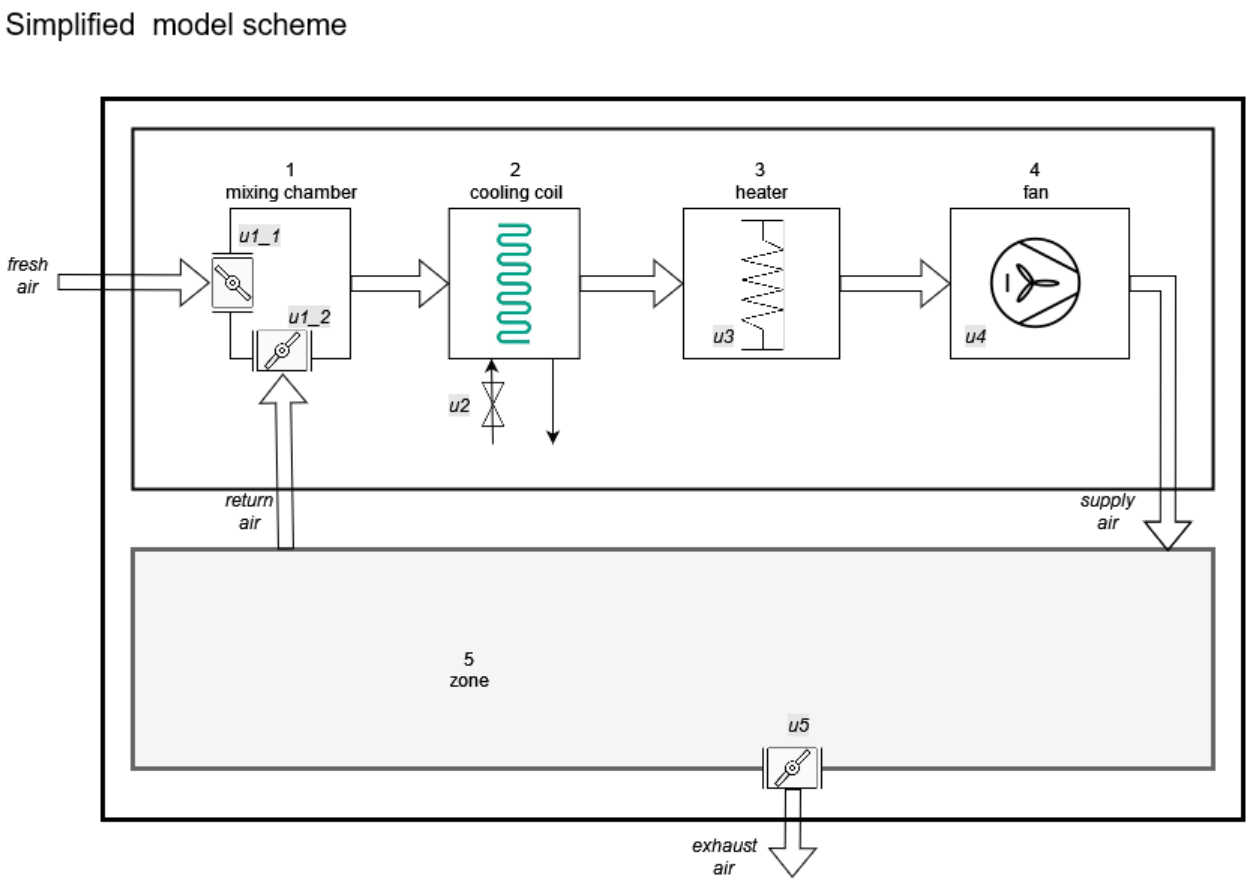

Figure 5: Scheme showing simplified model of the HVAC plant

Table 1: Control signals: mapping and ranges

CONTROL VARIABLE	PHYSICAL CONTROL SIGNAL	RANGE
u^{1_1} *(later* u^1*)*	Fresh air damper opening	0-100%
u^{1_2}	Return air damper opening	0-100%
u^2	Cooling coil valve opening	0-100%
u^3	Electric heater command	0-100%
u^4	Fan speed	0-100%
u^5	Extraction air damper opening	0-100%

4.1.2. System regulation

i) Control signals

In Table 1, all six control signals were listed. These signals are not independently defined due to the control logics, and this leads to a reduction of degrees of freedom. First, return air temperature (u^{1_2}) is set always so that it complements the fresh air damper opening (u^{1_1}) in total up to 100%. This is the strategy implemented in the current control strategy and is used in this research to reduce the complexity of the optimization problem. Second, the fan speed u^4 is fixed to 100%, as well for the simplification. Then, extraction air opening u^5 is always set to be equal to fresh air damper opening u^5 to insure balanced pressure in the building. This decision is based on controls experts' intuition and can be changed in future controller design actions. All the listed relations effectively lead to the set of three independent control signals to manipulate with: u^{1_1} (further u^1), u^2 and u^3.

ii) PID regulation

The current system is regulated with PI controllers (differential term is currently not used). Diagram is shown in Figure 6. There are two regulated temperatures: temperature in the zone *T zone* with controller PI1 and temperature at the outlet of fan *T supply* with controller PI2. The input for the PI controller is the error signal of the zone temperature. The output of PI1 represents the set point temperature for the supply air in degrees Celsius and is saturated between minimum supply temperature $T_{SUPPLY_{MIN}} = 15°C$, and maximum supply temperature $T_{SUPPLY_{MAX}} = 30°C$. In real system, supply temperature set point calculation is more complex and is not considered in this stage of the research. The error signal of the supply air temperature is the input for PI2. PI2 output is saturated between -200 and 200% and is an input for the next operation is the *split range* (Figure 7). Split range function defines the system controls variables: fresh air damper position u^1, cooling coil valve opening u^2 and the heater control signal u^3.

If the split range input falls into segment below -100%, fresh air damper openings are wide open. They are gradually closing as the input approaches 100%. With this, outside air is used for cooling, which can be done during winter, when outside air is cooler than return air. When the situation is the opposite, the commands are inversed and the first subplot of Figure 7 represent the opening of the return air damper instead of fresh air damper opening. On the second and third subplots cooling coil and heater regulation are shown. As expected, they are active on the opposite sides of split-range input signal.

What can be noticed is that in the central part of the graphs, when the split-range input is between -100% and 100%, only dampers are controlled, and active components (cooling coil, heater) are not used. This is configured so that the system manages to navigate regulation without using energy-intensive active components, but to use only damper management to maintain temperature at given setpoint. This is so-called 'free cooling', a wide-spread economical strategy to use available low external air temperatures to assist in cooling processes.

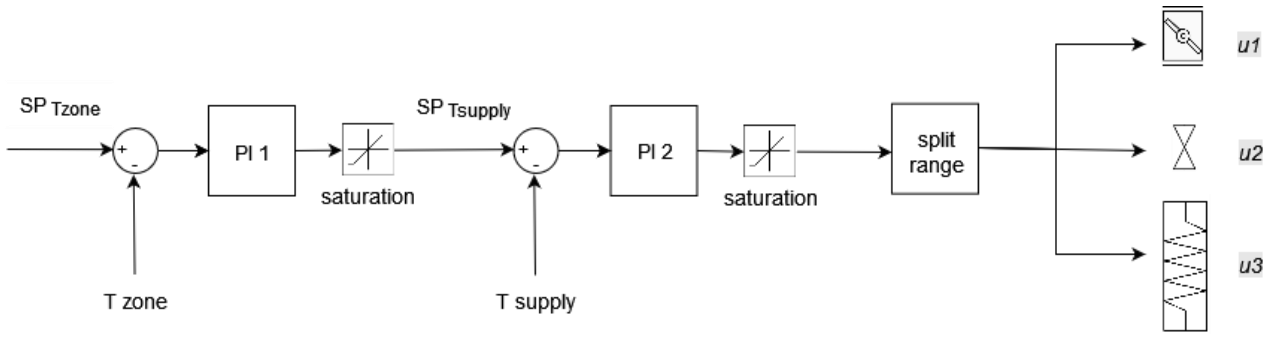

Figure 6: Regulation strategy in current system

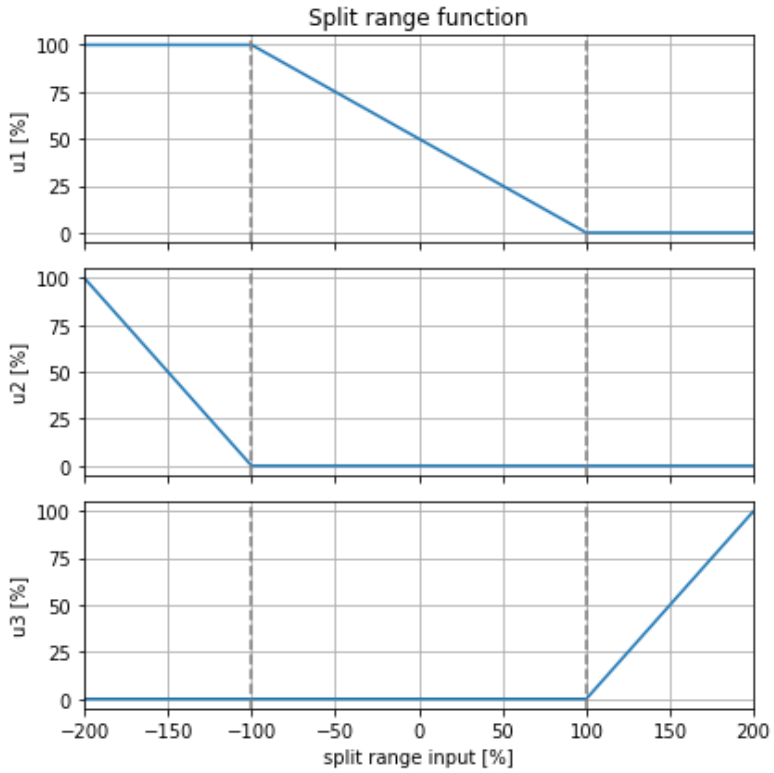

Figure 7: Split range function

iii) Implementation

In order to compare classical and advanced controls, both strategies need to be coded and run together with model in simulations. Therefore, PID strategy was also coded in Python. Changes made to the existing strategy in real system is that a wide dead band was applied for the first PI, define with limits $T_{ZONE_{MIN}} = 21°C$, and $T_{ZONE_{MAX}} = 24°C$. If zone temperature was anywhere between required minimum and maximum temperature (constraints are defined in 5.2.1.ii)), the error was considered to be 0.

Current PID regulation is real-time. Due to limited resources, computationally heavy MPC is creating new set of controls via optimization process every 15 minutes, and model is making predictions every 5 minutes. Therefore, to be comparable for this research purposes, PID digital implementation in Python gives new controls every 5 minutes. The parameters needed to be defined for each PI are the proportional gains P_1 and P_2 and integral time constants Ti_1 and Ti_2 (for details regarding discrete PID implementation, refer to [2]). Parameters used are P_1=0.9, Ti_1=10, P_2=0.7 and

Ti_2=20. Anti-windup [7] is also implemented for both controls, limiting integral term to the values that correspond to each saturation limits.

Code of digital PID implementation is available in appendix in A.3.

4.2. System modeling

As mentioned in chapter 6, model predictive control requires having a model which will predict future system behavior during simulations and be used to identify the optimal set of controls for a certain period in future. Digital representation for the plant is needed so that the optimization algorithms could be tested in a safe environment.

As shown in Figure 5, simplified system model is used in the study. Humidity regulation is disregarded, leaving the temperature regulation in focus. The selection and description of model architecture is provided in 4.2.1 and training and validation process in 4.2.2

4.2.1. Model architecture

A novel physics-informed recurrent neural network modeling methodology is applied, inspired by [5]. This is a data-driven methodology. The basic idea is to use recurrent neural networks (RNN) to model individual components of a system, then interconnect them based on the physical topology of the system.

i) Data-driven approach

For a complex process such an air-conditioning system, creating a model based on physical thermodynamic relations (white-box model, see 2.2.1.i)) would require a great number of parameters describing system design and functionality (e.g., pipe lengths, materials, diameters, dimensions of dampers, etc.). Besides being time-consuming, it would also be a non-scalable development process for other similar installations. On the other hand, data-driven approaches provide flexibility in modeling, and reduce development time for the later modeling of similar installations.

The disadvantages of data-driven approach often include lacking insight of internal processes. This is solved by having multiple models of different components, and therefore having the possibility to track variable (temperature) changes through the system and detect if components' predicted output temperatures are meaningful or not.

ii) Recurrent neural networks

The neural networks approach was chosen for modeling individual components. Their ability to learn nonlinear relationship of complex systems dynamics was a crucial factor. In specific, recurrent neural networks are tailored to such systems, because of the presence of internal loops in the paths from inputs to outputs [4]. These internal loops allow networks to learn from the past inputs and exploit that knowledge when making decisions for current processes, inherently making them suitable for representing complex dynamical processes.

The difference between a typical feed-forward neural networks (FFNN) architecture and RNN architecture is shown in Figure 8. The RNN hidden layers are clearly distinguishable from FFNN layer by their internal loops.

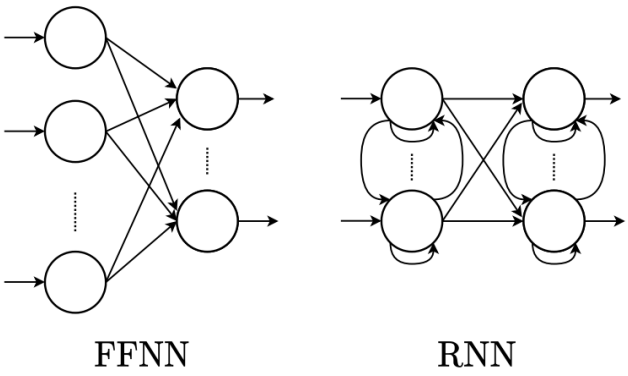

Figure 8: Illustration of FFNN and RNN architectures [4]

iii) Physics-informed neural networks

Physical knowledge of our model is embedded in interconnection between individual RNNs, based on the real plant topology. Schematic representation of model is shown in Figure 9.

There are six main blocks: 1) mixing chamber, 2) cooling coil, 3) heater, 4) fan, 5.1) zone 1, and 5.2) zone 2. Each block represents one physical component, except for the zone. The zone has two outputs: zone temperature and return air temperature. The idea was to use two neural networks to model these two outputs, so that the structure of each RNN remains uniform for all of the components. Following the block numbering, these six neural networks can be represented as RNN^{1}, RNN^{2}, RNN^{3}, RNN^{4}, $RNN^{5.1}$, $RNN^{5.2}$, with superscripts following the corresponding block numbering in Figure 9. The chosen number of hidden layers is one, as in [5] and it consists of 50 neurons.

Each of the RNNs has a set of inputs, which, for RNN^{k} in the point of time *t* is:

- a control variable for that component ($u^{k}[t]$),
- system state of the component *k* $x^{k}[t]$, which further consists of:
    a. a measurement - in this case, a fresh air temperature measurement ($x_{measurement}{}^{k}[t]$),
    b. an output of the previous neural network, with superscript *k_previous* based on the schematic ($T^{k\_previous}[t]$), and
    c. the past outputs from the RNN itself ($T^{k}[t\text{-}1]$).

Given that, the input of RNN *k*, in the point of time *t*, consists of the optimization variable $u^{k}[t]$ and the system state $x^{k}[t]$:

$$RNNinput^{k}[t] = \{u^{k}[t], x^{k}[t]\}, \tag{7}$$

where $x^{k}[t]$ is defined as:

$$x^{k}[t] = \{x_{measaurement}{}^{k}[t], T^{k\_previous}[t], T^{k}[t-1]\}. \tag{8}$$

The overview of inputs and outputs for each RNN in the point of time *t* is shown in Table 2. The controls mapping can be seen at model schematic in Figure 9, as well as in Table 1. It can be noted that RNN, when learning from its previous outputs, looks only one step in the past. Since the sampling time is 15 minutes (later discussed in 4.2.2), it is considered that previous outputs from two

and more steps in the past should not have an influence on the future system temperatures. Second, the input control for the block 4 – fan, is missing. That is because the fan speed is fixed to 100% for simplification.

Table 2: Inputs and outputs of RNNs

RNN	INPUTS [CONTROLS]	INPUTS [MEASUREMENTS]	INPUTS [OTHER RNN OUTPUTS]	INPUTS [RECURRENT VALUES]	OUTPUTS
1 – mixing chamber	$u^{1\_1}[t]$ $u^{1\_2}[t]$	Fresh air temperature in time t	$T^0[t]$	$T^1[t]$	$T^1[t+1]$
2 – cooling coil	$u^2[t]$	/	$T^1[t]$	$T^2[t]$	$T^2[t+1]$
3 – electrical heater	$u^3[t]$	/	$T^2[t]$	$T^3[t]$	$T^3[t+1]$
4 – fan	/	/	$T^3[t]$	$T^4[t]$	$T^4[t+1]$
5.1 – zone 1 (return temperature)	$u^5[t]$	Fresh air temperature in time t	$T^4[t]$	$T^0[t]$	$T^0[t+1]$
5.2 – zone 2 (zone temperature)	$u^5[t]$	Fresh air temperature in time t	$T^4[t]$	$T^5[t]$	$T^5[t+1]$

It can be noted that none of the models have flow as input, which is the unusual approach in modeling air handling units. That is because the fan speed is set on constant speed, and even with variation of other parameters, the flow estimation in the system remained mostly constant during the production of training and validation set (chapter 4.2.2), with less 10% of variation, which is considered to be insignificant.

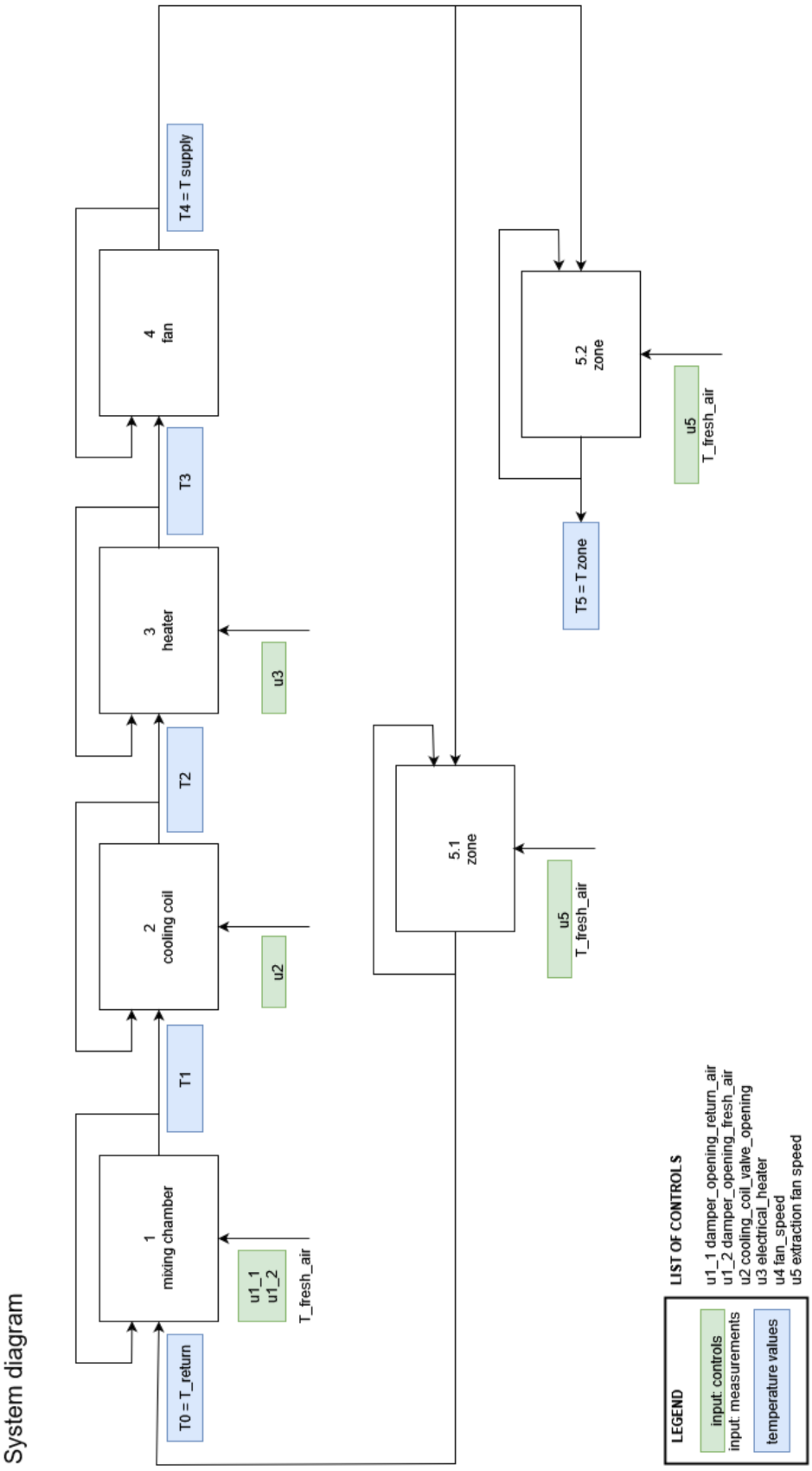

Figure 9: Schematic model representation

4.2.2. Training and validation

Each RNN was first trained individually, then all of them connected accordingly, based on the topology defined in Figure 9. Description of the training and validation process of individual components is given in the following chapter, followed by overview of the performance of the entire model.

i) Individual RNNs

In order to train and validate each of RNNs, balanced training and validation set were needed. Historical data couldn't be used in this purpose, since the control values were predominantly taking only certain range of values, not covering the complete range of operation (the range for all is 0-100%, as seen in Table 1). Balancing historical data sets was impossible to implement. For that reason, special data sets are generated by gradually changing the command from minimum to maximum value, resulting in uniform distribution over entire range, while other commands remain constant. For each three independent control values defined in 4.1.2.i), a dataset is produced, with size shown in Table 3. The sampling time is 15 minutes, which is sufficient to capture behavior of slow thermodynamic processes.

Table 3: Generated datasets

DATASET	CONTROL MANIPULATED THROUGH ITS RANGE	FIXED COMMANDS	TEST SET SIZE
A	u^1	$u^2=0$, $u^3=0$	452
B	u^2	$u^1=50\%$, $u^3=0$	242
C	u^3	$u^1=50\%$, $u^2=0$	152

For all the components, training hyperparameters were the same and are shown in Table 4. The results of individual training and validation mean square errors (MSE) for each of RNNs are shown in Table 5. Even on the validation set, the maximum error does not exceed 0.5°C, which is an acceptable error. An example of the individual validation performance was shown for the heater in Figure 10.

Code for training one singe RNN is available in A.1.

Table 4: RNN training hyperparameters

HYPERPARAMETER	VALUE
Train/validation set size ratio	50%
Number of neurons in hidden size	50
Initial learning rate	0.1
Learning rate schedule	After 35 epochs, exponentially decreasing with the low limit of 0.0001
Optimizer	Adam
Number of epochs	100
Batch size	32

Table 5 MSE on training and validation set on individual RNNs

RNN	DATASET	MSE ON TRAINING SET [°C]	MSE ON VALIDATION SET [°C]
1 – mixing chamber	Dataset A	0.061	0.125
2 – cooling coil	Dataset B	0.184	0.252
3 – electrical heater	Dataset C	0.065	0.403
4 – fan	Dataset B	0.057	0.077
5.1 – zone 1 (return temperature)	Dataset A	0.022	0.077
5.2 – zone 2 (zone temperature)	Dataset A	0.013	0.027

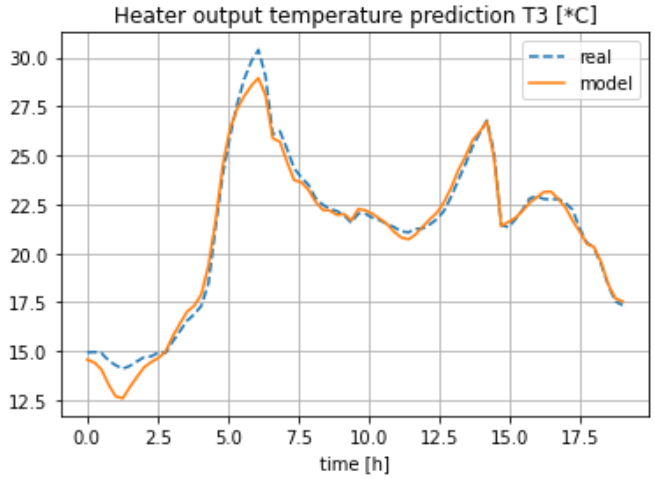

Figure 10: Individual RNN training and validation, example: heater output temperature prediction on its validation test

ii) Complete model

The complete model of the system has in total as inputs: 3 controls, 1 measurement and 6 predictions from the previous point in time. The first block produces first output, which is the input for the following one and so on until the loop is complete. The total output in time *t* consists of 6 predictions:

$$model\ inputs[t] = \{u[t], x_{measaurement}[t], T[t]\}, \tag{9}$$

$$model\_outputs[t] = \{T[t+1]\}, \tag{10}$$

where:

$$T[t] = \{T^k[t] | k = 1,2, \dots 6\}\ , \tag{11}$$

$$u[t] = \{u^k[t] | k = 1,2,3\}\ . \tag{12}$$

Code for complete model is available in appendix A.2, with model topology coded in A.2.1 and prediction function for the model in A.2.2.

5. OPTIMIZATION PROBLEM

This chapter discusses the optimization problem – electricity consumption within constraints. First, the type and the problem formulation are given, followed by presented constraints, optimization space and the optimization function formulation. Next, optimization algorithm is listed, together with implementation adaptation within MPC framework.

5.1. Problem type

The optimization objective is to minimize electricity consumption in an HVAC plant. The controls are considered in discrete steps, and the system is nonlinear due to the nonlinear nature of RNNs. This defines this optimization problem as:

- nonlinear optimization, due to the nonlinear model,
- constrained optimization, due to the presence of the state and input constraints, as is valid for all engineering problems,
- single-objective optimization, having one single optimization function that combines two requirements: minimizing electricity consumption and penalization from violating constraints,
- discrete optimization, since controls are considered in discrete steps.

5.2. Problem formulation

This optimization problem is minimizing the electricity consumption of an HVAC plant in the prediction horizon of 2 hours while maintaining supply and zone temperature within required limits. Optimization variables are a set of controls for the system for the entire prediction horizon. Detailed description of the optimization function, optimization space and variables, constraints and algorithms are following.

5.2.1. Constraints

i) Control constraints

Control (see mapping and details in Table 1 and 4.1.2.i)) constraints are defined as the following:

$$0 < u^k[t] < 100, \text{ k in } \{1,2,3\}, \forall t \tag{13}$$

$$u^2[t] * u^3[t] = 0, \forall t\,. \tag{14}$$

The first one defines the range of variables, which is between 0 and 100% for all commands. The second one concerns the fact that cooling coil and heat are not meant to work at the same time, to avoid energy waste. Therefore, in each time instance, the minimum one of these control signals is 0.

These constraints are already inherently satisfied thanks to optimization variable mapping, described in 5.2.3.i.2.

ii) Variable constraints

Zone temperature *T zone* and supply temperature *T supply* () are the controlled variables and in order for system to run properly, they need to be within certain limits, as defied in 4.1.2:

$$T_{SUPPLY_{MIN}} = 15°C < T_{SUPPLY} < T_{SUPPLY_{MAX}} = 30°C, \quad \textbf{(15)}$$

$$T_{ZONE_{MIN}} = 21°C < T_{ZONE} < T_{ZONE_{MAX}} = 24°C. \quad \textbf{(16)}$$

Violation of these constraints are penalized and included in optimization function, as described in 5.2.3.i.2.

5.2.2. Optimization variables and space

Optimization variables are control signals for the system for the entire prediction horizon. First, the coding of controls for individual point in time is defined.

As defined in 4.1.2.i), there are three independent control signals: u^1, u^2 and u^3, corresponding to fresh air damper opening, cooling coil valve opening and heater command (Table 1). These values are ranging between 0 and 100% (constraint (13)). In the real physical system, the precision of given digital control signals is the first decimal (0.1%). However, in order to simplify the problem in this research, the controls are coded in binary system. Besides simplification in representation for certain algorithms (like genetic algorithm), it also narrows search space, allowing faster convergence. The chosen number of bits to represent each command is 5 (precision of 3.125%). This would mean that default representation of control variables is size of 15 bits in total, but the default representation was not used.

The default controls representation size is reduced by exploiting the second listed control constraint (14). Since one of the signals u^2 and u^3 are 0, they don't need to be coded with two sets of 5 bits, but only one. This set of bits in controls coding is shared between u^2 and u^3. Which one will take this value, and which one is equal to 0 is decided by additional bit in coding, bit u^{23}. If it is equal to 0, cooling coil is active, otherwise, the heater.

Looking up to the process of free cooling implemented in current system regulation, described in 4.1.2, another modification to representation is done. The goal is to increase the probability of entering this area of regulation by damper management only, where less energy is consumed. A bit is added to the control variables representation, u^{off23}. If this bit is on, free cooling will be implemented, meaning that both u^2 and u^3 is set to 0 and only damper opening u^1 is regulating air conditions. If bit u^{off23} did not exist, the chances for both active commands to be 0 is equal to 3.125%, while the probability of u^{off23} being set is 50%. The possibility of entering free cooling drastically increases with u^{off23}. The final representation is depicted in Figure 11.

This mapping has two advantages over the trivial 15-bit coding: 1) it reduces the complexity of the problem by lowering number of bits from 15 to 12 and, 2) it increases the chance of finding the optimal solution by having inherently all controls satisfying the constraints 3) it is more probable to enter free cooling regulation.

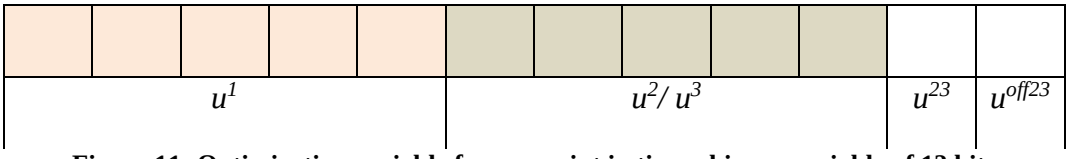

Figure 11: Optimization variable for one point in time - binary variable of 12 bits

The optimization space of an optimization variable in one point in time is represented by 12 binary bits and has 2^{12}=4096 possible states. Depending on the prediction horizon, the length of entire optimization variable (a set of individual components coding for each point of time) changes. In this research prediction period of 2h is used, which with sampling time of 15 minutes lead to 8 points of time for which controls need to be defined by optimization algorithm. Entire length of optimization variable is 12*8=96, with 2^{96} or around $8e^{28}$ possibilities.

5.2.3. Optimization function

The total optimization function is the sum of optimization functions during entire prediction horizon:

(17)

$$J = \sum_{t\ in\ prediction\ horizon} J[t].$$

The optimization function *J(t)* at time *t* is the sum of the electricity consumption the plant at time *t* to *t+1* and penalization when variable constraints are violated. This consumption is calculated as the sum of the electricity consumption of all individual components, between two points in time, *t* and *t+1*, between two runs of optimization algorithm, 15 minutes in real time. On this period,of time, components power is considered to be constant and consumption calculated by multiplying power estimate with 15 minutes = 0.25h. For the plant with *Nc* components modeled (*Nc* – number of components), the formula for optimization function is:

(18)

$$J[t] = \sum_{k=1}^{Nc} J^k(u^k[t], x^k[t])/J_{e\_max} + penalty,$$

where $J^k(u^k[t], x^k[t])$ is electricity consumption of component *k* for time *t* to *t+1* and operating state $x^k[t]$, with given set of controls $u^k[t]$, J_{e_max} is equal to 100kWh and serves to scale consumption value; *penalty* is the value penalized when variable constraints are violated. Estimation of electricity consumption of each component is described in 5.2.3.i.1 and penalty defined in 5.2.3.i.2.

5.2.3.i.1 Electricity consumption of the individual components/nodes

In the following paragraphs, electricity consumption of each component for given controls and component system state is calculated by estimating its electrical power and multiplied by time constant T_s, which is the time during which the component operates on constant power of 15minutes, or $0.25h$.

5.2.3.i.1.1 Block 1 – mixing chamber

No consumption. Actuators in mixing chamber are dampers. Their electricity consumption are negligible comparing to other components (there are only servomotors that move the dampers that consume energy), therefore:

(19)

$$J^1(u^1[t], x^1[t]) = 0, \forall t$$

5.2.3.i.1.2 Block 2 – cooling coil

The consumption of cooling coil, given the constant flow, is dependent only on the temperature drop over the component:

(20)

$$J^2(u^2[t], x^2[t]) = T_s * CoolingCoilConst\ * (T^2[t] - T^1[t]),$$

where *CoolingCoilConst* is a function of system design and operating parameters and is equal to 2, based on parameters from technical description.

5.2.3.i.1.3 Block 3 - heater

The electrical heater is equipped with a power sensor. To identify the relation between power and heater control, historical data of 5 days duration was used. This relation is shown in Figure 7. It can be noted that piecewise linear function would fairly represent the relation in question. However, for the simplicity of the model, completely linear function is used, shown in (21).

(21)

$$J^3(u^3[t]) = T_s * \frac{u^3[t]}{100\%} * 60kW$$

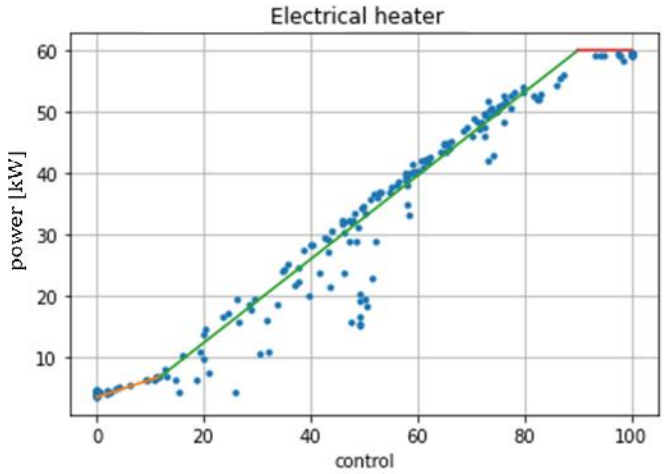

Figure 12: Electrical heater electricity power

5.2.3.i.1.4 Block 4 - fan

Fan is considered to consume constant electricity, since the flow is constant. Based on the training sets, power is equal to 2kW.

(22)

.

$$J^4 = T_s * 2kW, \forall t$$

5.2.3.i.1.5 Block 5.1 and 5.2 – zone

The only actuator in the zone is extraction air damper, so similar to block 1, the power and consumption are negligible comparing to other components:

(23)

$$J^5(u^5[t], x^5[t]) = 0, \forall t.$$

5.2.3.i.2 Penalty

Penalty was introduced to disfavor solutions when variable constraints are violated (5.2.1.ii)), meaning zone or supply temperature being outside of given range. In such case, penalty is set to be -1000, making optimization function extremely low comparing to scaled electricity consumption and in such way discouraging such solution.

5.2.4. Optimization algorithm

The chosen algorithm for optimization is genetic algorithm, as discussed in 3.3.1. Details about this algorithm performances can be found in [14].

5.2.5. Adaptations in MPC framework

Optimization variable minimizes optimization function for the time period of 2h, meaning *N_prediction_horizon* = 8 points of time. Therefore, in time *t* genetic algorithm searches for optimal controls for time *t,t+1,...t+N_prediction_horizon-1*. Controls in the very first point in time are implemented for the following 15 minutes and the others not. However, they can be used in the next point of time, *t+1*, since these controls once minimized the optimization function. There is a high probability they can be a good solution in the next point in time, together with randomly generated control in the last point in time, instead of randomly generating completely new solution. This is the first adaptation to genetic algorithm that is implemented.

The second adaptation checks if the solution with only free cooling for the entire prediction horizon can run a system within constraints. That means that before running genetic algorithm, models run the first solution consisting of random generated only damper openings, with inactive cooling coil and heater, and checks if the constraints are violated. If they are, the genetic algorithm runs and tries to find other solutions. If they are not, the generated free-cooling solution is accepted, and optimization exited. This speeds up the execution time drastically, as low as 2s with parameters used in this research (Table 6). However, if this option fails, the execution time of optimization algorithm goes up to 7 minutes. That's why this adaptation is introduced, even though, without it, it would still be possible to find a solution with free cooling only for the entire prediction horizon, but with a very low probability of less than 0.4%.

Code for genetic algorithm within MPC framework is available in A.4, with main and helper functions and classes.

Table 6: Genetic algorithm within MPC framework parameters

Variable	Value/Description	Explanation
PREDICTION_HORIZON	2 [hours]	The length of time over which future control actions are predicted and optimized.
SAMPLING_TIME	15 minutes	Sampling time of data
N_CONTROLS	2	The number of independent control variables in the system.
N_BITS_PER_CONTROL	5	The number of bits used to represent each control variable. With 5 bits, the control value can range from 0% to 100%, with 32 levels.
N_BITS_PER_ONE_ POINT_IN_TIME	*N_BITS_PER_CONTROL * N_CONTROLS + 1 + 1*	The total number of bits needed to represent all control variables and additional flag bits at one time step.
N_POINTS_OF_TIME_ IN_PREDICTION_ HORIZON	*60 * PREDICTION_HORIZON / SAMPLING_TIME*	The number of time points within the prediction horizon, determined by dividing the horizon duration by the sampling time interval.
IND_SIZE	*N_POINTS_OF_TIME _IN_PREDICTION_HORIZON * N_BITS_PER_ONE_ POINT_IN_TIME*	The size of each individual in the population, representing the total number of bits across all time points and control variables.

POP_SIZE	60	The population size for the genetic algorithm, indicating the number of potential solutions considered in each generation.
NUM_GENERATIONS	3	The number of generations (iterations) the genetic algorithm runs to evolve the population toward better solutions.
CROSSOVER_PROB	0.5	The probability that two individuals will exchange portions of their bit strings during crossover, promoting diversity in the population.
MUTATION_PROB	0.4	The probability that individual bits in the bit strings will be flipped during mutation, introducing new traits into the population.
TOURNAMENT_SIZE	3	The number of individuals competing in each tournament selection round, where the best individual is chosen for reproduction.

6. RESULTS

The research compares electricity consumption of the digital model of HVAC plant run by two control strategies: the current classical approach and new advanced algorithm. The observed period is two hours. These two methods, classical controls with PID (3.1.1) and model predictive control with genetic algorithm (3.1.2), were controlling the neural-network model of plant (4.2). Due to the limited resources in terms of computing power and time, only 10 starting data points from the generated data sets (4.2.2) were examined.

Current PID and MPC implementation on the chosen data set shows significant savings with advanced controls. Average electricity consumption is 5.96kWh with MPC, compared to 35.5kWh with standard controls, with average relative improvement of around 77%.

Results for three of them are presented in detail for closer analysis. For each of the cases, five graphs are shown, sharing a common x-axis with displayed time when the system was running: 2 hours. The first two graphs show zone and supply air temperature (4.1.1) of system driven by both strategies. The upper and lower temperature limits for each are also shown, marked as *MAX* and *MIN*, as well as the supply air set point marked as *supply T set point*. Zone temperature is controlled to stay within a temperature range between MAX and MIN, and hence has no single-value set-point (4.1.2.iii)). Graphs with control inputs follow (fresh air damper opening *u1*, cooling coil valve opening *u2* and heater control *u3*), for advanced controls (MPC with genetic algorithm) and classical controls (PID). The last graph shows air temperatures: fresh air temperature and return air temperature in case of PID regulation. This information is important input for the implemented regulation strategy (4.1.2.ii)).

6.1. Examples

Three cases have been chosen for detailed representation. Their electricity consumption is shown in Table 7.

Table 7: Electricity consumption for three chosen cases

CONSUMPTION OVERVIEW	CLASSICAL CONTROLS (PID)	MPC (GENETIC ALGORITHM)	RELATIVE IMPROVEMENT [%]
Case 1 [kWh]	21.88	8.85	59.5
Case 2 [kWh]	19.14	11.67	39.0
Case 3 [kWh]	32.26	4.0	89.6

Tracked variables for the case 1 are shown in Figure 13. Both strategies managed to keep zone and supply temperature within given limits. The difference is that genetic algorithm finds a solution that utilizes only damper openings and very few cooling coil power. On the other side, PID heavily activated cooling coil, trying to reach a given set point.

Case 2 results are shown in Figure 14. Here the starting zone temperature is above required values. Both systems successfully manage to lower down and maintain the temperature. The difference in consumption comes from the same cause as in case 1.

Case 3 (shown in Figure 15) is interesting and shows potential problems with neural-networks model. In case of PID, even though the supply air temperature was getting lower with time, zone temperature was increasing. This should be examined in detail and improved in future work, by

retraining the model and doing validation on a new dataset that will cover broader range of temperatures. Possibly, the sampling time of 15 minutes is causing temperature predictions to be unreliable and should be reviewed. Controls themselves seem to be too rigid in the point of crossover between fresh air and return air temperatures, 1.5h after the starting point, when the outside air becomes warmer than the inside one. PID suddenly changes the commands, closes fresh air dumpers and starts to heat the air. This suggests that more work should be put into finer controller tuning, possibly lowering proportional gain so that the controller response is less aggressive.

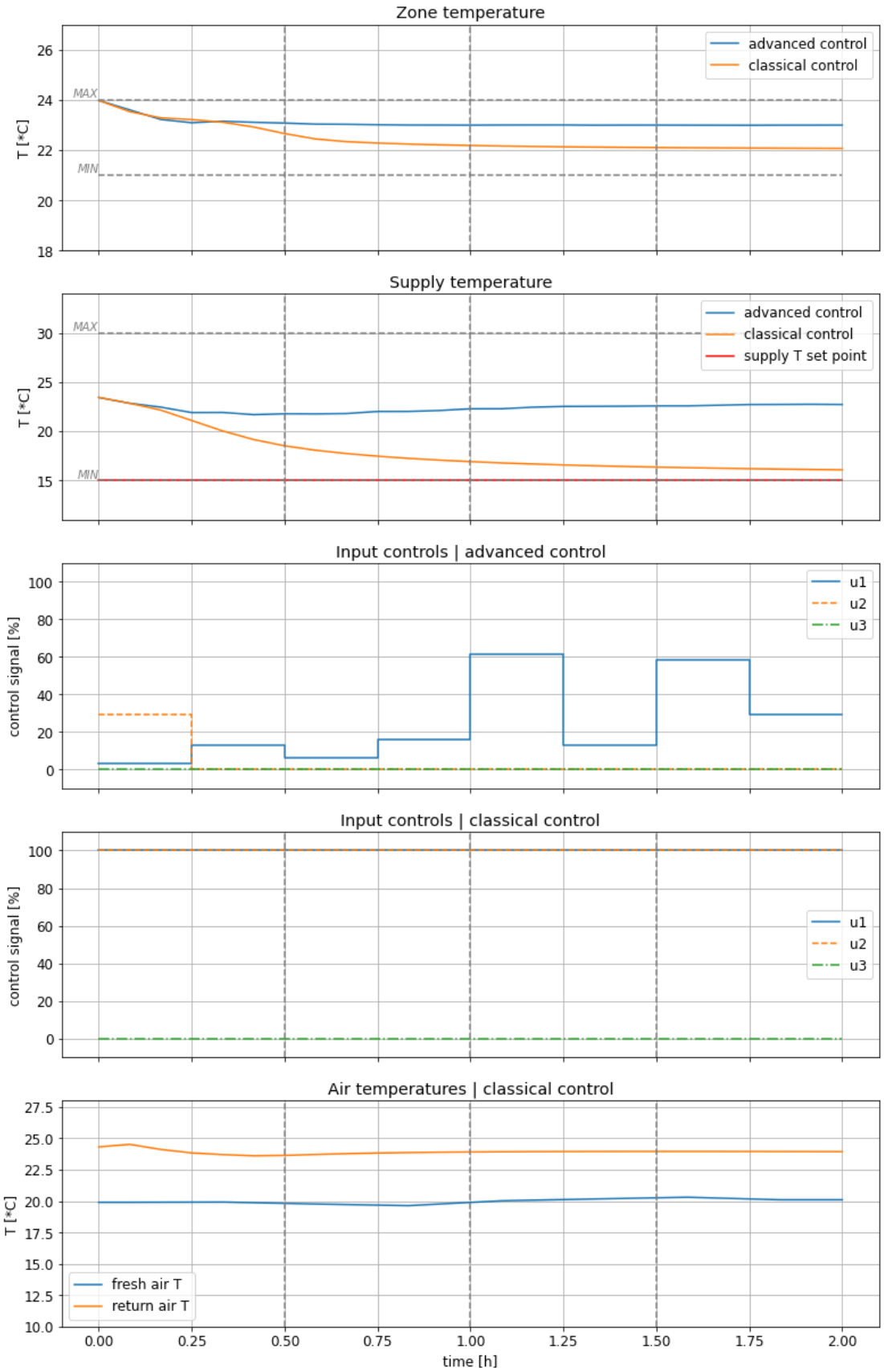

Figure 13: Results, case 1, temperatures and controls

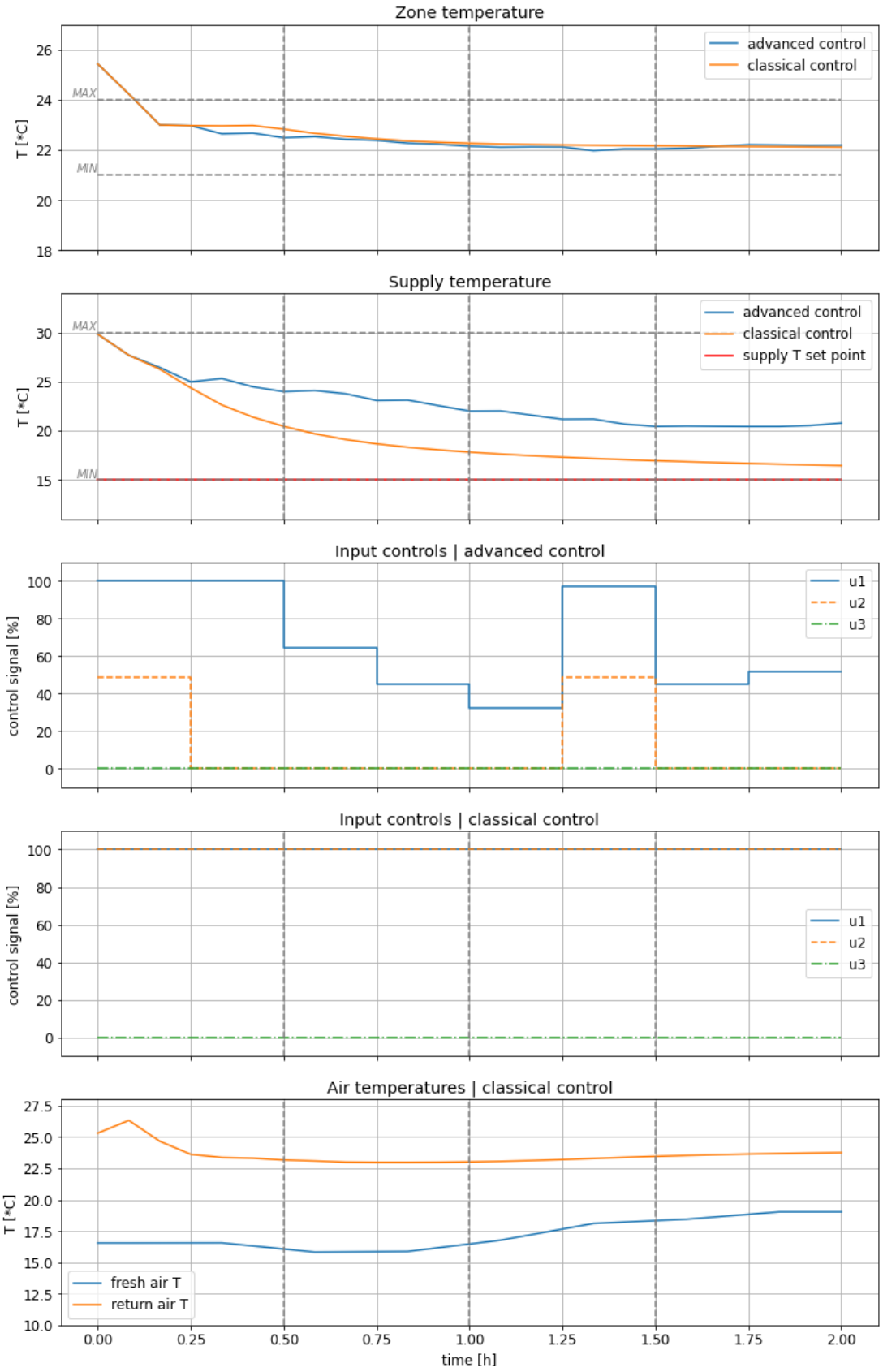

Figure 14: Results, case 2, temperatures and control

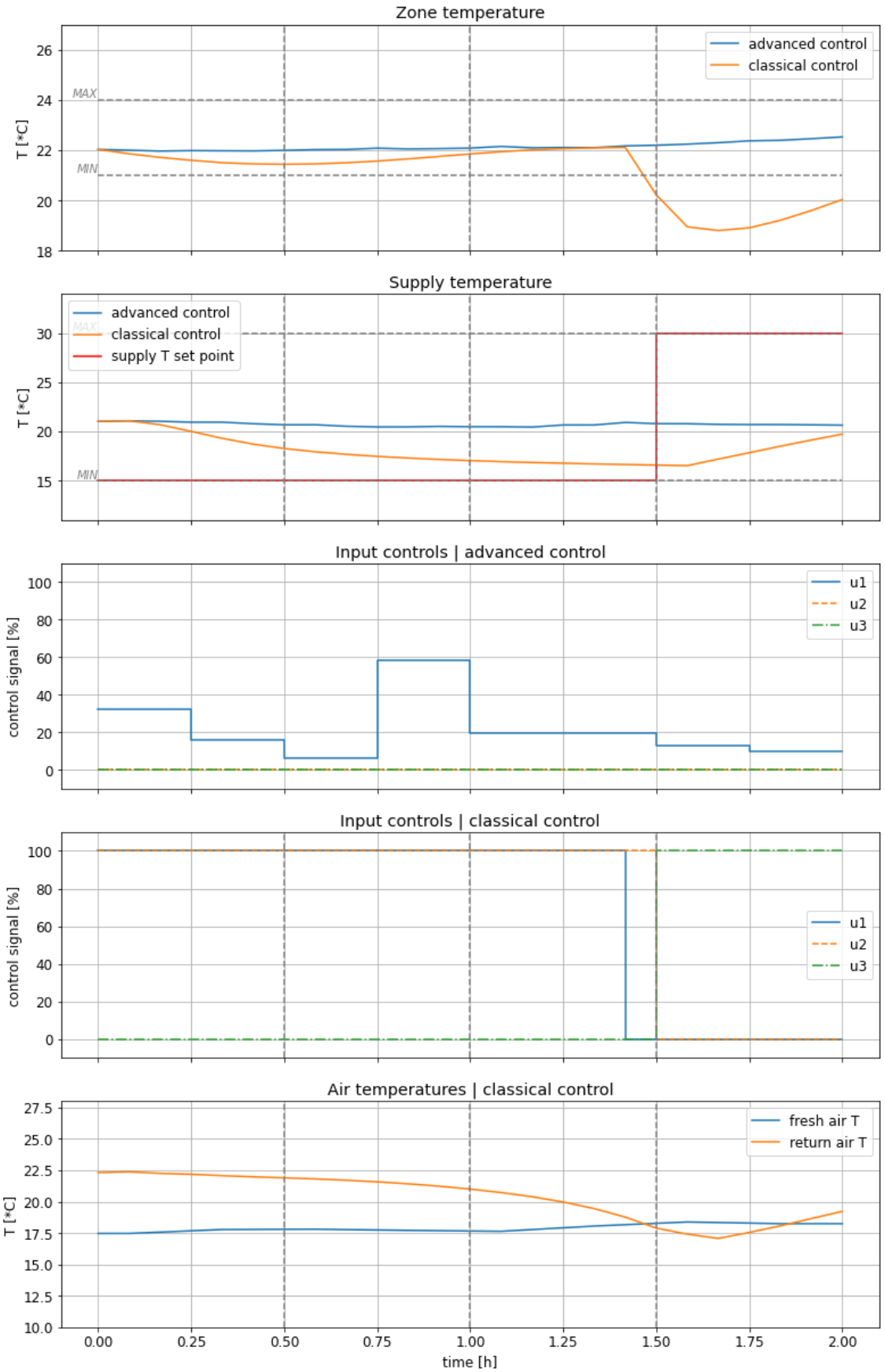

Figure 15: Results, case 3, temperatures and controls

7. CONCLUSION

This research provides a comparison between classical and advanced controls for HVAC systems regulation in terms of energy savings, as a base for further research on utilization of advanced model predictive controller (MPC) for cooling and ventilation systems. MPC was implemented for one HVAC unit at the European Organization for Nuclear Research (CERN). Method for modeling was data-driven, with physics-informed neural networks, genetic algorithm and optimization function that minimized electricity consumption while penalizing violating system constraints in terms of temperature requirements. Comparison was done for 10 datasets, each with the duration of two hours, with digitally implemented classical PID controller that represented the current system regulation strategy. Results show significant energy savings of around 77% improvement achieved by MPC, mostly by implementing damper management instead of using active components (heater, cooling coil) that consume the biggest amount of energy in the system.

Obtained results are far more promising than references in literature of savings up to 20% [18]. There, further research is needed to ensure a fair and comprehensive comparison, focusing on improving several aspects of implementation and methodology. First, digital models need to be validated on larger datasets. This could lead to the conclusion that model architecture needs to be changed or that model should be trained by optimizing all neural network together instead of individually. Sampling time should also be reviewed. Having a more precise model leads to more reliable results. Second, PID controller implementation needs to be fine-tuned, and supply temperature set point calculation upgraded to resemble calculation in real system. These changes would make the results of the comparison fairer. Regarding methodology, meaningful comparison requires a longer prediction horizon and running time, which can be easily executed if the code for MPC is optimized to run faster. In terms of the chosen configuration for comparison of controller implementations, more extreme scenarios should be tested, with outside air temperatures reaching maximum and minimum values. Exactly in these conditions it is expected to achieve the biggest savings, since it is usually the period of the year when the active components (cooling coil/heater) are the most intensively used. In order to test these conditions, they need to be included in training and validation sets for modeling. After establishing an appropriate approach regarding modeling and optimizing code, the entire HVAC unit of building is planned to be modelled and, depending on the results, the MPC deployed in real system.

The importance of this research is in setting the basis for further research on exploring MPC for energy savings in HVAC systems. Although the time and effort put into modeling and developing optimization algorithms are considerable, the potential benefit in terms of energy savings is the motivation that drives this research into further stages. This is a question important not only at CERN, but in other industrial and residential buildings. With the improvements suggested above, the study will provide reliable results that could motivate implementation of MPC on a larger scale in HVAC systems.

REFERENCES

[1] Ang KH, Chong G, Li Y (2005) Pid control system analysis, design, and technology. IEEE Trans Control Sys Technol 13(4):559–576, https://doi.org/10.1109/tcst.2005.847331

[2] Åström, K. J., & Hägglund, T. (1995). *PID Controllers, 2nd Edition*. Instrument Society of America. Research Triangle Park, NC, USA.

[3] Atabay, D. (2018). *Comparison of optimization methods for model predictive control: An application to a compressed air energy storage system*. Doctoral Dissertation, Technical University of Munich, Faculty of Electrical and Computer Engineering. (2.9.2024. https://mediatum.ub.tum.de/doc/1370275/document.pdf)

[4] Bonassi, F., Farina, M., Xie, J., & Scattolini, R. (2022). *On Recurrent Neural Networks for learning-based control: recent results and ideas for future developments*. *Journal of Process Control*, https://doi.org/10.1016/j.jprocont.2022.04.011 (2.9.2024. https://arxiv.org/pdf/2111.13557)

[5] Boca de Giuli, L., La Bella, A., & Scattolini, R. (2023). *Physics-informed Neural Network Modeling and Predictive Control of District Heating Systems*. arXiv preprint arXiv:2310.13937v1 [eess.SY]. (2.9.2024. https://arxiv.org/abs/2310.13937)

[6] CERN's main objectives for the period 2021-2025, (2.9.2024. https://home.cern/resources/brochure/cern/cernsmain-objectives-2021-2025)

[7] *ECE 486: Control Systems - Lecture 10B: Integrator Anti-windup*. University of Illinois at Urbana-Champaign. (2.9.2024. https://courses.grainger.illinois.edu/ece486/fa2021/documentation/lectures/slides/Lecture10B_IntegratorAntiwindup.pdf)

[8] Ghawash, F., Hovd, M., Schofield, B., & Monteiro, D. (2022). Model Predictive Control of Air Handling Unit for a Single Zone Setup. In *Proceedings of the 2022 IEEE International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, Vancouver, BC, Canada, 07-09 August 2022. IEEE. https://doi.org/10.1109/AdCONIP55568.2022.9894128 (2.9.2024. https://ieeexplore.ieee.org/document/9894128)

[9] González-Torres, M., Pérez-Lombard, L., Coronel, J. F., Maestre, I. R., Yan, D. (2022). A review on buildings energy information: Trends, end-uses, fuels and drivers. *Energy Reports*, 8, 626-637. https://doi.org/10.1016/j.egyr.2021.11.280 (2.9.2024. https://www.sciencedirect.com/science/article/pii/S235248472101427X)

[10] Jackson, R. B., Friedlingstein, P., Andrew, R. M., Canadell, J. G., Le Quéré, C., Peters, G. P. (2019). Persistent fossil fuel growth threatens the Paris Agreement and planetary health. *Environmental Research Letters*, 14(12), 121001. https://doi.org/10.1088/1748-9326/ab57b3 (2.9.2024. https://iopscience.iop.org/article/10.1088/1748-9326/ab57b3)

[11] Jones, C., Borrelli, F., & Morari, M. (2015). *Model Predictive Control Part I – Introduction*. Institut für Automatik, ETH Zurich; UC Berkeley; EPFL. (2.9.2024. https://ceid.utsa.edu/ataha/wp-content/uploads/sites/38/2017/10/MPC_Intro.pdf)

[12] Kim, D., Lee, J., Do, S., Mago, P. J., Lee, K. H., & Cho, H. (2022). Energy modeling and model predictive control for HVAC in buildings: A review of current research trends.

Energies, 15(19), 7231. https://doi.org/10.3390/en15197231 (2.9.2024. https://www.mdpi.com/1996-1073/15/19/7231)

[13] Monteiro, D., Bunijevac, N., Barillere, R., Rühl, I. (2023). *Controls Optimization for Energy Efficient Cooling and Ventilation at CERN*. *20th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2023)*, Cape Town, South Africa. https://doi.org/10.18429/JACoW-ICALEPCS2023-THPDP062 (2.9.2024. https://cds.cern.ch/record/2896067/files/document.pdf)

[14] Olćan D. Skripta iz optimizacionih algoritama (https://mtt.etf.bg.ac.rs/ms/osnovni.optimizacioni.algoritmi.htm#predavanja)

[15] Richalet J, Rault A, Testud JL, Papon J (1978) Model predictive heuristic control. Automatica 14(5):413–428, https://doi.org/10.1016/0005-1098(78)90001-8

[16] Schwenzer, M., Ay, M., Bergs, T. *et al.* Review on model predictive control: an engineering perspective. *Int J Adv Manuf Technol* **117**, 1327–1349 (2021). https://doi.org/10.1007/s00170-021-07682-3 (2.9.2024. https://link.springer.com/article/10.1007/s00170-021-07682-3)

[17] U.S. EIA. Global Energy & CO2 Status Report: The Latest Trends in Energy and Emissions in 2018; U.S. EIA: Washington, DC, USA, 2018.

[18] Yang, S., Wan, M. P., Ng, B. F., Dubey, S., Henze, G. P., Chen, W., & Baskaran, K. (2019). Experimental study of model predictive control for an air-conditioning system with dedicated outdoor air system. *Applied Energy*, 253, 113920. https://doi.org/10.1016/j.apenergy.2019.113920 (2.9.2024. https://www.sciencedirect.com/science/article/abs/pii/S0306261919316071?via%3Dihub)

LIST OF ABBREVIATIONS

CERN	*Organization for Nuclear Research*
CV	*Cooling and ventilation group (within CERN infrastructure)*
EN	*Engineering department (within CERN infrastructure)*
HVAC	*Heating, ventilation and air conditioning (systems)*
MPC	*Model predictive control*
LHC	*Large Hadron Collider*
PID	*Proportional–integral–derivative (controller)*

LIST OF FIGURES

LIST OF TABLES

A. Code

A.1. Individual RNN model train function

```
def train(x_train_sequences, y_train_labels, model_name, hidden_size=50,
initial_learning_rate=0.1, epochs=100, batch_size=32):
    """
    function that trains NN and saves the model
    inputs:
        x_train_sequences
        y_train_labels
        hidden_size - number of nodes in one and only hidden layer
        initial_lr=0.1
        epochs = 100
        batch_size = 32
    outputs:

    """

    # Define learning rate schedule
    def lr_scheduler(epoch, lr):
        if epoch < 35:
            return lr
        else:
            lr_ = lr * tf.math.exp(-0.05)
            if lr_ < 0.0001:
                lr_ = 0.0001
            lr = lr_
            return lr

    # Reshape x_train_set for RNN input
    x_train = x_train_sequences[:, :, :]

    # Reshape y_train_set for RNN input
    y_train = y_train_labels[:, np.newaxis]

    # Define input shape
    input_shape = x_train.shape[1:]

    # Define input layer
    inputs = tf.keras.Input(shape=input_shape)

    # Define RNN layer
    rnn_layer = tf.keras.layers.SimpleRNN(hidden_size)(inputs)

    # Define output layer
    outputs = tf.keras.layers.Dense(1)(rnn_layer)

    # Define model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    # Compile model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=initial_learning_rate)
    ,loss='mse')

    # Print model summary
    model.summary()

    # Define learning rate scheduler callback
    lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lr_scheduler)

    # Train model
```

```
    model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
    callbacks=[lr_scheduler])

    # Save model
    model.save(model_name + '.keras')

    return model
```

A.2. Complete model

A.2.1. Model topology mapping

```
def load_mapping():
    # loaded together with model
    # map to determine which controls from contros_t do i take as input for each block
    controls_dict = {
        1: [0],
        2: [1],
        3: [2],
        4: [],
        5: [0],
        6: [0],
    }

    # map to determine  which measurements from measurements_t do i take as input for
each block
    measurements_dict = {
        1: [0],
        2: [],
        3: [],
        4: [],
        5: [0],
        6: [0],
    }

    # map to determine which predictions from predictions in previous time step do i take
as input for each block
    y_t_prev_block_dict = {
        1: [5],
        2: [0],
        3: [1],
        4: [2],
        5: [3],
        6: [3],
    }

    return controls_dict, measurements_dict, y_t_prev_block_dict
```

A.2.2. Predict function

```
class SystemModel:
    def __init__(self, models_T_, T_mean_std, block_names_, controls_dict,
measurements_dict, y_t_prev_block_dict, flag_ONOFFctrl=False):
        self.models_T_ = models_T_
        self.T_mean_std = T_mean_std
        self.block_names_ = block_names_,
        self.controls_dict = controls_dict,
        self.measurements_dict = measurements_dict,
        self.y_t_prev_block_dict = y_t_prev_block_dict,
        self.flag_ONOFFctrl = flag_ONOFFctrl
        self.N_BLOCKS = len(self.models_T_)

    def predict(self, controls_t, measurements_t, y_t_prev_block, y_self_t_1):
        y_t = np.zeros([self.N_BLOCKS, 1])
```

```
        active_component_off = False
        with suppress_stdout():
            for j in range(0, self.N_BLOCKS):

                # load current measurements
                x_k_j = []

                for idx_controls in self.controls_dict[0][j + 1]:
                    x_k_j.append(controls_t[idx_controls])
                if j == 0:
                    x_k_j.append(100 - controls_t[idx_controls])

                # added so the components are off when controls <5% and temperature is
just forwarded from the previous components
                if j != 0 and (controls_t[self.controls_dict[0][j + 1]] < 2 or
(self.flag_ONOFFctrl and controls_t_[idx_controls + 1])):  # j!=0 because for mixing
chamber there is no off
                    y_t[j] = y_t[y_t_prev_block_dict[j + 1]]
                    continue

                for idx_measurements in self.measurements_dict[0][j + 1]:
                    x_k_j.append(measurements_t[idx_measurements])

                for idx_y_prev_predictions in self.y_t_prev_block_dict[0][j + 1]:
                    x_k_j.append(y_t_prev_block[idx_y_prev_predictions])

                x_k_j.append(y_self_t_1[j])

                X_test = (x_k_j - T_mean_std[j][0]) / T_mean_std[j][1]

                X_test = X_test[np.newaxis, np.newaxis, :]
                y_t[j] = self.models_T_[j].predict(X_test)

                # if cooling coil is heating or heater is cooling, keep previous value:
                if j == 1 and y_t[j] > y_t[y_t_prev_block_dict[j + 1]] or j == 2 and
y_t[j] < y_t[y_t_prev_block_dict[j + 1]]:
                    y_t[j] = y_t[y_t_prev_block_dict[j + 1]]
                    active_component_off = True

        return y_t
```

A.3. PID controller

```
def get_PID_controls(self, y_t, y_t_1, measurements_t):
    """
    @brief
        function to return control values based on previous and current system outputs
and measurements
    inputs
        y_t - output of system in time t
        y_t_1 - output of system in the previous point in time
        measurements_t - current measurements
    outputs: controls array [N_CONTROLS+1x1], ... # other PI values to save and track
changes
    """

    controls_t = np.zeros([N_CONTROLS + 1, ])
    T_fresh = measurements_t[0]

    K1 = K1_t;
    Ti1 = Ti1_t
    K2 = K2_t;
    Ti2 = Ti2_t
    SP_zone = (T_ZONE_MAX + T_ZONE_MIN) / 2
    Ts = 60 * factor  # 15*60 #sampling Time in seconds
```

```
PID_1_params = [K1, Ts * K1 / Ti1]  # P, Ti (Integral time constant in seconds)
PID_2_params = [K2, Ts * K2 / Ti2]
SP_zone = 23
I1_SATURATION = 30
I2_SATURATION = 200
PI2_SATURATION = 200
SPLIT_RANGE_LIMIT = 2 * CONTROLS_HIGH_LIM

T_zone = y_t[-1]
T_supply = y_t[3]
T_return = y_t[-2]

if T_zone > T_ZONE_MAX:
    SP_zone = T_ZONE_MAX
    error1_0 = SP_zone - T_zone

elif T_zone < T_ZONE_MIN:
    SP_zone = T_ZONE_MIN
    error1_0 = SP_zone - T_zone
else:
    error1_0 = 0
if np.abs(error1_0) < 0.01:
    error1_0 = 0

# Proportional term
P1 = PID_1_params[0] * error1_0
# Integral term
self.integral1 += error1_0 * Ts
I1 = PID_1_params[1] * self.integral1

# anti - windup 1
if I1 > I1_SATURATION:
    self.integral1 = I1_SATURATION / PID_1_params[1]
    I1 = I1_SATURATION

elif I1 < -I1_SATURATION:
    self.integral1 = -I1_SATURATION / PID_1_params[1]
    I1 = -I1_SATURATION

# Calculate PI1 output = SP_supply
SP_supply = P1 + I1
SP_supply_real = SP_supply

# saturation 1
if SP_supply > T_SUPPLY_MAX:
    SP_supply = T_SUPPLY_MAX
elif SP_supply < T_SUPPLY_MIN:
    SP_supply = T_SUPPLY_MIN

# ------------------- PI 2 : T_supply to split_range_input
error2_0 = SP_supply - T_supply

# Proportional term
P2 = PID_2_params[0] * error2_0
# Integral term
self.integral2 += error2_0 * Ts
I2 = PID_2_params[1] * self.integral2

# anti - windup 2
if I2 > I2_SATURATION:
    self.integral2 = I2_SATURATION / PID_2_params[1]
    I2 = I2_SATURATION
elif I2 < -I2_SATURATION:
    self.integral2 = -I2_SATURATION / PID_2_params[1]
```

```
        I2 = -I2_SATURATION

    # Calculate PI2 output = split_range_input
    split_range_input = P2 + I2
    split_range_input_real = split_range_input

    # saturation 2
    if split_range_input > SPLIT_RANGE_LIMIT:
        split_range_input = SPLIT_RANGE_LIMIT
    elif split_range_input < -SPLIT_RANGE_LIMIT:
        split_range_input = -SPLIT_RANGE_LIMIT

        # ------------------- split_range

    # heater
    if split_range_input > CONTROLS_HIGH_LIM:
        controls_t[2] = split_range_input - CONTROLS_HIGH_LIM

    # cooling coil
    if split_range_input < -CONTROLS_HIGH_LIM:
        controls_t[1] = np.abs(split_range_input) - CONTROLS_HIGH_LIM

    # fresh air damper
    if T_fresh < T_return:
        if split_range_input < -SPLIT_RANGE_LIMIT:
            controls_t[0] = CONTROLS_HIGH_LIM
        elif split_range_input < SPLIT_RANGE_LIMIT:
            controls_t[0] = 0.5 * CONTROLS_HIGH_LIM - 0.5 * split_range_input
    else:
        if split_range_input > SPLIT_RANGE_LIMIT:
            controls_t[0] = CONTROLS_HIGH_LIM
        elif split_range_input < SPLIT_RANGE_LIMIT:
            controls_t[0] = 0.5 * CONTROLS_HIGH_LIM + 0.5 * split_range_input

    controls_t[controls_t > 100] = 100
    controls_t[controls_t < 0] = 0
    print(SP_supply)

    return controls_t, SP_zone, SP_supply, split_range_input_real, T_fresh, T_return
```

A.4. Genetic algorithm within MPC framework

A.4.1. Class SystemState

```
class SystemState:

"""
        @brief: class to store system configuration within MPC framework; current system
state information, together with future predictions and optimal parameters
"""

    def __init__(self, system_model, measurements_t_prediction_horizon, y_t_prev_block,
y_self_t_1):
        self.system_model = system_model
        self.T_mean_std = T_mean_std
        self.measurements_t_prediction_horizon = measurements_t_prediction_horizon
        self.y_t_prev_block = y_t_prev_block
        self.y_self_t_1 = y_self_t_1
        self.predictions_final = []
        self.predictions_this_iter = []
        self.optimal_set_of_controls = None
        self.optimal_consumption = -float('inf')
```

```
        self.optimal_consumption_this_iter = -float('inf')
        self.optimal_fitness_score = -float('inf')

    def decode_set_of_controls(self, set_of_controls):
        """
        @brief: decode array of controls from binary to decimal (0-100)
        inputs:
            set_of_controls: array of binary numbers, shape
N_POINTS_OF_TIME_IN_PREDICTION_HORIZON x N_BITS_PER_ONE_POINT_IN_TIME
                             set of controls for the entire predictoin horizon
        outputs:
            decoded: array of binary numbers, shape
(N_POINTS_OF_TIME_IN_PREDICTION_HORIZON x N_CONTROLS) + 1
                each row consists of coded controls for that point of time
        """
        decoded = np.zeros((N_POINTS_OF_TIME_IN_PREDICTION_HORIZON, N_CONTROLS + 1))
        for i in range(0, N_POINTS_OF_TIME_IN_PREDICTION_HORIZON):
            binary_str = ''.join(
                map(str, set_of_controls[i * N_BITS_PER_ONE_POINT_IN_TIME:(i + 1) *
N_BITS_PER_ONE_POINT_IN_TIME]))
            # Separate the last bit as the code
            control_bits = binary_str[:-2]
            code_bit = binary_str[-2]
            both_off_bit = binary_str[-1]

            for ctrl in range(N_CONTROLS):
                value = int(control_bits[ctrl * N_BITS_PER_CONTROL:(ctrl + 1) *
N_BITS_PER_CONTROL], 2)  # Convert control bits to integer
                scaled_value = value / (2 ** N_BITS_PER_CONTROL - 1) * CONTROLS_HIGH_LIM
# Scale value to range 0-100

                if ctrl > 0:
                    # if code bit == 0, cooling coil [index 1] gets the control value,
heater remains off (ctrl = 0)
                    # if code bit == 1, heater [index 2] is on, cooling coil remains off
                    if int(both_off_bit) == 0:
                        # both remain off
                        continue
                    else:
                        decoded[i, ctrl + int(code_bit)] = scaled_value

                else:
                    # dampers
                    decoded[i, ctrl] = scaled_value

        return decoded

    def check_constraints_controls(self, controls):
        """
        @brief: check if given set of controls is valid (cooling and heating not on
simultaneously and controls within range)
        inputs:
            controls: array of binary numbers, shape
N_POINTS_OF_TIME_IN_PREDICTION_HORIZON x N_BITS_PER_ONE_POINT_IN_TIME
                each row consists of coded controls for that point of time
['u03_2_damper_opening_fresh_air','u08_cooling_coil_opening','u11_electrical_heater','ext
raction_air_opening']
        outputs:
            penalty:
                if == 0, controls constraints respected
                if == 1: controls constraints are violated
        """
        # controls constraint #1: each control is in range 0 to 100 (it is definitely in
this range, here upper limit is checked just in case scaling is invalid)
```

```
        penalty = 0
        for control in controls.ravel():
            if np.any(control > CONTROLS_HIGH_LIM):
                penalty = 1  # penalty for invalid set_of_controls

        return penalty

    def check_contraints_predictions(self, predictions):
        """
        @brief: check if supply temperature and zone temperature are within given range.
if not, discard solution (set penalty to 2)
        inputs:
            predictions:
['T_01_measure','T_04_measure','T_05_measure','T_06_measure','T_return_air','T_zone']
        outputs:
            penalty
        """
        penalty = 0
        if predictions[-1] < T_ZONE_MIN or predictions[-1] > T_ZONE_MAX:
            penalty = 2
        if predictions[3] < T_SUPPLY_MIN or predictions[3] > T_SUPPLY_MAX:
            penalty += 3

        return penalty

    def evaluate_set_of_controls(self, set_of_controls):
        """
        @brief:
        inputs:
            set_of_controls
        outputs:
            penalty: if > 0 , this individual (set_of_controls) constraints violated
            consumption_scaled
            predictions
        """
        # initializatoin
        penalty = 0
        consumption_scaled = 0
        predictions = []
        controls = self.decode_set_of_controls(set_of_controls)
        set_of_controls_new = set_of_controls.copy()
        penalty = self.check_constraints_controls(controls)
        if penalty > 0:
            return penalty, consumption_scaled, predictions

        measurements_t = self.measurements_t_prediction_horizon[0]
        y_t = self.y_t_prev_block
        y_t_1 = self.y_self_t_1
        predictions_ = []
        predictions_final = []
        predictions_this_iter = []

        for j in range(N_POINTS_OF_TIME_IN_PREDICTION_HORIZON):
            if j > 0:
                # update params for current itetation
                y_t_1 = y_t
                measurements_t = self.measurements_t_prediction_horizon[j]
                y_t = predictions.ravel()

            predictions, active_component_off = system_model.predict(controls[j, :],
measurements_t, y_t, y_t_1)

            penalty = self.check_contraints_predictions(predictions)
```

```
            predictions_.append(predictions)
            predictions_final.append(predictions[-1])

            if active_component_off:
                consumption_scaled += 0
            else:
                consumption_scaled += self.calculate_consumption(predictions, controls[j,
:])
            if j == 0:
                # store only predictions and controls for t+1
                predictions_this_iter = predictions.copy()
                consumption_scaled_this_iter = consumption_scaled

        consumption_scaled = consumption_scaled + penalty * MAX_CONSUMPTION * 10
        return penalty, consumption_scaled, consumption_scaled_this_iter,
predictions_final, predictions_this_iter

    def calculate_consumption(self, predictions, controls):
        """
        @brief
        inputs:
            predictions
            set_of_controls
        outputs:
            consumption_scaled
        """
        cc_const = 2
        deltaT = predictions[1] - predictions[0]
        cc_consumption = cc_const * np.abs(deltaT)

        heater_consumption = 60 / 100 * controls[2]

        consumption = heater_consumption + cc_consumption
        consumption_scaled = consumption / MAX_CONSUMPTION
        return consumption_scaled[0]

    def objective_function(self, set_of_controls):
        """
        @brief
        inputs:
            set_of_controls
        outputs:
            fitness_score
        """
        penalty, consumption_scaled, consumption_scaled_this_iter, predictions,
predictions_this_iter = self.evaluate_set_of_controls(
            set_of_controls)

        fitness_score = 1 - consumption_scaled  # Maximize sum of predictions, minimize
penalty

        # Check if this set_of_controls is the new best
        if fitness_score > self.optimal_fitness_score:
            self.optimal_fitness_score = fitness_score
            self.optimal_consumption = consumption_scaled
            self.optimal_consumption_this_iter = consumption_scaled_this_iter
            self.optimal_set_of_controls = set_of_controls
            self.predictions_final = predictions
            self.predictions_this_iter = predictions_this_iter

        return fitness_score
```

A.4.2. Helper genetic algorithm functions

```
# Create initial population
def create_individual():
    return [random.randint(0, 1) for _ in range(IND_SIZE)]


def create_population(pop_size):
    population = [create_individual() for _ in range(pop_size)]
    population_off_actuators_all = [0 for _ in range(IND_SIZE)]
    for i in range(N_POINTS_OF_TIME_IN_PREDICTION_HORIZON):
        population_off_actuators_all[
        i * N_BITS_PER_ONE_POINT_IN_TIME:i * N_BITS_PER_ONE_POINT_IN_TIME +
N_BITS_PER_CONTROL] = [random.randint(0, 1)

for _ in range(
                N_BITS_PER_CONTROL)]
    population = [population_off_actuators_all] + population
    return population


# Genetic operators
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_PROB:
        point = random.randint(1, IND_SIZE - 1)
        return parent1[:point] + parent2[point:], parent2[:point] + parent1[point:]
    return parent1, parent2


def mutate(individual):
    if random.random() < MUTATION_PROB:
        point = random.randint(0, IND_SIZE - 1)
        individual[point] = 1 - individual[point]
    return individual


# Tournament selection
def tournament_selection(population, fitnesses):
    selected = []
    for _ in range(TOURNAMENT_SIZE):
        i = random.randint(0, len(population) - 1)
        selected.append((population[i], fitnesses[i]))
    selected.sort(key=lambda x: x[1], reverse=True)
    return selected[0][0]


# Evaluate population
def evaluate_population(population, state):
    evaluate_ = [10 for _ in range(len(population))]
    for index, ind in enumerate(population):
        evaluate_[index] = state.objective_function(ind)
        if evaluate_[index] == 1.0:
            # maximum reached, 0 consumpiton
            break

    return evaluate_
```

A.4.3. Genetic algorithm function

```
# Main loop
def genetic_algorithm(pop_size, num_generations, state, u_optimal_previous=[]):
    """
    @brief
```

```
        Function that uses genetic algorithm to return optimal set of controls together
with other information
    inputs:
        Inputs for genetic algorithm

    outputs:
        Optimal set of controls and other
    """

    # vectors to track progress through generatoins
    optimal_fitness_score_vct = []
    optimal_consumption_vct = []
    optimal_set_of_controls_vct = []

    if u_optimal_previous == []:
        population = create_population(pop_size)
    else:
        population = create_population(pop_size - 1)
        individual_starting_with_u_previous = u_optimal_previous
        individual_remaining = [random.randint(0, 1) for _ in
range(N_BITS_PER_ONE_POINT_IN_TIME)]
        individual_starting_with_u_previous.extend(individual_remaining)
        population.append(individual_starting_with_u_previous)

    start_time = time.time()

    for generation in range(num_generations):
        fitnesses = evaluate_population(population, state)

        optimal_fitness_score_vct.append(state.optimal_fitness_score)
        optimal_consumption_vct.append(state.optimal_consumption)
        optimal_set_of_controls_vct.append(state.optimal_set_of_controls)

        updated_fitnesses = [-PENALTY_INCREMENT - fit if fit > 1 else fit for fit in
fitnesses]

        if np.max(updated_fitnesses) == 1:
            for i_finish in range(num_generations - generation - 1):
                optimal_fitness_score_vct.append(state.optimal_fitness_score)
                optimal_consumption_vct.append(state.optimal_consumption)
                optimal_set_of_controls_vct.append(state.optimal_set_of_controls)
            break

        new_population = []

        if ELIT_NUM > 0:
            paired_list = list(zip(updated_fitnesses, population))
            sorted_pairs = sorted(paired_list, key=lambda x: x[0], reverse=True)
            # Extract the top ELIT_NUM members with the highest fitness values
            elite_members = [pair[1] for pair in sorted_pairs[:ELIT_NUM]]
            new_population = elite_members

        for _ in range(pop_size // 2 - ELIT_NUM):
            parent1 = tournament_selection(population, updated_fitnesses)
            parent2 = tournament_selection(population, updated_fitnesses)
            offspring1, offspring2 = crossover(parent1, parent2)
            new_population.append(mutate(offspring1))
            new_population.append(mutate(offspring2))
        population = new_population


    end_time = time.time()
    execution_time = end_time - start_time
    return population, execution_time, optimal_fitness_score_vct,
```