

Data-driven Modeling and Black-box Optimization of MPC Parameters for Building Control

Electronic Systems - ES10-1027 - Spring 2024

Master's thesis





AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Fredrik Bajers Vej 7B
9000 Aalborg
<http://www.es.aau.dk>

Title:

Data-driven Modeling and Black-box Optimization of MPC parameters for Building Control

Project:

Master's thesis

Period:

February 2024 - May 2024

Group:

ES10-1027

Members:

Kasper Bruhn
Tristan Grusgaard Johnsen

Associated supervisors:

Jan Dimon Bendtsen

Pages: 99

Appendices: 88 to 99

Handed in 31-05-2024

Abstract:

A previously developed resistive-capacitive (nRnC) modeling approach was improved by incorporating delayed outputs, an efficient training method, and an improved scaling method for data preparation. A physics-informed neural network (PINN) was also developed using the trained nRnC model to generate training data. The PINN model's loss combined one-step prediction of the generated nRnC data and recursive prediction of the original training data, with automatic differentiation used for error gradients. Additionally, a Bayesian optimizer was implemented for online tuning of Model Predictive Control (MPC) hyperparameters, treating the problem as a black-box optimization. The Bayesian optimizer received performance signal feedback, based on the analysis of MPC performance for proposed hyperparameter values. Testing against the original nRnC model showed improved performance for both the improved nRnC model and the PINN across multiple building simulations. However, the hyperparameter tuning system did not improve performance due to misinterpretation of impact from concurrent ambient conditions.

Preface

The structure of this project is as follows: the overall report consists of three main parts: *Project Overview*, *Development*, and *Evaluation*. The first part, *Project Overview*, introduces the problem and provides background information on how this problem has been previously addressed, serving as the foundation for this project. The second part, *Development*, details the development and implementation of the proposed solution to the problem outlined in *Project Overview*. The final part is *Evaluation*, which presents the results of all tests conducted to determine whether the solution developed in *Development* was sufficient. Additionally, this section includes a discussion on potential areas for improvement and some of the challenges encountered during the project. Finally, a conclusion of the project is provided.

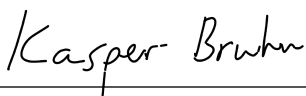
In this report the following notation is used:

Symbol	Description
x	Scalar
\mathbf{x}	Vector
\mathbf{X}	Matrix
\hat{x}	Prediction of x

Another important aspect of this report is that all illustrations and figures without citation were created by the authors. Furthermore, all software developed for this project was written in Python and executed on an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz with 8.0GB 2133 MHz speed RAM.

All source code can be found at:

<https://github.com/Tristan0017/MasterThesis.git>



Kasper Bruhn

kbruhn19@student.aau.dk



Tristan Grusgaard Johnsen

tjohna19@student.aau.dk

Contents

I	Project Overview	1
1	Introduction	2
2	Previous work with Ento	5
2.1	Data driven models	5
2.2	Model Predictive Control	8
2.3	Cost function	8
2.4	Simulation results	9
2.5	Trial of MPC on costumer HVAC system	12
3	Prerequisites	14
3.1	Energym Environment	14
3.2	Key Performance Indicators	17
4	Project Shaping	21
4.1	Delimitation	21
4.2	Problem Statement	21
4.3	Requirements	21
4.4	Accepttest	22
II	Development	25
5	System Overview	26
5.1	Project objective	26
5.2	Updated cost function	27
5.3	Sampling Frequency	28
5.4	Data Generation	29
5.5	Data Scaling	30
6	nRnC	32
6.1	Collection of denominator terms	32
6.2	Consideration of inertial effects	32
6.3	Parameter constraints	33
6.4	Training and selecting models	34
6.5	Increasing training efficiency	39
6.6	Model Validation	45

7	Physics-informed neural network	46
7.1	Background	46
7.2	Utilization of PINN concept for building modelling.	48
7.3	Training the model	53
7.4	Model Validation	55
8	Auto-tuning of hyperparameters	58
8.1	Problem definition	58
8.2	Candidate methods	59
8.3	Bayesian optimization	60
8.4	Implementation	65
III	Evaluation	70
9	Acceptance tests	71
9.1	Test of Req. 1	71
9.2	Test of Req. 2	74
9.3	Test of Req. 3	76
10	Discussion	78
10.1	Test results	78
10.2	Improved nRnC model	79
10.3	Physics-informed neural network	80
10.4	Hyperparameter tuning	80
11	Conclusion	82
	Bibliography	83
IV	Appendices	88
A	Effect of including delayed outputs in nRnC model	89
B	Plots of building control performance	92

Part I

Project Overview

1 | Introduction

In recent years, global efforts to mitigate climate change have intensified, with a particular focus on reducing carbon emissions across various sectors. Governments, businesses, and individuals are implementing strategies to transition towards more sustainable practices. One significant contributor to carbon emissions is the operation of buildings, which accounted for 26% of total emissions in 2022, according to the International Energy Agency (IEA) [1]. As outlined in the *European Green Deal*, the goal is to reduce carbon emissions by 55% by 2030 and achieve carbon neutrality by 2050 [2]. However, emissions related to buildings have unfortunately increased by an average of 1% every year since 2015 [1]. Therefore it is clear that this trend needs addressing, especially considering the projected increase in floor area which more than offsets efficiency gains and decarbonization efforts. The emissions related to buildings in recent years are shown in Figure 1.1

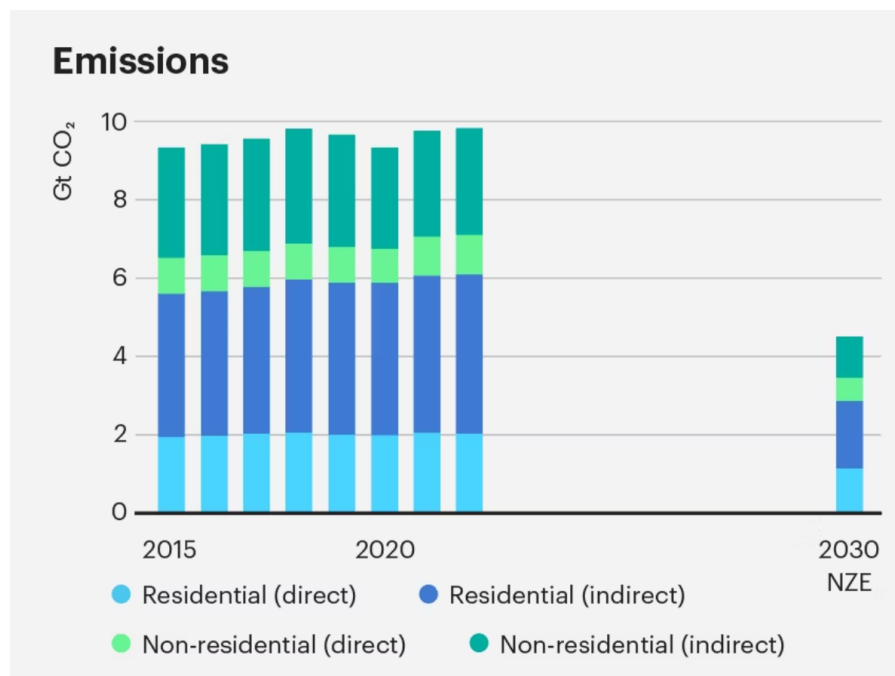


Figure 1.1. Development in building emissions from 2015 to 2022. As seen on the right, the emissions are to be halved by 2030 in order to be on track for Net Zero Emissions (NZE) by 2050 [1].

To deal with rising emissions from buildings, stringent policy measures have been put in place. Among these are building energy codes, such as the International Energy Conservation Code (IECC) in the United States, which mandate specific energy efficiency criteria for new constructions and major renovations [3]. Similarly, standards like ASHRAE Standard 90.1 prescribe minimum efficiency levels for building components and systems, which promote efforts towards improving performance and energy savings [4]. These legislative measures not only encourage the adoption of energy-efficient technologies but also establish clear targets and benchmarks for improving energy performance and reducing carbon emissions in the building sector. Furthermore, incentives for green building practices encourage developers and building owners to exceed minimum requirements and achieve higher levels of energy performance.

With the regulatory framework established to drive enhancements in energy efficiency, modern Building Management Systems (BMS) assume a crucial role in meeting and exceeding these objectives. Alongside the intensified focus on reducing carbon emissions in recent years, there has been a notable increase in available building data. This encompasses the deployment of smart meters capable of monitoring household natural gas or electricity consumption, among other data sources. For smart meter implementation in Europe specifically, a recent report by the European Commission stated that 95% of distribution systems operations have implemented plans for smart meter rollout on their grids [5]. Simultaneously, the Internet of Things (IoT) has emerged as a prominent concept, supporting digital interconnectivity among modules and systems linked to the internet. Once interconnected, the systems within the IoT ecosystem can be harnessed, thereby empowering the BMS to employ more advanced methodologies [6].

The goal of a BMS depends on the needs in the building in which it is installed, but typically BMS are implemented to control the temperature or air quality [7]. Controlling the air quality typically involves measuring different quantifiable things in the air, such as parts per million of CO₂ in the air or level of carbon monoxide due to running engines. Adjustment of these levels typically involve ventilating the indoor environment by introducing air from the outside. The controls for these subsystems in HVAC setups are typically based on feedback from sensors, since the pollutants cannot be removed from the building before they are introduced. According to a 2018 review by the U.S. Energy Information Administration, ventilation accounted for 11% of energy use in American buildings [8]. In the same review, heating accounted for 32% of the energy used, while cooling accounted for 9% [8]. As evident from these numbers, the efficiency of heating systems therefore play a large role in reducing the overall energy consumption of buildings. For control of heating subsystems in HVAC, BMS designers can take many different approaches. A widely implemented approach is scheduling, where the heating system is made to engage and disengage at certain times throughout the day. The schedule is typically based on occupancy patterns, and can be carefully selected to match precisely with these patterns. According to a review of scheduling methods, the most widely used HVAC scheduling approaches are either *basic* scheduling techniques, where the entire HVAC system is turned on or off at strategic times, or *conventional* techniques where the system is kept on but the temperature references are manipulated. Common among the approaches in the review, is the improvements to energy costs and comfort provided by pre-heating or pre-cooling the area before the building becomes occupied, since the occupied period overlaps with the period of peak energy costs. This strategy also helps reach temperature reference points in time for the occupied period starting. Similarly, improvements to energy costs were found when turning off the system or relaxing the reference points towards the end of the occupied period [9].

To implement a control system which can implement similar behaviour without the need for manual scheduling, an element of predictive behavior is required. This is the case, since most heating or cooling systems take a while to change the temperature inside the building. Therefore, the control system requires knowledge about future reference set points beforehand, in order to provide anticipatory heating or cooling actuation. One modern control method which has the ability to make predictive actions is Model Predictive Control (MPC). This approach utilizes a mathematical model of the building and actuators to predict future behavior, and provide control signals accordingly by solving an optimal control problem [10]. To obtain the required model for this method, the wealth of IoT and smart meter data can be leveraged for data-driven model discovery.

This can aid in avoiding the reliance on knowledge of the specific dynamics of a given building, which is usually necessary for a good mathematical model to be developed [11]. Some companies are already working on this problem of data-driven controller synthesis for buildings, among which is Ento [12]. Ento is a company which has worked with aggregated building data for years, where customized recommendations and analysis of savings can be provided based on historical building data. Recently they launched their new control product, which can utilize a sophisticated building model to provide optimal control for costumers [13]. The authors of this report were involved in the development of this product during the fall semester of 2023 as interns. This collaboration will be described later in this report. However, the effectiveness of MPC also depends on the tuning of the hyperparameters responsible for weighting the cost function used in the optimal control problem. Careful hyperparameter tuning ensures that the model is optimized to perform well under various conditions and scenarios, enhancing its adaptability and accuracy. Therefore a comprehensive approach can be taken, where the wealth of data from IoT and smart meter sources is used for data-driven model discovery, alongside the incorporation of automated hyperparameter tuning algorithms. With such an approach, a data-driven controller synthesis could become feasible. This project will focus on these topics.

2 | Previous work with Ento

The problem laid out in Chapter 1 was previously worked on by the authors of this report during fall 2023 while in an internship with the company Ento. Ento is a company founded in 2019, that specializes in machine learning aided analysis of aggregated building data. Through this, Ento is able to identify regular consumption patterns, and provide actionable recommendations based on these for a given building [12]. It was decided also to use this aggregated building data for other things, and therefore building control became a potential new service which Ento would offer to costumers. During the internship, a framework for MPC utilizing trained data-driven models of building dynamics was laid out. The project proved successful as a proof of concept, and the results of Ento's further work on the topic can be seen on their website, where costumers can now purchase ventilation and heating control services [13]. Since this project expands upon some of the methods used, this chapter will provide a brief synopsis of the work done by the authors during the internship.

2.1 Data driven models

In this section a brief introduction of each of the tested models and training methods will be given.

2.1.1 nRnC model

The first data-driven model attempted during the project was based on a resistive-capacitive (RC) circuit analogous model of the thermodynamics in the building. This model assumes that the rate of change in internal temperature ($\frac{dT_{int}}{dt}$) is controlled exclusively by the heating system in the building ($W_{heating}$), the difference between the internal and ambient temperature ($T_{amb} - T_{int}$), and the ambient solar radiation (W_{sun}). The setup is illustrated in Figure 2.1.

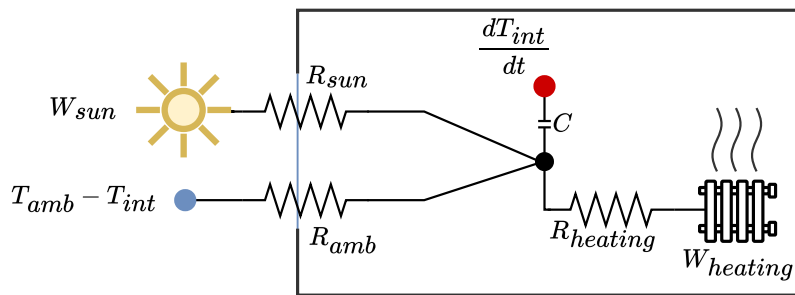


Figure 2.1. Simplified illustration of a building modelled with an RC circuit.

Based on the inputs, the predicted rate of change in internal temperature $\frac{dT_{int}}{dt}$ from the current time step to the next is found. A forward Euler integration scheme is then applied to obtain the next internal temperature [14] based on the rate of change.

This model is mathematically expressed as shown in (2.1).

$$C \cdot \frac{dT_{int}}{dt} = \frac{T_{amb} - T_{int}}{R_{amb}} + \frac{W_{sun}}{R_{sun} \cdot c_{sun} \cdot m_{sun}} + \frac{W_{heating}}{R_{heating} \cdot c_{heating} \cdot m_{heating}} \quad (2.1)$$

where,

Symbol	Unit	Description
$W_{heating}, W_{sun}$	$\frac{kg \cdot m^2}{s^3}$	Heating power of heating system and solar radiation
T_{int}, T_{amb}	K	Internal and ambient temperatures
$R_{heating}, R_{sun}$	$[.]$	Resistance related to heating system and solar radiation
C	$\frac{J}{K}$	Thermal capacitance of the building
$c_{heating}, c_{sun}$	$[\frac{J}{kg \cdot K}]$	Specific heat capacity of the heating and sun subsystems
$m_{heating}, m_{sun}$	$[kg]$	Mass of the materials in the heating and sun subsystems.

For brevity, the specific heat capacity and mass of the heating and solar radiation subsystems will be collected in the term $j = c \cdot m$. The data is scaled using standardization in preparation for training the model. To train the nRnC model, the values of the parameters R_{amb} , R_{sun} , $R_{heating}$ and C which allow the model to predict the internal temperature in the training data as accurately as possible, are estimated. One approach to estimating these values would be, to attempt to predict the entire data set with N samples using the first sample as an initial condition. However, the approach chosen was instead to mimic the way the model is to be utilized in the MPC, where a prediction of some moderate span of time into the future is made. During training, the model is therefore made to start a prediction of n_{ps} steps ahead, based on an initial condition. By iterating through the dataset and using every sample as the initial condition for a separate n_{ps} step prediction, the sum of the prediction error obtained starting from every initial condition can be found. This sum is then minimized using parameter estimation of the R and C parameters in (2.1). The minimization problem using this approach is shown in (2.2)

$$R_{amb}, R_{sun}, R_{heating}, C \quad \underset{\text{minimize}}{\sum_{j=1}^{N-n} \left(\sum_{k=1}^n \left(\|T_{int_{j+k}} - \hat{T}_{int_{j+k}}\|_2^2 \right) \right)} \quad (2.2)$$

$$\text{subject to} \quad \frac{dT_{int_{j+k}}}{dt} = \frac{T_{amb_{j+k}} - T_{int_{j+k}}}{R_{amb} \cdot C} + \frac{W_{sun_{j+k}}}{R_{sun} \cdot C \cdot j_{sun}} + \frac{W_{heating_{j+k}}}{R_{heating} \cdot C \cdot j_{heating}} \quad (2.3)$$

$$T_{int_{j+k+1}} = T_{int_{j+k}} + \Delta t \cdot \frac{dT_{int_{j+k}}}{dt} \quad (2.4)$$

For further clarification of the method, the code implementation of the prediction error sum can be seen in the lines shown in Snippet 2.1.

```

1 for i in range(1, (len(training_data)-n_ps)):
2     T_predicted = np.zeros(n_ps)
3     T_int = T_data[i-1]
4     for j in range (n_ps):
5         T_predicted[j] = T_int + (dt * r1c1_model(T_int, C, R_amb, R_q, R_sun,
6             T_amb[i+j], Q[i+j], Sun[i+j]))
7         T_int = T_predicted[j]
8     cost += np.sum((T_data[i:(i+n_ps)] - T_predicted)**2)

```

Snippet 2.1. Python code for training based on segmented prediction performance.

2.1.2 Multilayer Perceptron model

A model based on simply training a neural network to capture the dynamics of the system was also attempted. This model took the same inputs as the nRnC model described above, namely the heating power, the solar radiation power, and the difference between internal and ambient temperature. In addition to these, two time-lagged measurements of site temperature were given as inputs to the model, in order to provide the multilayer perceptron model (MLP) with knowledge regarding the recent dynamic of the site temperature [15]. To train the weights Λ and biases β of the MLP, the minimization problem shown in (2.5) was solved with a training set containing N_{train} samples.

$$\underset{\Lambda, \beta}{\text{minimize}} \quad \sum_{k=1}^{N_{train}} \left(T_{int_k} - \hat{T}_{int_k} \right) \quad (2.5)$$

2.1.3 Neural State-Space model

Another model which was attempted was a Neural State-Space model (NSSM). The basic idea behind this model is to replace the matrices in the classical state-space model with neural networks, which in theory enables the state-space model to capture nonlinearities in the training data better than with the typically linear matrices [16]. The discrete time NSSM with weights λ is shown in (2.6)

$$\frac{dT_{int}}{dt}(k) = A_{\lambda_A} x(k) + B_{\lambda_B} u(k) + D_{\lambda_D} d(k) \quad (2.6)$$

Here $u(k)$ the energy input to the system from the heating system and $d(k)$ is the disturbances to the system, which contain the difference between ambient and internal temperature, as well as solar radiation power. To train the model, a minimization problem including both the *one-step loss* and *tracking loss* was solved. The one-step loss is a loss similar to the one shown in (2.5) which penalizes errors in predicting the immediate output. The tracking loss was instead a prediction I steps ahead obtained by feeding the predicted states into the model recursively.

The accuracy at every step was included in the penalization. Through trial and error it was decided to place more emphasis on the tracking loss, hence this loss is scaled with 10. The minimization problem is shown in (2.7)

$$\underset{\lambda_A, \lambda_B, \lambda_D}{\text{minimize}} \quad \sum_{k=1}^{N_{train}-I} \left(\|T_{int}(k+1) - \hat{T}_{int}(k+1)\|_2^2 + 10 \cdot \sum_{i=1}^I \|T_{int}(k+i) - \hat{T}_{int}(k+i)\|_2^2 \right) \quad (2.7)$$

$$\text{subject to} \quad \frac{dT_{int}}{dt}(k) = A_{\lambda_A}(T_{int}(k)) + B_{\lambda_B}(u(k)) + D_{\lambda_D}(d(k)) \quad (2.8)$$

$$T_{int}(k+1) = T_{int}(k) + \frac{dT_{int}}{dt}(k) \quad (2.9)$$

2.2 Model Predictive Control

MPC operates on a principle called *receding horizon*, in which the current control action is obtained by solving a finite horizon open-loop optimal control problem at each sampling instant, using the current state of the plant as the initial state [10]. MPC determines the optimal control input sequence at a given step by minimizing a cost function using optimization methods. The cost function is evaluated based on a prediction of the future, which is obtained by using a trained model of the system. When obtaining the optimal control input, the MPC cost function is defined with three horizon parameters; H_p , H_u and H_w . The prediction horizon H_p determines how far into the future predictions should be computed. The control horizon H_u determines the final time step in the sequence where the control input can be changed. Finally H_w determines how many time steps after initialization must elapse, before the control input can begin to be changed. For the MPC in the previous project as well as this one, the value of $H_w = 0$, and $H_u = H_p$. The MPC problem is computing the vector of optimal control inputs \mathbf{u} , containing the optimal control input at each time step $k+i$ up to $i = H_p$. Once obtained, the first entry in the vector is given as a control input to the plant, after which the states of interest are measured at the next sampling time, and used as the next initial condition for the MPC.

2.3 Cost function

The cost function J in MPC is used to express the cost associated with the trajectory of different variables during the prediction horizon. Here the cost function is based upon reference tracking accuracy, defined as deviation in internal temperature $\mathbf{x}(k)$ from the reference vector \mathbf{r} , and size of control input change $\Delta u = u(k+i+1) - u(k+i)$ between all time steps. For the MPC used during the project with Ento, also penalization of the sum of control inputs is included, due to the desire to reduce energy consumption. This also provides a convenient way of tuning the emphasis placed on occupant comfort in relation to energy consumption reduction for a given building. Finally an increase in penalization is placed upon negative temperature deviations of more than \mathbf{t}_{lower} degrees from the reference during occupied hours. In the cost function, this lower bound value is defined as $\mathbf{r}_{lower} = \mathbf{r} - \mathbf{t}_{lower}$, and is only included during the occupied hours. This penalization was included, due to considerations related to the optimal control input obtained in the MPC, specifically during switching from the temperature reference used in unoccupied and occupied hours.

Without this penalization, the temperature curve would often be placed approximately halfway between the two temperature references at the switching time. While this might be optimal in terms of minimizing the distance from the reference before and after the switching time, it is not in line with the goal of reaching a certain temperature at a certain time. The full cost function can be seen in (2.10).

$$J(k) = \sum_{i=1}^{H_p} |\hat{\mathbf{x}}(k+i|k) - \mathbf{r}(k+i)| \cdot R \quad (2.10a)$$

$$+ \sum_{i=1}^{H_p} |\min(\hat{\mathbf{x}}(k+i|k) - \mathbf{r}_{lower}(k+i), 0)| \cdot R \cdot \xi \quad (2.10b)$$

$$+ \sum_{i=0}^{H_p-1} |\hat{\mathbf{u}}(k+i|k)| \cdot C \quad (2.10c)$$

$$+ \sum_{i=0}^{H_p-2} (\hat{\mathbf{u}}(k+i+1|k) - \hat{\mathbf{u}}(k+i|k))^2 \cdot C_{\Delta} \quad (2.10d)$$

where,

Symbol	Unit	Description
R	$[\cdot]$	Penalization weight of the error between the system trajectory to the reference
\mathbf{r}_{lower}	$[^{\circ}C]$	Lower reference boundary
ξ	$[\cdot]$	Penalty weight scaling variable for breaching the lower reference
\mathbf{u}	$[W]$	Control input
C	$[\cdot]$	Penalization weight of the control input
C_{Δ}	$[\cdot]$	Penalization weight of the change in control input

2.4 Simulation results

To verify the validity of the data driven approach, a test on a simulated building was performed. To obtain training data for the model, first the building was simulated with a proportional feedback controller which controls the heating system in the simulated building, by using measurements of the internal temperature as feedback. Data was collected over 365 days of simulation. After collecting the data, the models were trained by solving the minimization problem shown in (2.2). With a trained model, the MPC was implemented instead of the proportional feedback control. The tuning of the R and C penalization weights were determined through trial and error, where emphasis was placed on the ability of the controller to reach the reference during the rising edge of the occupied hours. The results for the nRnC model obtained are shown in Figure 2.2

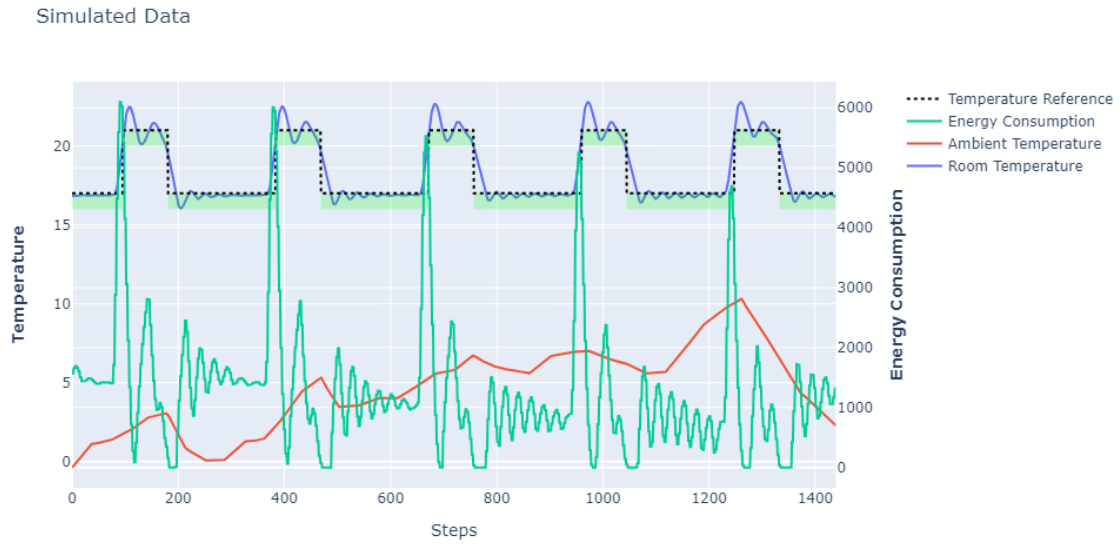


Figure 2.2. Performance of the MPC utilizing the nRnC model during simulation. The percentage of points above r_{lower} was 99.44%.

As seen, the MPC using the nRnC model as an internal model performs well, and achieves almost full time in range. From the performance it can also be seen, that the model seems to be able to capture the inertia of the building accurately enough, as to allow the MPC to raise the temperature in anticipation of the higher reference temperature during the day. However, it can also be observed that the model still has some degree of inaccuracy, since the steady state condition during the night exhibits oscillating behavior. The results for the MLP model are shown in Figure 2.3

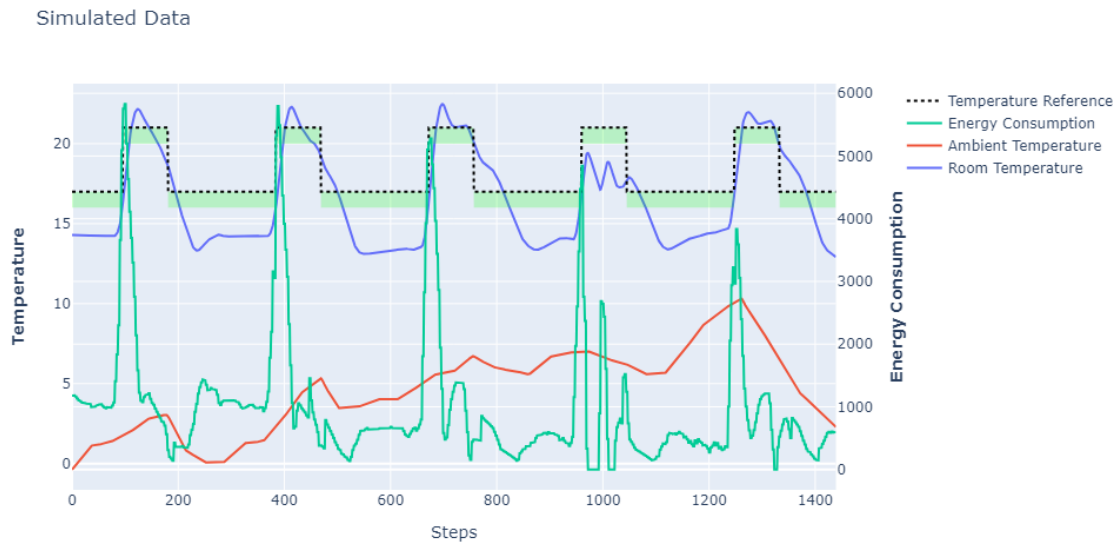


Figure 2.3. Performance of the MPC utilizing the MLP model during simulation. The percentage of points above r_{lower} was 34.51%.

From this performance it was evident, that the MLP model did not capture the dynamics of the building well enough to be used for predictive control. The results for the NSSM are shown in Figure 2.4

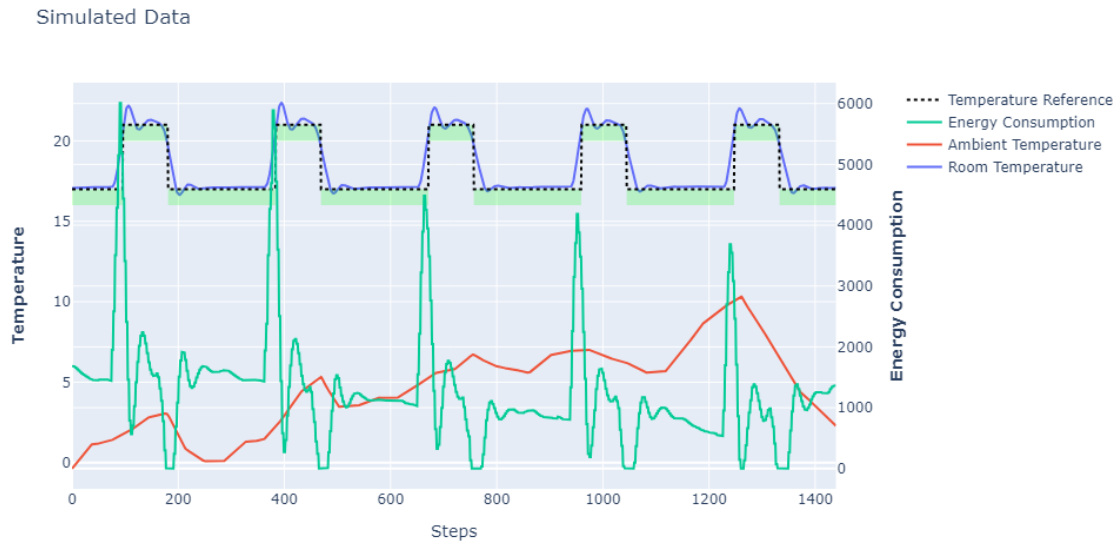


Figure 2.4. Performance of the MPC utilizing the NSSM during simulation. The percentage of points above r_{lower} was 99.38%.

As seen, the NSSM was able to achieve a very convincing performance. However, the time above r_{lower} achieved was actually marginally worse than the one obtained with the nRnC model. However, it is important to consider the performance of this model in context of its complexity, which caused the time required for simulation and training of this model to be very large, when compared to the other two models.

2.5 Trial of MPC on costumer HVAC system

In order to choose which model to test on a costumer HVAC system, a comprehensive simulation test was performed. The energy consumed and the TIR are calculated, and shown in Table 2.1.

Model	TIR	Energy Consumption	Unit
nRnC	99.44%	2882.637	[kJ]
MLP	35.41%	2296.316	[kJ]
NSSM	99.38%	2885.307	[kJ]

Table 2.1. Time in range and energy consumption for the MPC simulations with the three model types.

Based on these results, it was decided that the nRnC model was best suited for a test on a costumer system. In collaboration with Ento, the developed system was implemented on costumer HVAC systems. The system was implemented in the HVAC system at three different costumer sites.

One of these rooms had a heating system utilizing hydronic underfloor heating, which meant that the system had a much slower response time to heating input. Given a longer prediction horizon, the system was still able to provide decent performance despite this challenge. The obtained results for one of the sites is plotted in Figure 2.5, with equally long periods with and without the new control implemented.

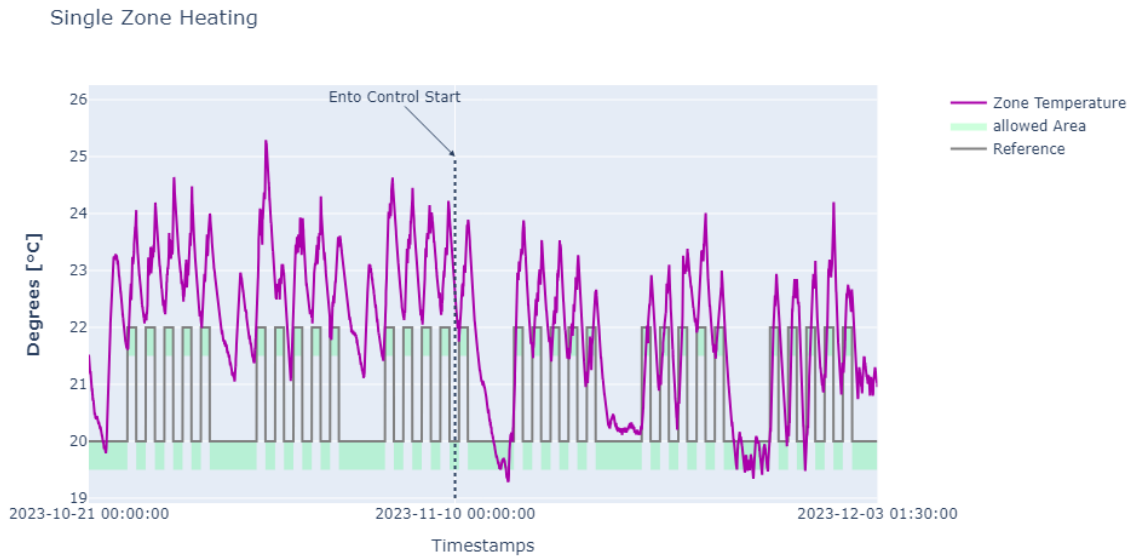


Figure 2.5. Site temperature measurements before and after control implementation with 10 minutes intervals.

When analysing the performance qualitatively, it was concluded that the proposed MPC control system with a data-driven model achieves good performance. However, it was difficult to give a quantitative evaluation of the performance achieved by the systems in all three zones, due to the unfortunate fact that the results were collected during a period, in which the ambient temperature dropped steadily.

The time in range, defined as being above r_{lower} , for the three zones along with the ambient temperature is shown in Figure 2.6

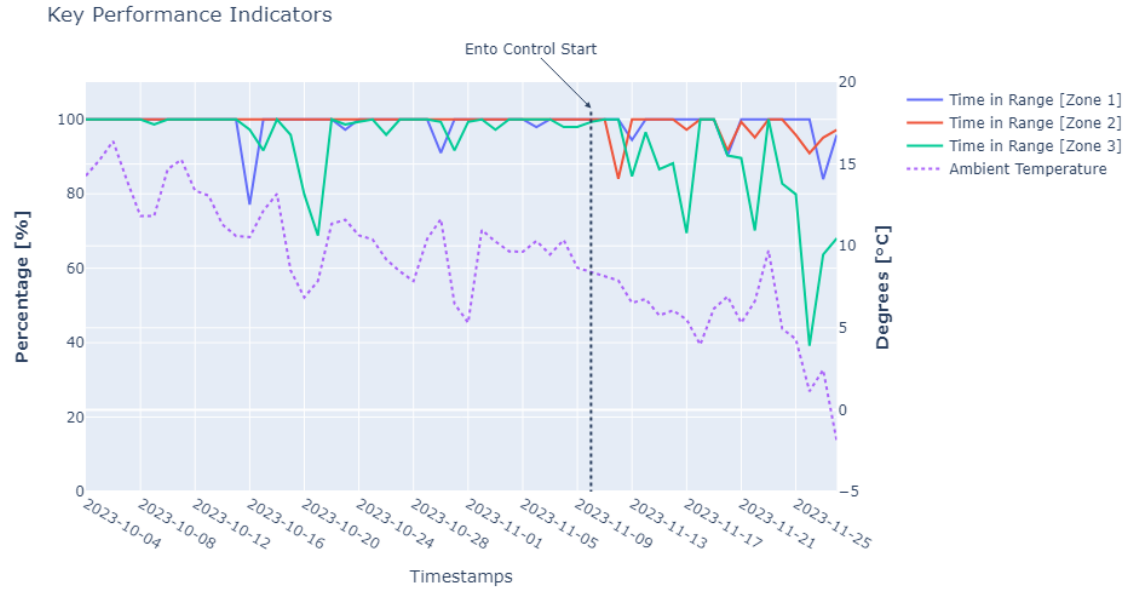


Figure 2.6. The time above r_{lower} evaluated each day, before and after implementation of new control.

Note that the heating system in *Zone 3* was known to have a capacity too small for the heating needs of the site during winter.

Chapter summary

In conclusion, the work with Ento served as a proof of concept for the feasibility of MPC controller synthesis based on data-driven models. Furthermore, this project highlighted the difficulty of assessing the performance of models during periods with significant change in the ambient conditions.

3 | Prerequisites

This chapter provides an overview of the simulation environment employed in the project, with a detailed description of each building model utilized. Additionally, a description is provided on how building performance is formalized using collected data.

3.1 Energym Environment

In this project, the Energym library is utilized for simulations of controller performance. Energym stems from the integration of two programs: EnergyPlus [17] and Modelica [18]. EnergyPlus, developed by the U.S. Department of Energy, serves as a widely-used building simulation program, while Modelica is a modeling language that excels in simulating complex cyber-physical systems, such as HVAC systems. Energym combines the capabilities of both EnergyPlus and Modelica to provide a versatile tool in the domain of building energy management, offering a comprehensive framework for simulating and fine-tuning various control strategies. Furthermore, Energym enables users to pause simulations, analyze real-time data, adjust control strategies, and resume simulations to assess the impact of changes [19]. While the option to design custom building models using third-party software exists, creating custom models is beyond the scope of this project. Therefore, the decision is made to utilize the predefined Modelica models provided within Energym. The alternative, EnergyPlus would also have sufficed and could be considered for future projects or alternative research objectives.

3.1.1 Environment Setup

Each simulation requires two key inputs: a building file and a weather file. The building file contains detailed descriptions of the building model's structural and thermal characteristics, including geometry, materials, and HVAC systems. The weather file provides meteorological data necessary for simulating real-world conditions [19]. The simulation environment operates with a default sampling interval of 5 minutes.

3.1.1.1 Building Models

To explore a variety of buildings, it is chosen to perform simulations with three different building models:

SimpleHouseRad

This simulation model is a one-zone residential building, thus treated as a single thermal zone, which is modelled based on a poorly insulated house. The zone is heated by a radiator supplied by a controllable water pump, and does not include a cooling system [20]. A schematic overview of this building model can be observed in Figure 3.1, where all factors influencing Zone temperature are highlighted including the Zone itself.

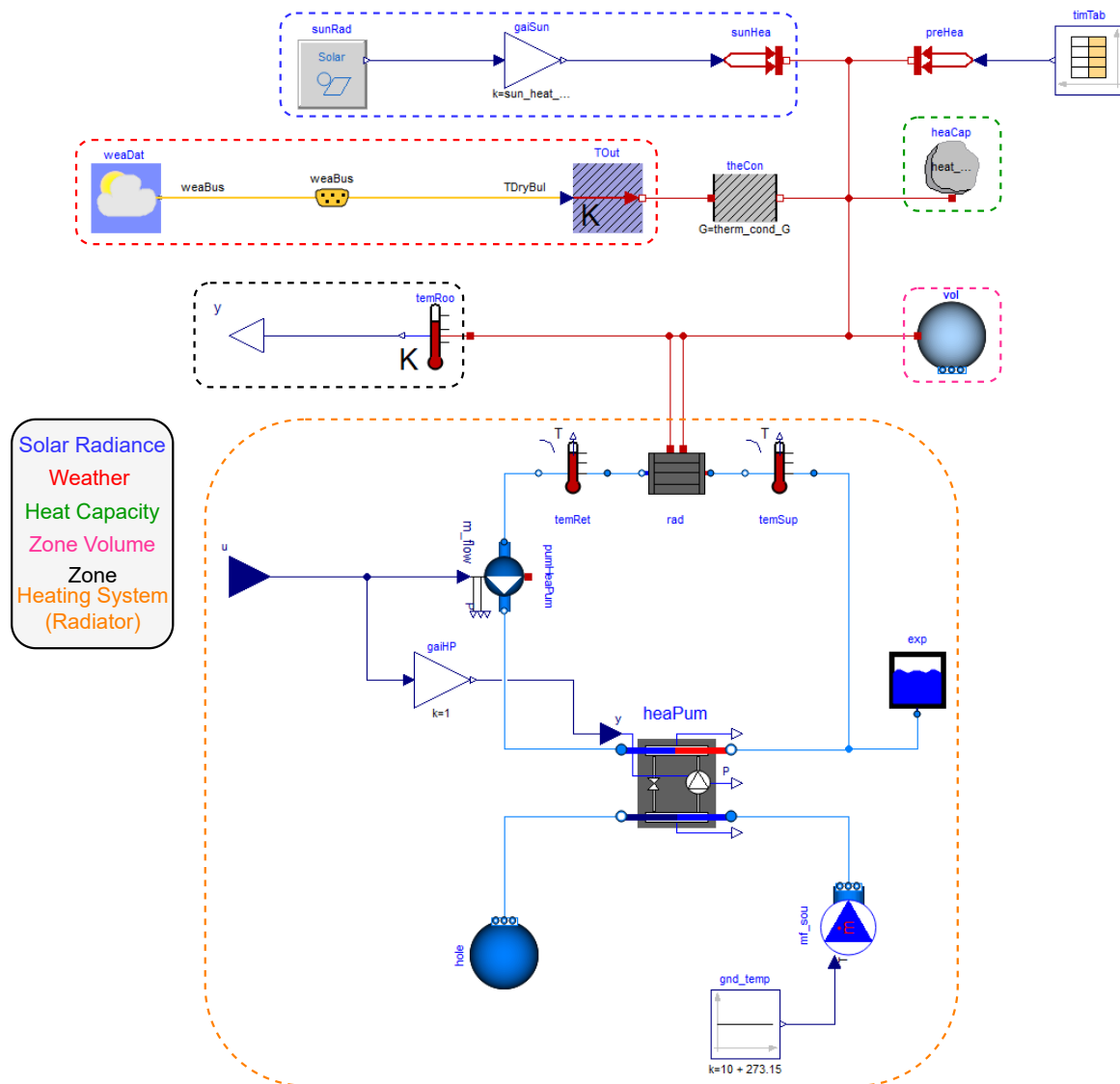


Figure 3.1. SimpleHouseRad building model schematic. Sub-systems are highlighted and linked by color with their corresponding title.

The schematic illustrates that the Zone temperature is influenced by factors such as *Solar Radiance*, *Weather*, *Zone Volume*, and the thermal energy stored in *Heat Capacity*. Central to these influences is the *Heating System* which for this model is a radiator. It raises the Zone temperature by circulating hot water supplied by a heat pump (heaPum).

SimpleHouseRSla

This simulation model is similar to the *SimpleHouseRad*, with the only distinction being the utilization of hydronic underfloor heating instead of a radiator [21]. This difference might cause the time constants related to the heating system to be slower, and therefore require a model with different capabilities. A schematic overview of the SimpleHouseRSla building model is illustrated in Figure 3.2. The schematic uses 'sla' to indicate the concrete slab where the underfloor heating is installed.

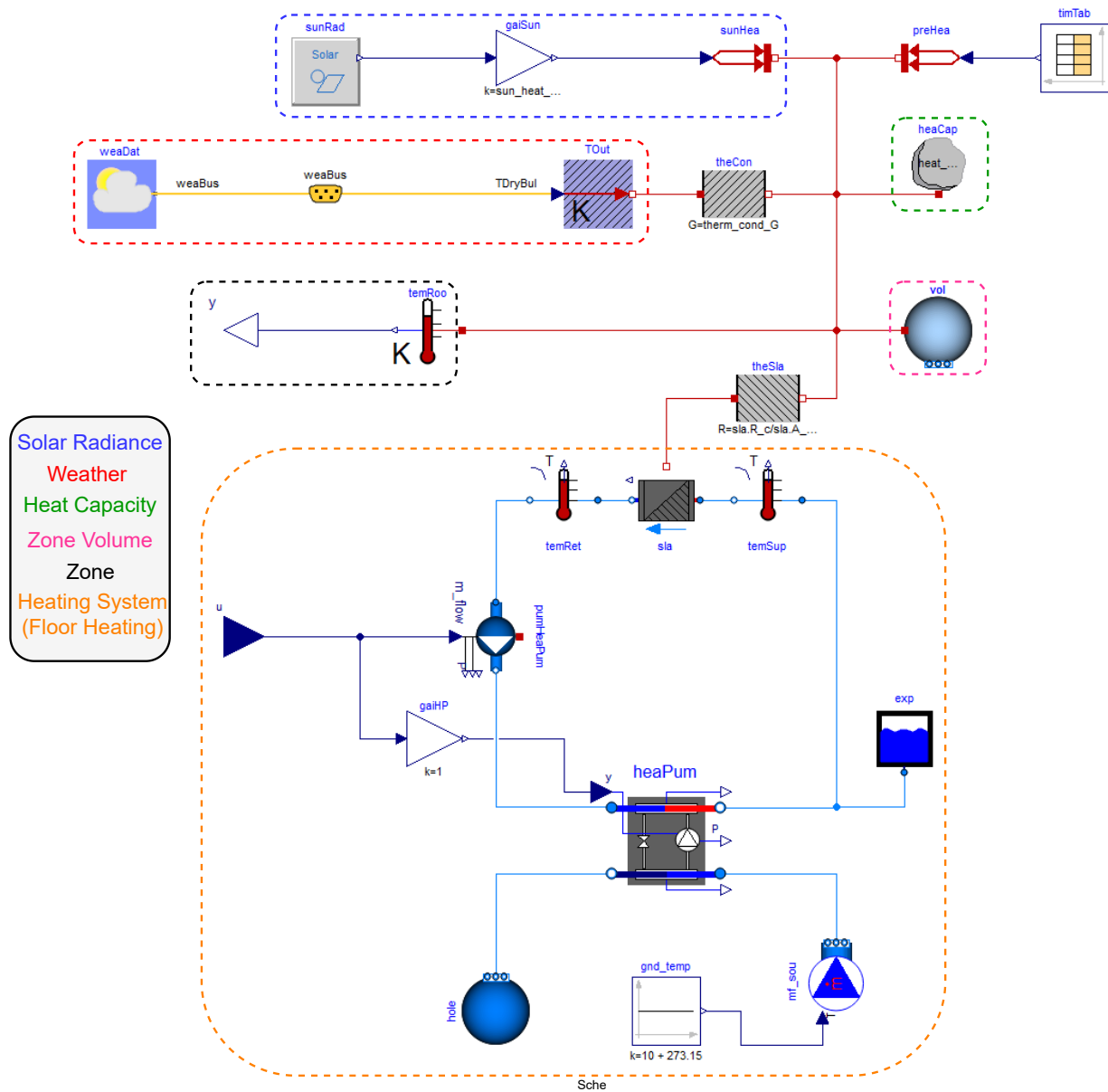


Figure 3.2. Schematic of SimpleHouseRSla with underfloor heating. Sub-systems are highlighted and linked by color with their corresponding title.

SwissHouseRSla

The two former building models represent variations of a general house approach, while this specific model is based on a Swiss "*Minergie*" house. Though it shares the same schematic as *SimpleHouseRSla*, its alignment with the Swiss Minergie house standards enforce a more energy-efficient design.

3.1.1.2 Weather

The weather file contains comprehensive meteorological data specific to the chosen location, including ambient temperature, solar radiance, wind patterns, and precipitation rates. For this project, the selected weather file is "CH_BS_Basel," representing the climatic conditions of Basel, Switzerland, to match the location of all building models.

3.2 Key Performance Indicators

As stated in Chapter 1, a goal of this project is to devise a method for automatic tuning of hyperparameters for an MPC implemented on HVAC systems. In order to do this, a method for assigning values to building performance throughout a day must be devised, since such values can be used as performance signals for the tuning algorithm. In order to provide a basis for comparing *good* and *bad* performance of the system, the chosen Key Performance Indicators (KPIs) will be described in this section. The values will be scaled to values in the single digit range, and found as per-sample values in order to enable analysis of multiple days at once. The data will be segmented such that a number of whole days is analysed, and not fractions of days. It is assumed, that the KPIs obtained will be more statistically reliable if data from several days are analyzed at once, since outliers in the ambient conditions that occur one day will have less of an impact. However, this would also increase the time between performance signals to the chosen hyperparameter tuning method, which might cause slower convergence [22]. Furthermore, weighting parameters θ will be included in order to be able to adjust the penalty assigned to each KPI. Finally, for all KPIs it is determined that a lower value indicates a better performance, in order to approach the hyperparameter tuning problem as a minimization. The notation used when formulation the KPIs in this section, is the same as the one in (2.10).

3.2.1 Reference tracking related KPIs

The two arbitrary temperature curves shown in Figure 3.3 will serve as a starting point for choosing the characteristics which are to be penalized.

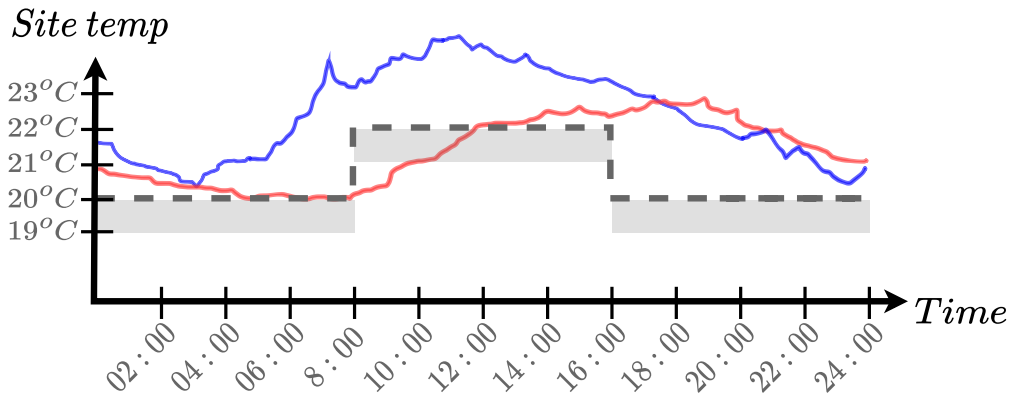


Figure 3.3. Example of two temperature trajectories. The red curve depicts a building which receives heating input too late, while the blue receives excessive heating input too early.

Time below range

As usual when analysing the performance of reference tracking control, deviations from the reference can be used for evaluation. However, when analysing performance of buildings subject to ambient influences, just finding the deviation or MSE might sometimes be misleading, for example on a hot summer day where the temperature outside the building is above the reference. Therefore, the chosen reference tracking KPI will be percentage below range (PBR).

Since the measurements included in the cost function are normalized, and the heavy penalty for points below the reference during occupied hours in (2.10) is included, a low PBR is expected of the MPC controller. To ensure this, the PBR-related KPI is defined such that a penalty value of one corresponds to a PBR of 20%. The calculation performed when determining the penalty related to this KPI for an amount of days containing V samples is shown in (3.1).

$$\text{PBR} = \left(1 - \frac{(\sum_{k=1}^V (\mathbf{r}_{lower}(k) < x(k)))}{V} - 0.8 \right) \cdot \theta_{PBR} \quad (3.1)$$

Undershoot

Additionally, a KPI related to the temperature reached in the first two hours, corresponding to o steps, during of the occupied period can be considered. The temperature measurements at these steps are collected into a dataset $V_{occupied}$ with V_o total samples. During the work with Ento described in Chapter 2, great importance was placed on these first steps of the occupied period, since this is the temperature the occupants will encounter first thing in the morning. This would serve as a particularly good indicator of the hyperparameter tuning's impact since ambient temperature usually rise throughout the occupied period. Therefore, the first steps during the occupied period is likely less correlated to the ambient effects, than the final steps. Similar considerations can be made regarding the impact of occupancy on the internal temperature, since the accumulated effects of occupancy are assumed to be more pronounced toward the end of the day. The calculation performed when determining the penalty related to this KPI is shown in (3.2).

$$\text{Undershoot} = \frac{\sum_{k=1}^{V_o} (\mathbf{r}_{lower}(k) > x(k))}{V_o} \cdot \theta_{under} \quad (3.2)$$

Overshoot

The final reference tracking consideration made relates to overshoot of the actuator. Similarly to the undershoot penalty described previously, overshooting of the internal temperature can be a good indicator of faulty hyperparameter tuning. This penalty will once again be evaluated based on the temperature during the first steps of the occupied period contained in $V_{occupied}$, where temperatures exceeding the reference will be penalized. The calculation performed when determining the penalty related to this KPI is shown in (3.3).

$$\text{Overshoot} = \frac{\sum_{k=1}^{V_o} (\mathbf{r}(k) < x(k))}{V_o} \cdot \theta_{over} \quad (3.3)$$

For an example of the implementation of the rising edge related penalties, the temperature curves shown in Figure 3.3 are now shown in Figure 3.4 with the overshoot and undershoot penalty areas highlighted.

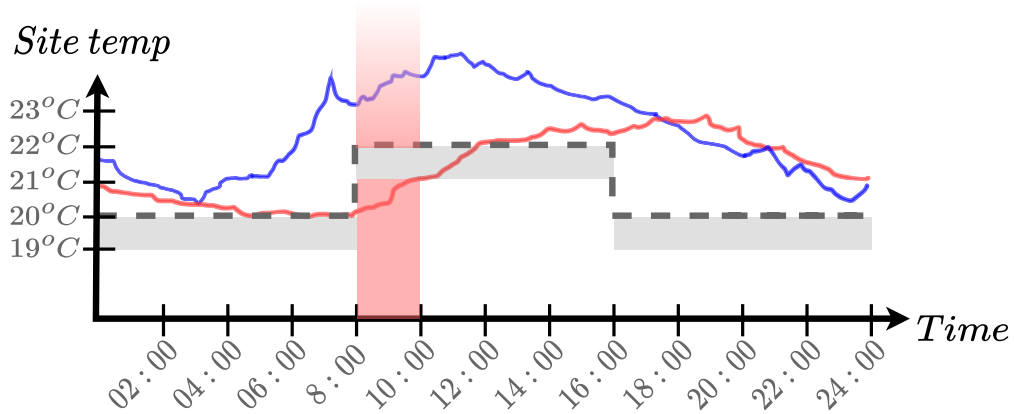


Figure 3.4. Example of the penalized areas above r and below r_{lower} during start of occupied periods, shown in red. The horizontal width of the penalized area corresponds to 2 hours.

3.2.2 Energy consumption related KPIs

As described in Chapter 1 and Chapter 2, reducing energy consumption is a priority when implementing optimal building control. However, when considering the energy consumption of a HVAC system, the ambient conditions often have a large effect on the amount of heating energy required. Therefore, a penalty on energy usage would likely be heavily correlated with ambient temperature. The described effect is illustrated in Figure 3.5

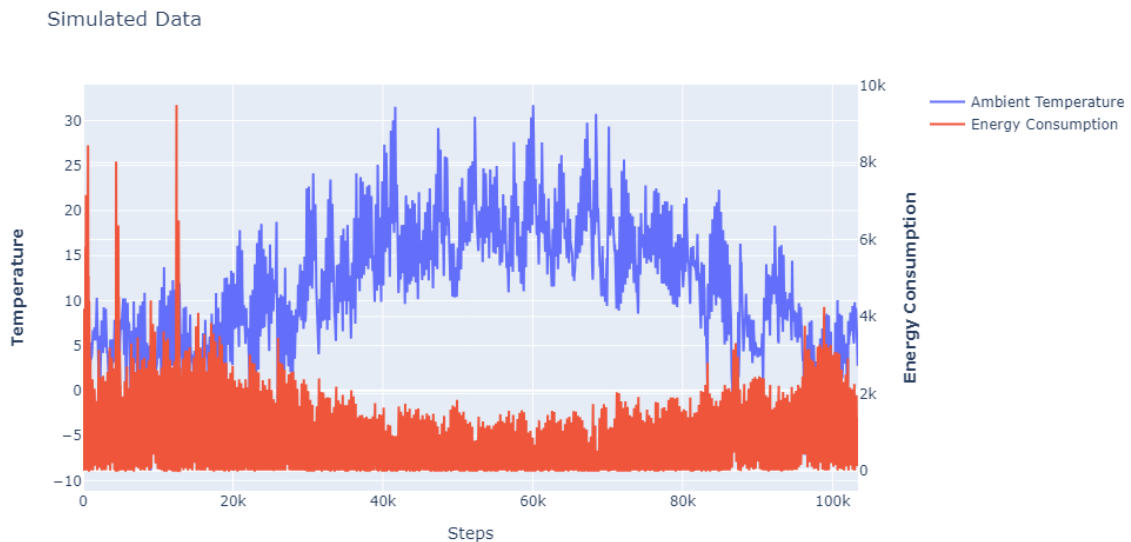


Figure 3.5. Energy consumption as controlled by MPC, plotted with ambient temperature throughout a full year.

Based on this, it is chosen to exclude this KPI. If a method of defining the KPI such that the energy use was evaluated based on the expected energy use given the ambient conditions was implemented, this KPI could be included.

Energy delta

Another consideration which can be made is regarding the rapidity of changes in the actuation, which is similar to the penalty on Δu in Section 2.3, since such behaviour will cause undesirable wear and tear on the equipment. Furthermore, rapid cycling of the heating system might cause a reduction in efficiency, due to the system not reaching steady-state operation [23]. To penalize this, the change in energy input during the days analyzed is found, and scaled down by a statistical indicator of the mean change in energy input in the training data. The value E_ρ used for scaling this penalty will be found in the training data for a given building, and based on the set of samples with non-zero changes in energy V_{delta} , as shown in (3.4)

$$E_\rho = \text{mean}(V_{delta}) + 3 \cdot \text{std}(V_{delta}) \quad (3.4)$$

The calculation performed when determining the penalty related to this KPI is shown in (3.5).

$$\text{Energy}_\Delta = \left(\sum_{k=1}^{V-1} \frac{\max(u(k+1) - u(k), 0)}{E_\rho} \right) \theta_\Delta \quad (3.5)$$

3.2.3 Summary of KPIs

Based on the discussions in this section, the four KPIs which will be utilized for hyperparameter tuning have been described. These are shown in Table 3.1.

KPI name	KPI Description	KPI Formula
PBR	Penalize deviations from no PBR	$\left(1 - \frac{(\sum_{k=1}^V \text{sum}(\mathbf{r}_{lower}(k) < x(k)) - 0.8)}{1 - 0.8} \right) \cdot \theta_{PBR}$
Undershoot	Penalize failure to reach reference at start of occupied hours	$\frac{\sum_{k=1}^{V_o} \text{sum}(\mathbf{r}_{lower}(k) > x(k))}{V_o} \cdot \theta_{under}$
Overshoot	Penalize exceeding reference at start of occupied hours	$\frac{\sum_{k=1}^{V_o} \text{sum}(\mathbf{r}(k) < x(k))}{V_o} \cdot \theta_{over}$
Energy $_\Delta$	Penalize excessive variation in energy input	$\left(\sum_{k=1}^{V-1} \frac{\max(u(k+1) - u(k), 0)}{E_\rho} \right) \theta_\Delta$

Table 3.1. List of KPIs which will be tested as penalty signals for a hyperparameter tuning system.

Chapter summary

The three buildings models used for simulation during this project have been defined to be: *SimpleHouseRad*, *SimpleHouseRSla*, and *SwissHouseRSla*. These buildings feature some variety for simulating different simple buildings. Furthermore, the KPIs which will be used for hyperparameter tuning in this project have been formulated in Table 3.1.

4 | Project Shaping

This chapter will outline the objective of this project by detailing the problem statement, requirements, and finally the tests carried out to verify whether the requirements are met.

4.1 Delimitation

The development and testing of the designed system in this project will be done within the simulated environment described in Section 3.1. However, the simulations only contain buildings consisting of a single zone, and therefore multi-zone control is not investigated. While such an investigation might be valuable, a single-zone modelling and control approach is also applicable in many cases, where only a single sensor and heating system is present in a building. These types of setups were commonly seen in Ento's costumer portfolio. Furthermore, it has been decided not to include occupants in the modeling of this system. Occupants affect the temperature both through their presence and their actions, such as opening windows. Due to the difficulty of predicting human behaviour, a model for this would have to be probabilistic in nature. Implementing this would require formulating a Stochastic Model Predictive Control (SMPC), which involves probability constraints that allow for a trade-off between control performance and the likelihood of violating constraints in a stochastic setting [24]. Additionally, the impact of occupants is observed to be greatest later in the day, usually from noon onward. Therefore, it has been decided to exclude the modeling of occupants.

4.2 Problem Statement

How can the data-driven building modelling for MPC described in Chapter 2 be further developed, and how can performance indicators be leveraged for online tuning of MPC hyperparameters?

4.3 Requirements

The requirements for this project is split into two subparts; modelling and hyperparameter tuning.

Modelling requirements

Compared to the original nRnC model introduced in Chapter 2, the developed model(s) must:

- | | |
|---------------|---|
| Req. 1 | demonstrate improved predictive performance when used on training data |
| Req. 2 | demonstrate improved key performance indicator values when used in building simulation with MPC |

Hyperparameter tuning requirement

The implementation of automatic tuning of hyperparameters must:

- Req. 3** demonstrate improved key performance indicator values when compared with the same system utilizing constant hyperparameter values during building simulation with MPC

4.4 Accepttest

With the requirements described in Section 4.3, tests can now be defined to evaluate whether the designed system meets these requirements. However, first some key definitions are described.

Prediction performance evaluation

The predictive performance of the models will be assessed using a method similar to the training of the nRnC model described in Chapter 2. Each step in the dataset will serve as an initial condition, and the model will predict n_{ps} samples ahead, such that a 3 hour prediction is made from each initial condition. However, instead of summarizing the squared cost across all initial conditions and predictions as done during training, only the final prediction will be measured as the Mean Absolute Error (MAE). The formula for obtaining the MAE is shown in (4.1)

$$\frac{\sum_{i=0}^{N_{Sim}-n_{ps}} \left| \hat{T}_{int_i+n_{ps}} - T_{int_i+n_{ps}} \right|}{N_{Sim} - n_{ps}} \quad (4.1)$$

Where,

$$\begin{array}{l|l} N_{Sim} & \text{Total number of samples used during training} \\ n_{ps} & \text{Number of predicted samples ahead from initial condition} \end{array}$$

Thus, the final cost will represent the prediction error 3 hours ahead for each initial condition. Furthermore, the tests will be carried out once for each of the building models described in Section 3.1.

Simulation length

The simulation length will be consistent for all acceptance tests performed for all requirements, with each simulation lasting half a year. Only half a year is used, since the second half of the year would mirror the first due to the periodicity of the seasons, and therefore contribute little additional information. Furthermore, the first two days of each simulation will be discarded due to initial conditions interfering with the overall performance of the simulations.

Evaluation of KPIs

For Req. 2 and Req. 3 the success criteria is based on having improved KPI values. These KPI value are listed in Table 4.1 with a description of each. Note that these KPIs are different from the ones described in Section 3.2, since the KPIs described in that section were specifically designed as performance signals for hyperparameter tuning.

KPI	Description
PIR	Percentage of points during the simulation which are in the range of $\pm 1^\circ$ of the internal temperature reference.
PBR	Percentage of points during the simulation which are below r_{lower} .
Energy	Total energy consumed during simulation.
Energy $_{\Delta}$	Indicator of the changes in energy input.
Energy $_{\Delta}^2$	Indicator of the changes in energy input, where outliers are emphasized.

Table 4.1. Description of KPIs used for evaluating the performance of the MPC.

With these definitions made, the acceptance tests can be described.

4.4.1 Req. 1

To evaluate whether the requirement '*The model must demonstrate improved predictive performance when used on training data*' is met by the system, the following procedure will be tested.

Procedure

1. Train the developed models and the original nRnC model on building training data
2. Use each sample as initial condition and predict 3 hours ahead for each initial condition.
3. Compute the average MAE for the last prediction from each initial condition.

Criteria of success

The average MAE for the models must be lower than the original nRnC.

4.4.2 Req. 2

To evaluate whether the requirement '*The model must demonstrate improved key performance indicator values when used in building simulation with MPC*' is met by the system, the following procedure will be tested.

Procedure

1. Train and run the MPC simulation for all models with constant hyperparameters.
2. Compute the KPIs in Table 4.1 for each model

Criteria of success

The KPIs for both the models must be smaller than those of the original nRnC.

4.4.3 Req. 3

To evaluate whether the requirement '*The implementation of automatic tuning of hyper parameters must demonstrate improved key performance indicator values when compared with the same system utilizing constant hyperparameter values during building simulation with MPC*' is met by the system, the following procedure will be tested.

Procedure

1. Train and run the MPC simulation for the developed models with online hyperparameter tuning.
2. Compute the KPIs in Table 4.1 for each model

Criteria of success

The KPIs for both the improved nRnC and PINN models' KPIs are to be smaller than those of the original nRnC.

Part II

Development

5.2 Updated cost function

As mentioned in Section 4.2, model predictive control has been chosen for managing internal temperature in this project. To achieve this, the cost function introduced in (2.10) is being re-employed with some adjustments. It is decided only to penalize changes in energy input when a rise in energy input occurs, instead of penalizing any change. The penalization of rapid changes in energy input was originally introduced, due to considerations regarding the potential damage short-cycling can cause to actuators [23]. Therefore, $\Delta \hat{\mathbf{u}}$ is introduced to mitigate such risks. However, it has been observed that including this penalty during the time of the day where the actuator is turned down leads to higher energy consumption than necessary, since the cost of changing the actuation offsets some of the cost of superfluous energy input. The penalization would still make short-cycling expensive without this penalization on lowering energy input. Therefore, the penalization term $\Delta \hat{\mathbf{u}}$ in (5.1d) is eliminated for reductions in energy consumption. This is implemented by employing the 'max' function to return the highest of either 0 or $\Delta \hat{\mathbf{u}}$, since a decrease in energy input would map to 0 and disable the penalization of the term. The cost function with this change is shown in (5.1)

$$J(k) = \sum_{i=1}^{H_p} |\hat{\mathbf{x}}(k+i|k) - \mathbf{r}(k+i)| \cdot R \quad (5.1a)$$

$$+ \sum_{i=1}^{H_p} |\min(\hat{\mathbf{x}}(k+i|k) - \mathbf{r}_{lower}(k+i), 0)| \cdot R \cdot \xi \quad (5.1b)$$

$$+ \sum_{i=0}^{H_p-1} |\hat{\mathbf{u}}(k+i|k)| \cdot C \quad (5.1c)$$

$$+ \sum_{i=0}^{H_p-2} (\max(0, \hat{\mathbf{u}}(k+i+1|k) - \hat{\mathbf{u}}(k+i|k)))^2 \cdot C_{\Delta} \quad (5.1d)$$

where,

Symbol	Unit	Description
R	[.]	Penalization weight of the error between the system trajectory to the reference
\mathbf{r}_{lower}	[°C]	Lower reference boundary
ξ	[.]	Penalty weight scaling variable for breaching the lower reference
\mathbf{u}	[W]	Control input
C	[.]	Penalization weight of the control input
C_{Δ}	[.]	Penalization weight of the change in control input

5.3 Sampling Frequency

As mentioned in Section 3.1.1, the simulation environment utilizes a sampling frequency of 5 minutes. In the context of building temperature control, 5 minutes is a relatively short amount of time, and therefore the possibility of a lower sampling frequency is investigated. Decreasing the sampling frequency decreases the required computational power for MPC, but also lessens the speed at which the MPC can react to disturbances. A 2022 paper by Zhe Wang [25] addresses the trade-off between sampling frequency and unnecessary waste of resources by investigating the optimal sampling frequency for monitoring indoor temperature. It is found that any sampling period shorter than 30 minutes achieves similar performance. In Section 3.1.1 it is stated that the three buildings simulated in the Energym environment contain different heating systems, and consequently the time required to heat up the building varies from building to building [26]. Thus, to accommodate for this variation in systems a balanced sampling frequency of 15 minutes is chosen. However, the inability to modify the sampling frequency directly in the simulation environment prevent an easy implementation of this decision. To solve this issue, every N 'th sample of the data set, generated with a sampling frequency of 5 minutes, is utilized. Hence, to achieve a sampling frequency of 15 minutes, every third sample is used. The choice of allowing a difference in sampling periods between the simulations and models of the buildings cause some unforeseen effects. The difference in sampling frequencies results in a temporal resolution disparity. As a consequence of this, some emergent dynamics might be introduced to the models of the buildings, since these are trained with longer sampling period. For example these emergent dynamics might arise, if the impact of a constant energy input over three simulation steps cause some nonlinear behavior, which could be caused by thermal inertia in the heating system or building thermal mass. This phenomenon is illustrated in Figure 5.2.

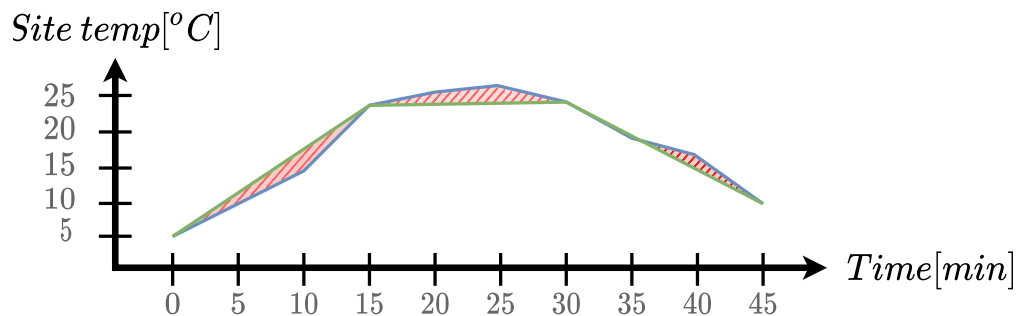


Figure 5.2. The blue trace illustrates a sampling frequency of 5 minutes, while the green trace illustrates a sampling frequency of 15 minutes. The red shaded area marks the mismatch.

5.4 Data Generation

In order to utilize the simulation tool within the context of the objectives outlined in Section 5.1, historical data for training the models is required. To achieve this, a similar method to the one described in Section 2.4 is used, where the simulation is run for a full year with reactive control to simulate a building with some sort of controlled heating system. For this purpose, a P controller with a gain of 0.3 and sampling period of 15 minutes is implemented. Furthermore, the temperature reference used for the proportional controller will be changed several times during the year, in order to generate training with varied scenarios. The temperature references for both occupied and non-occupied hours are generated by choosing two numbers; a and b . The value of a can be interpreted as a baseline value, from which the occupied temperature reference value is taken as $a + b$ and the un-occupied temperature reference value is taken as $a - b$. The timing of the occupied hours remains consistent at [08 : 00 – 16 : 00] across all temperature reference variations. An example of such a reference is provided in Figure 5.3.

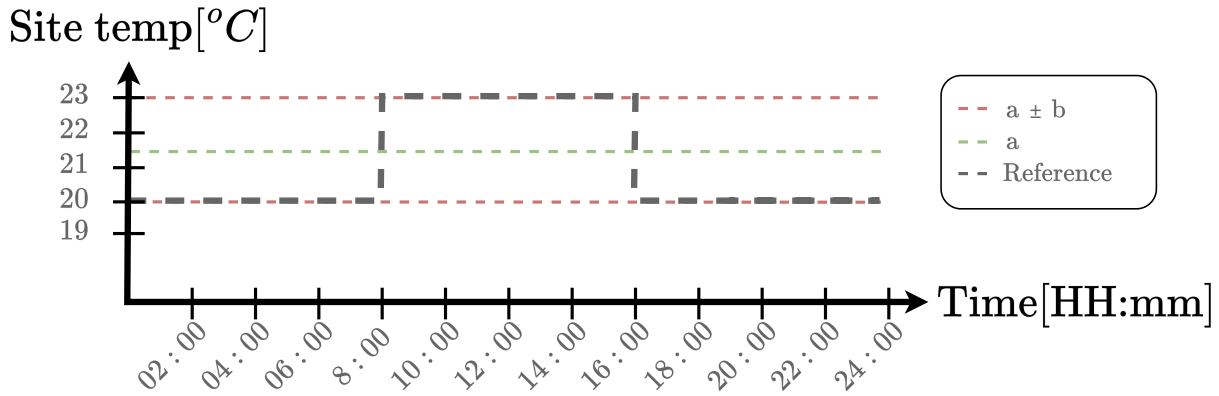


Figure 5.3. Example of temperature reference with a mean of 21.5 and a deviation of 1.5.

The sequence of temperature references is equal for all buildings, meaning that the sequence is only chosen once, and then used in all building data simulations. For all training data simulations in this project, the sequence of values a and b are selected beforehand, where the next pair of a and b values in the sequence are used once every 30 days. The sequence of temperature references for the variable a is selected within predefined range:

$$a = \{19, 20, 21, 22, 23, 24\}$$

Similarly, for b , the sequence is selected within the predefined range:

$$b = \{1, 2\}$$

5.5 Data Scaling

The oscillations observed when running the MPC with the model described in Figure 2.2 are a possible case for improvements. Further investigation reveals that the oscillations observed in Figure 2.2 are caused by the method chosen for scaling the data, which maps the data to values near zero by assuming a normal distribution with mean zero and a standard deviation of one. This scaling method is generally suitable for neural networks, where the sign of numbers is not crucial. However, it proves problematic for the linear nRnC model, which is sensitive to the signs of the data. To address this, the scaling method is switched from standardization to simply scaling the data with a scalar, in order to maintain the original sign of the data. One of the most straightforward scaling methods is to divide all the samples of a given feature by the numerically biggest value observed for that feature, ensuring that the highest value of any data point is either one or minus one. However this approach is sensitive to spikes in the data, and therefore a statistical approach is chosen instead, where the "maximum" value of a feature $x_{maximum}$ is found from the feature mean μ_x and feature standard deviation σ_x as

$$x_{maximum} = |\mu_x| + 3 \cdot \sigma_x \quad (5.2)$$

The scaling method then becomes as shown in (5.3).

$$x_{scaled} = \frac{x}{x_{maximum}} \quad (5.3)$$

To investigate if this change removed the oscillating behavior, a test simulation of the nRnC model with the new scaling method is shown in Figure 5.4.



Figure 5.4. Validation of new scaling method on the nRnC model in the MPC.

As observed in Figure 5.4, the oscillation observed in Figure 2.2 are no longer present, which indicates that the new method for scaling the data alleviates the issue.

Chapter Summary

In this chapter, the disparate parts of the system which are to be designed have been laid out, and their interconnections have been visualised in the diagram shown in Figure 5.1. Furthermore, a description of the method used for generating training data has been given, alongside the chosen scaling method used to prepare the data for model training. The description of the system and training data will serve as a starting point for the further development in this project.

6 | nRnC

In Chapter 2 the nRnC model developed in collaboration with Ento is described. Based on evaluations of the MPC performance achieved with this model, as well as further considerations regarding the training and structure of this model, several changes have been made which will be described in this chapter.

6.1 Collection of denominator terms

In some terms of the nRnC model described in (2.1), the thermal capacitance of the building C , resistance value R , specific heat capacity c and mass m were included. However, the c and m terms were only included due to considerations related to units, but never assigned any meaningful values. In practice, the parameter estimation simply estimated the product of all these terms. For simplicity, a unique τ parameter is therefore substituted for all the terms in the denominator of each of the different fractions in (2.1), resulting in (6.1).

$$\frac{dT_{int}}{dt} = \frac{T_{amb} - T_{int}}{\tau_{amb}} + \frac{W_{sun}}{\tau_{sun}} + \frac{W_{heating}}{\tau_{heating}} \quad (6.1)$$

6.2 Consideration of inertial effects

While the C parameter was initially incorporated into the nRnC model to represent the building's inertia, it became evident that relying solely on this term for capturing inertia was based on flawed reasoning. This is even more evident now with the inclusion of the τ parameters in (6.1). Another attempt is made at modelling inertial effects, by including terms that introduce scaled versions of the recent outputs, thereby giving the model knowledge of the recent temperature trajectory. The effect of including delayed outputs is investigated in Appendix A, where it is concluded that this inclusion raises the order of the nRnC model to be equal to the number of delayed inputs included. By incorporating this change, (6.1) becomes (6.2)

$$\frac{dT_{int}}{dt} = \frac{T_{amb} - T_{int}}{\tau_{amb}} + \frac{W_{sun}}{\tau_{sun}} + \frac{W_{heating}}{\tau_{heating}} + \sum_{i=0}^{\pi} \frac{\frac{dT_{int_{k-i}}}{dt}}{\tau_i} \quad (6.2)$$

With this addition, the minimization problem initially described for the standard nRnC model in (2.2) is redefined as (6.3).

$$\underset{\tau_{amb}, \tau_{sun}, \tau_{heating}, \tau_i}{\text{minimize}} \quad \sum_{k=0}^{N-n_{ps}} \left(\sum_{j=1}^{n_{ps}} \left(\|T_{int_{k+j}} - \hat{T}_{int_{k+j}}\|_2^2 \right) \right) \quad (6.3)$$

$$\text{subject to} \quad \frac{dT_{int_{k+j}}}{dt} = \frac{T_{amb_{k+j}} - T_{int_{k+j}}}{\tau_{amb}} + \frac{W_{sun_{k+j}}}{\tau_{sun}} + \frac{W_{heating_{k+j}}}{\tau_{heating}} + \sum_{i=0}^{\pi} \frac{\frac{dT_{int_{k+j-i}}}{dt}}{\tau_i} \quad (6.4)$$

$$T_{int_{k+j+1}} = T_{int_{k+j}} + \Delta t \cdot \frac{dT_{int_{k+j}}}{dt} \quad (6.5)$$

Having access to an unlimited number of delayed outputs ($\pi = \infty$) would be ideal, however, the computational overhead of including every delayed output would probably become infeasible. Therefore it is chosen to dynamically choose the amount of delayed outputs included, based on the improvement in the fit to training data gained for every extra delayed output included. In a paper by Kasper Vinther et. al. it is shown that two delayed outputs often captured the majority of the effects of inertia in the system[15]. However, it is crucial to account for the diversity among building types, therefore it is chosen to test up to a maximum of $\pi_{max} = 4$ delayed outputs.

6.3 Parameter constraints

Furthermore, to avoid overfitting, it is decided to enforce a constraint that ensures the impact of past outputs lessens over time, such that newer outputs always have a higher impact on the model. Furthermore, values of τ_1 below 1 are also prohibited, due to the stability results obtained in Appendix A. The constraints are shown in (6.6) and (6.7)

$$1 < \tau_1 < \tau_2 < \tau_3 < \tau_4 \quad (6.6)$$

$$0 < \tau_{amb}, \quad 0 < \tau_{sun}, \quad 0 < \tau_{heating} \quad (6.7)$$

6.4 Training and selecting models

With the additions to the minimization problem and constraints on the τ values established, the nRnC model is trained with the code shown in Snippet 6.1, which is an adjusted version of the code shown in Snippet 2.1.

```

1  # Soft constraint for tau values
2  if tau_amb < 0:
3      return 1e12
4  if tau_q < 0:
5      return 1e12
6  if tau_sun < 0:
7      return 1e12
8  if any(tau < 1 for tau in tau_delayed):
9      return 1e12
10 if pi > 1:
11     if np.min(np.diff(tau_delayed[:pi]))<0:
12         return 1e12
13
14 for i in range(1, (len(training_data) - n_ps)):
15     #Initialization for each step:
16     T_predicted = np.zeros(n_ps)
17     T_int = T_data[i - 1]
18     Tamb_Tint_temp = Tamb_Tint[i - 1]
19     delayed_output_array_temp = delayed_output_array.iloc[i - 1].values
20
21     #Predict n_ps steps ahead
22     for j in range(n_ps):
23         # Compute change in temperature:
24         dTdt = r1c1_model(tau_amb, tau_q, tau_sun, Tamb_Tint_temp, Q[i + j - 1], Sun[i + j
25             - 1], delayed_output_array_temp, tau_delayed, pi)
26
27         # Compute the next temperature
28         T_predicted[j] = normalize(denormalize(T_int, max_x) + denormalize(dTdt, max_y),
29             max_x)
30
31         # Update parameters:
32         T_int = T_predicted[j]
33         Tamb_Tint_temp = normalize(denormalize(T_amb[i + j], max_d[0]) - denormalize(T_int,
34             max_x), max_d[2])
35         delayed_output_array_temp = np.insert(delayed_output_array_temp[:-1], 0, dTdt)
36
37 cost += np.sum((T_data[i:(i + n_ps)] - T_predicted)**2)

```

Snippet 6.1. Modification of the nRnC training to accommodate delayed outputs as input.

The modification introduce a number of delayed outputs into the nRnC model, according to the value π . The delayed outputs are stored in *delayed_output_array_temp* and their associated τ values are stored in *tau_delayed*. The nRnC model function *r1c1_model* includes the delayed outputs into the calculation of $dTdt$ as shown in Snippet 6.2.

```

1  def r1c1_model(tau_amb, tau_q, tau_sun, Tamb_Tint, Q, Sun, delayed_output_array,
    tau_delayed, pi):
2      # Original nRnC model equation:
3      dTdt = (Tamb_Tint / R_amb) + (Q / R_q) + (Sun / R_sun)
4
5      # Adding delayed outputs based on the chosen pi:
6      for i in range(pi):
7          dTdt += delayed_output_array[i] / tau_delayed[i]
```

Snippet 6.2. nRnC model equation with integrated Inertia.

An algorithm has been designed to automatically train five different models with zero to four delayed outputs, and afterwards chose the model which performs sufficiently well with fewest delayed outputs. The steps involved in this mechanism is provided in the following list, along with a code snippet of the implementation of each step.

1. **Train five nRnC models with an incrementally increasing number of delayed outputs included:** Train a series of five models, where each model should incorporate an increasing number of delayed outputs corresponding to its π value. A code snippet demonstrating the implementation of this step is provided in Snippet 6.3.

```

1      for i in range(pi_max):
2          min_Loss, tau_amb, tau_heating, tau_sun, tau_delayed =
            nRnC_model_trainer(training_data, n_ps, i)
3
4          # Append the minimum losses
5          min_Loss_list.append(min_Loss)
```

Snippet 6.3. Step 1: Training models.

2. **Determine a Performance Threshold:** After training the models, identify the model with the lowest loss (best performance) and compute a threshold as 110% of this lowest loss. This threshold serves as a performance cut-off for selecting a model which balances complexity and prediction performance

```

6      best_loss = min(min_Loss_List)
7      threshold = best_loss * 1.10
```

Snippet 6.4. Step 1: Define Threshold.

3. **Select a model:** Review each model, starting from lowest to highest number of delayed outputs, and select the first model where the loss does not exceed the threshold.

```

8     # Iterate through models from lowest order to highest
9     for i, loss in enumerate(min_loss_values):
10        # Check if threshold is obliged and exit if true
11        if loss <= threshold:
12            pi = i
13            break

```

Snippet 6.5. Step 1: Choosing a model.

6.4.1 Evaluation

To evaluate the effectiveness of this feature, a test is conducted to compare the performance across all π 'th order nRnC models for all building models introduced in Section 3.1.1. The performance is evaluated based on the MAE test described in (4.1). For each building model, each nRnC model will be fed identical data, ensuring that any observed differences in effectiveness are directly caused by the inclusion of delayed outputs. The obtained results of these tests are seen in Figure 6.1, Figure 6.2, and Figure 6.3 with their corresponding training durations in Table 6.1, Table 6.2, and Table 6.3, respectively.

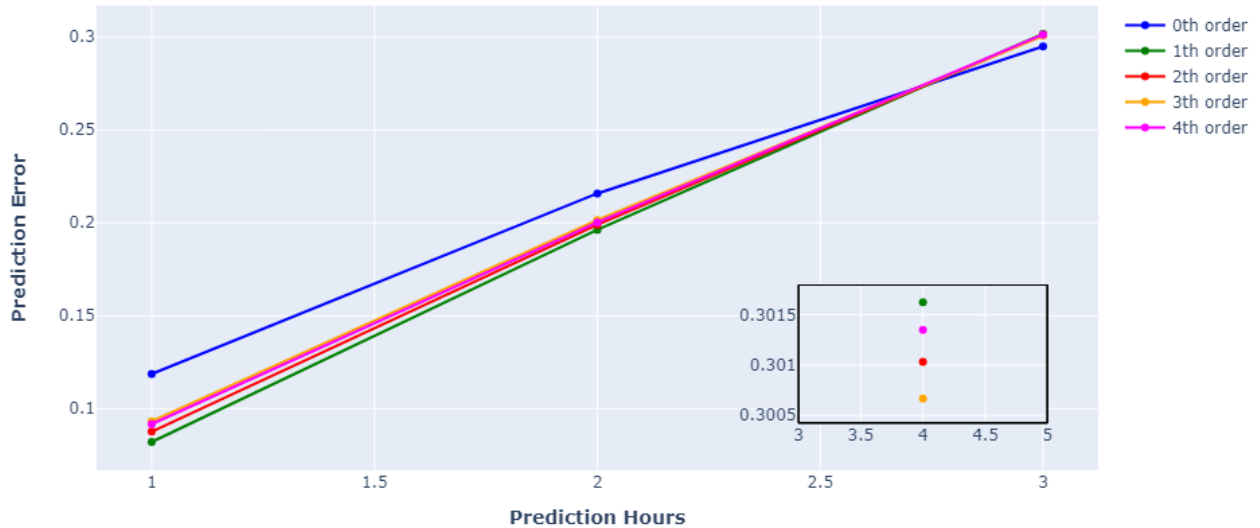


Figure 6.1. Performance test for *SimpleHouseRad* of π 'th order nRnC models over a 3-hour prediction horizon. This simulation, with a threshold of 110%, determined the optimal model order to be 1.

Model Order	Training duration [HH:mm:ss]
0	00:02:04
1	00:05:50
2	00:06:10
3	00:06:12
4	00:06:10

Table 6.1. Training duration for each nRnC model order tested on *SimpleHouseRad*

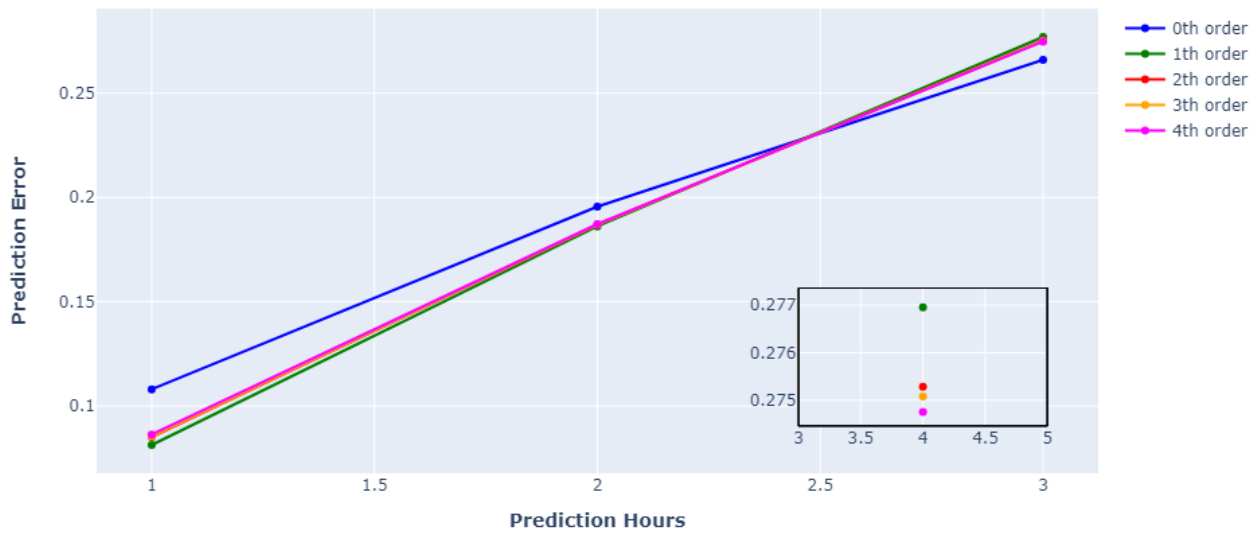


Figure 6.2. Performance test for *SimpleHouseRSla* of π 'th order nRnC models over a 3-hour prediction horizon. This simulation, with a threshold of 110%, determined the optimal model order to be 1.

Model Order	Training duration [HH:mm:ss]
0	00:02:04
1	00:05:10
2	00:04:21
3	00:05:30
4	00:06:14

Table 6.2. Training duration for each nRnC model order tested on *SimpleHouseRSla*

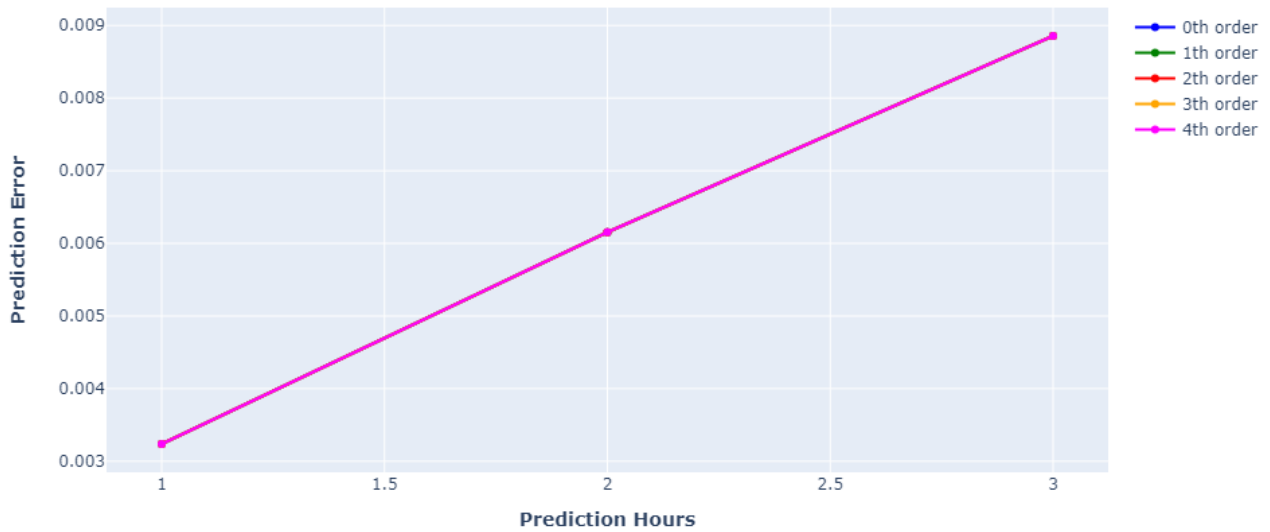


Figure 6.3. Performance test for *SwissHouseRSla* of π 'th order nRnC models over a 3-hour prediction horizon. Note that all traces overlap with each other.

Model Order	Training duration [HH:mm:ss]
0	00:01:44
1	00:06:08
2	00:06:10
3	00:06:11
4	00:06:15

Table 6.3. Training duration for each nRnC model order tested on *SwissHouseRSla*

As seen in Figure 6.1 and Figure 6.2, both simulations determined their optimal model order to be 1. This confirms that the inclusion of inertia enhances the predictive capability of the nRnC model for those building types. However, in Figure 6.3, the third building model exhibits no significant difference between orders 0 to 4, indicating that for certain building types, the inclusion of inertia may not be beneficial.

6.5 Increasing training efficiency

As outlined in Chapter 2, the nRnC model is trained by predicting n_{ps} steps ahead from each data point. As the amount of training data N_{Sim} increases, this approach quickly becomes computationally intensive. To avoid this, the possibility of implementing a method which is less computationally intensive, while remaining similar to the training procedure, is investigated. The proposed training procedure involves subset of the of samples from the training data, which will serve as the initial conditions from which the n_{ps} -step prediction ahead is made. The samples will be chosen uniformly, such that every sample is spaced evenly wrt. the closest other drawn samples. The total number of samples involved in the computation N_{Train} is shown in (6.8).

$$N_{Train} = N_{Sim} \cdot n_{ps} \cdot P_{init} \quad (6.8)$$

Where,

$$n_{ps} = \frac{m_p}{m_s}$$

N_{Train}	Total number of samples used during training
N_{Sim}	Number of samples in simulated data
P_{init}	Percentage of original training data used as initial conditions
n_{ps}	Number of predicted samples ahead from initial condition
m_p	Minutes to predict ahead
m_s	Minutes per sample

The selection of P_{init} is crucial, as it significantly impacts the training process. This variable represents the percentage of training data to be utilized, and can equivalently be interpreted as employing every $\frac{1}{P_{init}} = H$ 'th sample. Due to the decision of spacing the initial condition samples evenly, an improper selection of P_{init} , and therefore H , will degrade the training process. This is the case when H is chosen as a linear combination of daily sample count and any whole number. The degradation occurs, because H aligns with the daily cycle, resulting in a phenomenon observed on the left side of Figure 6.4, where the model's knowledge is restricted to only use the times 12 : 00 and 06 : 00 as initial conditions. Consequently, the model training will never use data points that are not in the intervals after the chosen initial conditions. If H is chosen properly this phenomenon is avoided, as shown on the right side of Figure 6.4, which ensures that the training process gradually covers an entire day.

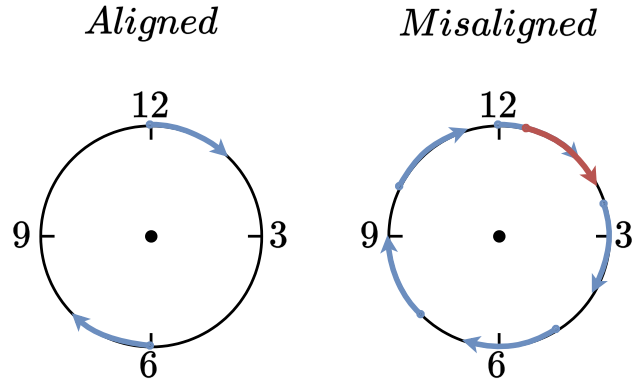


Figure 6.4. Daily cycle with a number of predictions of 90 minutes marked as arrows. Left side shows a 12 hour sampling interval, where the right side shows a 2.5 hour sampling interval.

To prevent this phenomenon, the constraints shown in (6.9) are introduced for H .

$$H = \frac{1}{P_{init}} \neq \begin{cases} S_{pd} \cdot K_1 & | K_1 \in \{\mathbb{W} | n_{ps} < S_{pd}\} \\ \frac{S_{pd}}{K_2} & | K_2 \in \{\mathbb{W} | K_2 \leq \frac{S_{pd}}{n_{ps}}\} \end{cases} \quad (6.9)$$

Where,

S_{pd} | Number of samples per day

With this implementation, the minimization problem is once again redefined as (6.10), building upon the previous redefinition in (6.3), to incorporate these changes.

$$\underset{\tau_{amb}, \tau_{sun}, \tau_{heating}, \tau_i}{\text{minimize}} \quad \sum_{k=0}^{\frac{N-n_{ps}}{H}} \left(\sum_{j=1}^{n_{ps}} \left(\|T_{int(k \cdot H)+j} - \hat{T}_{int(k \cdot H)+j}\|_2^2 \right) \right) \quad (6.10)$$

$$\text{subject to} \quad \frac{dT_{int(k \cdot H)+j}}{dt} = \frac{T_{amb(k \cdot H)+j} - T_{int(k \cdot H)+j}}{\tau_{amb}} + \frac{W_{sun(k \cdot H)+j}}{\tau_{sun}} + \frac{W_{heating(k \cdot H)+j}}{\tau_{heating}} \quad (6.11)$$

$$+ \sum_{i=0}^{\pi} \frac{\frac{dT_{int(k \cdot H)+j-i}}{dt}}{\tau_i}$$

$$T_{int(k \cdot H)+j+1} = T_{int(k \cdot H)+j} + \Delta t \cdot \frac{dT_{int(k \cdot H)+j}}{dt} \quad (6.12)$$

The code structure for this implementation is shown in Snippet 6.6, demonstrating the creation of an array containing uniformly chosen indices within the training dataset. These indices are then used, when training the nRnC model, to selectively utilize samples from the training dataset.

```

1 # Compute upper bound:
2 N = len(training_data) - n_ps
3
4 # Number of samples to draw:
5 nPointsDrawn = math.floor(p_init * len(training_data))
6
7 # Draw nPointsDrawn numbers from a uniform distribution within the range [1, N]:
8 startingPoints = np.roun(np.linspace(1, N, nPointsDrawn)).astype(int)

```

Snippet 6.6. Creation of uniformly chosen indices array

Here, p_{init} is the fraction of the training data used as initial conditions. Furthermore, *startingPoints* is the array containing the indices chosen to be used as initial conditions. Implementing this into the code shown in Snippet 6.1 results in the code shown in Snippet 6.7.

```

1 for i in range(nPointsDrawn):
2     #Initialization for each step:
3     T_predicted = np.zeros(n_ps)
4     T_int = T_data[startingPoints[i] - 1]
5     Tamb_Tint_temp = Tamb_Tint[startingPoints[i] - 1]
6     delayed_output_array_temp = delayed_output_array.iloc[startingPoints[i] - 1].values
7
8     #Predict n_ps steps ahead
9     for j in range(n_ps):
10        # Compute change in temperature:
11        dTdt = r1c1_model(tau_amb, tau_q, tau_sun, Tamb_Tint_temp, Q[startingPoints[i] + j
12                           - 1], Sun[startingPoints[i] + j - 1], delayed_output_array_temp, tau_delayed, pi)
13
14        # Compute the next temperature
15        T_predicted[j] = normalize(denormalize(T_int, max_x) + denormalize(dTdt, max_y),
16                                   max_x)
17
18        # Update parameters:
19        T_int = T_predicted[j]
20        Tamb_Tint_temp = normalize(denormalize(T_amb[startingPoints[i] + j], max_d[0]) -
21                                   denormalize(T_int, max_x), max_d[2])
22        delayed_output_array_temp = np.insert(delayed_output_array_temp[:-1], 0, dTdt)
23
24    cost += np.sum((T_data[startingPoints[i]:(startingPoints[i] + n_ps)] - T_predicted)**2)

```

Snippet 6.7. Training of nRnC model with integrated inertia and percentage chunk training.

6.5.1 Evaluation

To evaluate the impact of these changes and to estimate a lower bound on the amount of initial conditions chosen, the MAE test described in (4.1) will be performed to compare the performance of the nRnC model using varying percentages of the training data. These percentages will follow a geometric progression with a ratio of $\frac{1}{10}$. However, the minimum percentage in which the training data can be used is expressed as:

$$P_{init_{min}} = \frac{1}{N_{sim}} = \frac{1}{35040} \approx 0.0000285 \quad (6.13)$$

Where N_{sim} represents the number of samples based on a 15-minute sampling frequency over the course of a year. Thus, using a geometric progression with a ratio of $\frac{1}{10}$, the decimal range becomes 1 to 0.0001, which translates to a range of percentages from 100% to 0.01%. Furthermore, this evaluation will be performed on each building model introduced in Section 3.1.1 to validate the approach across multiple building types. For consistency, the evaluation process will be based on second order nRnC models. The results for the second order nRnC model across all building models are shown for *SimpleHouseRad* in Figure 6.5 and Table 6.4, for *SimpleHouseRSla* in Figure 6.6 and Table 6.5, and for *SwissHouseRSla* in Figure 6.7 and Table 6.6.

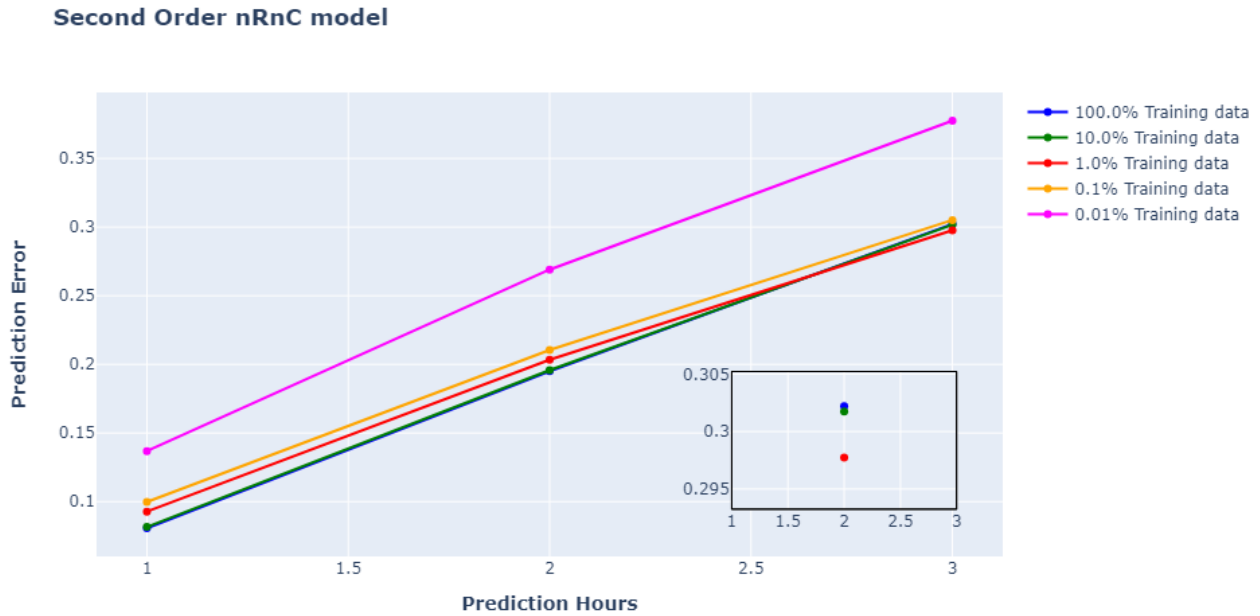


Figure 6.5. Performance trajectory for *SimpleHouseRad* through decreasing percentages training data, including a zoomed-in view for detailed analysis.

Training data %	Training Time [HH:mm:ss]
100 %	11:32:30
10 %	00:58:00
1 %	00:06:16
0.1 %	00:01:08
0.01 %	00:00:35

Table 6.4. Training times for each incremental percentage of training data used in model training for *SimpleHouseRad*.

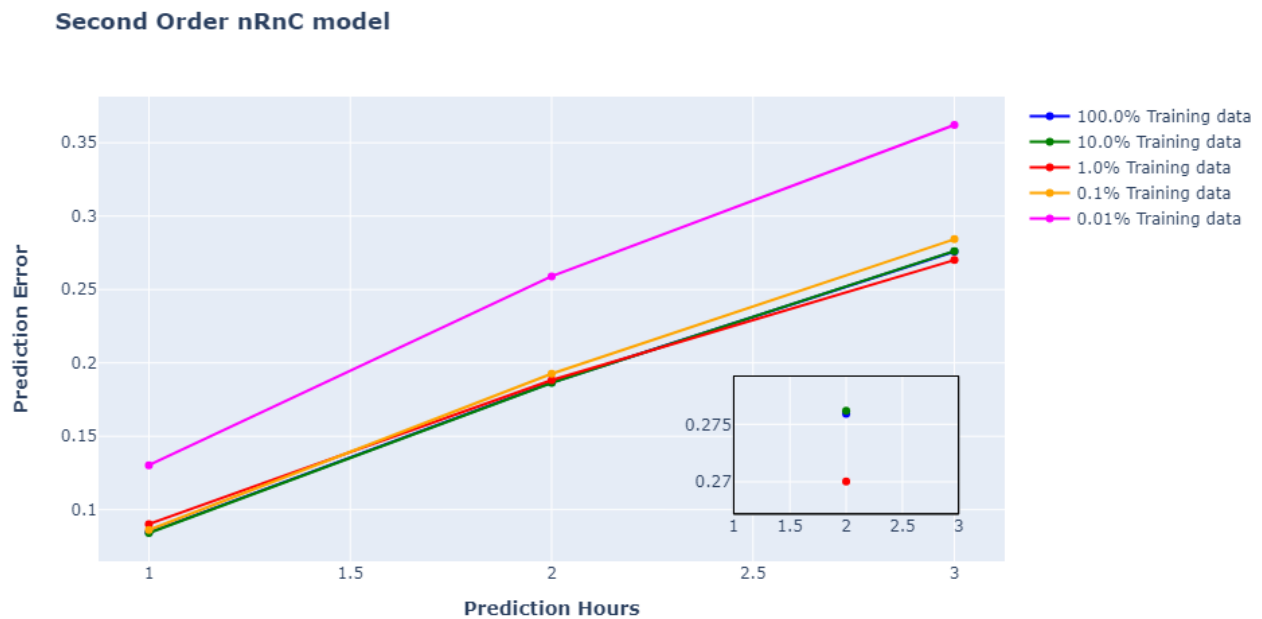


Figure 6.6. Performance trajectory for *SimpleHouseRSla* through decreasing percentages training data, including a zoomed-in view for detailed analysis.

Training data [%]	Training Time [HH:mm:ss]
100 %	08:06:24
10 %	00:55:49
1 %	00:04:20
0.1 %	00:00:59
0.01 %	00:00:35

Table 6.5. Training times for each incremental percentage of training data used in model training for *SimpleHouseRSla*.

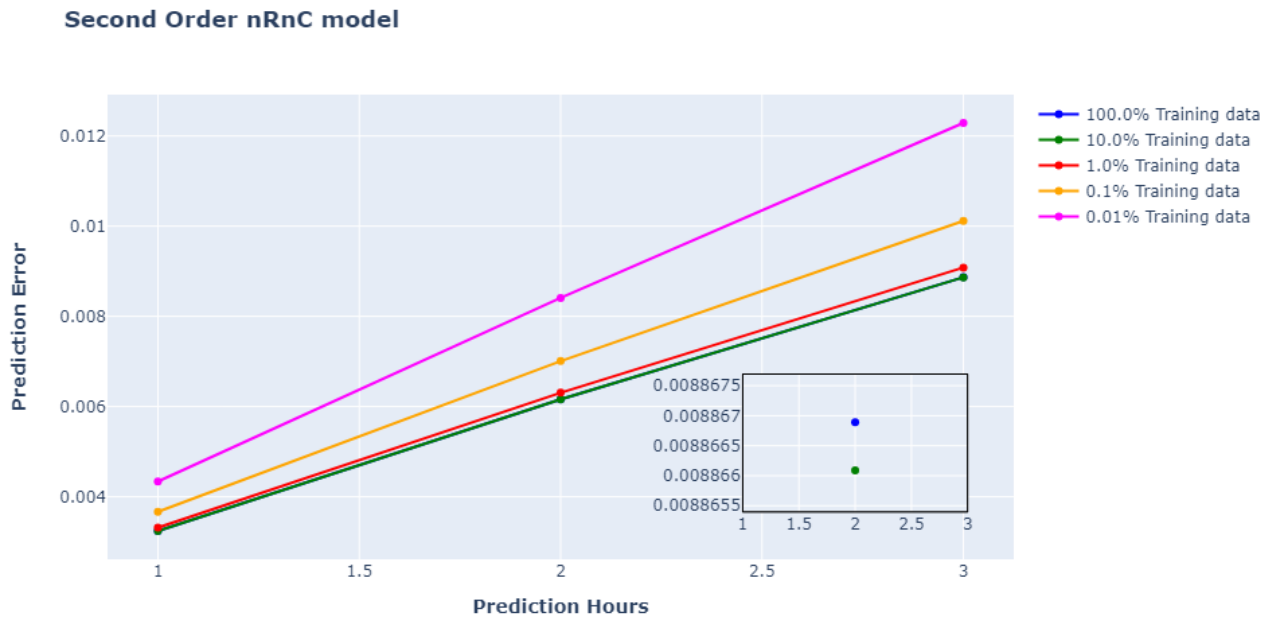


Figure 6.7. Performance trajectory for *SwissHouseRSla* through decreasing percentages training data, including a zoomed-in view for detailed analysis.

Training data %	Training Time [HH:mm:ss]
100 %	09:26:52
10 %	00:57:30
1 %	00:06:16
0.1 %	00:01:07
0.01 %	00:00:36

Table 6.6. Training times for each incremental percentage of training data used in model training for *SwissHouseRSla*.

As observed in Figure 6.5 and Figure 6.6, utilizing 100%, 10%, 1%, or 0.1% of the original training data, has little to no significant impact on the predictive performance of the nRnC model. In contrast to Figure 6.7, which demonstrates a performance impact when utilizing 0.1% of the original training data. When comparing these performances with their respective computational times listed in Table 6.4, Table 6.5 and Table 6.6, it becomes clear that utilizing 1% of the original training data as initial conditions represents a good trade-off between the computation time required to train a nRnC model, and the predictive performance of the trained model. Therefore, for all future simulations and each simulation within Section 6.2, it is decided to utilize only 1% of the training data.

6.6 Model Validation

With the enhancements made to the nRnC model, a validation of the model is to be made in the form of a comparison of the former nRnC model introduced in Chapter 2. Both models are trained using the *SimpleHouseRad* dataset, and their predictive performance are evaluated by comparing their prediction with the actual data for the corresponding time period. The comparison test is shown in Figure 6.8 along with the corresponding MSEs for each model compared to the actual data in Table 6.7.

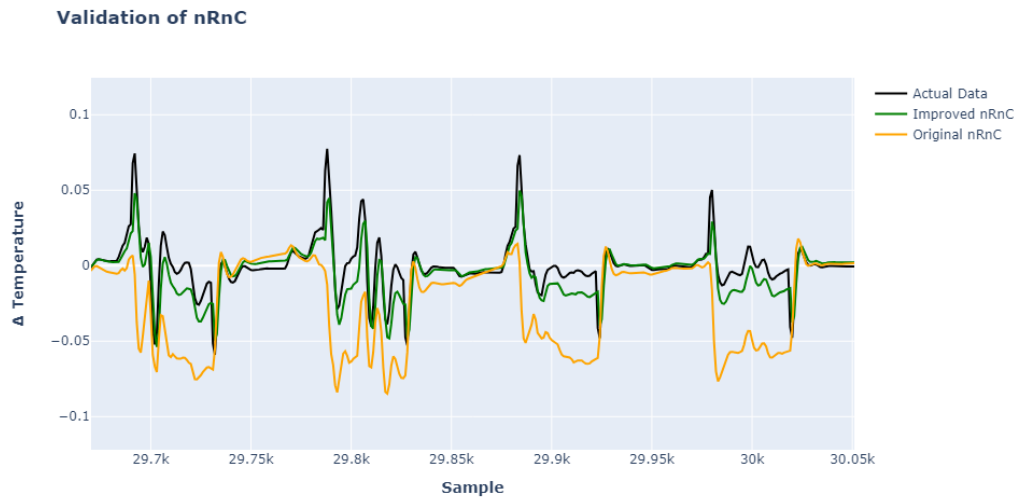


Figure 6.8. Comparison of the two nRnC models trained on *SimpleHouseRad* plotted against the actual data, demonstrating their accuracy in predicting temperature changes.

Model	MSE
Original nRnC	0.00141
Improved nRnC	0.00087

Table 6.7. Predictive performance of the two nRnC models trained on *SimpleHouseRad* compared to the actual data.

Thus, Table 6.7 confirms an improvement with the changes made to the original nRnC model, nearly doubling the predictive accuracy.

Chapter summary

In this chapter the further development of the nRnC from Chapter 2 has been described. The development included inclusion of scaled versions of delayed outputs, a mechanism for obtaining the number of delayed inputs to include which balances performance and complexity, and finally a method for improving the efficiency of the training method previously used. With these changes, the nRnC demonstrates improved predictive performance and increased efficiency. Henceforth in this project, the version of the nRnC model described in Chapter 2 will be referred to as the *original nRnC*, while the version of the nRnC described in this chapter will be referred to as the *improved nRnC*.

7 | Physics-informed neural network

This chapter will describe the implementation of Physics-Informed Neural Networks (PINNs) for learning the dynamics of the simulated buildings in this project. First the theoretical background of this type of neural network is given, after which the training of the PINNs in this project will be described.

7.1 Background

In recent years, the combination of deep learning techniques and traditional physics-based modeling has paved the way for new methods in data-driven modelling. Among these innovative approaches PINNs stand out as a promising method, combining the expressiveness of neural networks with the foundational principles of physics. The concept of PINNs was introduced in a 2017 paper by M. Raissi et. al. [27]. PINNs have a structure identical to a standard multi-layer perceptron (MLP), as shown in Figure 7.1

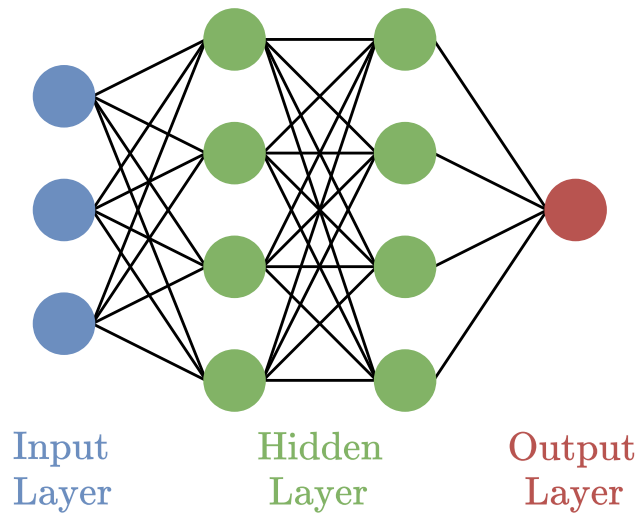


Figure 7.1. MLP with 3 inputs, 2 hidden layers with 4 neurons each, and 1 output.

In a MLP, the input propagates through the layers from left to right. In each hidden layer of the MLP a number of neurons are present, where the output of that neuron is generated by the input from the previous layer, which is weighted and summed with a bias. The output produced for a given input to a neuron, is based on an activation function, for example a rectified linear unit or sigmoid function [28, c.7]. During the training process of a neural network, the concept of epochs plays a crucial role. An epoch refers to a single pass through the entire training dataset. After processing all data points in the dataset, one epoch is completed, and the process may repeat for a predefined number of epochs or until a convergence criterion is met. Multiple epochs allow the network to progressively learn from the data [29, p.7].

In much of the literature available regarding PINNs, the goal is to make a PINN that can approximate the solution to a PDE which is expensive to solve using numerical methods. In a 2021 paper by F. Arnold and R. King, an example of the use of PINNs is given, where the training objective of the PINN is to provide approximate solutions to Burgers' equation [30]. Here, $h(p, t)$ denotes the solution to the equation as obtained through utilizing numerical methods at position p and time t . The goal is then to train a PINN $\psi(p, t)$, such that

$$h(p, t) \approx \psi(p, t)$$

In the mentioned article, the data collected to train the PINN is composed of boundary conditions and collocation points, which provide solutions to the PDE at points of interest. The boundary conditions of t consist of the initial condition at $h(p, 0)$, as well as the terminal condition at $h(p, t_{end})$. Similarly, the boundaries of p contain the solutions to the PDE at the lower and upper spatial bounds, p_{lb} and p_{ub} , of the equation $h(p_{lb}, t)$ and $h(p_{ub}, t)$. The collocation points consist of a selection points where the solution to $h(p, t)$ is known within the boundaries on t and p . The setup of these training points is shown in Figure 7.2, where the blue and white points are collocation and boundary points, respectively.

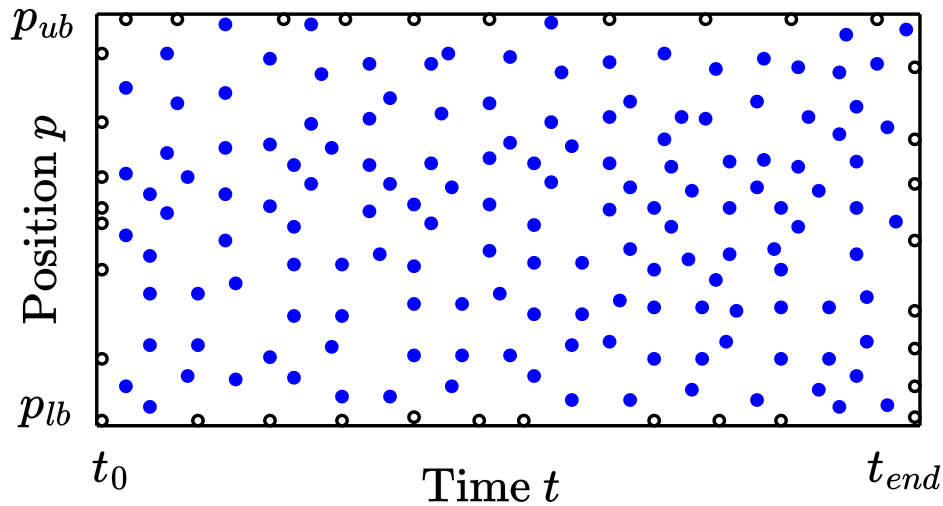


Figure 7.2. Example of distribution of points used for training PINN in the 2021 paper by F. Arnold and R. King. The blue points represent collocation points, and the white points represent boundary conditions [30].

Before training the PINN, numerical solutions to Burger's equation at these points are obtained by numerical solvers. Once the PINN has been trained on the input-output pairs, solutions to any arbitrary point chosen within the boundary conditions can be obtained by evaluating the PINN $\psi(p, t)$. This will be computationally cheap, when compared to a numerical solving method [30]. In this approach, incorporation of fundamental governing equations ensure that PINNs respects the underlying physics of the system, while providing generalization capability and fast evaluation of points within the boundary conditions.

7.2 Utilization of PINN concept for building modelling.

In this project, a similar approach is taken. As mentioned above, usually an accurate and detailed model of some underlying physical interaction is utilized as a ground truth, which the PINN is trained to mimic. The proposed PINN training strategy in this project will be, to use the improved nRnC model to provide these outputs at points of interest, which will be distributed similarly to how the data in Figure 7.2 is distributed. The similarity being, that the entire relevant input space will be populated with points. At each of these points the improved nRnC model will provide a predicted output, which then will be used to produce a *physics* training loss $\mathcal{L}_{physics}$ based on the difference in the outputs of the PINN and nRnC models. However, a PINN trained only with output data given by the improved nRnC model will only be as good as the improved nRnC model. To further improve the accuracy of the PINN, the data obtained in the simulation described in Section 5.4 will be utilized. This is the same data used when training the improved nRnC model, however the structure imposed by the choice of expressions in (6.2) might omit some important details. To allow the PINN to capture these details, the *data* training loss \mathcal{L}_{data} will be obtained by use of the back-propagation through time (BPTT) algorithm [31]. The adjustment to the weights and biases in the PINN will therefore be chosen based on (7.1)

$$\mathcal{L}_{total} = \mathcal{L}_{physics} + \text{sum}(\mathcal{L}_{data}) \quad (7.1)$$

7.2.1 Batching

For the data loss \mathcal{L}_{data} utilizing the BPTT algorithm, a length of time with q samples will be predicted from an initial condition, and consequently the entire simulated dataset will be split into batches of size q . Since the error used for training the PINN is \mathcal{L}_{total} , the data generated for $\mathcal{L}_{physics}$ will also have to be split. It is decided not to limit the size of this data to be equal to the simulation data, and instead ensure that an equal amount of total batches is created for both losses. The error of $\mathcal{L}_{physics}$ and \mathcal{L}_{data} will consequently be found as an average across the batch, such that a large dataset for either loss will not cause the associated error to dominate the training.

7.2.2 Physics loss

For the physics loss, first a dataset containing input samples is constructed. This dataset is constructed in a similar manner as the one mentioned in Section 7.1, however this dataset is not constrained by spatial and temporal boundaries, and instead constrained by statistically determined upper and lower bounds on each input feature. The upper and lower bound for each feature is found, by finding the mean value μ and standard deviation σ of the feature x in the simulated training data. The bounds \underline{x} are then defined as:

$$\bar{x} = \mu_{feature} \pm 5 \cdot \sigma_{feature} \quad (7.2)$$

The number of input features are equal to the input size of the improved nRnC model developed in Section 6.2. With the input size defined, a number of desired input points N is chosen. These points will be distributed according to the size of the input layer ν , by taking the ν 'th root of N , N_ν . The value of N_ν will dictate how many unique values of a given input feature will be distributed evenly within the \bar{x} of that feature. The concept can be illustrated as shown in Figure 7.3 by considering the case with only two features.

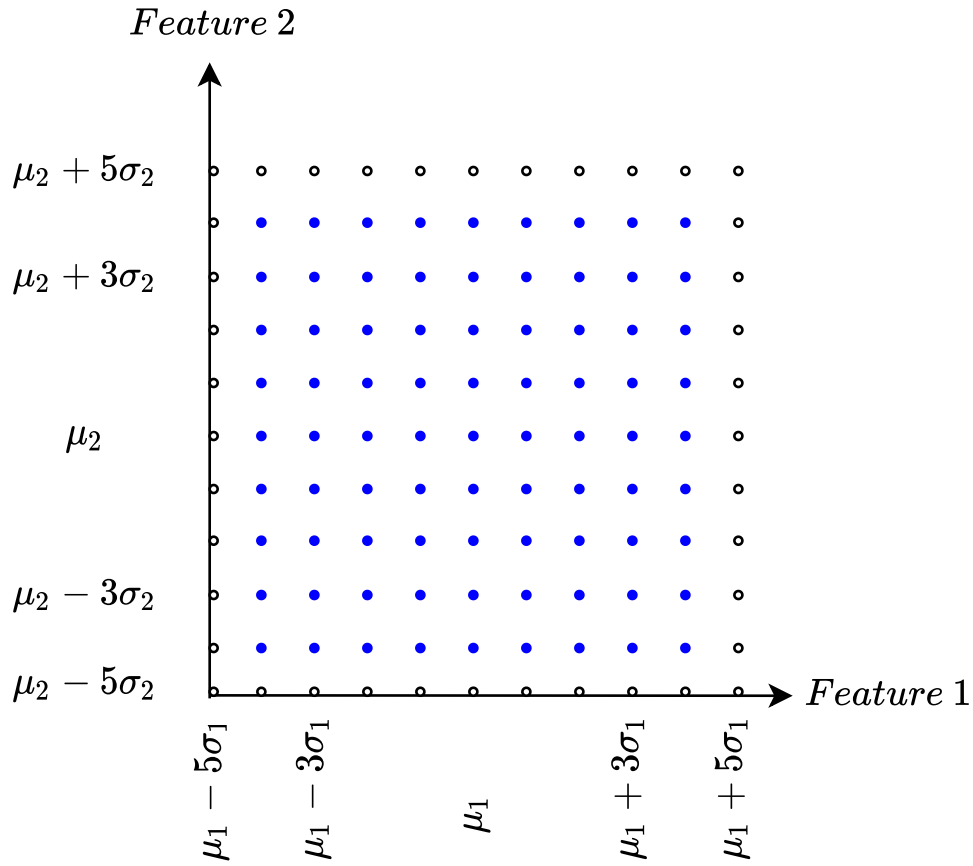


Figure 7.3. Illustration of method chosen for distributing points in relevant area of input space for two arbitrary features. The white dots represent points along the boundaries of one or multiple features, while the blue points represents points which are not placed on the boundary of any feature.

Once the input data has been generated, it is collected into a matrix \mathbf{D}_p^i . The matrix \mathbf{D}_p^i is then given sample by sample to the trained nRnC model, in order to obtain matching input and output pairs. The outputs are collected into the vector \mathbf{d}_p^o . As stated previously, the data will be segmented into batches. Therefore \mathbf{D}_p^i and \mathbf{d}_p^o are shuffled, while maintaining coherent input-output pairs, and segmented into batches, such that the total number of batches produced from the simulated data and physics data is equal. The length of the batches created from the physics data will be q_p , and the input matrix and output vector will be named \mathbf{B}_p^i and \mathbf{b}_p^o respectively, and are defined as:

$$\mathbf{B}_p^i = \begin{bmatrix} T_{Amb_k} - T_{Int_k} & T_{Amb_{k+1}} - T_{Int_{k+1}} & \dots & T_{Amb_{k+q_p}} - T_{Int_{k+q_p}} \\ W_{Sun_k} & W_{Sun_{k+1}} & \dots & W_{Sun_{k+q_p}} \\ W_{Heating_k} & W_{Heating_{k+1}} & \dots & W_{Heating_{k+q_p}} \\ \Delta T_{Int_{k-1}} & \Delta T_{Int_k} & \dots & \Delta T_{Int_{k+q_p-1}} \\ \Delta T_{Int_{k-2}} & \Delta T_{Int_{k-1}} & \dots & \Delta T_{Int_{k+q_p-2}} \\ \Delta T_{Int_{k-3}} & \Delta T_{Int_{k-2}} & \dots & \Delta T_{Int_{k+q_p-3}} \\ \Delta T_{Int_{k-4}} & \Delta T_{Int_{k-3}} & \dots & \Delta T_{Int_{k+q_p-4}} \end{bmatrix}$$

$$\mathbf{b}_p^o = \begin{bmatrix} \Delta T_{Int_k} & \Delta T_{Int_{k+1}} & \dots & \Delta T_{Int_{k+q_p}} \end{bmatrix}$$

The concurrent input-output pair at an arbitrary step is then taken as a column at the same index in both \mathbf{B}_p^i and \mathbf{b}_p^o . The physics loss $\mathcal{L}_{physics}$ for one batch is then obtained as shown in (7.3)

$$\mathcal{L}_{physics} = \frac{\sum_{k=1}^{q_p} (\mathbf{b}_p^o[k] - \psi(\mathbf{B}_p^i[k]))^2}{q_p} \quad (7.3)$$

The code implementation of how the physics loss is obtained is shown in Snippet 7.1

```

1 def PINN_loss_physics(PINN, input_data, output_data):
2
3     physics_loss = torch.tensor(0.0, requires_grad=True)
4     for i in range(batch_size_physics):
5         predicted_output = PINN(input_data[i,:])
6         loss_term = (predicted_output - actual_output[i])**2
7         physics_loss = physics_loss + loss_term
8     return physics_loss/q_p

```

Snippet 7.1. Code for obtaining physics loss.

7.2.3 Data loss

For the data loss, the simulation data is first segmented into batches. The batches will contain q_d temporally sequential samples, and be split into input \mathbf{B}_d^i and output \mathbf{b}_d^i . When predicting using the simulation data with the PINN, a prediction from an initial condition at step k is made. After the first input, the output of the model ΔT_{int_k} in step k , is used to predict the internal temperature T_{int} at step $k+1$ by applying a Forward Euler integration scheme. $T_{int_{k+1}}$ is then used as an input for the PINN ψ at step $k+1$, after which the process is repeated. Similarly, the delayed outputs are updated, such that the output at step k will become the once-delayed output when predicting the output at step $k+1$. The setup is shown in Figure 7.4

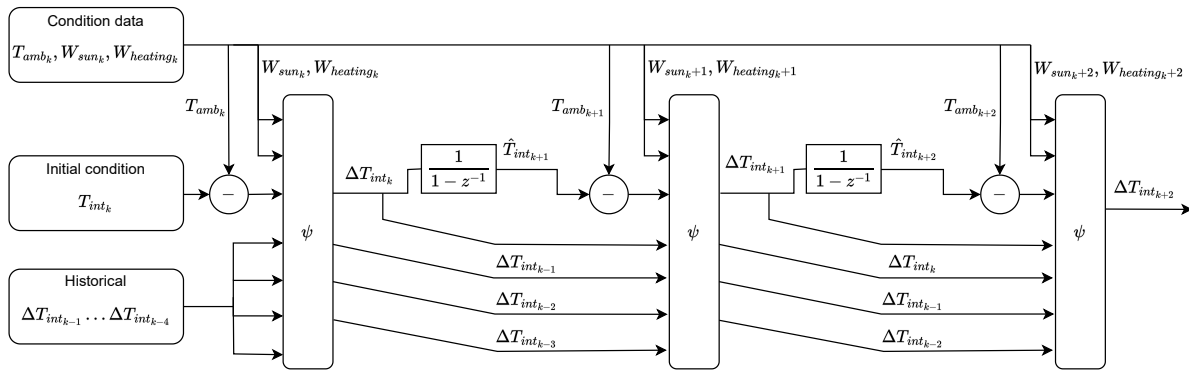


Figure 7.4. Illustration of the propagation of states during prediction from initial condition using the PINN network ψ . The condition data contains values for ambient temperature, solar radiation and heating input.

With the method of recursively predicting the temperature trajectory in a batch illustrated, the BPTT algorithm used to obtain the loss gradient through one batch will be described. The BPTT algorithm is an algorithm used with recurrent neural networks (RNNs). While the PINN in this project is not a RNN, the training method employed for the simulation data loss will be similar in concept. This is due to the multi-step prediction error obtained by both evaluating the temperature \hat{T}_{int} at each intermediate prediction step in the batch and the terminal $\hat{T}_{int_{k+q_d}}$, as shown in Figure 7.4. The BPTT algorithm aims to update the weights \mathbf{W} of the neural network ψ , denoted as \mathbf{W}_ψ . These weights are updated using the weight update rule shown in (7.4).

$$\mathbf{W}_{\psi_{k+1}} = \mathbf{W}_{\psi_k} + lr \cdot \frac{\delta \mathcal{L}_{data}}{\delta \mathbf{W}_{\psi_k}} \quad (7.4)$$

Where,

Symbol	Unit	Description
lr	$[\cdot]$	Learning Rate
\mathcal{L}_{data}	$[\cdot]$	Batch loss

With \mathcal{L}_{data} found as the difference between the predicted and actual internal temperature at step k as shown in (7.5)

$$\mathcal{L}_{data} = (\hat{T}_{Int_k} - T_{Int_k})^2 \quad (7.5)$$

To compute the gradients of \mathcal{L}_{data} with respect to \mathbf{W}_ψ , the chain rule of differentiation is applied through the network unrolled over time, as illustrated in Figure 7.4. Using the chain rule, the gradient of \mathcal{L}_{data} with respect to the weights \mathbf{W}_ψ is given by (7.6) [31].

$$\frac{\partial \mathcal{L}_{data}}{\partial \mathbf{W}_\psi} = \sum_{i=1}^{q_d} \sum_{j=0}^i \frac{\partial \mathcal{L}_{data_i}}{\partial \Delta \hat{T}_{Int_k+i}} \frac{\partial \Delta \hat{T}_{Int_k+i}}{\partial \Delta \hat{T}_{Int_k+j}} \frac{\partial \Delta \hat{T}_{Int_k+j}}{\partial \mathbf{W}_\psi} \quad (7.6)$$

Where \mathcal{L}_{data_i} represents the loss at time step i , and the summation accounts for the contributions from each time step to the overall gradient. Thus (7.6) reflects how the gradient at time step i depends on the outputs and weights of all previous time steps up to j . The weights \mathbf{W}_ψ are then updated based on the loss gradient, ensuring that the model learns to minimize the loss over time. This description covers how BPTT would be implemented for the \mathcal{L}_{data} alone, however in this project the loss used for training is instead the total loss \mathcal{L}_{total} . To obtain the data loss \mathcal{L}_{data} for a single batch, the true sequence of q_d outputs in the training data is compared with the sequence predicted by the PINN from the initial condition. The calculation of \mathcal{L}_{data} is shown in (7.7).

$$\mathcal{L}_{data} = \frac{\sum_{k=1}^{q_d} (\Delta T_{Int_k} - \Delta \hat{T}_{Int_k})^2}{q_d} \quad (7.7)$$

The code implementation of how the data loss is obtained is shown in Snippet 7.2

```

1  def PINN_loss_data(PINN, input_data, actual_temperature, condition_data):
2
3      data_loss = torch.tensor(0.0, requires_grad=True)
4      for i in range(batch_size_data-1):
5          predicted_output = PINN(input_data[i,:])
6          predicted_temperature = normalize(denormalize(condition_data[i, 0], data_max[0]) +
7              denormalize(predicted_output, data_max[4]), data_max[0])
8          Tamb_Tint = normalize(denormalize(condition_data[i,1],
9              max_d[0]) - denormalize(predicted_temperature, max_x), max_d[2])
10         loss_term = (predicted_temperature - actual_temperature[i])**2
11         data_loss = data_loss + loss_term
12
13         # update input corresponding to difference between ambient and internal temperature,
14         # based on predicted temperature
15         input_data[i+1, 2].data.copy_(Tamb_Tint.data.item())
16     return data_loss/q_d

```

Snippet 7.2. Code for obtaining data loss.

7.3 Training the model

With $\mathcal{L}_{physics}$ and \mathcal{L}_{data} defined as per-batch losses, the model training will be described. The training will be performed by utilizing the Pytorch library, which is a highly developed and commonly used open-source machine learning framework [32]. First the two losses are combined into \mathcal{L}_{total} as shown in (7.1). To adjust the weights in accordance with the \mathcal{L}_{total} obtained for one batch, auto-differentiation is utilized. The `.backward()` function call in Pytorch is the automatic differentiation technique used in this project to find the gradients of the PINN loss function [33]. Automatic differentiation is a method used to obtain partial derivatives of code variables wrt. other variables, by exploiting the fact that any computer program is constructed using a finite set of simple mathematical operations. By repeatedly applying the chain rule of differentiation through the chain of operations, the desired partial derivative is obtained [34]. Based on the obtained gradient, the Adam optimizer will be utilized to implement gradient descent optimization in this project [35]. The code implementation of the PINN training is shown in Snippet 7.3

```

1 # Obtaining losses
2 data_loss = PINN_loss_data(PINN, batch_input_data, batch_output_data, batch_condition_data)
3 physics_loss = PINN_loss_physics(PINN, batch_input_physics, batch_output_physics)
4 loss = data_loss + physics_loss
5
6 # Backpropagation and optimization
7 loss.backward()
8 optimizer.step()

```

Snippet 7.3. Code implementation of autodifferentiation using Pytorch.

7.3.1 Determining Network Size

With the training method and loss functions defined, the PINN can be trained and tested. However, first the width and depth of the network has to be defined. While simply choosing a very large network size would allow the PINN to model the dynamics very accurately, however such a large network would take longer to train, be more prone to overfitting, and take longer to evaluate. This reduction in evaluation speed would become relevant when used in combination with the MPC, since the operation of MPC requires a very large amount of model evaluations every time the optimal control problem is solved. Therefore, the desired size of the PINN is the size, where the network is *just* big enough to accurately model the dynamics of the building. To ensure this, an approach similar to the method in Section 6.2 is used. Specifically a value L_e , which is initialized as zero, is used to control this size, by determining the amount of neurons γ_n in each layer according to (7.8)

$$\gamma_n = 2 + 5 \cdot (\text{mod}(L_e, 2)) + 5 \cdot [L_e/2] \quad (7.8)$$

And the amount of hidden layers γ_l in the network as shown in (7.9)

$$\gamma_l = 1 + [L_e/2] \quad (7.9)$$

This method of expanding the network was defined based on heuristic considerations, namely that the network size should not shrink when adding additional layers. As seen, first the amount of neurons in every hidden layer is increased by 5 when incrementing L_e once. The next increment of L_e adds another layer with the same amount of neurons. By training the PINN with increasing values of L_e , a systematic exploration of different network shapes will be made. To evaluate the trained PINN, 20% of the simulation data batches constructed will be used as validation data and the remaining 80% will be used as training data. Once a network has been trained, the MSE between the output in the validation data and the output calculated by the PINN when fed the validation data inputs will be found. The systematic network size increase will follow the procedure shown below:

1. Train the PINN with γ_l hidden layers and γ_n neurons per layer
2. Calculate validation loss as MSE of PINN output compared with validation dataset output
3. Compare with previous best validation loss
4. If the obtained validation loss is more than 1% less than the best validation loss, increment L_e and repeat from step 1.
5. If the validation loss improvement does not exceed the threshold, but the most recent iteration before the current provided an improvement, increment L_e and repeat from step 1.
6. If the validation loss has failed to improve twice in a row, end the training and select the network which the last two networks were compared with.

As shown, the improvement between consecutive trained PINN networks has to be above a threshold of 1%. This is a tuning value, which has been chosen based on a weighting between network performance and training time.

7.3.2 Determining Learning Rate

The weights in the PINN network are randomly initialized, with a large initial learning rate. However, a large learning rate will probably prohibit the weights of the PINN from converging fully. To address this, a scheduler is introduced. The role of a scheduler is to monitor a loss for improvement. If no improvement is detected over a set number of epochs, it decreases the learning rate, which causes the training algorithm to take smaller steps. This scheduling mechanism is implemented using the '**torch.optim.lr_scheduler library**', specifically the '**ReduceLROnPlateau**' function. While this function offers several adjustable parameters, for this purpose, only the following are utilized [36]:

```
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                           mode='min',
                                           factor=0.1,
                                           patience=2)
```

In this context, the '**mode**' parameter determines whether the learning rate should increase or decrease. Here, the setting '**min**' indicates that a decrease is desired. The '**patience**' parameter dictates the number of epochs to wait, before adjusting the learning rate. Lastly, the '**factor**' parameter specifies the magnitude of change in the learning rate, calculated as [36]:

$$lr_{\text{new}} = lr \cdot \text{factor}$$

With this implementation, the PINN model commences with randomly initialized weights and progressively decreases step sizes as it approaches convergence.

7.4 Model Validation

To evaluate the performance of the PINN model, the MAE test described in (4.1) is conducted to compare its predictive performance with the improved nRnC model developed in Chapter 6. The test will be performed for all three building models: *SimpleHouseRad*, *SimpleHouseRSla*, and *SwissHouseRSla*. Additionally, the training duration for the PINN model is provided. Unlike the nRnC model, the training duration of the PINN model is not reliant on model order, but rather on the ability to find a suitable minimum that satisfies the threshold. The variability in finding this minimum stems from, among others, the randomized initialization of weights in the neural network of the PINN, as detailed in Section 7.3.2. To address this variability, the PINN is trained five times with a model order of 1, allowing for a comparison of duration variations specific to this model. The predictive performance plots for each building model are displayed in Figure 7.5, Figure 7.6, and Figure 7.7. Additionally, the training duration and network size for five PINN trainings on a single building model, *SimpleHouseRad*, are presented in Table 7.1.

Run #	Network Size [Neurons:Layers:Epochs]	Training Duration [HH:mm:ss]
1	7:2:77	00:37:22
2	7:2:89	00:44:29
3	7:2:52	00:26:57
4	12:2:96	00:49:10
5	7:2:61	00:29:32

Table 7.1. Network size, epochs, and training duration for *SimpleHouseRad*, highlighting the training process parameters during the optimization process.

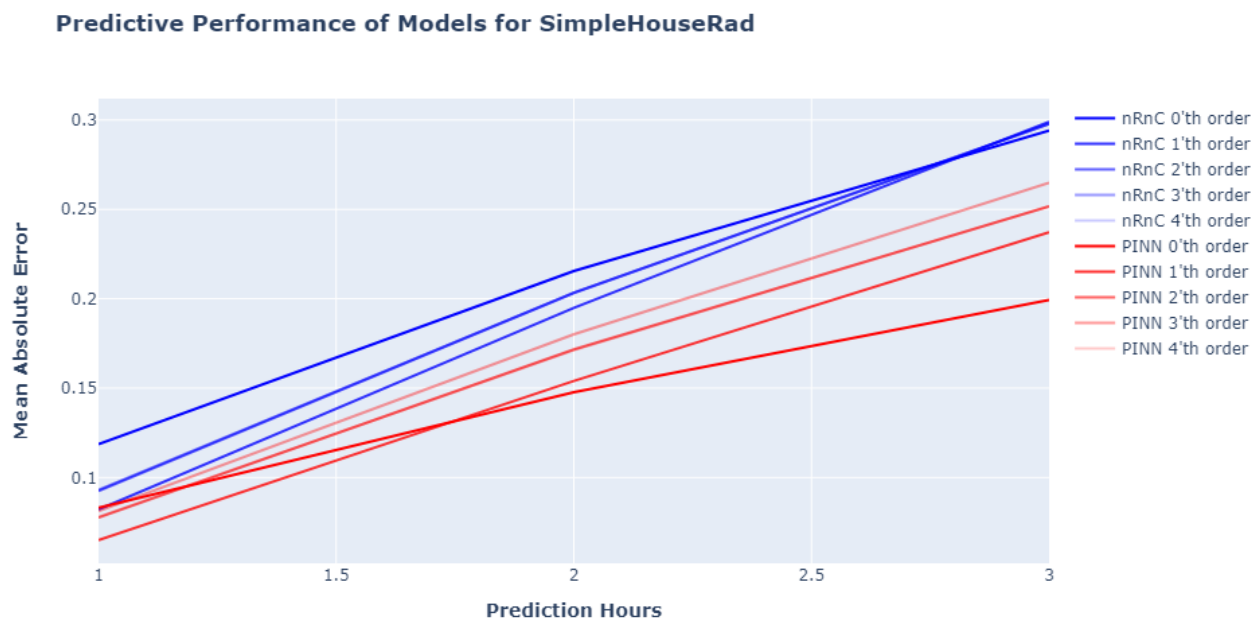


Figure 7.5. Predictive performance comparison between the π th order improved nRnC and PINN models on the *SimpleHouseRad* dataset.

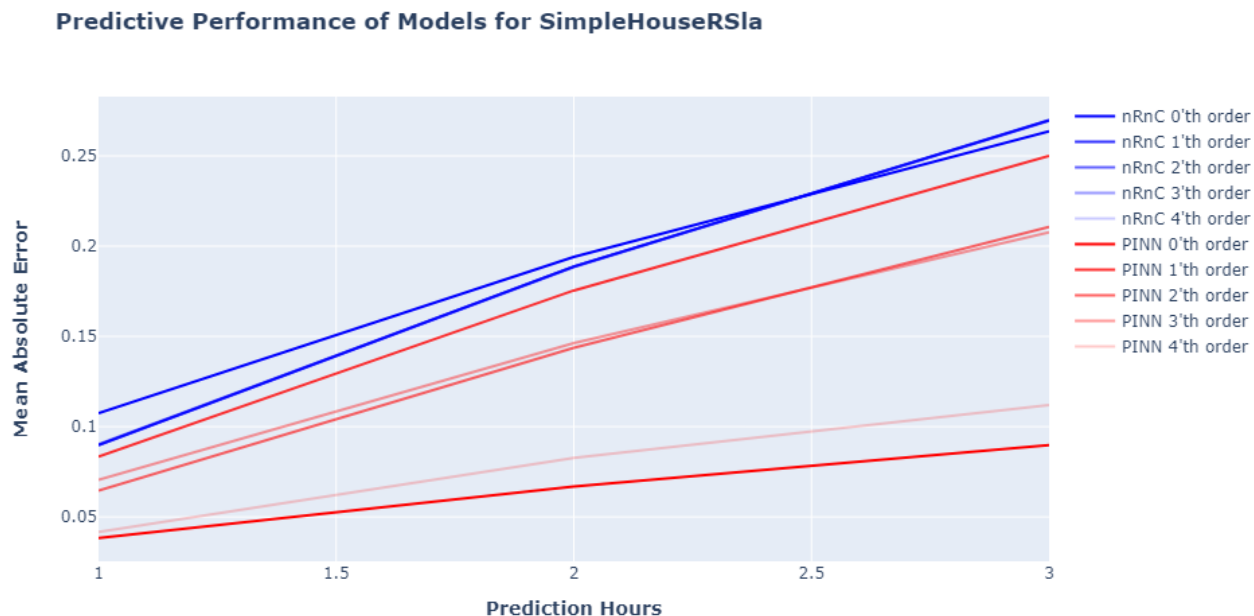


Figure 7.6. Predictive performance comparison between the π th order improved nRnC and PINN models on the *SimpleHouseRSla* dataset.

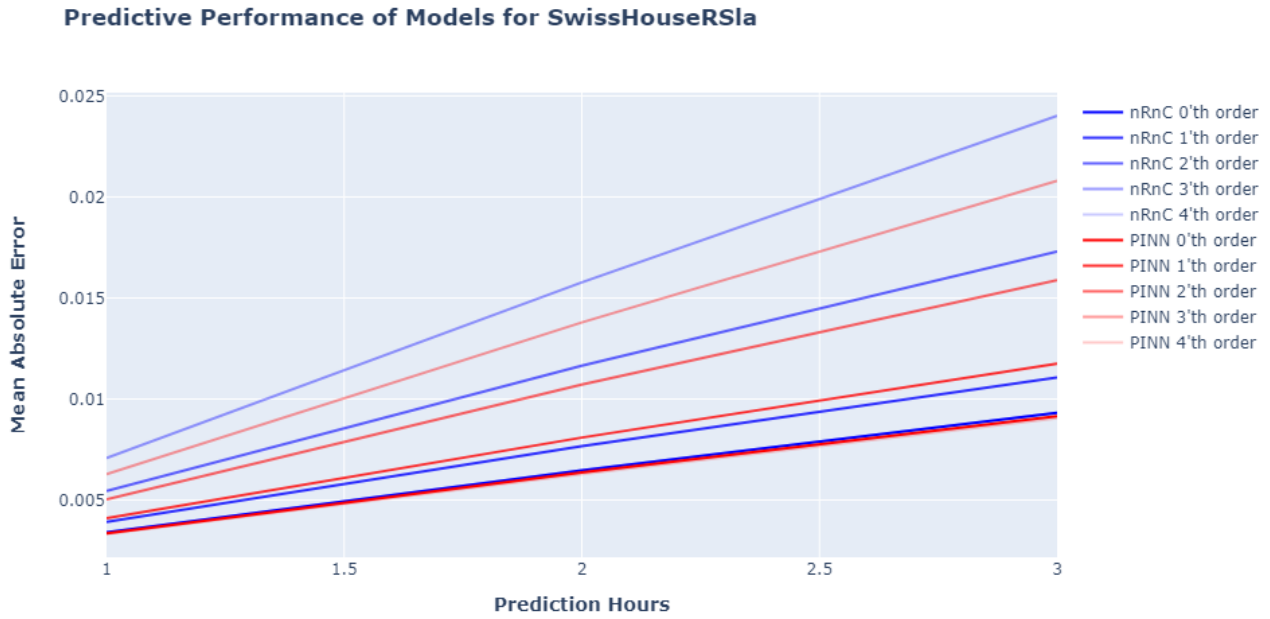


Figure 7.7. Predictive performance comparison between the π th order improved nRnC and PINN models on the *SwissHouseRSla* dataset.

Chapter summary

In this chapter the development of a PINN, which utilizes a combination of the predictions made by the improved nRnC model, described in Chapter 6, and predictions over consecutive steps during training. Furthermore, a method for dynamically determining the size of the network and the learning rate based on prediction performance on verification data is described. Finally, the predictive performance of the PINN is compared with the performance of the improved nRnC, where the PINN demonstrates better performance in the buildings *SimpleHouseRad* and *SimpleHouseRSla*, while the performance in *SwissHouseRSla* is almost identical between the models.

8 | Auto-tuning of hyperparameters

This chapter will describe the selection of a system for online tuning of the hyperparameters in the MPC cost function described in (5.1). The designed system will use the KPIs described in Section 3.2 as performance signals to aid the tuning. Additionally, the implementation of the chosen system will be described, after which a demonstration of the system used in a building simulation will be given.

8.1 Problem definition

As seen in the MPC cost function described in Section 2.3, multiple hyperparameters are present. The H_w and H_c hyperparameters are set to constant values, zero and H_p respectively, regardless of the characteristics of the current building in this project. H_p is also relatively simple to determine, where the value is initially set based on a priori assumptions regarding how long it typically takes to heat up a building. The value can then be raised, if for example the heating system is not able to raise the temperature in time for the rising edge of the occupied hours, even with high actuation throughout the prediction horizon. Finally the ξ parameter is chosen as a constant value. Based on this, only the R , C and C_Δ will be chosen as the hyperparameters to be tuned by the system described in this chapter. However, a trade-off is present between the R and both the C and C_Δ hyperparameters, which can help simplify the problem if exploited. The trade-off being, that the cost of energy input into the system is weighted against the cost of deviating from the reference in the minimization problem solved in the MPC. Since these two costs are weighted linearly by the scalar values R and C , a ten-fold increase in both values would make no difference to the optimal control input found. A similar consideration can be made regarding R and C_Δ . Based on this, it is chosen to find the optimal values of C and C_Δ *given a specific R* .

8.1.1 Choice of strategy

A decision has to be made regarding whether a model of the interaction between chosen hyperparameters and measured KPI values will be made. Such a model would have to include the disturbances caused by variations in ambient conditions. The severity of these disturbances would depend on the specific building, and have to be known in advance in order for the model to accurately account for them. However, the task of defining a building-specific transfer function which takes hyperparameters as inputs and outputs KPIs would be in conflict with the problem statement in Section 4.2. One possible solution would be to train a model of this interaction based on historical data. However this would require a systematic search of the hyperparameter space with corresponding output values, which is difficult to obtain due to the stochastic nature of the impact of ambient conditions. Therefore, it is chosen to treat the problem as a black-box optimization problem, which implies that the transfer function between the hyperparameter inputs and the resulting KPIs is not explicitly defined.

8.2 Candidate methods

With the problem specified as a black-box optimization problem, the possible candidates for solving such a problem will be described in this section.

8.2.1 Sampling methods

A simple approach to this problem is choosing the best performing point observed, based on a sampling of the space of possible hyperparameter values. Grid search or random sampling are examples of methods which uses this approach [37]. Common for these methods is, that a large amount of hyperparameter values have to be tested in order to provide a solid basis for choosing the best set of hyperparameters. As stated in Section 3.2, the fact that each set of hyperparameter values requires at least one full day of measurements before KPI values can be produced, makes these methods poorly suited. Furthermore, the changing ambient conditions also impact the problem, which causes points previously sampled to become less reliable over time.

8.2.2 Metaheuristic methods

Metaheuristic methods offer an alternative approach to tackling this problem. Unlike traditional sampling methods, they systematically explore the search space with the flexibility to make assumptions about the problem. This enables them to focus on promising areas of the hyperparameter space rather than exhaustively examining every possibility [38, p.2]. In these methods, an initial broad sampling of the hyperparameter space is conducted. The most successful points from this sampling guide further exploration. For instance, in particle swarm optimization particles move through the hyperparameter space, adjusting their positions based on performance feedback [39, c.1]. Similarly, genetic algorithms select new values to explore based on combinations of evaluated points with good performance, and discards poor points in order to iteratively refine the search [40, c.1]. These approaches have the potential to reduce the number of evaluations needed compared to traditional sampling methods, especially if an efficient initial exploration is performed. However, sparse initial sampling may overlook promising areas in the hyperparameter space.

8.2.3 Surrogate-assisted methods

Surrogate-assisted methods are a type of sequential method, where points in the hyperparameters space which have been evaluated, are used to approximate a surrogate model of the black-box function. Optimization techniques can then be used on the surrogate model, in order to obtain a suggestion for a new optimal point to evaluate [41]. Over time, as new points are suggested by the model, and subsequently evaluated in the black-box, the number of known points available for constructing the surrogate model increases. Due to this, these methods are very sample-efficient, since each additional evaluation aids the process by increasing the information captured by the surrogate model [42]. One popular surrogate method is Bayesian Optimization (BO), which is based on Bayesian inference. BO is a popular and effective method for hyperparameter optimization in neural networks, to the extent that a variation of BO was the winning algorithm in the "black-box optimization challenge" at the 2020 NeurIPS conference [43]. Furthermore, frameworks like AutoMPC, which implement automatic hyperparameter tuning, utilize BO techniques for this task [44].

8.2.4 Selection of black-box optimization method

As mentioned, there is a high cost associated with evaluation of the black-box function, since a proposed set of hyperparameter values has to be used in the HVAC of a building for at least a full day, before KPIs can be calculated using the functions in Table 3.1. Based on this, it is desired to select a method which is as sample-efficient as possible. Therefore BO is chosen as the black-box optimization method in this project.

8.3 Bayesian optimization

With BO chosen as the black-box optimization method in this project, a description of the method will be given. As described, BO utilizes a surrogate model to capture the assumed behaviour of the objective function. By applying a so-called acquisition function to this surrogate model, a surface of the value assigned to choosing a given point in the surrogate model is obtained. By applying optimization techniques to this surface, a suggested point which either minimizes or maximizes the acquisition function is chosen. The setup is illustrated in Figure 8.1.

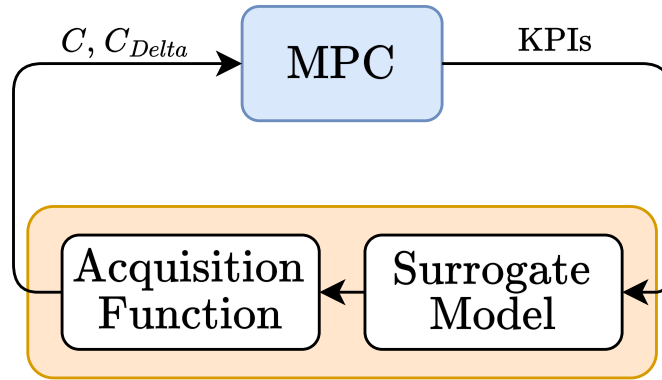


Figure 8.1. Illustration of a BO with a surrogate model representing the knowledge of the objective function, which in this project is the MPC, and an acquisition function used in combination with optimization techniques to determine points of interest. The colors indicate where the described components of BO are to be found in the system diagram in Figure 5.1

For clarity, a step-by-step description of Bayesian Optimization is given. The Bayesian Optimization algorithm proceeds as follows:

1. Initialize a surrogate model of the objective function with initial data points.
2. Assign value to the surrogate function using an acquisition function.
3. Select the next point for evaluation by applying an optimizer to the acquisition function output.
4. Evaluate the objective function at the selected point.
5. Update the surrogate model with the new data.
6. Either repeat step 2 to 4 until a stopping criteria is met.

The stopping criteria mentioned in step 5 of this procedure is chosen to fit the specific problem which is solved. BO is often used for determining the size and shape of neural networks, and in that context a stopping criteria could for example be a performance threshold. In this project no stopping criteria is defined, where BO is simply allowed to operate throughout the simulation. However, in a real world building implementation which utilizes BO for hyperparameter tuning, the BO algorithm could be turned off after multiple years of historical data has already been collected. As described, the surrogate model plays a crucial role in BO. In this project Gaussian Process (GP) regression is chosen for obtaining such a model, which is a common choice for BO [45]. A description of (GP) regression will be given below.

8.3.1 Gaussian Process Regression

GP regression is a powerful tool for constructing surrogate models in Bayesian Optimization. It defines a *distribution over functions*, allowing the BO to capture uncertainty and make probabilistic predictions about the objective function $f(x)$. The distribution over functions is defined by assuming the observations of the objective function $f(x)$ are all originating from the same function evaluated at a point x . This function can be thought of as an infinite vector, specifying the value of the function at any chosen point x . Therefore any finite set of observations would only partially describe the function. Fortunately, Gaussian inference allows predictions of the behaviour of the function at a finite set of points of interest to be determined based on a finite set of observations, and the results obtained will be the same as those obtained, if all the infinite points not evaluated were included [46, p.2]. More specifically, it is assumed that the observations from the objective function at point x are obtained as noisy observations, shown in (8.1)

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2) \quad (8.1)$$

To construct the GP, first the following declarations are made for a GP with v features:

Symbol	Description
σ_ϵ^2	Variance of the observation noise.
$(x_i, y_i)^N$	Set of N observations
$(x_i^*)^M$	Set of M test inputs
$\mathbf{X}^{N \times v}$	Matrix with evaluated inputs, each row represents one input
$\mathbf{X}^{*M \times v}$	Matrix with test inputs, each row represents one input
$\mathbf{y}^{N \times 1}$	Observed noisy outputs at the inputs \mathbf{X}
$\mathbf{f}^{N \times 1}$	True function value at the inputs \mathbf{X}
$\mathbf{f}^{*M \times 1}$	True function value at the test inputs \mathbf{X}^*
$\mathbf{K}_{x,x}^{N \times N}$	Cross-correlations between evaluated points, obtained with a chosen kernel function as $k(x_i, x_j)$
$\mathbf{K}_{x,x^*}^{N \times M}$	Cross-correlations between evaluated points and test points, obtained as $k(x_i, x_j^*)$
$\mathbf{K}_{x^*,x}^{M \times N}$	Cross-correlations between evaluated points and test points, obtained as $k(x_i^*, x_j)$
$\mathbf{K}_{x^*,x^*}^{M \times M}$	Cross-correlations between test points, obtained as $k(x_i^*, x_j^*)$

The GP assumption then is, that the observations \mathbf{y} and the true function values at the test points \mathbf{f}^* are *jointly distributed* as an $N + M$ dimensional multivariate normal [46, p.14]. The distribution is shown in (8.2)

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^* \end{bmatrix} \sim \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 & \mathbf{K}_{\mathbf{x},\mathbf{x}^*} \\ \mathbf{K}_{\mathbf{x}^*,\mathbf{x}} & \mathbf{K}_{\mathbf{x}^*,\mathbf{x}^*} \end{bmatrix} \right) \quad (8.2)$$

With this definition, a GP is fully specified by its mean function $\mu(x)$ and covariance function (kernel) $k(x, x')$. Formally, a GP is denoted as shown in (8.3)[46, p.15-16]

$$\mathbf{f}^* | \mathbf{X}\mathbf{y} \sim \mathcal{N}(\mu_{f^*}, \sigma_{f^*}^2) \quad (8.3)$$

where:

Symbol	Equation
μ_{f^*}	$\mathbf{K}_{\mathbf{x}^*,\mathbf{x}}[\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2]^{-1}\mathbf{y}.$
$\sigma_{f^*}^2$	$\mathbf{K}_{\mathbf{x}^*,\mathbf{x}^*} - \mathbf{K}_{\mathbf{x}^*,\mathbf{x}}[\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2]^{-1}\mathbf{K}_{\mathbf{x},\mathbf{x}^*}.$

For evaluating a single test point x^* , as will be done in this project, the cross-correlation matrices can be compactly defined as vectors of covariances \mathbf{k}^* . The equations for μ_{f^*} and $\sigma_{f^*}^2$ then become as shown (8.4) and (8.5)[46, p.17]

$$\mu_{f^*} = \mathbf{k}^{*T}(\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2)^{-1}\mathbf{y} \quad (8.4)$$

$$\sigma_{f^*}^2 = k(x^*, x^*) - \mathbf{k}^{*T}(\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2)^{-1}\mathbf{k}^* \quad (8.5)$$

8.3.1.1 Kernel function

The choice of kernel in this project was between the Radial Basis Function (RBF) and the Matérn kernel. While the Matérn kernel provides good robustness to outliers, the generality of the RBF kernel was deemed as the better choice, since assumptions regarding the characteristics of the noise in the functions described in Table 3.1 were not made [46, p. 83-85]. The RBF kernel assumes that the function values become less correlated as the distance between the input points increases, and it provides a smooth interpolation between observed data points. Furthermore the RBF kernel is more computationally efficient, which makes for a scalable implementation. The RBF kernel is defined as shown in (8.6)[46, p.84]

$$k_{RBF}(\mathbf{x}, \mathbf{x}') = \sqrt{\pi}\ell\sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2(\sqrt{2}\ell)^2}\right) \quad (8.6)$$

where:

Symbol	Description
σ_f^2	Signal variance, controlling the overall variance of the function.
ℓ	Length scale parameter, which determines the distance over which the function values are correlated.

8.3.2 Acquisition Functions in Bayesian Optimization

In Bayesian optimization, acquisition functions are critical for guiding the search for the optimal solution. These function assign value to the surrogate function surface based on the mean from (8.4) and variance from (8.5) by balancing exploration, which promotes searching new areas with high uncertainty, and exploitation, which focuses on areas where good function values have been observed and the uncertainty is low. In order to select an acquisition function for this project, some possible choices for these functions will be described. The descriptions will be formulated for maximizing as in [47], however once an acquisition function has been selected the conversion to minimization will be described.

8.3.2.1 Probability of Improvement (PI)

The Probability of Improvement (PI) acquisition function focuses on the likelihood that a new sample will yield a better result than the current best observation [47, p.681]. It is defined as:

$$PI(x) = \Phi \left(\frac{\mu_{f^*}(x) - f(x^+) - \kappa}{\sqrt{\sigma_{f^*}^2(x)}} \right) \quad (8.7)$$

where:

Symbol	Description
$\Phi(\cdot)$	Cumulative distribution function of the standard normal distribution.
μ_{f^*}	Predicted mean of the objective function at point x .
$\sqrt{\sigma_{f^*}^2}$	Square root of the GP covariance.
$f(x^+)$	Current best observation.
κ	Exploration parameter.

The PI function prioritizes points that have a high probability of improving upon the best known value, thus focusing on exploitation. Since the uncertainty of the prediction made with (8.3) increases as the distance from a known point does, the denominator of (8.7) also increases. Therefore, this acquisition function sometimes becomes "greedy" and avoids exploring areas that might yield even better results but have low certainty. However, tuning of the κ parameter can adjust this behaviour.

8.3.2.2 Expected Improvement (EI)

The Expected Improvement (EI) acquisition function considers not only the probability of improvement, but also the magnitude of the improvement. It is designed to find points that provide the greatest expected improvement over the current best observation [48, p.82-85]. It is given by:

$$EI(x) = (\mu_{f^*}(x) - f(x^+) - \kappa)\Phi(Z) + \sqrt{\sigma_{f^*}^2(x)}\phi(Z) \quad (8.8)$$

given by:

$$Z = \frac{\mu_{f^*}(x) - f(x^+) - \kappa}{\sqrt{\sigma_{f^*}^2(x)}} \quad (8.9)$$

where:

Symbol	Description
$\phi(\cdot)$	Probability density function of the standard normal distribution.

The EI function is effective at balancing exploration and exploitation, because it takes into account both the uncertainty (through $\sqrt{\sigma_{f^*}^2(x)}$) and the potential reward (through $\mu_{f^*}(x) - f(x^+)$). It tends to select points where the model predicts both a high mean and high uncertainty. Due to this, the improvement obtained with the acquisition function is both an improvement in terms of potentially finding a point with a high function value, but also an improvement in terms of reducing the total uncertainty of the surrogate model [48, p.82-85].

8.3.2.3 Upper Confidence Bound (UCB)

The Upper Confidence Bound (UCB) acquisition function explicitly controls the trade-off between exploration and exploitation by considering both the predicted mean and the uncertainty of the prediction [47, p.682]. It is defined as:

$$UCB(x) = \mu_{f^*}(x) + \kappa\sqrt{\sigma_{f^*}^2(x)} \quad (8.10)$$

A higher value of κ encourages exploration by placing more emphasis on the uncertainty (higher $\sqrt{\sigma_{f^*}^2(x)}$), while a lower value of κ encourages exploitation by focusing more on the mean ($\mu_{f^*}(x)$). This flexibility allows UCB to be tailored to different optimization problems, but the choice of κ is crucial and can significantly affect the performance of the optimization process.

8.3.2.4 Summary and selection of acquisition function

In summary, these three acquisition functions have different ways of balancing exploration and exploitation. A summary of the functions is given below:

- **Probability of Improvement (PI)**: Simple and focuses on the probability of improvement, but may be overly exploitative.
- **Expected Improvement (EI)**: Considers both the improvement in terms of reducing uncertainty and obtaining good function values.
- **Upper Confidence Bound (UCB)**: Explicitly controls the exploration-exploitation trade-off through the parameter κ , offering flexibility but requiring careful tuning.

In this project the UCB acquisition function is chosen. This is the case, since an exhaustive exploration of the hyperparameter space is not required, instead it is desired to implement an acquisition function which will exploit a proven set of hyperparameters once such a set has been observed. This excludes the EI acquisition function, since this acquisition function explicitly assigns value to exploration, which might in turn lead to poor occupant comfort. The choice between PI and UCB came down to how the exploration parameter κ is used in each formula, where it is evaluated that the implementation in UCB is more intuitive. As described, the formulations are described in the context of maximization. To convert the UCB to minimization, and therefore Lower Confidence Bound (LCB), the formulation instead becomes as shown in (8.11):

$$LCB(x) = \mu_{f^*}(x) - \kappa \sqrt{\sigma_{f^*}^2(x)} \quad (8.11)$$

8.4 Implementation

The BO in this project will be a single-objective Bayesian optimizer, meaning that the KPIs will be summed during evaluation of a proposed set of hyperparameter values. The BO is implemented using the Scikit-Optimize library [49]. As shown, the limited memory Broyden–Fletcher–Goldfarb–Shanno optimization algorithm is chosen as the optimizer in the BO implementation in this project due to its popularity and availability in the Scikit library [47, p.679]. The implementation is shown in Snippet 8.1

```

1  kernel = "RBF" # Define the kernel
2  gp = GaussianProcessRegressor(kernel=kernel) # Create a Gaussian Process Regressor
3  result = gp_minimize(MPC,
4                      param_space,
5                      acq_func="LCB",
6                      acq_optimizer="lbfgs",
7                      kappa=1,
8                      n_initial_points=3,
9                      n_calls=(int(final_sim_days -
10                                ((initialIters*paramTuningInterval)+1))/paramTuningInterval),
11                      x0=[obs[0] for obs in observations],
12                      y0=[obs[1] for obs in observations])

```

Snippet 8.1. nRnC model equation with integrated Inertia.

In this code, the *observations* contains the observed points $(x_i, y_i)^N$. Neither the parameter σ_ϵ^2 in the GP noisy measurement assumption in (8.1) or the σ_f^2 from the RBF kernel definition in (8.6) will be specified here, since the Scikit library implementation is able to estimate these values from the observations automatically. Similarly the length scale ℓ is also determined by the library, since a manual tuning of this parameter requires many observations. In a full implementation, parameters like these could be estimated after first running the system for a while, since a good ℓ parameter value is often found through fitting the surrogate model to historical observations. As mentioned, the BO implementation will input hyperparameter values to the MPC controlling the simulated buildings in this project, after which the sum of the KPIs described in Table 3.1 will be calculated and returned to the BO.

8.4.1 Determining κ

With this decision made, four **SimpleHouseRad** building simulations with the MPC using the improved nRnC and different κ parameter values for the UCB acquisition function are run for 80 days from January 1st, to investigate the impact of this tuning parameter. Table 8.1

Simulation run	κ	R	θ_{PBR}	θ_{under}	θ_{over}	θ_Δ	Range of allowed C and C_{delta} values
#1	0.1	500	1	4	2	1	[0.0, 2000.0]
#2	0.5	500	1	4	2	1	[0.0, 2000.0]
#3	1.0	500	1	4	2	1	[0.0, 2000.0]
#4	2.0	500	1	4	2	1	[0.0, 2000.0]

Table 8.1. Parameter values used during test of different κ values.

The simulation is made to gather data over three days at every suggested hyperparameter value, before analyzing the performance and producing KPI values. To initialize the BO, first some initial points have to be manually tested to initialize the GP regression. For this purpose, three days with a C and C_{delta} value of 1900 is run, followed by three days with a value of 950, and finally three days with a value of 0. After this, the BO algorithm controls the value of the hyperparameters for the remainder of the simulation. The κ parameter value is made to vary from the 0.1 to 2, since these values represent small and large κ values. The results observed can be seen in Figure 8.2, Figure 8.3, Figure 8.4 and Figure 8.5

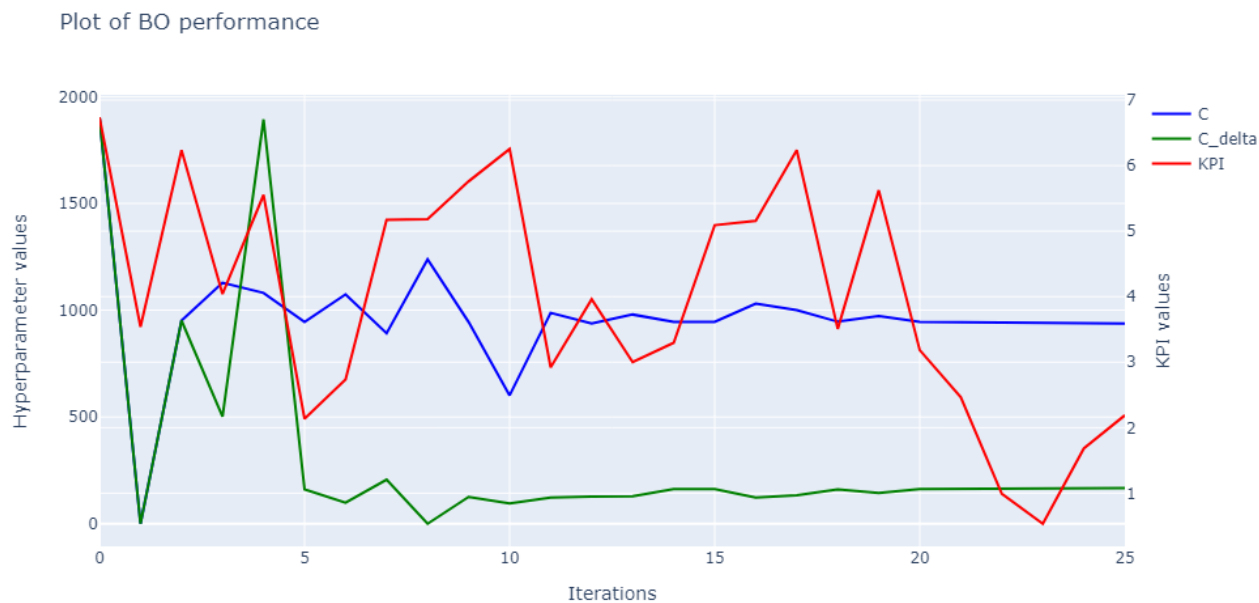


Figure 8.2. Performance with a $\kappa = 0.1$. As observed, the hyperparameters tend towards stable values after roughly 8 iterations.

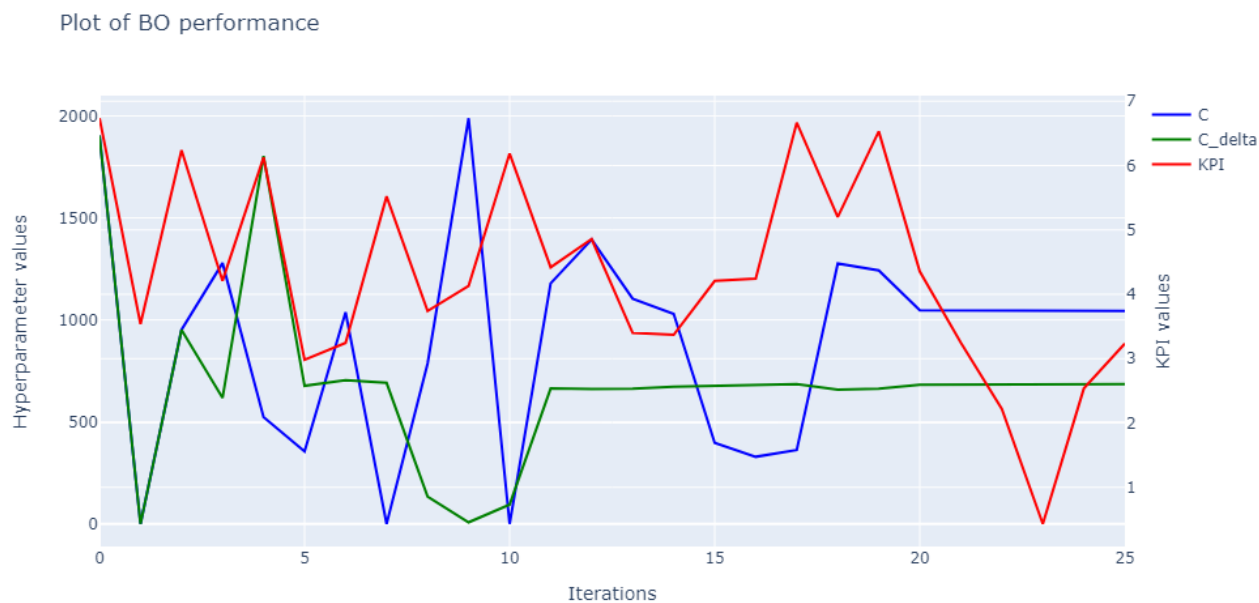


Figure 8.3. Performance with a $\kappa = 0.5$. As observed the C_{Δ} hyperparameter tends toward a stable value after roughly 12 iterations, while C is not stable until iteration 20.

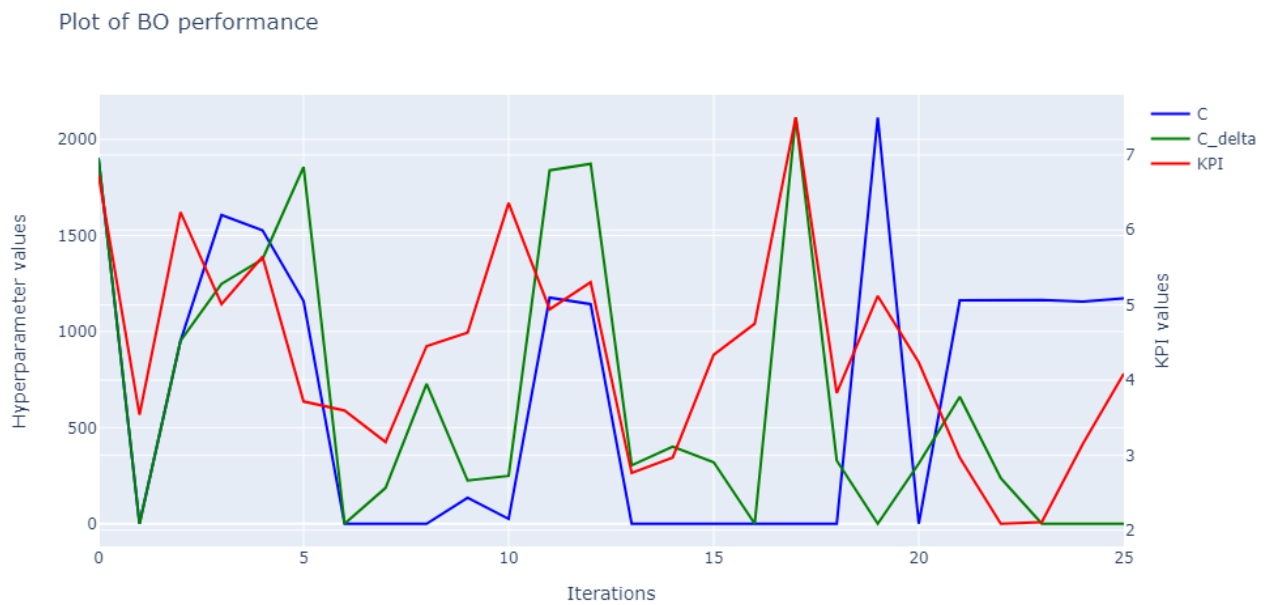


Figure 8.4. Performance with a $\kappa = 1$. As observed, the hyperparameters does not stabilize for longer periods, but does have short periods of stability.

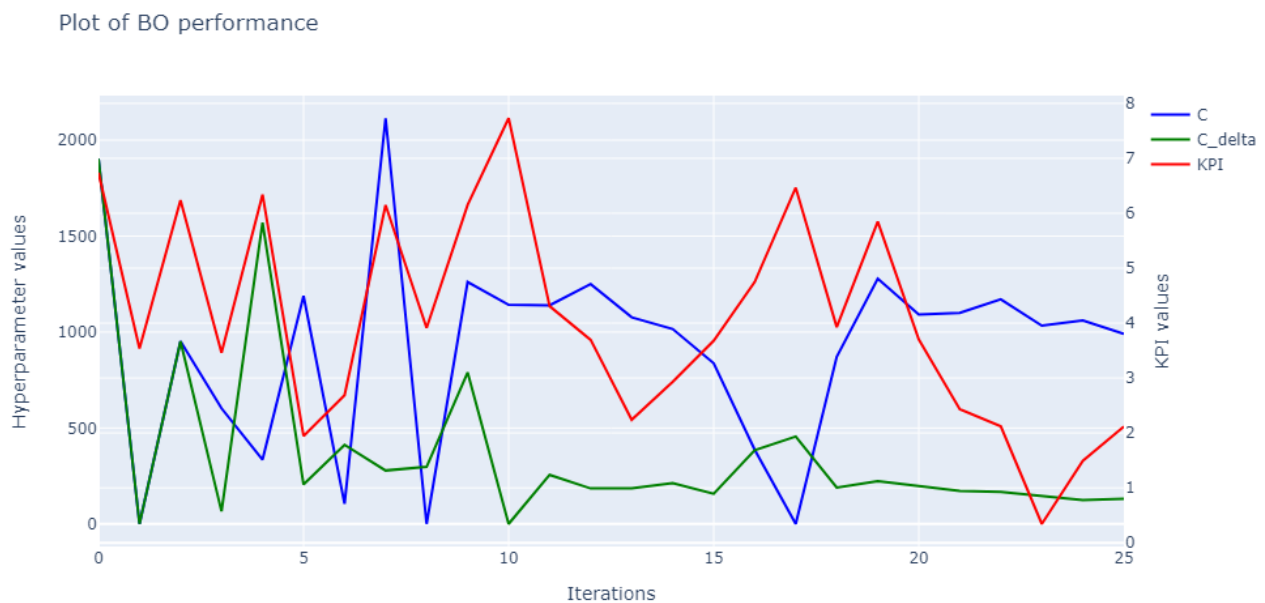


Figure 8.5. Performance with a $\kappa = 2$. It can be observed that the hyperparameters seem to stabilize after roughly 20 iterations.

To evaluate the results, the sum of KPIs in each simulation is found and shown in Table 8.2. The value of this sum can be seen as the overall penalty during the 80 day simulation, and therefore a good indicator of performance.

κ value	0.1	0.5	1	2
Sum of KPI values	103.172	111.57	113.12	104.936

Table 8.2. Table showing the KPI value summed over all iterations at different values of κ .

As seen, the variation over iterations in the proposed hyperparameter values increase, as the value of κ does. In all the simulations except Figure 8.3, it appears that the hyperparameters stabilize at a value of C of around 1000, and a low C_{delta} of around 100. Based on the inconclusive results shown in Table 8.2, a κ value of **1** is chosen. This choice is made, since no value of κ clearly stood out as the best in this test. Therefore a choice of a high or low κ value would be arbitrary, where low values potentially cause the BO to avoid exploration of better hyperparameters given changes to the ambient conditions, and higher values might be exceedingly sensitive to noise in the KPI functions in Section 3.2.

Part III

Evaluation

9 | Acceptance tests

In this chapter the tests described in Section 4.4 will be performed. For each test, first the setup of the test is described, after which the results of the test is shown for each of the three buildings described in Section 3.1. Finally a brief evaluation of the results is given. For the tests related to simulation of a building with MPC, plots of the performance during the same four days in each simulation is shown in Appendix B, to enable visual inspection of the performance.

9.1 Test of Req. 1

The following test results relate to the requirement:

Compared to the original nRnC model introduced in Chapter 2, the model(s) must demonstrate improved predictive performance when used on training data

9.1.1 Setup

To perform this test, the procedure described in (4.1) is used. The results of this procedure are shown, in order to investigate the predictive performance of the models on training data. A fourth trace is included, which serves as a baseline and demonstrates the prediction errors obtained if the initial condition value is unchanged and no model is utilized. The model order determined with Section 6.2 and the size of the PINN network determined with Section 7.3.1 in each building are shown in Table 9.1

Parameter Building	Model order	Network width	Network depth
SimpleHouseRad	1	12	2
SimpleHouseRSla	1	7	2
SwissHouseRSla	0	7	2

Table 9.1. Chosen nRnC model order, network size for each building in the tests.

9.1.2 Results

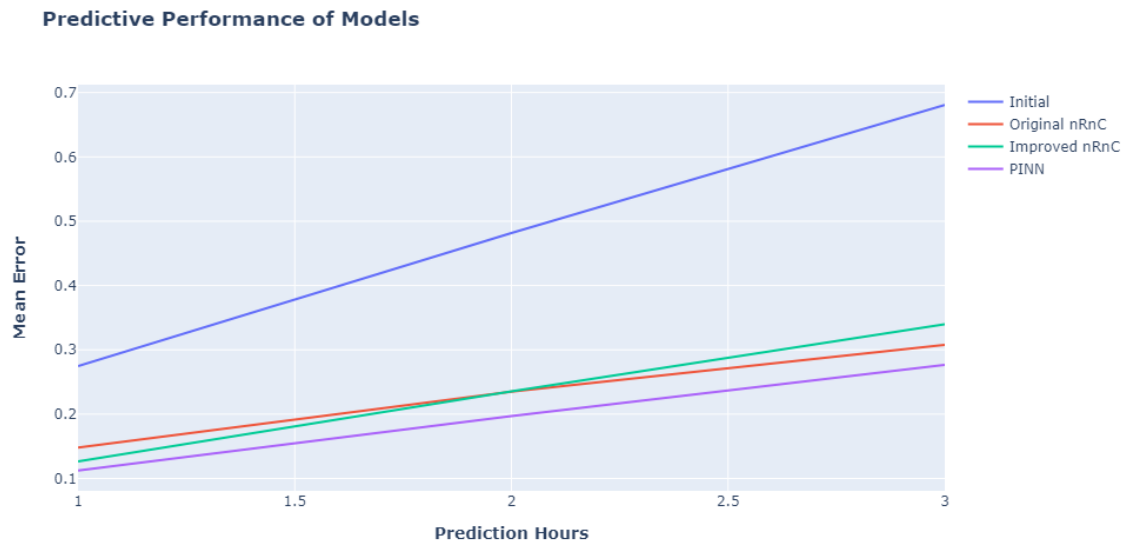


Figure 9.1. Results for "SimpleHouseRad" of the MAE test, for all three models including a fourth initial condition trace.

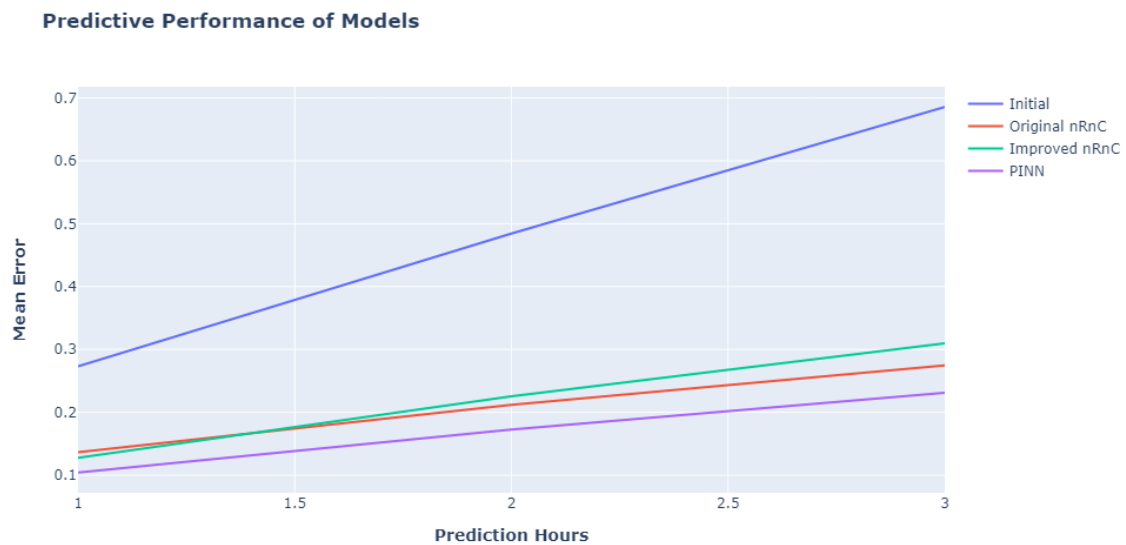


Figure 9.2. Results for "SimpleHouseRSla" of the MAE test, for all three models including a fourth initial condition trace.

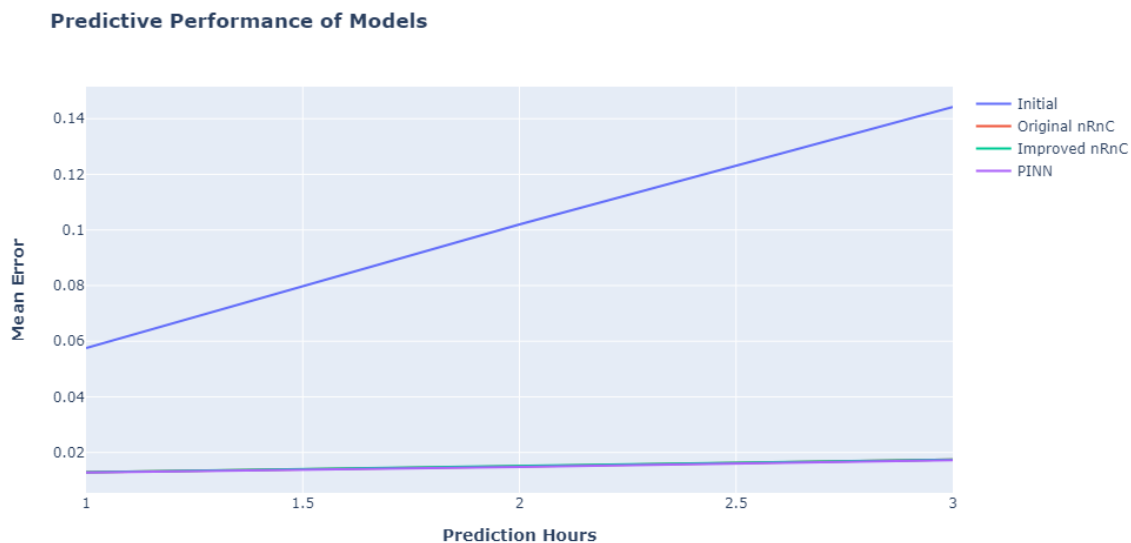


Figure 9.3. Results for "**SwissHouseRSla**" of the MAE test, for all three models including a fourth initial condition trace.

9.1.3 Evaluation

As seen, the predictive performance of the original and improved nRnC model are very similar across all the tests. However, the PINN demonstrates superior predictive performance in this test for the **SimpleHouseRad** and **SimpleHouseRSla** buildings. The results obtained for **SwissHouseRSla** are less conclusive, where each model attains almost exactly the same predictive performance. Therefore it is evaluated that the requirement is fulfilled for the **SimpleHouseRad** and **SimpleHouseRSla** buildings, but failed for **SwissHouseRSla**.

9.2 Test of Req. 2

The following test results relate to the requirement:

Compared to the original nRnC model introduced in Chapter 2, the model(s) must demonstrate improved key performance indicator values when used in building simulation with MPC when compared to the original nRnC introduced in Chapter 2

9.2.1 Setup

To perform this test, each building is simulated with each model without BO enabled. The hyperparameters H_p , C and C_Δ were therefore manually tuned. The values chosen are shown in Table 9.2. Furthermore, the models used for this test are the same as those in the first test, and therefore the order of the improved nRnC model, as well as the network size of the PINN can be seen in Table 9.1.

Parameter Building	H_p	C	C_Δ	R	ξ
SimpleHouseRad	8	400	400	500	10
SimpleHouseRSla	8	100	100	500	10
SwissHouseRSla	8	0	0	500	10

Table 9.2. Chosen hyperparameters for MPC across all building models.

9.2.2 Results

Model \ KPI	PIR	PBR	Energy [J]	Energy $_\Delta$	Energy $_\Delta^2$	Time [HH:MM:ss]
Original nRnC	68.08%	1.48%	3.35 E7	1.2 E6	8.29 E8	01:02:49
Improved nRnC	48.53%	0.52%	4.07 E7	1.52 E6	5.28 E8	01:08:58
PINN	47.17%	0.47%	4.06 E7	1.08 E6	4.96 E8	08:26:10

Table 9.3. KPIs obtained for the building model **SimpleHouseRad** for all three models, including their simulation duration.

Model \ KPI	PIR	PBR	Energy [J]	Energy $_\Delta$	Energy $_\Delta^2$	Time [HH:MM:ss]
Original nRnC	52.8%	3.21%	3.38 E7	7.41 E5	1.8 E8	00:51:23
Improved nRnC	60.31%	1.1%	3.38 E7	9.85 E5	3.57 E8	01:05:56
PINN	42.08%	0.75%	3.69 E7	1.51 E6	1.04 E9	08:10:02

Table 9.4. KPIs obtained for the building model **SimpleHouseRSla** for all three models, including their simulation duration.

Model \ KPI	PIR	PBR	Energy [J]	Energy Δ	Energy Δ^2	Time [HH:MM:ss]
Original nRnC	23.79%	4.13%	7.3 E6	6.54 E5	4.18 E8	00:50:25
Improved nRnC	25.77%	1.87%	6.98 E6	7.32 E5	5.9 E8	00:50:40
PINN	26.92%	0.00%	6.09 E6	1.35 E6	2.4 E9	12:05:57

Table 9.5. KPIs obtained for the building model **SwissHouseRSla** for all three models, including their simulation duration.

9.2.3 Evaluation

As observed, the KPIs obtained through simulating the MPC with all three models on all three buildings, the *PBR* have improved compared to the original nRnC model for all PINN and improved nRnC models. This indicates, that the improved nRnC and the PINN model both have an increased ability to accurately include the inertia of the buildings in the model. However, the remaining KPIs show very similar performance, compared to the original nRnC model, and therefore it is concluded that the requirement is fulfilled for all buildings.

9.3 Test of Req. 3

The following test results relate to the requirement:

The system must demonstrate improved key performance indicator values when compared with the same system utilizing constant hyperparameter values during building simulation with MPC

9.3.1 Setup

To perform this test, each building is simulated with the BO algorithm enabled. The results obtained are then compared with the simulations from Section 9.2, where the BO algorithm was disabled. For ease of comparison, the results are therefore repeated in the tables in this section. As mentioned in Section 3.2, weighting parameters θ were present in the KPIs used for BO penalty signals. The values these were set to are shown in Table 9.6. Once again, the order of the improved nRnC model and the network size of the PINN can be seen in Table 9.1.

Parameter Building	θ_{PBR}	θ_{under}	θ_{over}	θ_{Δ}	R	H_p	ξ	Days per iteration.
SimpleHouseRad	1	4	2	1	500	8	10	3
SimpleHouseRSIa	1	4	2	1	500	8	10	3
SwissHouseRSIa	1	4	2	1	500	8	10	3

Table 9.6. Chosen hyperparameters across all building models.

9.3.2 Results

Model \ KPI	PIR	PBR	Energy [J]	Energy $_{\Delta}$	Energy $_{\Delta}^2$	Time [HH:MM:ss]
nRnC without BO	44.16%	0.25%	4.11 E7	1.15 E6	6.18 E8	01:08:58
PINN without BO	47.17%	0.47%	4.06 E7	1.08 E6	4.96 E8	08:26:10
nRnC with BO	46.11%	1.46%	4.02 E7	1.35E6	1.36 E9	00:53:07
PINN with BO	47.06%	1.28%	4.02 E7	1.93 E6	3.73 E9	08:57:25

Table 9.7. KPIs obtained for the building model **SimpleHouseRad** for the PINN and improved nRnC model with BO enabled and disabled, including their simulation duration.

Model \ KPI	PIR	PBR	Energy [J]	Energy $_{\Delta}$	Energy $_{\Delta}^2$	Time [HH:MM:ss]
nRnC without BO	60.31%	1.1%	3.38 E7	9.85 E5	3.57 E8	01:05:56
PINN without BO	42.08%	0.75%	3.69 E7	1.51 E6	1.04 E9	08:10:02
nRnC with BO	53.37%	5.38%	3.39 E7	1.11 E6	1.08 E9	00:53:07
PINN with BO	32.83%	0.45%	3.91 E7	1.57 E6	1.28 E9	11:24:15

Table 9.8. KPIs obtained for the building model **SimpleHouseRSIa** for the PINN and improved nRnC model with BO enabled and disabled, including their simulation duration.

Model \ KPI	PIR	PBR	Energy [J]	Energy $_{\Delta}$	Energy $_{\Delta}^2$	Time [HH:MM:ss]
nRnC without BO	25.77%	1.87%	6.98 E6	7.32 E5	5.9 E8	00:50:40
PINN without BO	26.92%	0.00%	6.09 E6	1.35 E6	2.4 E9	12:05:57
nRnC with BO	23.57%	35.19%	4.3 E6	2.92 E5	1.67 E8	00:29:15
PINN with BO	22.33%	34.11%	3.97 E6	3.96 E5	6.88 E8	08:02:19

Table 9.9. KPIs obtained for the building model **SwissHouseRS1a** for the PINN and improved nRnC model with BO enabled and disabled, including their simulation duration.

9.3.3 Evaluation

As observed, the inclusion of BO unfortunately does not improve the performance of the MPC with neither the improved nRnC or the PINN model for any building. Therefore this requirement is failed.

10 | Discussion

10.1 Test results

This section will describe points of discussion for the test results shown in Chapter 9.

10.1.1 SwissHouseRSla

When manually tuning the hyperparameters for this building, it became evident that this building was very different from the others. This could be due to the Swiss "*Minergie*" building regulations used when constructing this building model. Upon inspection of the training data, it became clear that this building required little energy, and reacted quickly to heating input. Because of this, the statistical method chosen for obtaining a value with which to scale the data described in Section 5.5 caused some problems. Specifically, the cost associated with energy input and changes to the energy input in the cost function described in (5.1) became large. This is the reason for choosing $C = 0$ and $C_{\Delta} = 0$ as tuning parameters for this building. While unfortunate, this provided an opportunity to illustrate the advantage of the BO algorithm in Figure 10.1. As seen, the BO algorithm is able to eventually reach the value where the MPC was able to provide acceptable heating inputs. Overall, the characteristics of this building made it difficult to evaluate the performance of the models and BO.

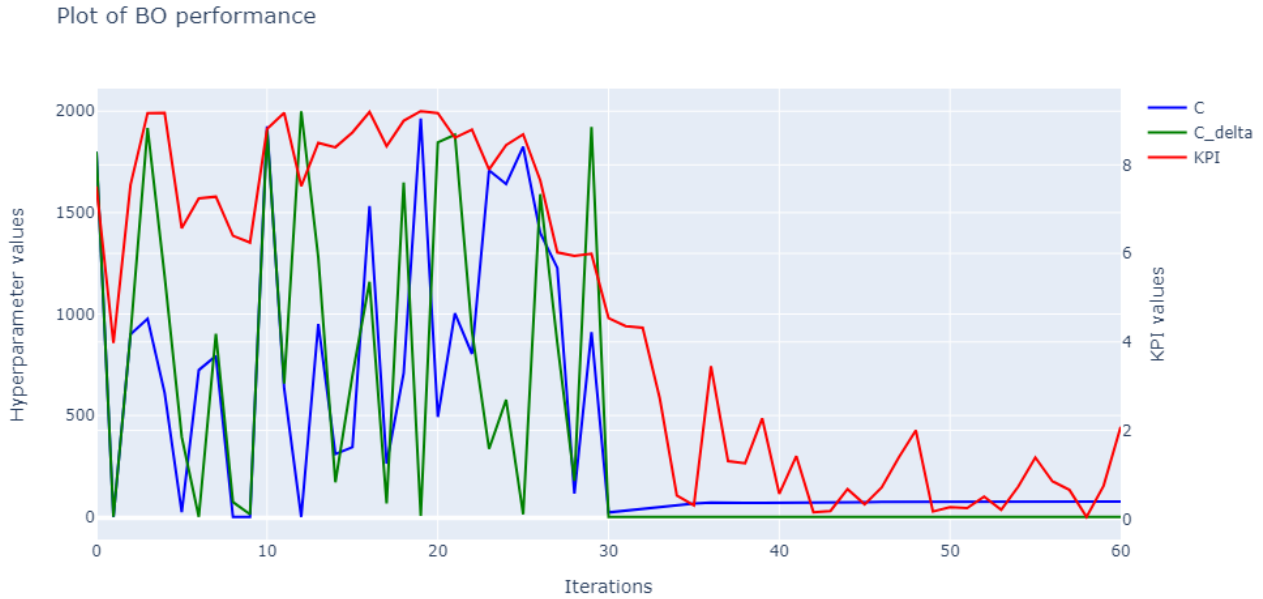


Figure 10.1. Trajectory for hyperparameters and KPIs from Section 3.2.

10.1.2 Test of Req. 2

When comparing the results obtained with the original nRnC with those obtained with the improved nRnC and PINN models, an improvement in performance was observed. However for PIR specifically, it can be observed that the original nRnC sometimes performed better than the other models. Upon inspection, it became clear that this result is related to the increased PBR of this model, which caused the trajectory of the temperature to enter the range later, and therefore accrue more time in range before exceeding the upper limit.

10.1.3 Hyperparameter space

When manually tuning the hyperparameters for all building models in Chapter 9, the values chosen for each building were quite distinct from one another. Thus, the choice of defining the space of possible hyperparameters values for online tuning as $[0.0, 2000.0]$ for all the buildings might have been a bad decision. This is the case, since the exploration performed by the BO would have to search a large range of unnecessary values which are likely to cause poor performance for the *SimpleHouseRSla* and *SwissHouseRSla* buildings.

10.1.4 Computation Time

The developed PINN model provided an increase to predictive performance, even when compared to the improved nRnC, demonstrating a consistently lower PBR in the tests described in Chapter 9. However, when weighing the computation time required to obtain these results with the comparatively lower computation time for the improved nRnC, the need for a faster PINN is evident. For example, automatic differentiation techniques can be used to find the gradients of the PINN and cost function in the MPC, in order to utilize gradient descent optimization.

10.2 Improved nRnC model

This section will describe points of discussion for the adjustments made to the original nRnC model introduced in Chapter 6.

10.2.1 Choice of P_{init}

As described in Section 6.5, a choice was made to only select every $\frac{1}{P_{init}} = H$ 'th sample as an initial condition during training, and constraints on specific values which would degenerate the training process were derived. However, when considering the $H = 100$ value chosen in this project and the sampling interval of 15 minutes ($S_{pd} = 96$), it becomes clear that only a difference of one hour will pass between the instances chosen as initial condition over two consecutive days. Due to this, the initial condition time during a day will have a period of 24 days. While this choice still causes some misalignment, as desired, a number which is further from 96 might have been better.

10.2.2 Impact of delayed outputs

As described in Section 6.2, the inclusion of previous outputs was motivated by the poor performance of the original nRnC model on buildings with dynamics that feature a significant amount of inertia. The inclusion provided little, if any, benefit to predictive performance on training data, when analyzed with the test described in (4.1). However, when utilizing the improved nRnC model as the predictive model in MPC for the acceptance tests, the improved nRnC indeed demonstrated improved ability to reach the reference at the start of the occupied period. Furthermore, as shown in Table 6.7 the improved nRnC model demonstrated drastically improved performance when used for one-step prediction.

10.3 Physics-informed neural network

This section will describe points of discussion for the PINN described in Chapter 7.

10.3.1 Physics data generation strategy

The method utilized for generating PINN training data based on statistically generated input samples and a trained nRnC model shown in (7.2) and Figure 7.3 proved effective for producing a generalizable model, especially when compared to the attempt with a MLP in Figure 2.3. However, for some features this approach might have to be amended slightly. Specifically in this project, it was observed that the range of energy input values obtained with (7.2) for some buildings included negative values, due to a relatively low mean and high variance. Upon further investigation, it was discovered that the many days during summer, which often required zero heating, impacted the mean value obtained significantly. Especially for the *SwissHouseRSla* building this effect was noticeable, due to the very low average energy consumption. To avoid such effects, the statistical values for energy consumption could be found based on a dataset, where zero and near-zero entries of energy consumption are removed.

10.3.2 Effect of local minima during training

When training the PINN model repeatedly on the same data, the method of dynamically adjusting the network depth and width described in Section 7.3.1 was observed to sometimes cause the training to terminate prematurely. A proposed cause for this might be, that the training enforced weights which overfitted the model to the training data, at the cost of predictive performance on the verification data, which triggers the early stopping mechanism.

10.4 Hyperparameter tuning

This section will describe points of discussion for the Bayesian optimizer described in Chapter 8.

10.4.1 Online tuning of BO parameters

Future work for the BO could be the implementation of a mechanism to tune the strategy chosen for trade-off between exploration and exploitation in the BO algorithm. One proposed strategy could be to analyze the variance of the recent ambient conditions, and based on the analysis prioritize either exploitation or exploration.

10.4.2 Effect of noise on BO

The variability in the weather is currently a problem for the BO in this project, since the BO cannot discern between sudden changes in weather which negatively effect the KPI values, and a bad choice of parameter values. Specifically, the noise $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$ might be too large, and moreover not normally distributed around zero. To amend this behaviour, one possible strategy could be to save separate datasets $(x_i, y_i)^N$ in the BO based on discretization of the ambient conditions. With this method, a historical dataset of similar conditions could be used for creating more relevant surrogate models. The choice of surrogate model would then come down to the weather forecast.

11 | Conclusion

This project addresses the following problem statement:

How can the data-driven building modelling for MPC described in Chapter 2 be further developed, and how can performance indicators be leveraged for online tuning of MPC hyperparameters?

To answer the first part of this problem statement, the nRnC modelling approach described in Chapter 2 was further developed. The development included the inclusion of delayed outputs in the model, a revised training method which emphasized efficiency during training, as well as a change to the method used for scaling during data preparation. Furthermore, a physics-informed neural network was developed, which utilizes the trained nRnC model to generate training data. When training the PINN model, the loss was obtained from a combination of one-step prediction of the generated nRnC data and recursive prediction of the original data used for training the nRnC model. To train the model based on the obtained loss, automatic differentiation was utilized to obtain error gradients.

To answer the second part of this problem statement, a Bayesian optimizer was implemented for online tuning of the MPC hyperparameters. The problem was specified as a black-box optimization problem, where the Bayesian optimizer received performance signals generated based on analysis of the MPC performance at a chosen set of proposed hyperparameters values.

The developed models were tested against the nRnC model described in Chapter 2 in simulation three Modelica building models. The tests for the models showed improved performance for both the improved nRnC model and the physics-informed neural network across multiple building models. The developed hyperparameter tuning system was tested against simulation models implementing manually tuned constant hyperparameter values. The test for the hyperparameter tuning system did not provide an increase in performance, where it was observed that the effect of concurrent ambient conditions was often misinterpreted in the hyperparameter tuning system. Overall, the models developed in this project provided good results, however the improved performance gained from utilizing the PINN model instead of the improved nRnC might not outweigh the significant increase in training time and computational effort demonstrated when used with MPC. Furthermore, the developed hyperparameter tuning system did not provide satisfactory results, and further development is required before implementation on real buildings can be made.

Bibliography

- [1] Buildings – breakthrough agenda report 2023 – analysis - iea. <https://www.iea.org/reports/breakthrough-agenda-report-2023/buildings>, 2023. (Accessed on 26/02/2024).
- [2] The european green deal - european commission. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en. (Accessed on 26/03/2024).
- [3] Digital codes. <https://codes.iccsafe.org/content/IECC2021P2>. (Accessed on 03/28/2024).
- [4] Standard 90.1. <https://www.ashrae.org/technical-resources/bookstore/standard-90-1>. (Accessed on 03/28/2024).
- [5] De Paola A, Andreadou N, and Kotsakis E. Clean energy technology observatory: Smart grids in the european union - 2023 status report on technology development trends, value chains and markets. (KJ-NA-31-673-EN-N (online)), 2023. ISSN 1831-9424 (online). doi: 10.2760/237911(online). URL <https://publications.jrc.ec.europa.eu/repository/handle/JRC134988>.
- [6] Three essential elements of next generation building management systems (bms) — tmbsa. <https://www.tmbsa.com/blog/three-essential-elements-of-next-generation-building-management-systems-bms>. (Accessed on 04/01/2024).
- [7] MEP Academy Instructor. Basic hvac controls - mep academy. <https://mepacademy.com/basic-hvac-controls/>, 07 2022. (Accessed on 04/11/2024).
- [8] Use of energy in commercial buildings - u.s. energy information administration (eia). <https://www.eia.gov/energyexplained/use-of-energy/commercial-buildings.php>, 06 2023. (Accessed on 04/11/2024).
- [9] Mohamad Fadzli Haniff, Hazlina Selamat, Rubiyah Yusof, Salinda Buyamin, and Fatimah Sham Ismail. Review of hvac scheduling techniques for buildings towards energy-efficient and cost-effective operations. *Renewable & sustainable energy reviews*, 27:94–103, 2013. ISSN 1364-0321. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_fao_agris_US201500126932.
- [10] Jan M. Maciejowski. *Predictive control : with constraints*. Prentice Hall, Harlow, 2002. ISBN 9780201398236.
- [11] D. Mariano-Hernández, L. Hernández-Callejo, A. Zorita-Lamadrid, O. Duque-Pérez, and F. Santos García. A review of strategies for building energy management system: Model predictive control, demand side management, optimization, and fault detect & diagnosis. *Journal of Building Engineering*, 33:101692–, 2021. ISSN 2352-7102. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_crossref_primary_10_1016_j_job_2020_101692.

- [12] Ento. about us. <https://www.ento.ai/about/company>, . (Accessed on 09/13/2023).
- [13] Ento control. optimising hvac systems. <https://www.ento.ai/product/ento-control>, . (Accessed on 04/18/2024).
- [14] Uri M. Ascher, Lind R. Petzold, Uri M. Ascher, Linda Ruth. Petzold, and Linda R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, Philadelphia, 1998. ISBN 0898714125. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9920620936105762.
- [15] Kasper Vinther et al. Predictive control of hydronic floor heating systems using neural networks and genetic algorithms. *IFAC-PapersOnLine*, 50(1):7381–7388, 2017. ISSN 2405-8963. doi: <https://doi.org/10.1016/j.ifacol.2017.08.1477>. URL <https://www.sciencedirect.com/science/article/pii/S2405896317320542>. 20th IFAC World Congress.
- [16] Elliott Skomski, Ján Drgoňa, and Aaron Tuor. Automating discovery of physics-informed neural state space models via learning and evolution. In *Proceedings of Machine Learning Research*, volume 144, pages 980–991, 2021. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_scopus_primary_2_s2_0_85126880554.
- [17] Energyplus. <https://energyplus.net/>. (Accessed on 04/29/2024).
- [18] Michael Wetter, Wangda Zuo, Thierry S. Noudui, and Xiufeng Pang. Modelica buildings library. *Journal of building performance simulation*, 7(4):253–270, 2014. ISSN 1940-1493. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_webofscience_primary_000329775000002.
- [19] Paul Scharnhorst, Baptiste Schubnel, Carlos Fernández Bandera, Jaume Salom, Paolo Taddeo, Max Boegli, Tomasz Gorecki, Yves Stauffer, Antonis Peppas, and Chrysa Politi. Energym: A building model library for controller benchmarking. *Applied sciences*, 11(8):3518–, 2021. ISSN 2076-3417. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_doaj_primary_oai_doaj_org_article_de59a13a317d409c8efd89184ab8dc89.
- [20] Simplehouserad — energym 2020.02.19 documentation. <https://bs1546.github.io/energym-pages/sources/houserad.html>, . (Accessed on 08/04/2024).
- [21] Simplehousersla — energym 2020.02.19 documentation. <https://bs1546.github.io/energym-pages/sources/houseslab.html>, . (Accessed on 08/04/2024).
- [22] Panos M Pardalos, Varvara Rasskazova, and Michael N Vrahatis. Black-box optimization: Methods and applications. In *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, volume 170 of *Springer Optimization and Its Applications*, pages 35–65. Springer International Publishing AG, Switzerland, 2021. ISBN 9783030665142. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_springer_books_10_1007_978_3_030_66515_9_2.
- [23] Bryan Orr. Short-cycling - hvac school. <https://hvacschool.com/short-cycling/>, 05 2019. (Accessed on 04/23/2024).

- [24] Tor Aksel N. Heirung, Joel A. Paulson, Jared O’Leary, and Ali Mesbah. Stochastic model predictive control — how does it work? *Computers & chemical engineering*, 114:158–170, 2018. ISSN 0098-1354. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_scopus_primary_2_s2_0_85033553426.
- [25] Zhe Wang. How frequent should we measure the indoor thermal environment. *Building and environment*, 222:109464–, 2022. ISSN 0360-1323. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_elsevier_sciencedirect_doi_10_1016_j_buildenv_2022_109464.
- [26] Chris Hedges. Underfloor heating vs radiators – what’s better?, Jan 2022. URL <https://www.thermosphere.com/blog/underfloor-heating-vs-radiators/>. (Accessed on 05/01/2024).
- [27] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv.org*, 2017. ISSN 2331-8422. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_arxiv_primary_1711_10566.
- [28] John Paul. Mueller and Luca Massaron. *Deep learning. For dummies. For Dummies*, Hoboken, New Jersey, 1st edition edition, 2019. ISBN 1-119-54303-7. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9920848396805762.
- [29] Charu C Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing AG, Cham, second edition edition, 2023. ISBN 9783031296413. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_askewsholts_vlebooks_9783031296420.
- [30] Florian Arnold and Rudibert King. State–space modeling for control based on physics-informed neural networks. *Engineering applications of artificial intelligence*, 101:104195–, 2021. ISSN 0952-1976. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_webofscience_primary_000672478100004.
- [31] J. Mazumdar and R.G. Harley. Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents. *IEEE transactions on industrial electronics (1982)*, 55(9):3484–3491, 2008. ISSN 0278-0046. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_proquest_miscellaneous_903637716.
- [32] Pytorch. URL <https://pytorch.org/>. (Accessed on 05/07/2024).
- [33] torch.tensor.backward — pytorch 2.3 documentation. URL <https://pytorch.org/docs/stable/generated/torch.Tensor.backward.html>. (Accessed on 05/06/2024).
- [34] H. Martin. Bücker, George. Corliss, Paul. Hovland, Uwe. Naumann, and Boyana. Norris. *Automatic Differentiation: Applications, Theory, and Implementations*. Lecture Notes in Computational Science and Engineering, 50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 2006. edition, 2006. ISBN 1-280-46094-6. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9920773368205762.

- [35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv.org*, 2017. ISSN 2331-8422. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_arxiv_primary_1412_6980.
- [36] torch.optim.lr_scheduler — pytorch 2.3 documentation. https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#ReduceLRonPlateau. (Accessed on 05/20/2024).
- [37] Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv.org*, 2019. ISSN 2331-8422. URL <https://www.proquest.com/docview/2325614336?pq-origsite=primo&sourcetype=Working%20Papers>.
- [38] Ali Kaveh and Taha Bakhshpoori. *Metaheuristics*. Springer International Publishing AG, Cham, 1 edition, 2019. ISBN 9783030040666. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_proquest_ebookcentral_EBC5742727.
- [39] Aleksandar Lazinica and Alex Lazinica. *Particle swarm optimization*. IntechOpen, Rijeka, Croatia, 2009. ISBN 953-51-5747-7. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9920813822205762.
- [40] Nilanjan. Dey. *Applied Genetic Algorithm and Its Variants Case Studies and New Developments*. Springer Tracts in Nature-Inspired Computing. Springer Nature Singapore, Singapore, 1st ed. 2023. edition, 2023. ISBN 981-9934-28-1. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9921465001705762.
- [41] Ky Khac Vu, Claudia D’Ambrosio, Youssef Hamadi, and Leo Liberti. Surrogate-based methods for black-box optimization. *International transactions in operational research*, 24(3):393–424, 2017. ISSN 0969-6016. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_crossref_primary_10_1111_itor_12292.
- [42] Antonio Candelieri. A gentle introduction to bayesian optimization. In *2021 Winter Simulation Conference (WSC)*, volume 2021-, pages 1–16. IEEE, 2021. ISBN 9781665433112. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_ieee_primary_9715413.
- [43] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *Proceedings of Machine Learning Research*, volume 133, pages 3–26, 2020. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_scopus_primary_2_s2_0_85162645190.
- [44] William Edwards, Gao Tang, Giorgos Mamakoukas, Todd Murphey, and Kris Hauser. Automatic tuning for data-driven model predictive control. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2021-, pages 7379–7385. IEEE, 2021. ISBN 1728190770. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_ieee_primary_9562025.
- [45] Peter I Frazier. A tutorial on bayesian optimization. *arXiv.org*, 2018. ISSN 2331-8422. URL https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_arxiv_primary_1807_02811.

- [46] Carl Edward Rasmussen and Christopher K. I Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, Cambridge, 1 edition, 2005. ISBN 9780262256834. URL https://kjdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_skillsoft_books24x7_bks00005224.
- [47] Jungtaek Kim and Seungjin Choi. On local optimizers of acquisition functions in bayesian optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12458 of *Lecture Notes in Computer Science*, pages 675–690. Springer International Publishing, Cham, 2021. ISBN 3030676609. URL https://kjdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/159qapk/cdi_springer_books_10_1007_978_3_030_67661_2_40.
- [48] Peng Liu. *Bayesian Optimization Theory and Practice Using Python*. Apress, Berkeley, CA, 1st ed. 2023. edition, 2023. ISBN 1-4842-9063-1. URL https://kjdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9921395776605762.
- [49] The scikit-optimize contributors. scikit-optimize · pypi. <https://pypi.org/project/scikit-optimize/>, 03 2024. (Accessed on 05/22/2024).
- [50] R. Anand. *Digital Signal Processing : An Introduction*. Mercury Learning and Information, Dulles, Virginia, 2022 - 2022. ISBN 1-68392-801-6. URL https://kjdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n411aj/alma9921457331505762.

Part IV

Appendices

A | Effect of including delayed outputs in nRnC model

To investigate the effect of including past outputs as states in the nRnC model, the impact of these inclusions in a state space formulation of the nRnC will be analyzed. In this formulation, the state vector $\mathbf{x}(k)$, and therefore also the output $y(k)$, will only be the output of the model, $\frac{dT_{int}}{dt}$ at step k . Without any inclusions, the explicit time-invariant discrete time state space representation of the nRnC model is as shown in (A.1)

$$\mathbf{x}(k+1) = \begin{bmatrix} 0 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} \frac{1}{\tau_{amb}} & \frac{1}{\tau_{sun}} & \frac{1}{\tau_{heating}} \end{bmatrix} \begin{bmatrix} T_{amb} - T_{int} \\ W_{sun} \\ W_{heating} \end{bmatrix} \quad (A.1)$$

$$\mathbf{y}(k) = \begin{bmatrix} 1 \end{bmatrix} \mathbf{x}(k)$$

The matrices here reflect the definition of the model from (6.1), where the output is solely defined as a sum of the inputs scaled by the corresponding τ values. As shown, the output of the model $y(k)$ is taken simply as the state $x(k)$, which is the derivative of the internal temperature T_{int} . The state space model for the extended nRnC model shown in (6.2) is defined, by extending the state vector $\mathbf{x}(k)$ of the model to include the π previous values of $x(k)$. The extended state space model of the nRnC model with, for example, $\pi = 2$ previous values of $x(k)$ then becomes the one shown in (A.2). For clarity, the state vector \mathbf{x} will be written out explicitly.

$$\begin{bmatrix} x(k+1) \\ x(k) \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1} & \frac{1}{\tau_2} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ x(k-1) \end{bmatrix} + \begin{bmatrix} \frac{1}{\tau_{amb}} & \frac{1}{\tau_{sun}} & \frac{1}{\tau_{heating}} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} T_{amb} - T_{int} \\ W_{sun} \\ W_{heating} \end{bmatrix} \quad (A.2)$$

$$\mathbf{y}(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ x(k-1) \end{bmatrix}$$

To analyse the impact of the inclusion of the previous inputs, the discrete time transfer function $H(z) = \frac{Y(z)}{U(z)}$ will be derived from the state space model. For brevity during the derivation, the matrices in (A.2) will be substituted as shown in (A.3)

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \quad (A.3)$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k)$$

First the Z-transform is applied [50]

$$z\mathbf{x}(k) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k)$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k)$$

Next, the state vector $\mathbf{x}(k)$ isolated, such that it can be inserted in the output equation $\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k)$. These steps are shown below

$$\begin{aligned} z\mathbf{x}(k) - \mathbf{A}\mathbf{x}(k) &= \mathbf{B}\mathbf{u}(k) \\ (z\mathbf{I} - \mathbf{A})\mathbf{x}(k) &= \mathbf{B}\mathbf{u}(k) \\ \mathbf{x}(k) &= (z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{u}(k) \end{aligned}$$

Inserting this definition of $\mathbf{x}(k)$ into the output equation we obtain

$$\mathbf{y}(k) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{u}(k) \quad (\text{A.4})$$

To invert the term $(z\mathbf{I} - \mathbf{A})^{-1}$, first the matrix $(z\mathbf{I} - \mathbf{A})$ is found

$$(z\mathbf{I} - \mathbf{A}) = \begin{bmatrix} z - \frac{1}{\tau_1} & -\frac{1}{\tau_2} \\ -1 & z \end{bmatrix}$$

The inverse of this matrix is found by using the the method of matrix inversion shown with an arbitrary matrix in (A.5)

$$W^{-1} = \frac{1}{\det(W)} \text{adj}(W) \quad (\text{A.5})$$

Applying this to (A.4) yields

$$(z\mathbf{I} - \mathbf{A})^{-1} = \frac{1}{z^2 - \frac{z}{\tau_1} - \frac{1}{\tau_2}} \begin{bmatrix} z & \frac{1}{\tau_2} \\ 1 & z - \frac{1}{\tau_1} \end{bmatrix}$$

Inserting this into (A.4) and reintroducing the original input and output matrices \mathbf{B} and \mathbf{C} yields

$$\mathbf{y}(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \frac{1}{z^2 - \frac{z}{\tau_1} - \frac{1}{\tau_2}} \begin{bmatrix} z & \frac{1}{\tau_2} \\ 1 & z - \frac{1}{\tau_1} \end{bmatrix} \begin{bmatrix} \frac{1}{\tau_{amb}} & \frac{1}{\tau_{sun}} & \frac{1}{\tau_{heating}} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} T_{amb} - T_{int} \\ W_{sun} \\ W_{heating} \end{bmatrix}$$

Performing the matrix multiplications and multiplying with $\frac{z^{-2}}{z^{-2}}$ to ensure causality yields the desired transfer function, shown in (A.6)

$$H(z) = \frac{z^{-1}(\frac{1}{\tau_{amb}} + \frac{1}{\tau_{sun}} + \frac{1}{\tau_{heating}})}{1 - \frac{z^{-1}}{\tau_1} - \frac{z^{-2}}{\tau_2}} \quad (\text{A.6})$$

As seen, this transfer function has an order of two. It can be derived from this result, and observations regarding the determinant one would obtain from the term $(z\mathbf{I} - \mathbf{A})^{-1}$, that the order of the transfer function obtained from (A.2) would always correspond to the amount of delayed inputs included in the calculation of $\mathbf{y} = \frac{dT_{int}}{dt}$ in this project. For further investigation, the poles of the obtained transfer function are found by applying the quadratic formula on the denominator polynomial. The resulting roots are shown in (A.7)

$$z = \frac{\frac{1}{\tau_1} \pm \sqrt{\frac{1}{\tau_1^2} + \frac{4}{\tau_2}}}{2} \quad (\text{A.7})$$

As seen, the value of τ_1 is very important for the stability of the system, since a τ_1 approaching zero would quickly move both the poles out of the unit circle. Furthermore, a low value of τ_2 will also cause instability, however the constraint in (6.6) prohibits the value of τ_2 from becoming less than τ_1 . From this it can be determined that the system becomes unstable when either τ_1 becomes too small, or both τ_1 and τ_2 become too small. If only τ_1 is considered, the τ_1 value at which a pole of the system lies directly on the unit circle is $\tau_1 = 1$, with values of $\tau_1 < 1$ causing the system to become unstable.

B | Plots of building control performance

In this appendix, plots of the controller performance for the same day across the three models will be shown. Furthermore, for the improved nRnC and the PINN, plots of the performance with MPC implementing BO during those specific days are also shown.

B.1 Performance without BO

B.1.1 Performance on *SimpleHouseRad*

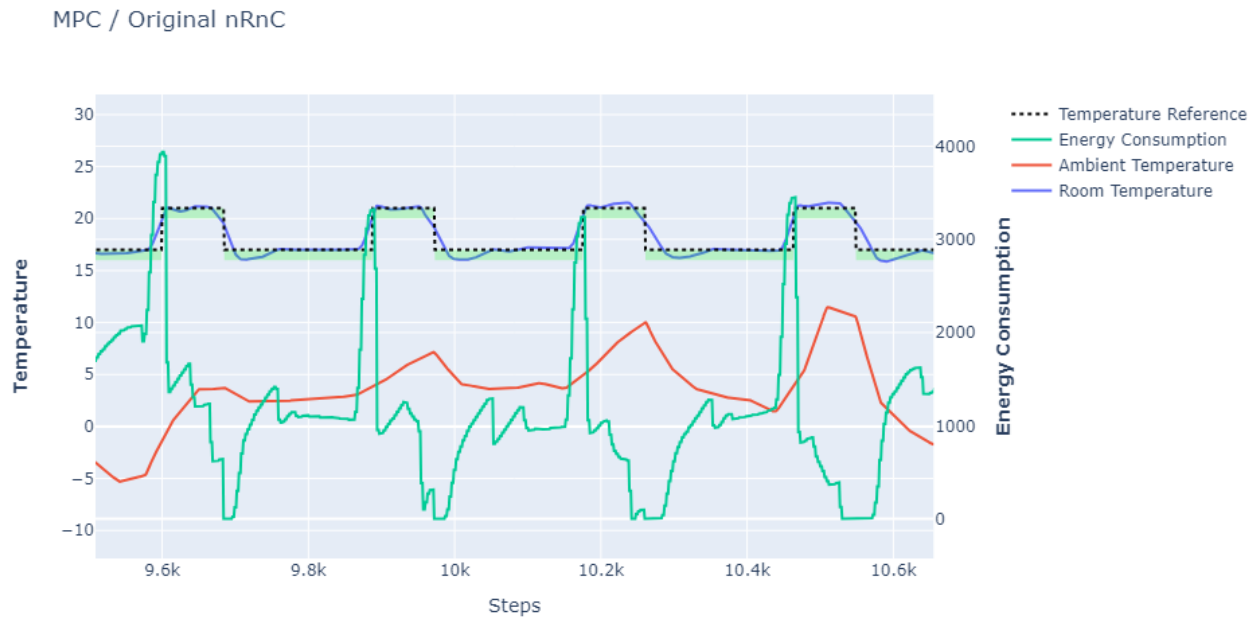


Figure B.1. Results for "SimpleHouseRad" with original nRnC.



Figure B.2. Results for "SimpleHouseRad" with improved nRnC.

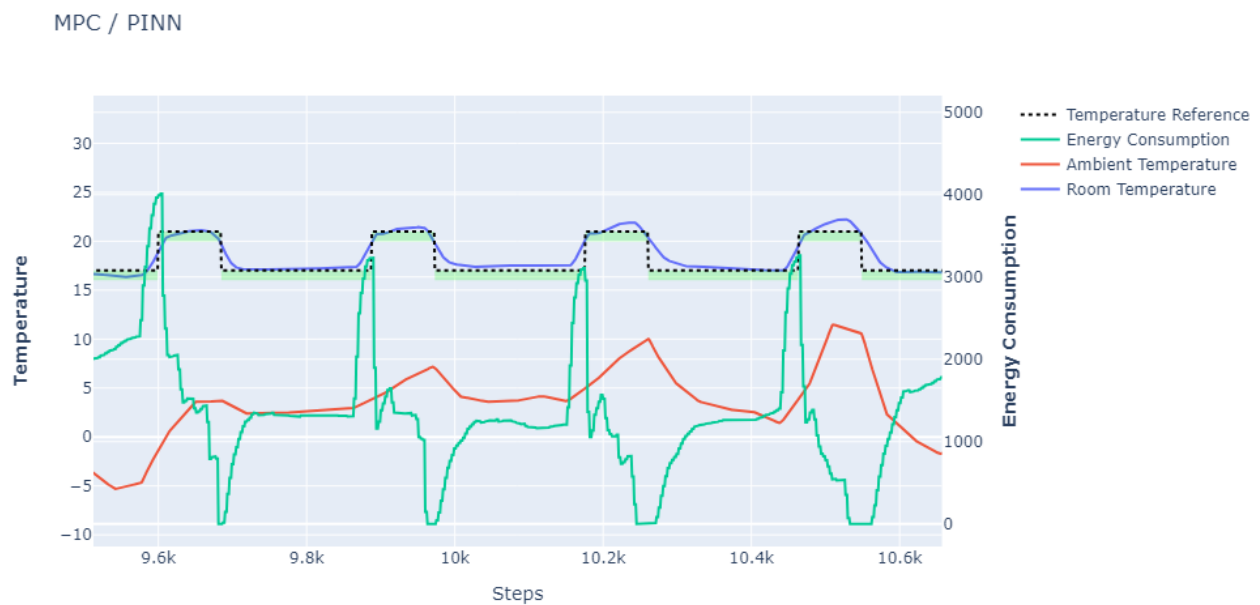


Figure B.3. Results for "SimpleHouseRad" with PINN.

B.1.2 Performance on *SimpleHouseRSla*

MPC / Original nRnC

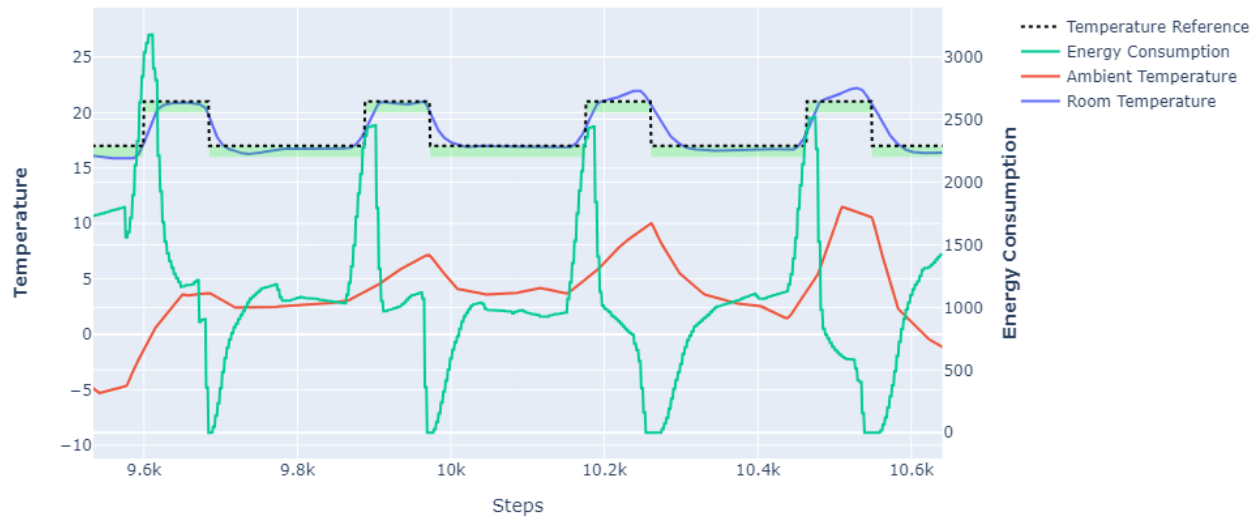


Figure B.4. Results for "SimpleHouseRSla" with original nRnC.

MPC / Improved nRnC

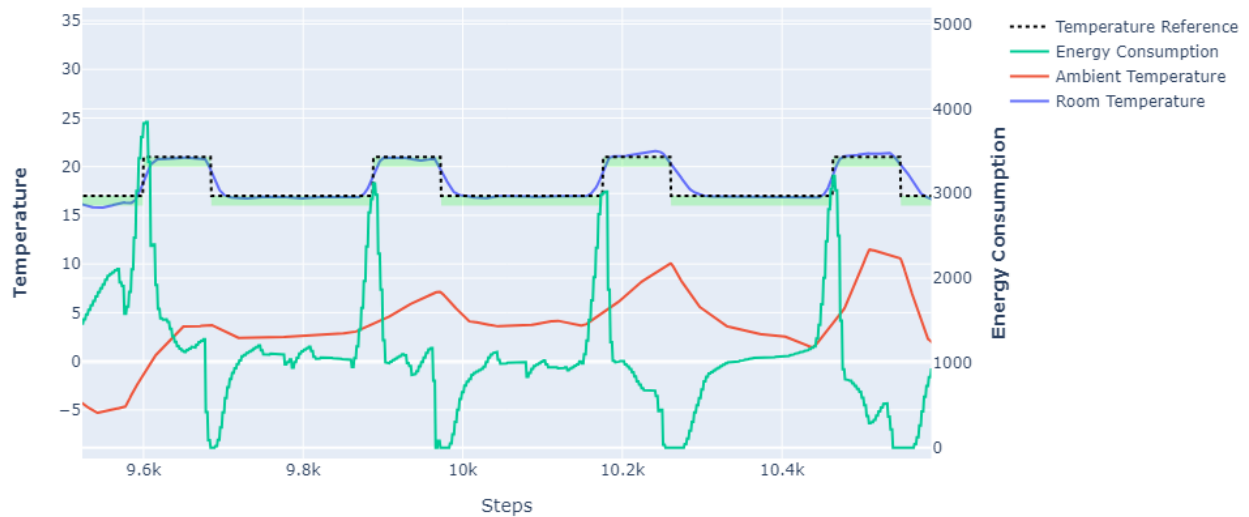


Figure B.5. Results for "SimpleHouseRSla" with improved nRnC.

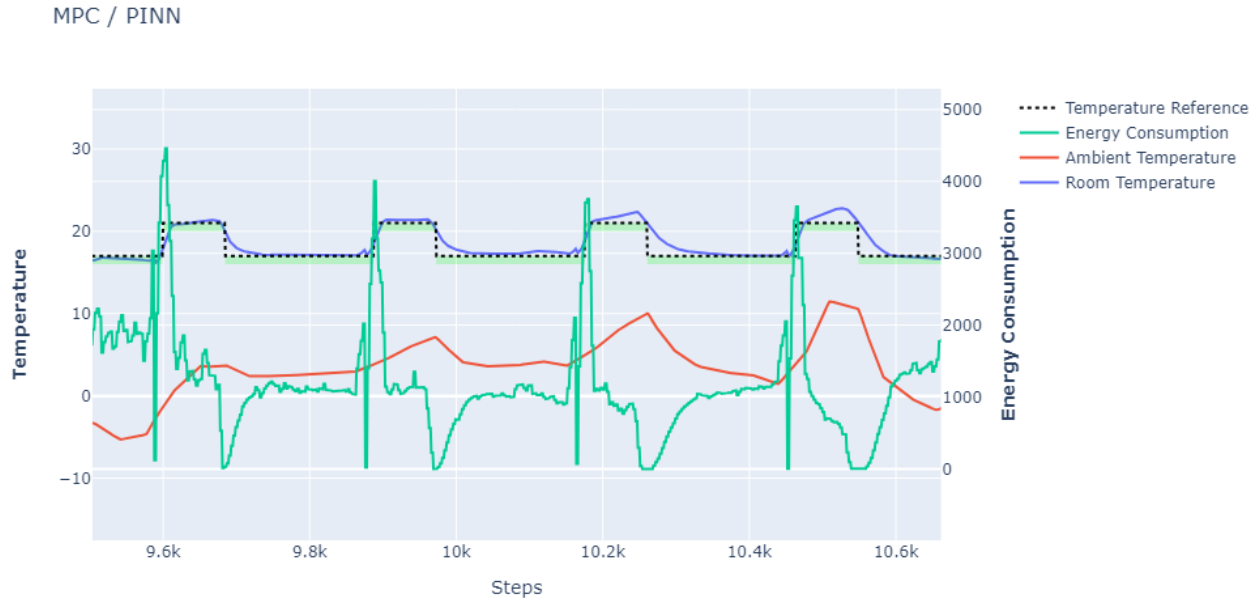


Figure B.6. Results for "SimpleHouseRSla" with PINN.

B.1.3 Performance on *SwissHouseRSla*

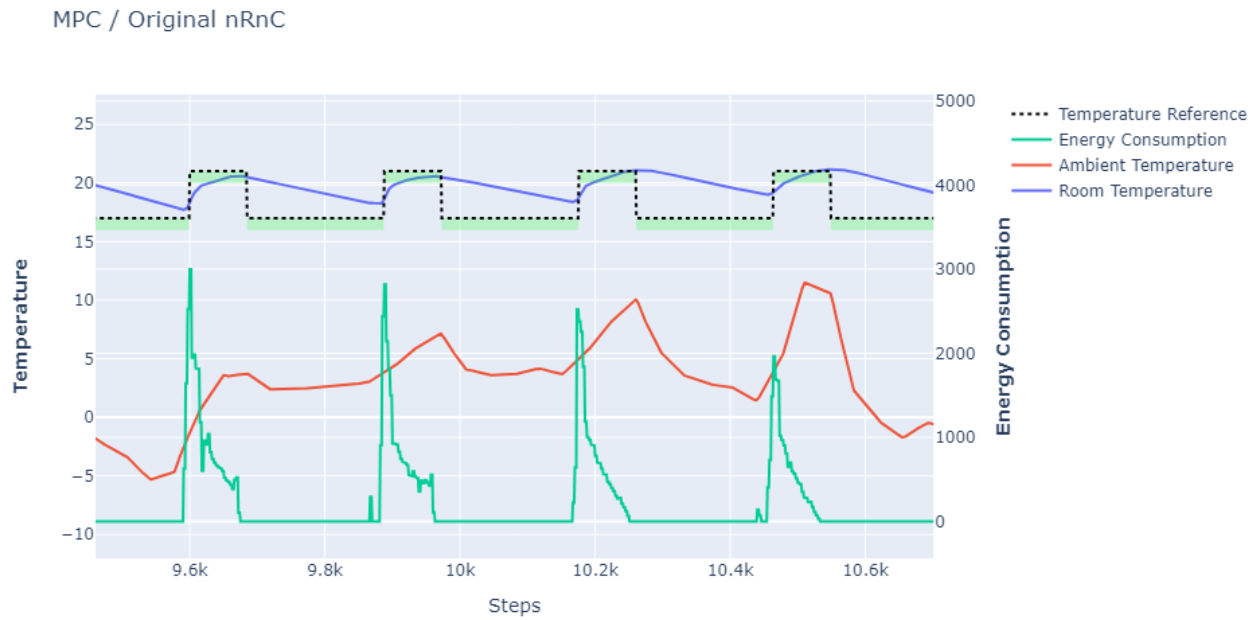


Figure B.7. Results for "SwissHouseRSla" with original nRnC.

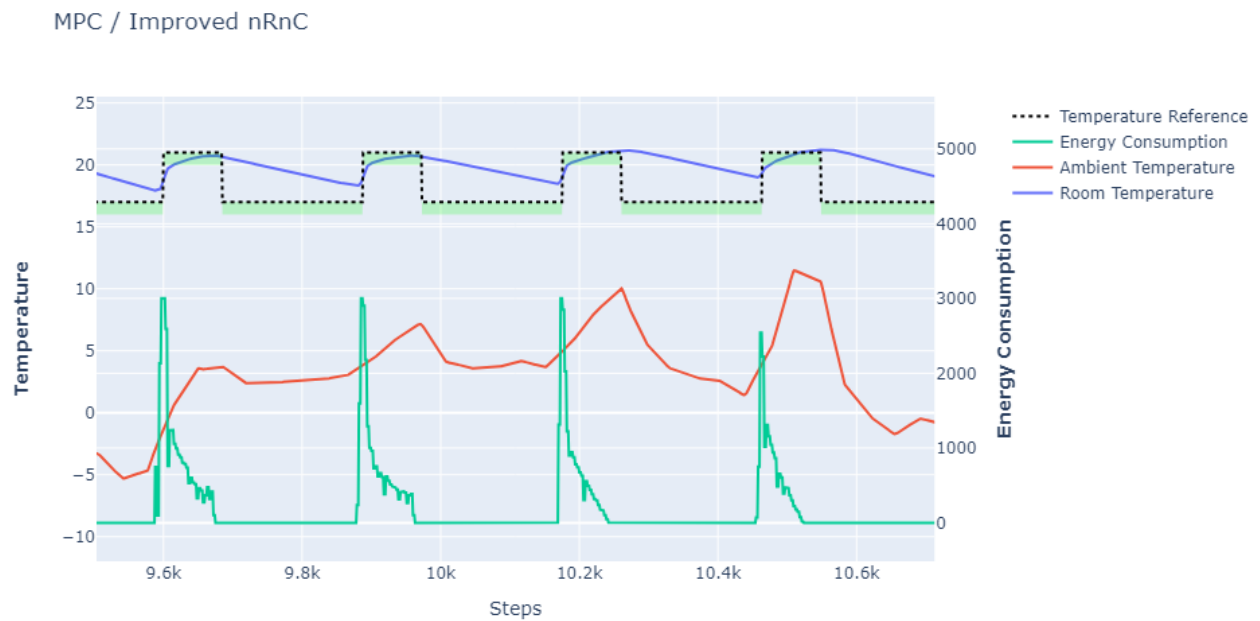


Figure B.8. Results for "SwissHouseRSla" with improved nRnC.

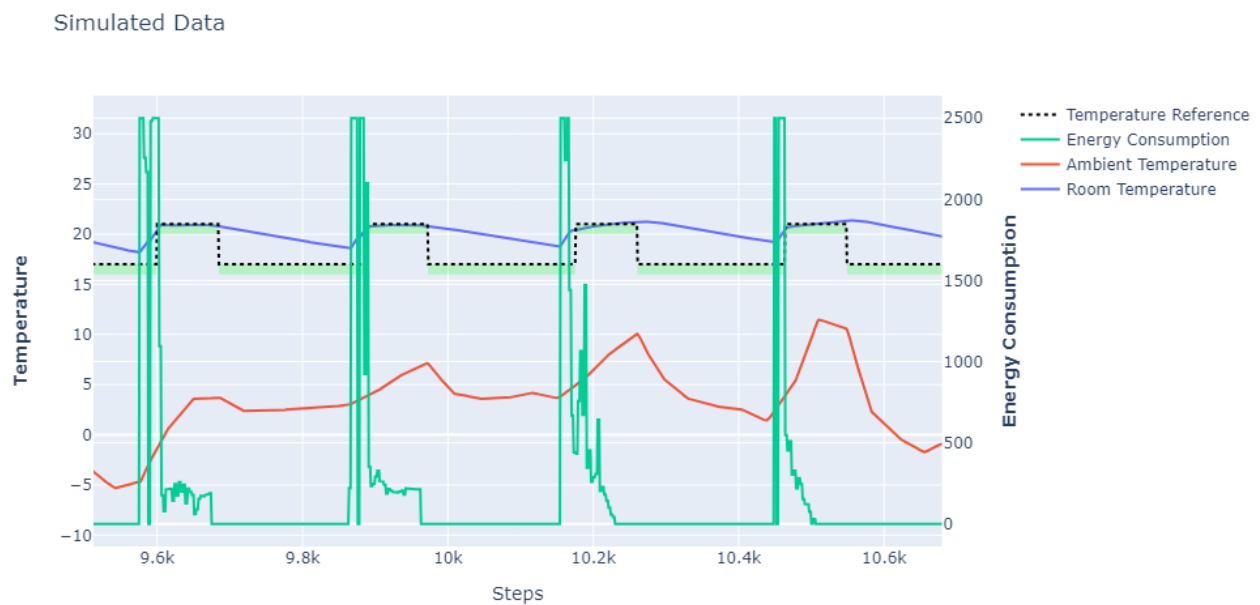


Figure B.9. Results for "SwissHouseRSla" with PINN.

B.2 Performance with BO

B.2.1 Performance on *SimpleHouseRad*

MPC / Improved nRnC

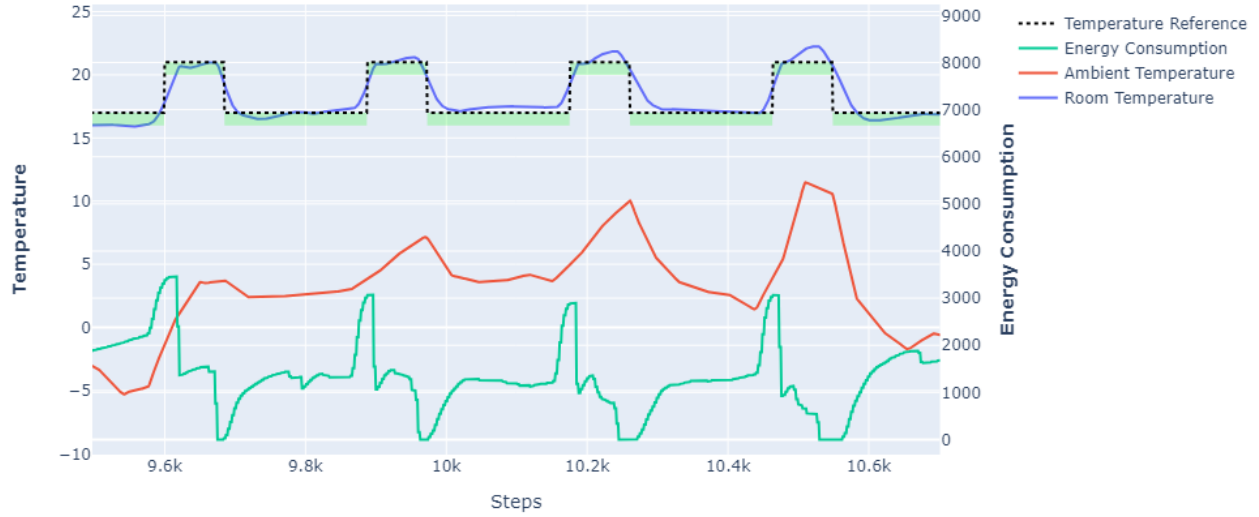


Figure B.10. Results for "SimpleHouseRad" with improved nRnC and BO.

Simulated Data

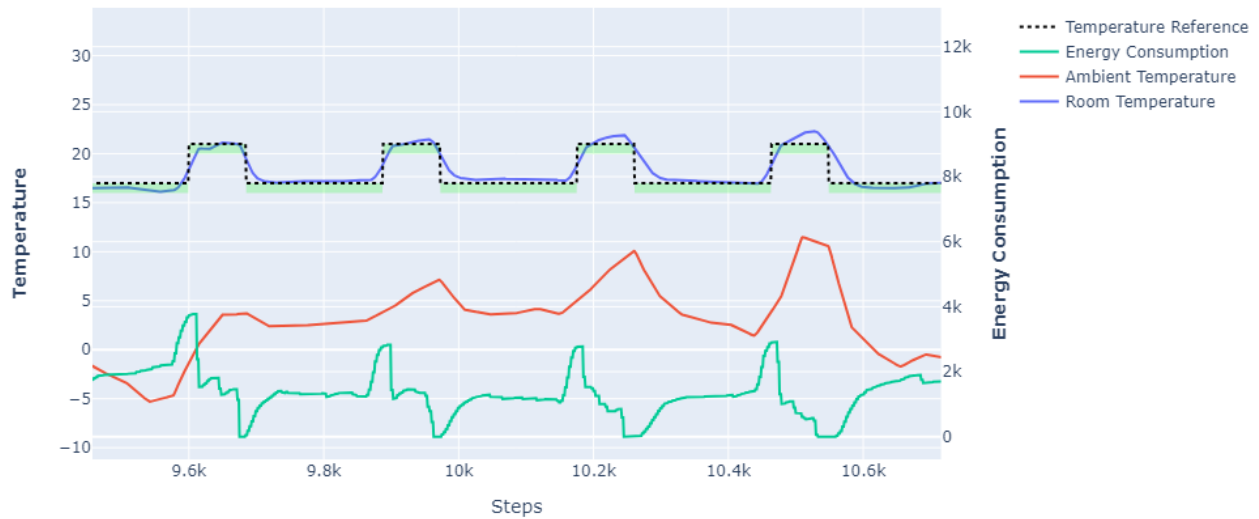


Figure B.11. Results for "SimpleHouseRad" with PINN and BO.

B.2.2 Performance on *SimpleHouseRSla*

MPC / Improved nRnC



Figure B.12. Results for "SimpleHouseRSla" with improved nRnC and BO.

Simulated Data

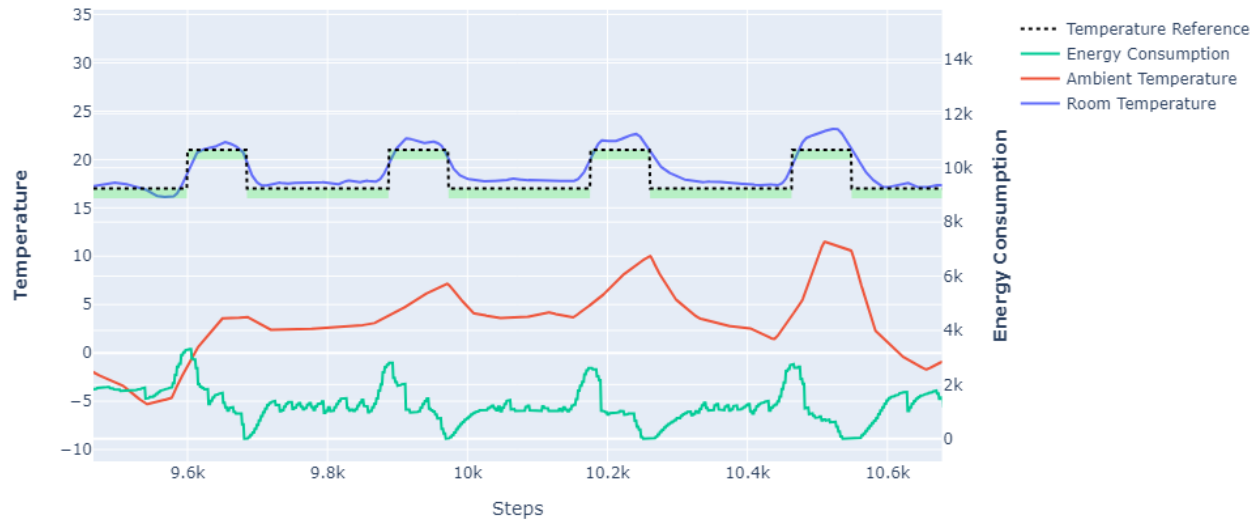


Figure B.13. Results for "SimpleHouseRSla" with PINN and BO.

B.2.3 Performance on *SwissHouseRSla*

Simulated Data

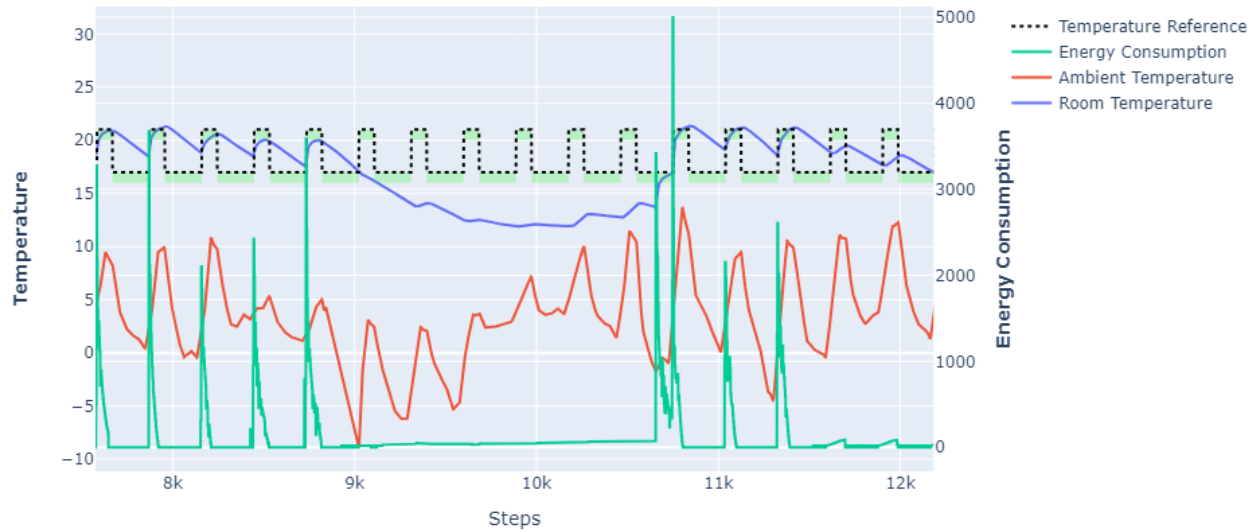


Figure B.14. Results for "SwissHouseRSla" with improved nRnC and BO.

Simulated Data

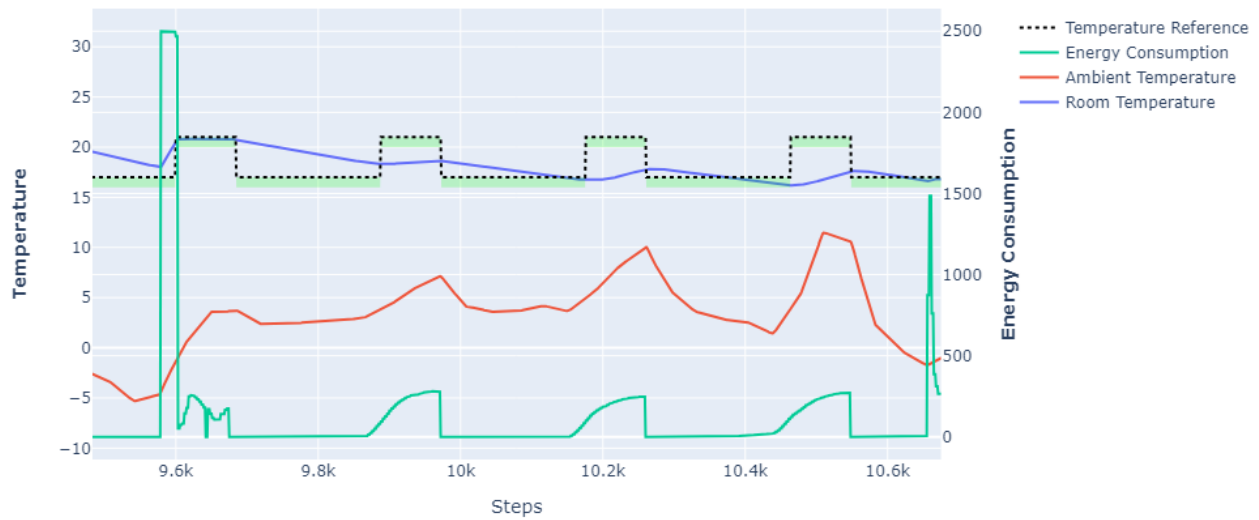


Figure B.15. Results for "SwissHouseRSla" with PINN and BO.