university of groningen

faculty of science and engineering

MASTER'S THESIS
ARTIFICIAL INTELLIGENCE

From Data To Control: Learning An Optimal HVAC Control Policy Using A Deep Learning Based Neural Twin

Author: Bram de Wit *(s3151654)*

First supervisor: prof. dr. L.R.B. Schomaker

External supervisor: Jaap Broekhuizen

November 14, 2021

Contents

Acknowledgments

I think a project always builds upon numerous things. I probably can not name all developments required before this project could even start, but there are a few things I would like to mention. First of all, I would like to acknowledge the many years of work that went into digitizing the building environment. From the physical placement of sensors to the log records stored in a cloud database. Without these years of development performed by my colleagues at ACS (Jaap Broekhuizen, Wisse Hooghiem, Sietse Damstra and many more), there would be no data for me to work with and this project would not even be here. I would like to thank ACS in general for providing me with the opportunity of performing this study under their supervision. It has provided me with many new insights about how research might be done within a company, a valuable experience!

Furthermore I would like to thank my supervisor prof. dr. Lambert Schomaker. He provided me with a lot of new information and many new ways to think about the problems in this study (even if this meant prolonging our meetings every once in a while), for which I am very thankful!

Jaap Broekhuizen fulfilled the same role, but from the perspective of the company. He helped me keep the project practical and applicable, which helped me personally to find a balance between performing research and providing usable work.

Abstract

The building and construction sector is responsible for 36% of the total energy usage and for 39% of energy/process related emissions. A large part of this energy is used by heating, ventilating and air conditioning (HVAC) systems that regulate a building's indoor climate. Most of the HVAC systems however are still using classic control systems, which leaves room for improvement in the form of lowered emissions and decreased energy usage. Although many solutions are already proposed in the literature, the proposed solutions fail to transfer to the real world HVAC systems.

This thesis aims at developing smart HVAC controllers that can be used at scale. The proposed controllers achieve this by deriving the control solely from data. First a Neural Twin of the process is learned. This can be seen as a simulator of the process, learned from the data using a GRU encoder-decoder neural network. The results show that the Neural Twin framework is able to simulate several distinct processes with an average absolute error close to 0.2 °C for all processes, even when predicting many hours ahead. The Neural Twin is used to develop two different control algorithms.

The first algorithm learns a control policy for a process using a neural network. The network is trained using Proximal Policy Optimization (PPO) by gathering experience from the simulated environment provided by the Neural Twin.

The second algorithm performs Model Predictive Control (MPC) using the Neural Twin at real time during control. It uses the Neural Twin to choose an optimal control sequence, given a set of possible control sequences. It selects the optimal control sequence based on the horizon, which is usually a few hours ahead. Both control algorithms are inspected in several environments and for one of those environments the controllers are tested on a physical room.

The results show that the control algorithms are able to handle a wide variety of different processes, without manual tuning. The controllers achieve improved performance compared to the conventional control algorithms, which were manually tuned, mainly in terms of energy usage. It is estimated that the proposed controllers can lead to a 5% - 40% decrease in effective energy usage, while retaining the thermal comfort and stability. The controller trained using reinforcement learning showed the best performance. From the results it is concluded that the control methods pose an attractive alternative compared to conventional controllers.

1 Introduction

1.1 Problem description and goals

Within a building the indoor climate is often actively regulated. This regulation is performed by Heating, Ventilating and Air Conditioning (HVAC) systems. There are various motivations for indoor climate regulation such as providing thermal comfort for occupants of the building, maintaining good conditions in greenhouses for growing crops or storing wares which require certain climate conditions.

According to the National Human Activity Pattern Survey, a person spends approximately 87% of their time indoors [1]. Furthermore, several studies have already shown the negative impact of poor indoor climate conditions, which can cause health issues and decreased productivity for occupants of the building (also known as the Sick Building Syndrome (SBS)) [2] [3] [4]. For these reasons, maximizing the quality of the indoor climate is desirable.

In general, regulating the indoor climate in buildings results in both energy costs and carbon dioxide emissions. Obviously it is desirable to reduce both of these as much as possible. This is one of the main goals of the International Energy Agency (IEA) organization, whose goal it is to shape a world with secure and sustainable energy. IEA provides analyses of energy consumption and carbon dioxide emissions globally and for specific countries. In their (at the time of writing) most recent report they claimed the following: ”The buildings and construction sector should be a primary target for green house gasses (GHG) emissions mitigation efforts, as it accounted for 36% of final energy use and 39% of energy- and process-related emissions in 2018.” ([5] p. 12). The same report showed that 30% of the process related energy is used by buildings alone and 28% of the energy-related carbon dioxide emissions come from buildings. Furthermore it is expected that the energy demand of buildings over the next 30 years will increase further with 50% [6].

The goals above pose a conflict, as regulating the indoor climate causes always increases energy demand (compared to simply not regulating the indoor climate). This means that in the end a tradeoff has to be made between the quality of the indoor climate regulation and the energy consumption. Recent studies show that currently it is often possible to use machine learning approaches in order to improve on the current control policies with respect to both goals [7] [8] [9] [10] [11].

1.2 Possible improvements

Several actions can improve the energy performance of buildings, and in order to achieve optimal energy performance a building should invest in multiple actions. Two straightforward actions one can take are:

1. Improve the insulation of the building (for an example see [12]). This results in less energy loss which means less energy is needed to maintain the building's indoor climate.

2. Improve the energy management of a building, i.e. improving the current HVAC system of the building. One way of achieving this is by taking advantage of the sensor data of a building in order to optimize its control policies.

This study focuses on the latter by developing a method to improve the energy management of buildings, mainly due to the reason that changing the software which runs a building is easier than re-building parts of a building (which is required when improving insulation). This makes this approach more practical and generalizable. However, in the end both approaches can lead to a significant decrease in energy costs.

1.3 Building characteristics

A building's indoor climate can be regulated in many ways. For example, one can use a radiator as shown in Figure 1 to heat up a room. One can run hot water (heated by a boiler, located somewhere else in the building) through the radiator, after which it will slowly heat up the room.

How fast a radiator heats up a room is dependent on many factors, most of which can be measured using sensors. For example, the temperature of the water as it leaves the boiler can be measured using a temperature sensor. Furthermore, it is common that there is a valve which controls how much of the hot water is used for the water flowing through the radiator. This way, one can control the process of heating the room.

However, almost every process is unique, and in order to optimally control the process, one needs to know something about this process. For example; how fast does the room temperature react on a change in position of the valve? How much time does it take to increase the room temperature with 1 °C? As this differs for all processes, one needs different control policies for all of them in order to control them optimally.

Figure 1: Example of a heater (radiator), a factor that can control the room temperature.

The method presented in this study relies solely on data acquired from sensors. Before data of sensors can be gathered, it is required to setup a network of sensors describing the building and letting them communicate according to a protocol, such that data can be stored in a database. Several protocols exist to provide guidelines about how this data should be communicated. The data used in this study was gathered according to the Modbus and BACnet guidelines. The precise workings/principles of these guidelines are not relevant for this study, but can be found in [13], [14]. The data was stored using Climatics[1], from which the data was retrieved using its Application Programming Interface (API). An example of sensor data from a valve sensor and a corresponding room air temperature sensor over time is shown in Figure 2.

The sensors measure characteristics of (parts of) a building by logging values at a regular interval. The logged values of all sensors of a building at a specific time t then effectively describe the state of the building at that time. Storing this data essentially means gathering a descriptive data set of the building, capturing the relations of the different processes of the building (and possibly external factors such as weather conditions). Some examples of sensors in buildings are:

1. Temperature sensors measuring the air temperature of a room, the outside air temperature or the supply (exit) temperature of a boiler.
2. Feedback sensors, measuring the control value of certain equipment, such as how much % a valve is open or a binary value for on/off controllers.
3. Pressure sensors, measuring relative pressure in Pascal.
4. Flow sensors, which measure the volume of water passing the sensor every second.
5. CO2 sensors, which measure the concentration of the air in PPM.

[1]https://climatics.nl

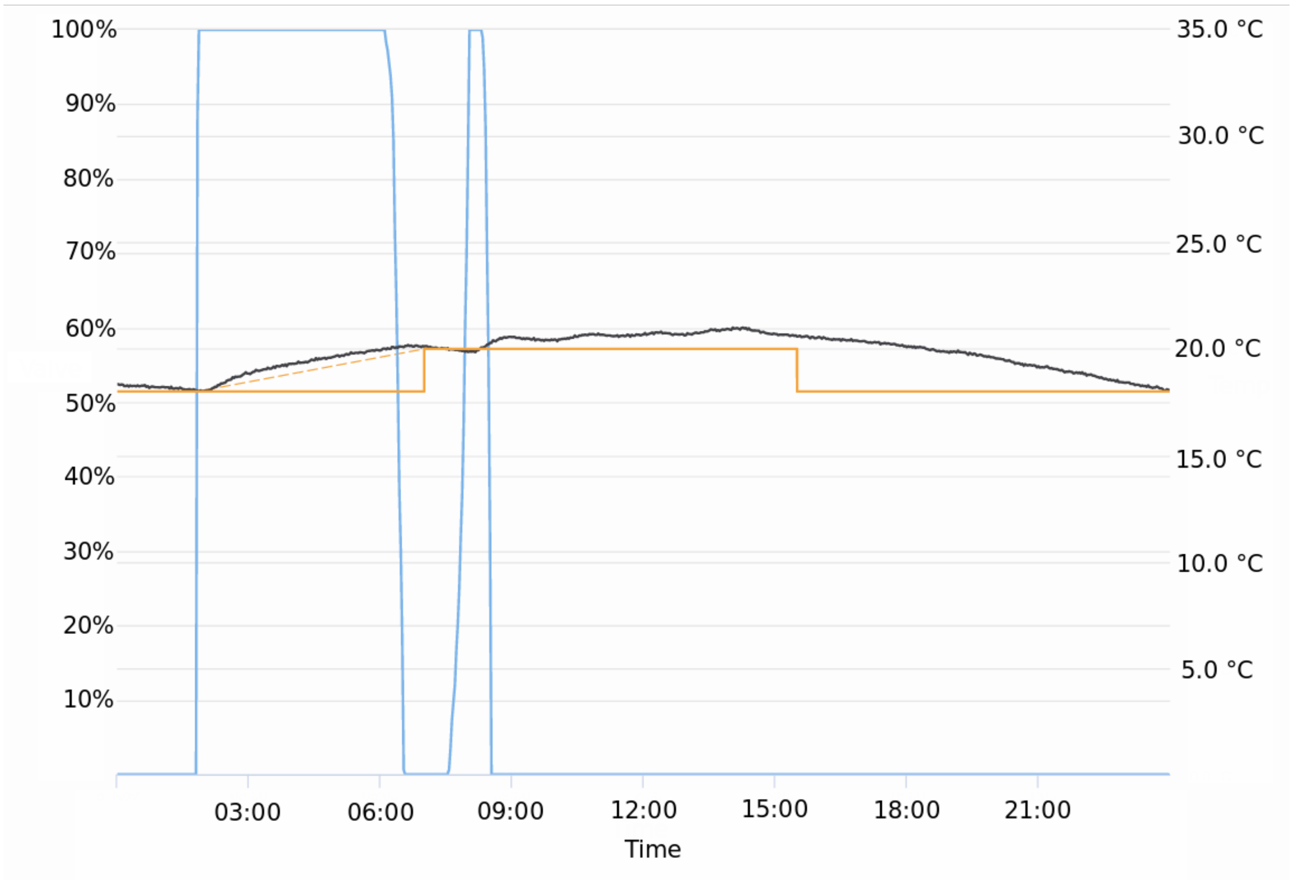


Figure 2: An example of sensor data over time. The black line corresponds to the room air temperature in °C of a sports hall, and the blue line corresponds to the valve position in %, showing how much hot water from the boiler was send into the radiator. The orange line denotes the setpoint of the room. A clear relationship between the sensors can be identified; when the valve opens, the temperature rises. Furthermore, it can be seen that the room air temperature is influenced by external factors as well (the weather conditions in this case).
The figure was taken from `https://climatics.nl/`.

A data set with sensor values over time shows the result of the *dynamical system* of that particular building. The dynamics of a building are in general quite complex to model accurately, as it depends on many varying factors such as weather conditions, volume of the room, location of the room within the building, physical properties of the walls and occupancy of certain zones in the building. Manual construction of such a model is possible, however, and there are multiple simulation programs available for this. Examples of such programs are EnergyPlus [15], TRNSYS [16] and WUFI [17]. Although these programs facilitate the modeling of buildings, they still require a user to specify the complete building in a configuration, which for complex buildings takes time and is error prone. Furthermore, from an industrial/practical point of view, this is not a scalable approach when many buildings have to be put into production. This study aims at finding a scalable approach using only the data available from the sensors of a building to find an optimal control policy. A detailed description of the data used in this study can be found in Section 3.

1.4 System identification and learning of control policies

Extracting the parameters that determine the dynamics of a system using measured data is called *system identification*. Part of this study aims at learning a model which approximates these dynamics. Much research has already been devoted to performing system identification using machine learning (for more details the reader is referred to Section 2). The model should be reliable enough such that control policies can be learned and evaluated on it. When the model approximates the actual dynamics close enough, it is expected that the learned control policies will also be able to control the process in real life. This way, control policies can be learned from the data alone. This has two main advantages. First, this approach does not require the time consuming setup of simulation software. Second, this approach will be scalable as long as enough data is available from the process.

1.5 Research questions and contributions

To summarize, this study aims at answering the following research questions:

- Can a reliable simulator, which captures the underlying system dynamics of a building be trained, solely from historical sensor data?
- Given such a simulator, is it possible to use this simulator in order to train a model to exert a control policy π which improves the control performance as compared to classic control?

The contributions of this study are:

- Introducing a scalable approach for learning control policies, based on data alone;
- Inspecting the performance and reliability of system identification using machine learning, for four different simulated environments;
- Demonstrating the performance of the learned control policy π for an actual physical room.

1.6 Thesis Outline

The remainder of this thesis is structured as follows. Section 2 describes the relevant theoretical background needed for the rest of the study. General concepts are introduced here as well as previous literature related to this study. Section 3 describes the details of the methods and design choices for the system identification models and the learning algorithms used for finding optimal control policies. Section 4 describes the setup of the experiments performed in this study. Section 5 presents the results of the experiments and in Section 6 these results are interpreted and a conclusion is drawn.

2 Theoretical Background

In this section, the theoretical background for this study is given. First, a very basic introduction is given concerning the general concepts which are needed to understand the methods used in this study. Then, previous literature is presented on which this study is based.

2.1 Control systems

The process of indoor climate regulation can be viewed as a *control system*. A control system aims at controlling a certain process variable (PV) such that it reaches and maintains a given value, often called the setpoint (SP). In a control system, the PV is dependent on one or more factors, some of which can be controlled. The algorithm that determines the values of controllable factors in order to control the PV will be called the *control policy* in this study.

A concrete example of an indoor climate control system can be the following: A room has a certain temperature, which will be the PV. The room has a certain setpoint, i.e. the desired temperature of the room (for example 21°C). The room temperature is influenced by a heating component (for example a radiator as shown in Figure 1). This heating component has a temperature which is influenced by a valve, which determines the amount of hot water coming from the boiler that flows through the radiator. This valve has a *control value* (CV), often ranging from 0% to 100%. By controlling the valve, one influences the room temperature. Of course, in a realistic setting the room temperature is furthermore influenced by the combination of the outside temperature, the insulation of the room, the size of the windows, the material of the walls, the number of occupants in the room, and many more factors.

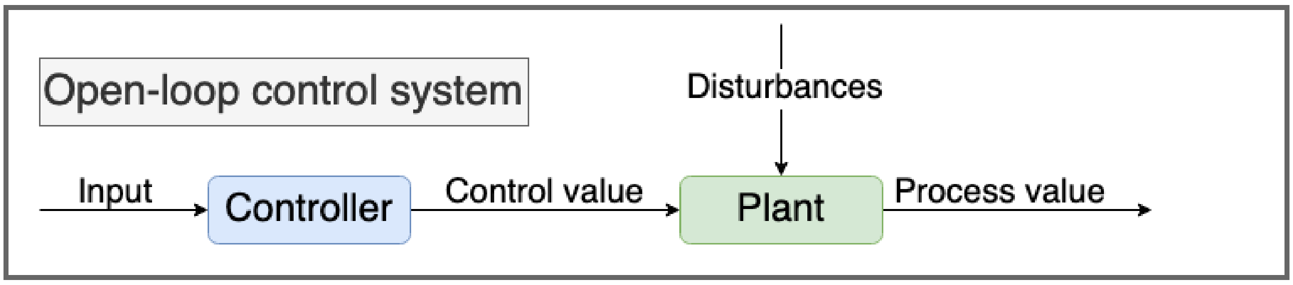
From this the following question arises: How can one optimally control the room temperature? Note that the term *optimally* leaves room for defining how a control system behaves optimally. In this study (as introduced in Section 1) a trade-off will be made between energy efficiency and quality of the indoor climate. Furthermore, the optimality of a control policy can vary in different use cases. For example, for humans working indoor a comfortable temperature lies (depending on the region) between 18-24°C [18]. For controlling greenhouses, depending on the crops, this variation may be much higher, but certain other aspects may weigh more heavily, such as the average temperature over a day.

2.2 Types of control systems

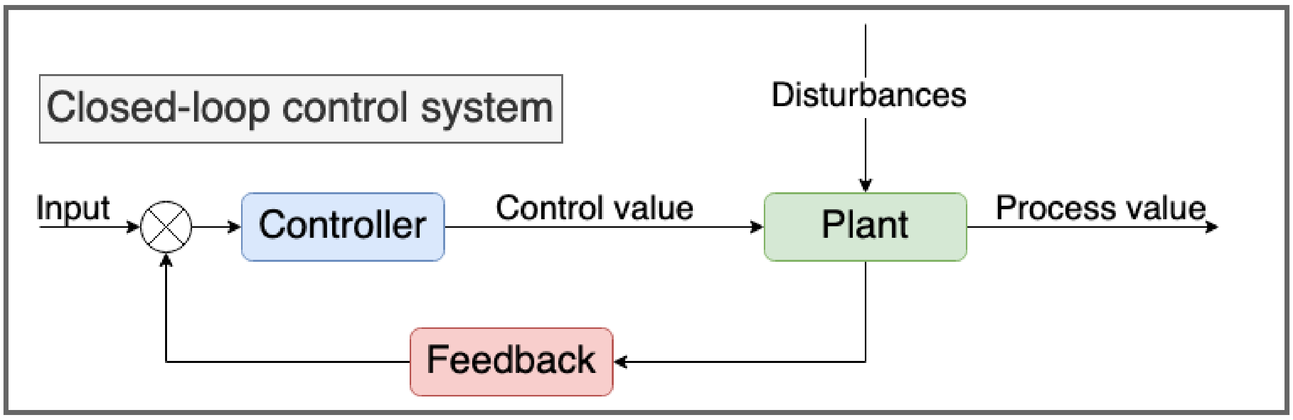
Two types of common control systems exist: open-loop control systems and closed-loop control systems.

2.2.1 Open-loop control systems

In open-loop control systems (also known as non-feedback systems), the control value is determined independent of the process value. A simple example of an open-loop control system is an on/off boiler which simply turns on at 06:00 AM and turns off at 08:00 PM. The process value (the temperature of the water leaving the boiler) is not used in determining the control value. A schematic view of an open-loop control system is shown in Figure 3a.



(a) Schematic view of an open-loop control system.



(b) Schematic view of a closed-loop control system.

Figure 3: This figure shows two common types of control systems. Figure 3a shows an open-loop control system whereas Figure 3b shows a closed-loop control system.

2.2.2 Closed-loop control systems

In closed-loop control systems (also known as feedback systems), the control value is dependent on the value of the process value and its relation to the setpoint. A simple example of an closed-loop control system is an on/off boiler which turns on if the process value is a certain value x °C below setpoint, and turns off if the process value is x °C above setpoint. This control policy is often called dead-band control. A schematic view of an closed-loop control system is shown in Figure 3b. The feedback must have a negative sign somewhere in the loop in order to guarantee convergence to a desired set point of the process value (negative feedback). Positive feedback would be undesirable, because it would create a divergence or oscillating avalanche in the process value.

2.3 Examples of classic control policies

Dead-band controller A very simple control policy is the dead-band controller. This control policy is mostly used for on/off controllers. As described above, one specifies a (not necessarily symmetric) 'band' around the setpoint. In mechanical thermostats the dead band is realized through hysteresis, i.e., the switch-on moment being at a lower temperature than the switch-off moment. Whenever the PV falls below the band, the controller will turn on. Whenever the PV exceeds the band, the controller will turn off. An advantage of the dead-band controller is that it is very simple to implement. A disadvantage is that it can not easily be adapted for continuous control, which is often encountered in realistic building control settings.

PID controller The proportional-integral-derivative (PID) controller is one of the most popular controllers today, together with its more simpler versions: the P and PI controllers. The PID controller is an old concept (first formally introduced in 1922 [19]) but one that is still widely adopted due to its robustness and stability (that is, when tuned correctly). A PID controller produces new control values based on the error term $e(t)$, i.e. the difference between the current temperature and the setpoint. The control value $u(t)$ is then given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{1}$$

The three parameters K_p, K_i and K_d determine the behaviour of the control policy. The proportional term, K_p times $e(t)$, allows a system to react proportionally to an observed error: Larger errors should give rise to larger corrections. The integral term with coefficient K_i integrates (averages) the error over time, yielding a low-pass effect. The derivative term with coefficient K_d allows, conversely, for a faster response if the rate of change in the error signal is high. In control, the goal is to find optimal values for the three coefficients, such that the response is proportional, smoothed, but fast enough.

Although the PID controller is relatively simple, determining the optimal value of the weights of three terms in order to achieve 'optimal' behaviour is difficult. Even worse, when the three terms are not properly balanced, the system might not even stabilize and have a tendency to oscillate. Several methods have been developed in order to determine proper PID parameters such as the Ziegler-Nichols method which aims at Quarter-Amplitude Damping (QAD) of the process value [20]. This is achieved by extracting certain process characteristics (for example the *deadtime* of a process, the time it takes for the process to react to a step jump in the control value). Other PID tuning methods are for example the Cohen Coon method [21] or Lambda tuning [22]. Cohen Coon works also by extracting the same process characteristics, but is more suited for large time delay systems (where the process value reacts relatively slow to the control value). Lambda tuning is a method which can be used for systems in which overshoot is very costly. It yields slow behaviour and uses no derivative term ($K_d = 0$).

In general, the PID controller allows for robust control once one has found proper parameter values. Tuning methods exist but these still do not provide a general solution and manual tuning often is still desirable. Another disadvantage of PID controllers is that they are reactive, in a sense that there the PID controller will never look ahead at what will happen, which means the control is always lagging. Furthermore, the parameters of a PID controller are static, and often in different seasons during the year, different behaviour is required, which means that the controller needs to be tuned again.

2.4 Model-Predictive Control

A more sophisticated and still upcoming branch of control policies is Model-Predictive Control (MPC). Although these types of controllers outperform classic controllers, they are less widely adapted as they require a model of the process, which is often hard to obtain. However, when available, the model is used to plan a trajectory τ for a fixed number of time steps ahead. It will try to optimize the τ over this horizon h according to a certain cost function, which needs to be designed manually. The advantage of these types of control is that they use information of the system in order to plan ahead, making it able to handle non-linear control systems well. Of course, the performance of the MPC depends on the quality of the underlying model and the length of the horizon over which the trajectory is optimized. Two examples of how to utilize MPC are given below.

2.4.1 Random-Sampling Shooting Method

One way to utilize MPC is to use the random-sampling shooting method [23]. This generates K sequences of control values for the given horizon h. For all these sequences the resulting trajectory is predicted using the model. Then for all these trajectories the cost is calculated and the first action of the control sequence with the lowest cost is executed. This process repeats for every time step. An advantage of this approach is that it requires no extra knowledge besides the model (which still can be the bottleneck). A disadvantage of this approach is that it is computationally demanding, although one could argue that this is irrelevant when new control values are not needed that often (once every 10 minutes for example). Furthermore, as it is randomized, one needs to make sure that enough sequences are generated, as too few control sequences will almost inevitably lead to poor performance. As a result, it is impossible to guarantee a certain degree of performance, other than by evaluation. An application of this approach to the control of robotics can be found in [24].

2.4.2 Genetic Algorithms

Another approach is to use a Genetic Algorithm (GA). The population will consist of a list where every element is sequence of control values for the upcoming timesteps (a member equals a sequence of control values). The fitness of the members (control sequences) in the population can then be determined by predicting the corresponding trajectories and their cost. The lower the cost, the higher the fitness of the member. Then, the next population is generated based on the fitness scores of the members of the previous population, together with mutation and crossover. This process can continue until the new control values are required. Then the first action of the member/control sequence with the highest fitness score is executed. Although in this way an optimization procedure takes place, it is still hard to determine whether or not the found control values are optimal. An example of the usage of GA for MPC to provide smart control of HVAC systems can be found in [25].

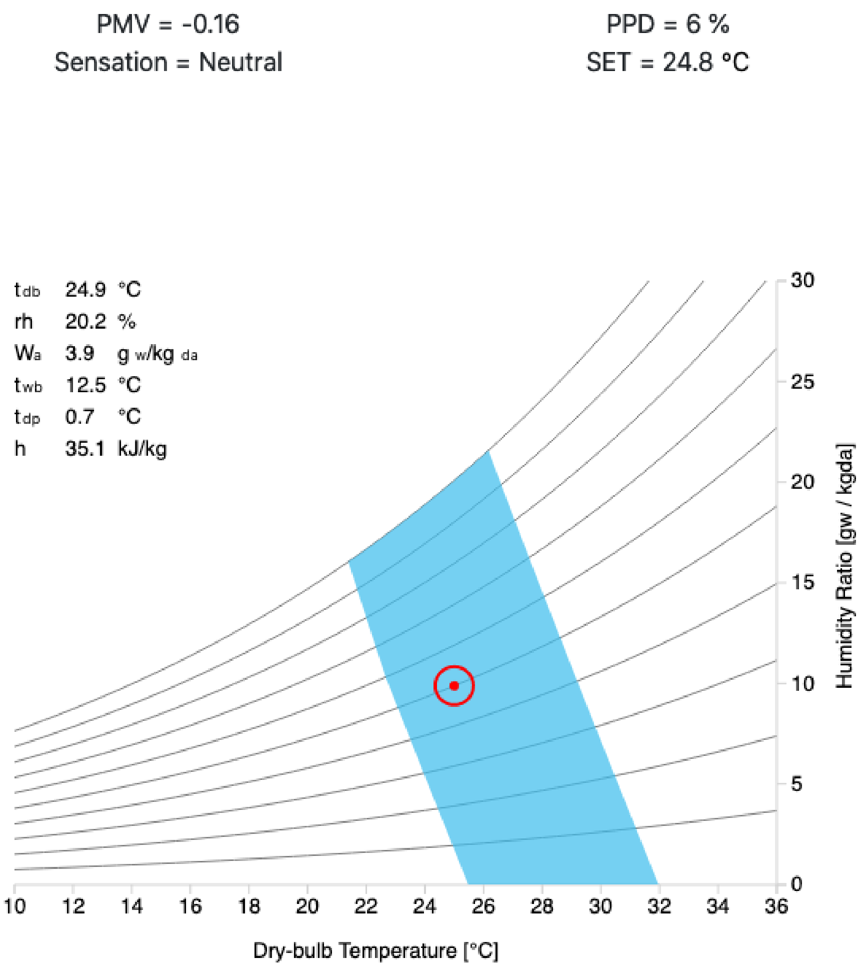
Naturally, the optimization of the control sequences can also be done using other optimization techniques such as Particle-Swarm Optimization [26], Simulated Annealing [27] or Bee-Colony Optimization [28].

2.5 Performance of control

As described in Section 1, there are two conflicting goals that determine the performance of a control system. Note that the weighting of these two goals can vary over time, as during the night in a regular office comfort maximization should not be weighted as much as during the day. Lastly, a different but related goal is described as well, regarding the stability of control.

2.5.1 Energy minimization

The first goal is to minimize the energy used during control. The energy used can vary for different processes, as it might be the case that one building provides its hot water supply using a boiler for which the gas consumption can be measured, whereas another building might use a heat pump for which the electricity usage should be measured. Sensors that measure the energy usage of the system were not present in the buildings of this study, but it is relatively easy to find a measurement proportional to the energy usage which can then be used to minimize.


PMV = -0.16
Sensation = Neutral
PPD = 6 %
SET = 24.8 °C

t_{db}	24.9	°C
rh	20.2	%
W_a	3.9	g_w/kg_{da}
t_{wb}	12.5	°C
t_{dp}	0.7	°C
h	35.1	kJ/kg

Figure 4: This figure shows the accepted area in terms of temperature and humidity in terms of thermal comfort as provided by the ASHRAE standards [29]. It links the PMV and PPD to a specific spot within the blue shaded area. The reader is invited to play around with the different factors on the website to see how it influences the PMV and PPD. This figure is taken directly from the site `https://comfort.cbe.berkeley.edu/`.

2.5.2 Comfort maximization

The second goal is to maximize the comfort of the occupants in the building. This goal is harder to measure, although much research has been devoted to this specific goal. The main question is: how can one measure objectively the performance of indoor climate regulation with respect to occupant comfort?

One well known model for measuring this quantitatively is by determining the Predicted Mean Vote (PMV), and deriving the Percentage of People Dissatisfied (PPD) from it. The Predicted Mean Vote was developed by Fanger and introduced in [30]. The PMV estimates the average vote of persons about the temperature of a room. It is a number ranging from -3 to 3 where -3 represents cold and 3 represent warm (note that the number itself does not imply anything yet about whether the temperature is *too* cold/hot). From the PMV, the PPD can be determined, which represents the estimated % of people which would be (thermally) dissatisfied, given the PMV. Often, ASHRAE standards [29] are then consulted to infer reasonable bounds for these values, see Figure 4.

The PMV is calculated using the following factors:

1. Dry-bulb air temperature in °C.
2. Mean radiant temperature in °C.
3. Relative air speed in m/s (caused by body movement).
4. Relative humidity in %.
5. The metabolic rate, a number proportional to the physical activity of occupants.
6. Clothing insulation of occupants.

The PPD is then calculated using a predefined distribution over the PMV. The exact calculations for the PMV and PPD can be found in [29].

Other comfort models exist as well such as the adaptive ASHRAE comfort model. This method yields acceptable bands for the inside air temperature taking the running average of the outside temperature into account.

Although these metrics are well studied, it remains the case that many aspects of such models are either based on assumptions or otherwise hard to measure factors, such as the clothing insulation. Even though there exist ASHRAE approved methods for estimating (under certain conditions) the clothing insulation based on the outside temperature [31], one can imagine a situation where an individual with its own office deviates from these clothing conditions, and prefers a somewhat hotter environment. In such a case, the person will be dissatisfied when optimizing for such thermal comfort models.

Therefore, this study will refrain from placing such assumptions on the occupants of buildings, and assumes that the control of rooms will be regulated with setpoints, which can then be determined by the occupants themselves in order to optimize their thermal comfort. In that case, the goal of optimizing thermal comfort will align with the goal of tracking the given setpoint as closely as possible. This comes at the cost of requiring occupants of the building (or the building managers) to determine setpoints for rooms at specific times.

2.5.3 Stability

This goal is related to the physical constraints a system have. For example, a valve operates in the physical worlds. Without these valves, control is not possible. The more stable a control policy is, the longer the lifetime of such a valve. Furthermore, if a control policy is more stable, this means that it will be more predictable as well, and as a result the building's control system as a whole will be more stable if all of the sub processes are stabler as well. Hence, having a stable controller is desirable. Another way of expressing this is by 'roughness' of control, i.e. how rough does the valve open and close over time in order to reach its goal.

2.6 System identification

The process of (nonlinear) *system identification* can be described as the process of building mathematical models of a specific dynamical system. The mathematical models are build using measured data. In this study this data is gathered by the sensors placed in a building. In theory, creating a model based on first principles (using known equations) is possible, but in general yields highly complex

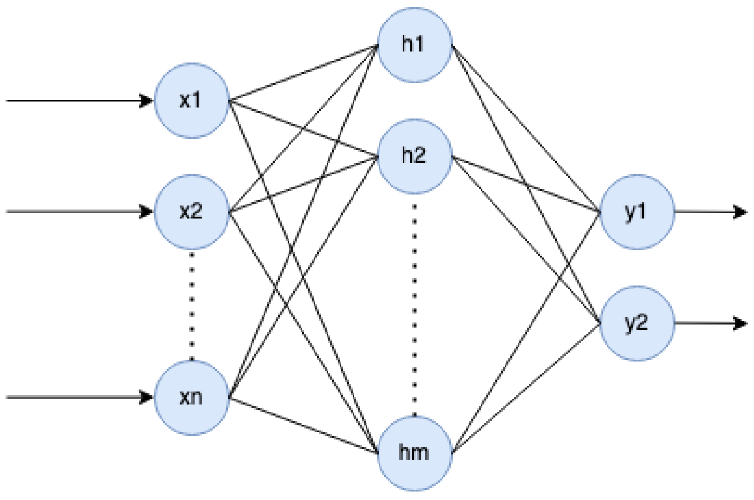


Figure 5: Architecture of a multilayer perceptron with n inputs an 2 outputs, with a single hidden layer of m neurons. The circles represent neurons and the lines represent weights.

models. Often building such a model is too time consuming. However, the advantage of such a model would be that one will be able to understand the dynamical system in full detail. Therefore these models are often called *white-box* models.

A different approach is to make (almost) no prior assumptions about the form of the model, and infer it solely from the data. Articial Neural Networks (ANN) can be used for this purpose. They learn a function mapping inputs to outputs, and in the form of system identification this would mean that it receives the current state of the system as input and predicts the next state of the system. The use of ANNs for system identification has already been studied extensively and the idea of using ANNs as nonlinear function approximators for this purpose was already introduced in the previous century [32] [33]. Recent deep learning advances has motivated researches to apply these techniques with the same purpose [34] [35] [36] [37] [38]. A recent survey covering and supporting the use of neural networks for nonlinear system identification is given in [39].

2.7 Artificial Neural Networks

The models used in this study to perform system identification are called Artificial Neural Networks (ANNs). ANNs have received much attention during the past decades, as they are widely applicable and with large amounts of data they have proven to be able to perform tasks with high (sometimes even superhuman) performance.

One of the attractive properties of ANNs is that they are *universal approximators*, as already shown in 1989 by [40]. Informally, the theorem in [40] states that an ANN with a single layer with a sigmoidal activation function can approximate any function f that is well behaved arbitrarily well, as long as it has enough neurons. Over the years, many different types of architectures have proven empirically to work best for different types of problems.

2.7.1 Multilayer perceptron

One of the simplest and most straightforward architectures is the Multilayer Perceptron (MLP), which maps vectors to vectors. An MLP consists of a certain number of so called *hidden layers*, which are connected by weights (represented as matrices). An example schematic of an MLP with input vector

$\mathbf{x} \in \mathbb{R}^n$, output vector $\mathbf{y} \in \mathbb{R}^2$ and a single hidden layer with m neurons is shown in Figure 5. The lines connecting the neurons are the weight matrices, which means that this network consists of two weight matrices: W^{ih}, connecting the input layer with the hidden layer and W^{ho}, connecting the hidden layer with the output layer.

One of the key parts in the expressive power of neural networks is the nonlinear activation function used in the layers. Examples of such activation functions are the sigmoid function $\sigma = 1/(1+e^{-\mathbf{x}})$, the tanh function or the rectified linear activation function $ReLU(\mathbf{x}) = \max(0, \mathbf{x})$. The operations for activation functions are applied element-wise.

Note: For simplicity, this text will omit biases, which can be used to introduce affine transformations between layers. Biases can simply be applied by adding an extra neuron at every layer (introducing an extra dimension) whose value will always be 1. Then the extra weight can scale this in order to learn an affine transformation.

More formally, an MLP can be described as a function consisting of weight multiplications and activation functions. Suppose the network of Figure 5 uses the sigmoid function as activation function for the hidden layer, and no activation function for the output layer. Then the network can be written down as a function f:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = W^{ho} \cdot \sigma(W^{ih} \cdot \mathbf{x}) \tag{2}$$

The network acts as a nonlinear mapping $f : \mathbb{R}^n \to \mathbb{R}^2$. The weight matrices can be thought of as mappings as well. A 'deep' neural network (where 'deep' means several hidden layers) therefore transforms the input vector through many different 'spaces', before reaching a space from which a final inference can be made, yielding $\mathbf{y}$.

In order to train an MLP, a certain metric is needed which evaluates how 'good' a prediction of the MLP is. Suppose our MLP has K hidden layers. Furthermore, Suppose we have a data set $\mathcal{D}$ of size N consisting of pairs $(\mathbf{x}_i, \mathbf{y}_i)$ for $i = 1, 2, \ldots, N$, where $\mathbf{x}_i \in \mathbb{R}^n$ is an input vector for which the target vector is given by $\mathbf{y}_i \in \mathbb{R}^2$. Furthermore we denote the parameters (the weights) of the network as θ and the MLP as f_θ. Then the ultimate goal is to minimize the empirical risk based on the data set $\mathcal{D}$:

$$R^{emp}(f_\theta) = \frac{1}{N} \sum_{i=1,\ldots,N} L(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{3}$$

$L(\hat{\mathbf{y}}, \mathbf{y}_i)$ represents the *loss function*, which determines the error of a prediction. A standard loss function is the mean squared error (MSE) which is defined as:

$$\text{MSE}_i = (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{4}$$

Then, the empirical risk becomes:

$$R^{emp}(f_\theta) = \frac{1}{N} \sum_{i=1,\ldots,N} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{5}$$

ANNs are optimized by an iterative training procedure, using gradient descent. During the training procedure, the parameters of the network are updated as follows:

$$\theta^{t+1} = \theta^t - \alpha \nabla R^{emp}(f_\theta) \tag{6}$$

This means that the gradient is obtained by averaging the gradients for all input-output pairs in the data, and this gradient is used to perform a step update. However, this requires a pass through the complete data set in order to obtain this estimate of the gradient, and for large data sets this yields slow training. Therefore, the gradient is often estimated using *stochastic gradient descent*, which simply uses the gradient of a single example, sampled randomly from $\mathcal{D}$. Although this allows for more and faster updates, the convergence is more noisy. Another alternative is to use *minibatches*, where a random minibatch is selected of a certain size (for example 32 input-output pairs), which is used to estimate the gradient. As more examples are used for an update, the updates are often more stable and representative for the data set, yielding more stable convergence.

For the purpose of explaining the training process, here the gradient of the empirical risk will be used, with the mean squared error loss function. This gradient can be expressed as:

$$\nabla R^{emp}(f_\theta) = \nabla\left(\frac{1}{N}\sum_{i=1,\dots,N}(\mathbf{y}_i - f_\theta(\mathbf{x}_i))^2\right) = \frac{1}{N}\sum_{i=1,\dots,N}\nabla(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{7}$$

In order to perform the weight update, one needs the partial derivative with respect to every individual parameter in the network. Due to the convenient structure of an MLP, these partial derivatives can be calculated efficiently using the *backpropagation* algorithm [41]. The backpropagation algorithm calculates the error for the output layer and then backpropagates this error through the network.

The partial derivative of the loss with respect to a specific weight connecting neuron i from layer $k-1$ to neuron j from layer k is given by (using the chain rule of calculus)

$$\frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial w_{ji}^k} = \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial x_j^k}\frac{\partial x_j^k}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ji}^k}, \tag{8}$$

where x_j^k denotes the output activation of neuron j in layer k and a_j^k denotes the *potential* of the neuron, the value the neuron received *before* passing it through the activation function. $\frac{\partial x_j^k}{\partial a_j^k}$ corresponds to the derivative of the activation function, and $\frac{\partial a_j^k}{\partial w_{ji}^k}$ will simply be x_i^{k-1}.

Now let us define:

$$\delta_j^k = \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial x_j^k}\frac{\partial x_j^k}{\partial a_j^k} \tag{9}$$

During backpropagation, first the error of the output layer is calculated. For the output layer, x_j^k corresponds to y_j. For simplicity, let us assume we have linear output neurons. Then for the output layer:

$$\delta^K = \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}\frac{\hat{\mathbf{y}}}{\partial \mathbf{a}^K} = \frac{\partial(\mathbf{y}-\hat{\mathbf{y}})^2}{\partial \hat{\mathbf{y}}}\cdot 1 = -2(\mathbf{y}-\hat{\mathbf{y}}) \tag{10}$$

Then, for every hidden layer k, the error is propagated back through the network using the error of the next layer as follows (again, using the chain rule). This is recursively defined as

$$\delta^k = \delta^{k+1}\frac{\partial \mathbf{a}^{k+1}}{\partial \mathbf{x}^k}\frac{\partial \mathbf{x}^k}{\partial \mathbf{a}^k} \tag{11}$$

where $\frac{\partial \mathbf{a}^{k+1}}{\partial \mathbf{x}^k}$ is represented by the weights W^k connecting those two layers and $\frac{\partial \mathbf{x}^k}{\partial \mathbf{a}^k}$ is the derivative of the activation function of layer k.

In general, the partial derivatives for a specific weight w_{ji}^k of can then be calculated with:

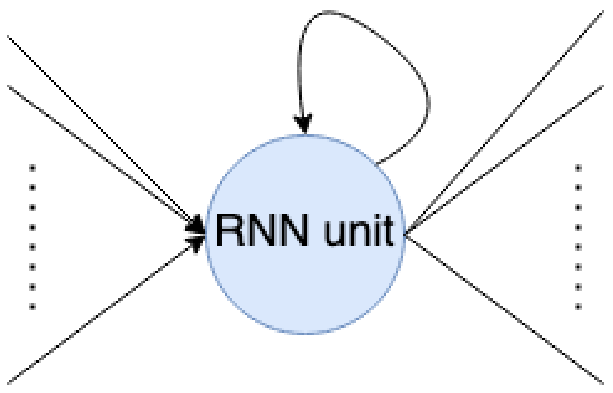


Figure 6: Example of a hidden neuron of an RNN. The unit accepts new inputs at a new timestep (the arrows coming from the left), and combines them with its previous state, denoted by the circular arrow.

$$\partial w_{ji}^{k} = \delta_j^k \frac{\partial a_j^k}{\partial w_{ji}^k} = \delta_j^k x_i^{k-1} \tag{12}$$

This procedure is repeated for all layers and for the input layer the values $\mathbf{x}^{k-1}$ are the same as the inputs provided by the data set.

Considering this, there are *forward passes* in which input is fed to the neural network which yields predictions, and there are *backward passes*, in which partial derivatives for all weights are calculated.

When all input-output pairs in the data set are used exactly one time each in order to perform updates, this is considered an *epoch*. Several epochs of training (thus multiple passes through the data set) are usually required in order to obtain good performance.

2.7.2 Recurrent Neural Networks

The MLP structure allows for a clean and convenient structure for most simple input-output pairs. However, how well it will perform on a problem is also dependent on the nature of the problem. Suppose for example the goal is to train a neural network that maps the state of a room to the temperature of the room at $t+1$, where a state $\mathbf{s}_t$ is defined by the values of all sensors at time t related to that room (and additionally some external factors such as weather conditions might be provided). To learn such a mapping, the information provided at time t by all the sensors might not be enough, as previous states $(\mathbf{s}_{t-n},\dots,\mathbf{s}_{t-1})$ might also have an influence on $\mathbf{s}_{t+1}$, meaning that if only $(\mathbf{s}_t,\ \mathbf{s}_{t+1})$ is given as input-output pair, the neural network would not have enough information to form an accurate prediction. Instead, a number of sequential inputs $\mathbf{s}_{t-n},\dots,\mathbf{s}_t$ should be given when predicting $\mathbf{s}_{t+1}$. Especially for processes with a large time-delay (under which for example room temperatures) this is important.

One could argue that a way to solve this is to concatenate all state vectors into a single state vector $\hat{\mathbf{s}}_t$ (which then contains information of previous states as well) and use this to predict $\mathbf{s}_{t+1}$. In regression form this is often called window-based regression. However, then the question arises how many states in advance should be included, which might vary from process to process. It also might lead to many weights which effectively describe the same relation between two factors but at different time steps, which do not necessarily need to vary, and thus the same weight need to be learned many times.

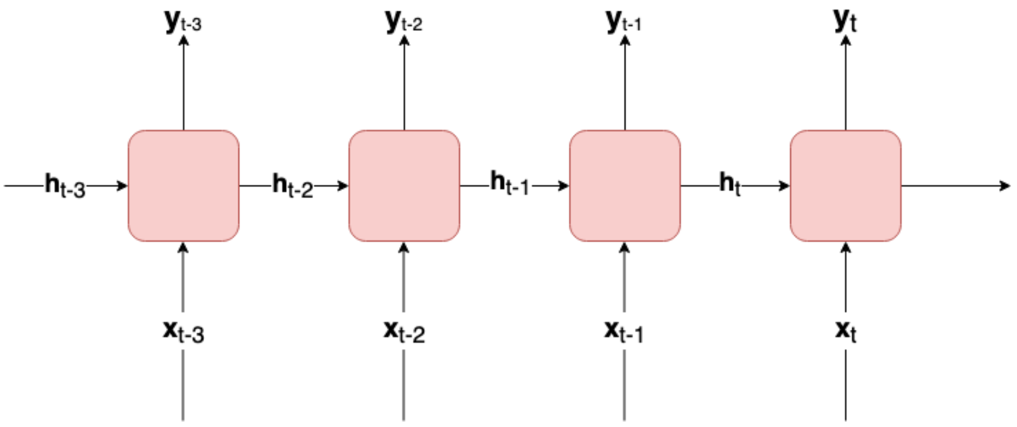


Figure 7: An RNN network rolled out over time. It shows a sequence of inputs and a sequence of outputs, and how the hidden state is carried through time.

Another alternative is to introduce some form of *memory* in the network which carries over information from previous inputs to the current input. Networks that do this are called *Recurrent Neural Networks* (RNNs). In such a network, next to weights connecting neurons from one layer to the next layer, there are also *recurring* weights, which connect the hidden neurons of a layer with themselves. An example of a hidden unit is given in Figure 6. These recurring connections represent the memory structure of the network, and its memory is often referred to as the 'hidden state' or 'internal state'. The hidden state needs to be initialized at the begin of a sequence, and often it is initialized as the zero vector $\mathbf{0}$. It allows the network to remember (a part of) the activation of the hidden layer during the previous input. Thus, it introduces a time component in the neural network.

Suppose the same network structure as in Figure 5 is used, but now recurrent connections for the hidden neurons are added. The neural network is then parameterized by the weight matrices W^{ih}, W^{ho} and the recurrent weights W^{hh}.

More formally, this network is presented as a function f_θ parameterized by the weights θ. The hidden state $\mathbf{h}_t$ of a hidden layer in an RNN at time t is given by the previous output activation of that layer at $t-1$. This hidden state is multiplied by W^{hh} and added to the input of the next time step. Thus, the output of the neural network, given input $\mathbf{x}_t$ can be written down as:

$$\hat{\mathbf{y}} = f_\theta(\mathbf{x}_t) = W^{ho} \cdot \sigma(W^{ih} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}) \tag{13}$$

Performing backpropagation for RNNs is very similar to the derivation of backpropagation for the MLP given above. It can be helpful to think of 'rolling out' the network over time, such as in Figure 7. As the loss of the network now depends on multiple timesteps, the loss of a predicted sequence can be expressed as:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{T} \sum_{t=0,\ldots,T} l(\mathbf{y}_t, \hat{\mathbf{y}}_t) \tag{14}$$

Calculating the derivative with respect to this loss is mostly the same as for a MLP, except for the partial derivative of recurrent weights. Their partial derivatives depend on both W^{hh} and $\mathbf{h}_{t-1}$, where $\mathbf{h}_{t-1}$ is again dependent on W^{hh}. Therefore, patterns from previous inputs in the sequence are learned in order to affect the predictions at later timesteps. In practice however, for long sequences propagating a gradient through a network can cause the problem of vanishing/exploding gradients

[42]. Research has shown that vanilla RNNs, as described above, have problems learning long-term dependencies due to this problem. In order to solve this, several alternatives which preserve a memory over longer input sequences have been proposed.

2.7.3 Long Short-Term Memory networks

A solution to the vanishing/exploding gradient problem was already proposed in 1997 [43]. This study introduced a Long Short-Term Memory (LSTM) network architecture, in which a constant error flow is provided for the memory of the network, often called the cell state. LSTM networks are designed specifically such that the cell state, which contains the memory of the sequence, is updated in a way such that the error flows back without causing the vanishing or exploding of gradients. It does so by introducing certain gates. The core component of an LSTM cell is the cell state C_t. This cell state is updated every timestep according to a sequence of operations. The following will present the architecture of the LSTM cell, already including a forget gate, which was not originally presented in [43] but suggested 2 years later in [44].

1. When a new input arrives, first the LSTM layer determines which parts of the cell state are forgotten. In the equation below, $\mathbf{f}_t$ will hold for every unit in the cell state a number between 0 and 1 which states how much should be remembered of that unit:

$$\mathbf{f}_t = \sigma(W^f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{15}$$

2. Then, it is determined which parts of the inputs are added to the cell state, and their corresponding values:

$$\mathbf{i}_t = \sigma(W^i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{16}$$

$$\hat{C}_t = \tanh(W^C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{17}$$

3. Now the cell state can be updated as follows:

$$C_t = \mathbf{f}_t \odot C_{t-1} + \mathbf{i}_t \odot \hat{C}_t \tag{18}$$

4. With the updated cell state, a new output (and hidden state) can be generated. The part of the cell state that will be output is based on the current input and hidden state:

$$\mathbf{o}_t = \sigma(W^o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{19}$$

5. The final output and the new hidden state of the LSTM cell is then given by:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(C_t) \tag{20}$$

In the following years, several alternatives and improvements were suggested and experimented with, such as forget gates [44] and peephole connections [45]. A large scale study has shown that the most critical components of the LSTM cell are the forget and output gates [46].

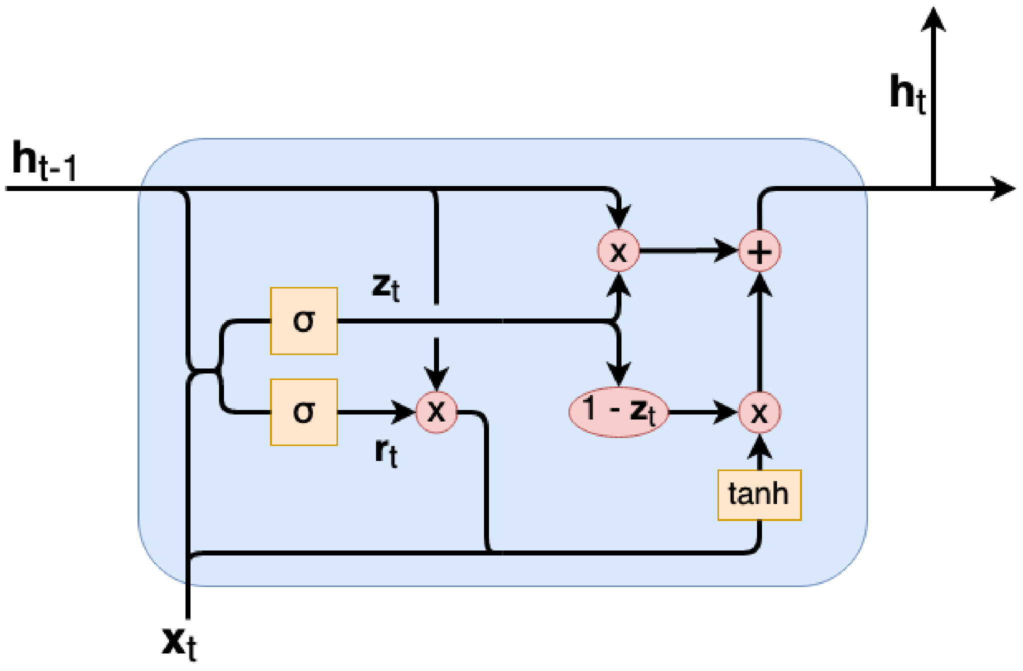


Figure 8: A schematic view of an GRU.

2.7.4 Gated Recurrent Units

In 2014, an alternative architecture closely related to the LSTM was introduced, called the Gated Recurrent Unit (GRU) [47]. The architecture retains the most important concepts of the LSTM, but requires less parameters and while showing comparable performance [48]. An GRU does not hold a cell state like the LSTM does, but operates only using a hidden state h_t. It combines the input and forget gate. For a GRU, every input the following steps are performed:

1. Determine which values of the hidden state to pass through to potentially add to the new hidden state, called the *reset* gate:
$$\mathbf{r}_t = \sigma(W^r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \tag{21}$$

2. Determine what to update in this timestep, the *update* gate:
$$\mathbf{z}_t = \sigma(W^z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \tag{22}$$

3. Then, the hidden state is updated according to:
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{z}_t) \odot \tanh(W^h \cdot [\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t]) \tag{23}$$

As can be seen, a GRU model has fewer parameters than an LSTM model, while obtaining comparable performance. For this reason, GRU networks will be used in this study. A schematic view of an GRU is shown in Figure 8.

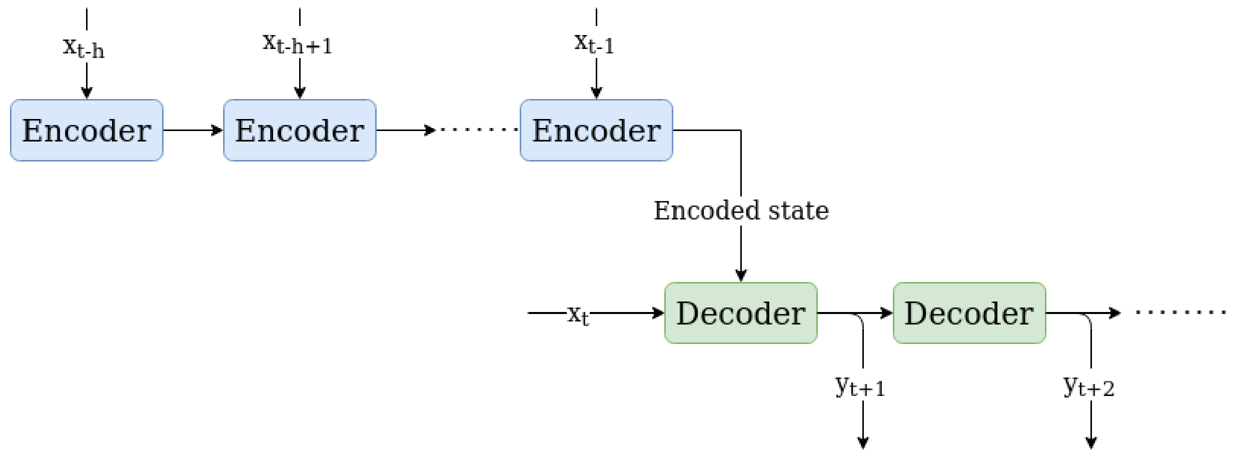


Figure 9: A schematic view of an Encoder-Decoder model.

2.7.5 Encoder-Decoder Models

Although a network based on GRU units allows one to incorporate memory in the network, it cannot handle certain scenarios in its current form. Suppose the goal is to predict the trajectory of certain process values of a building n timesteps ahead at time t, given a number of past observations h (known as sequence-to-sequence prediction). Available are all observations of sensor values of the past $\{\mathbf{o}_{t-h}, \ldots, \mathbf{o}_{t-1}, \mathbf{o}_t\}$. Only a part of these observational values can be modelled and predicted, whereas others cannot, as they are provided by external processes (for example weather conditions or room occupation). The process values to predict, for example room temperatures, are dependent on these external variables. This means that for future timesteps, no input for external variables is available during future timesteps, and the network that is used to feed past observations cannot be used for recurrent predictions in future timesteps. One can still model this using neural networks by using an Encoder-Decoder architecture. Furthermore this architecture allows to predict target sequences of arbitrary length, regardless the length of the input sequence.

Encoder-Decoder models were first introduced in the context of machine translation, where the target sequence (the translated sentence) might be of a different length then the input sequence (the sentence in the original language) [47] [49]. Encoder-Decoder architectures consist of two networks: an encoder and a decoder. The main idea is to first encode the input sequence word by word using the encoder (in the papers LSTMs were used). Then the cell state and hidden state of the encoder are used as initial hidden state and initial cell state of the decoder network, which are LSTMs with the same memory size as the encoder network. Then first a ’Start Of Sentence’ (SOS) token is fed to the decoder, marking the start of the target sequence. The decoder then predicts the first output, which will also be its own input for the next timestep. This repeats until an ’End Of Sentence’ (EOS) token is produced by the network, marking the end of the prediction sequence. During training, instead of its own outputs sometimes the original target tokens are fed to the decoder in order to speed up training. This is called *teacher forcing*. Of course, during inference these target tokens are not available, so then the output of the decoder at time t will be the input for the decoder at time $t+1$.

Encoder-Decoders can just as well be constructed using GRUs. The main difference then lies in that now only the hidden state is passed as the initial state to the decoder, and the rest remains the same. A schematic view of an Encoder-Decoder network is shown in Figure 9.

Many other types of architectures have been studied extensively such as Convolutional Neural

Networks (CNNs) [50], Bidirectional Recurrent Neural Networks [51] and neural networks equipped with attention mechanisms allowing the networks to focus on specific parts of the input at different times during prediction [52]. In this study, only GRUs and MLPs will be used.

2.8 Reinforcement learning

Reinforcement Learning (RL) deals with an agent learning how to handle sequential decision-making tasks. As opposed to supervised learning, often there is no clear target is available (such as ground truth classes or actions). Instead, the agent interacts with an environment, gathers experience, and then tries to learn from this experience. The environment is equipped with a reward function, which tells the agent how well it is currently doing. How this agent acts influences the environment, and thus which states will be visited. The experiences that are gathered tell the agent something about its actions. For example if an agent is in state s_t and takes action a_t, it might receive a high reward corresponding with the next state s_{t+1}. Then, in the future, selecting this action in this state should be more likely than other actions, as it yielded a high reward. Reinforcement Learning deals with automating the process of learning which actions to select in which state, in order to maximize the rewards it gets. Rewards are experienced in *episodes*. Sometimes the problem has episodes which can terminate, such as a game of chess which is finished after some time. Other problems such as regulating a building's environment are continuing, meaning an episode never ends. A schematic view of an agent interacting with the environment is shown in Figure 10. Formally, Reinforcement Learning is formulated as a Markov decision process (MDP).

2.8.1 Markov Decision Process

An MDP is defined by the tuple $< \mathcal{S}, \mathcal{A}, P(), \mathcal{R}() >$ where:

- $\mathcal{S}$ is a set containing all possible states. Elements of the set will be denoted as s_t, where t represents the timestep.
- $\mathcal{A}$ is a set containing all possible actions. Elements of the set will be denoted as a_t, where t represents the timestep.

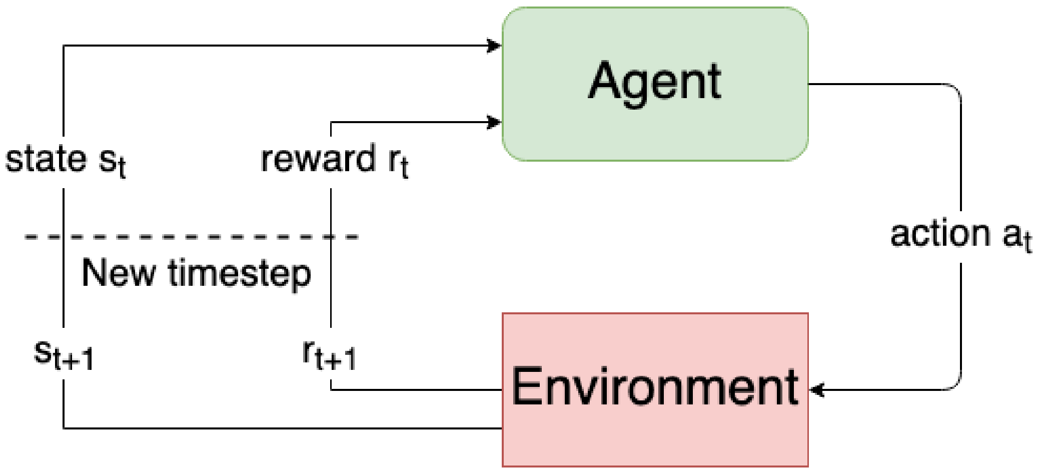


Figure 10: An agent that interacts with an environment through actions.

- $P()$ is the state transition function, and expresses the probability of a next state given the current state and an action: $P(s_{t+1}|s_t,a_t)$.
- $\mathcal{R}()$ is the reward function, yielding a number: $\mathcal{R}(s_t,a_t,s_{t+1})$.

An MDP yields *trajectories* τ, which consist of a sequence of states and actions:

$$\{s_0,a_0,s_1,a_1,\dots,a_{T-1},s_T\}$$

The function that determines what will be the next state is the transition function. Important to notice is that it is defined as $P(s_{t+1}|s_t,a_t)$, which expresses the *Markov property*. Formally the Markov property means:

$$P(s_{t+1}|s_0,a_0,s_1,a_1,\dots,s_t,a_t) = P(s_{t+1}|s_t,a_t) \tag{24}$$

This states that only the current state and action have an influence on which state will be encountered next. The transition function expresses an probability distribution over next states, and hence allows the environment to be stochastic. Generally, the transition function and the reward function are not known for the agent at the start of training.

Furthermore, it should be noted that the next state depends on the action, meaning the agent has influence on the upcoming states, and hence on the reward that comes with those states. How an agent selects its actions is represented by a *policy* which maps states to actions, hence $\pi : s_t \mapsto a_t$. The policy may be stochastic, meaning the probability of an certain action can be written as $\pi(a_t|s_t)$ and sampled actions as $a \sim \pi$.

2.8.2 Learning Objective

As every state action pair comes with an reward, the return of an episode is defined as follows (note that $R \neq \mathcal{R}$):

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t \tag{25}$$

γ is called the *discount* factor, and is required such that the expectation of the return given a policy π is bounded for problems with an infinite time horizon. Furthermore it influences how an agent weighs rewards from future states. γ is set somewhere between 0 and 1, where a value of 0 means that at time t only the next reward matters to the agent. r_t simply denotes the reward received at timestep t in the episode.

The goal of the agent is then to maximize the *expected* value of the return of a trajectory τ:

$$J(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{26}$$

Note here that the trajectory τ is generated by a specific policy π, which makes sense since the agents actions generate the new states and hence the trajectory. Therefore, the agent should modify its policy such that the expected return is maximized. It is expressed as an expectation as the initial states might differ when starting trajectories.

With this information two learnable functions can be derived, called value functions:

$$V^{\pi}(s) = \mathbb{E}_{s_0=s,\tau\sim\pi}[R(\tau)] \tag{27}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0=s,a_0=a,\tau\sim\pi}[R(\tau)] \tag{28}$$

Intuitively they mean: the total return the agent expects to receive when it starts in state s (and for $Q^\pi(s,a)$ when it starts with action a) and from there acts according to policy π. Note that the two functions are closely related. For example $Q^\pi(s,a)$ can be expressed as:

$$Q^\pi(s,a) = \mathbb{E}_{s_0=s,a_0=a,\tau\sim\pi}[r_0 + V^\pi(s_1)] \tag{29}$$

The latter is often used as target during training. With this equation, targets on which a value network can learn can be generated for every tuple (s_t, a_t, s_{t+1}). The network parameterized by θ then tries to minimize the following loss function:

$$L(\theta) = \frac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2 \tag{30}$$

where $\hat{Q}(s_t,a_t)$ is calculated in a manner similar to 29 (many variations exist).

2.8.3 Types of Algorithms

With these building blocks, learning algorithms can be distinguished in the following classes.

Policy-Based Algorithms These algorithms learn only the policy π, directly from experience. An advantage of policy-based algorithms is that they try to optimize the objective directly (maximizing $J(\tau)$). One of the first policy-based algorithms is REINFORCE [53], which still forms the basis for many of the algorithms developed in later years.

Value-Based Algorithms These algorithms try to learn one of the value functions $V^\pi(s)$ or $Q^\pi(s,a)$. When such a value function is learned correctly, a policy can be constructed from it. This is easy especially for $Q^\pi(s,a)$, as then one can simply take the action a in state s which maximizes that function. For $V^\pi(s)$ this is less trivial, as this function does not contain information about actions. Hence, often simply $Q^\pi(s,a)$ is learned. Examples of value-based algorithms are SARSA [54] and Deep Q-Networks (DQN) [55].

Model-Based Algorithms These algorithms try to learn (or use, if available) a model of the environment's transition function. If this function is known, the agent can use it to predict and plan the optimal trajectory for some timesteps ahead. This can then be done without actually executing the actions. Examples of such algorithms are Monte Carlo Tree Search (MCTS) as used in AlphaGo [56] and iterative Linear Quadratic Regulators (iLQR) [57].

Combined Algorithms Algorithms in this family combine algorithms described above. Actor-Critic algorithms are known to learn both a policy and a value function. In such algorithms, the policy (actor) learns from the value function (critic) instead of only from the rewards observed in the corresponding trajectory. Examples of Actor-Critic algorithms are Proximal Policy Optimization (PPO) [58] and Soft Actor-Critic (SAC) [59].

2.8.4 On-Policy and Off-Policy Algorithms

An important distinction that can be made between algorithms is whether they are *on-policy* or *off-policy*. The difference between on-policy and off-policy algorithms lies in their use of data. On-policy algorithms can only make use of data/experience gathered with the policy that is being updated. Thus, when an updated is performed, new experiences must be gathered and old experiences are discarded. The result is that on-policy algorithms have a low *sample efficiency*. Off-policy algorithms however, can use experiences generated by any policy, making them much more sample efficient. This can impact the choice of the algorithm, as in some environments it is costly to gather experience, hence an off-policy algorithm would be preferable in that case.

2.8.5 Deep Reinforcement Learning

One might have already noted that the value functions and the policies are both simply functions, meaning that they can be approximated using a neural network. This means that the value functions and the policy will be parameterized by all the weights θ of the neural network. The input-output pairs needed to train a neural network will be constructed when gathering experience. From here on, the techniques described above can be used to train the neural network.

2.8.6 Difficulties Of Reinforcement Learning

Since there is often no ground truth on how to act optimally in an environment, the agent needs to explore the state-action space by itself. These spaces can be extremely large, meaning the agent can never visit all state-action pairs (let alone multiple times) in some environments. This is where the generalization of neural networks plays an important role, where the hope is that the learned functions generalize to unseen states.

As the policy is always changing, some previously learned information might no longer be correct. However, if an agent always acts deterministically in a state, this might mean that it will never explore another action in that state. Most of the time it is desirable that the agent takes the action it thinks is best, but this might be wrong, and hence it is important to build in an exploration component, such that the agent does not get stuck in locally optimal policies. This is called the *exploration* versus *exploitation* tradeoff. Good exploration is required to let the agent explore the state-action space enough, exploitation is required to let the agent act optimally.

Another difficulty of RL, already touched upon lightly, is the data generation process. The data distribution from which data is generated constantly changes (as the policy changes), and sometimes of the training data and targets are based on *bootstrapping*, using current estimates as part of the target value.

2.8.7 REINFORCE

The REINFORCE algorithm [53] on itself is not stable enough to solve complex problems. However, it still forms the basis for most of the algorithms developed later. For this reason, its core concept will be discussed briefly here. REINFORCE aims at optimizing the policy by estimating its gradient (in order to maximize the objective). Remember that the objective is to maximize the returns given a certain policy π parameterized by θ:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{31}$$

REINFORCE then tries to find the estimate of the gradient of the objective with respect to the parameters of the policy:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{32}$$

The problem in Equation 32 is that the return is not directly dependent on θ, and it cannot be sampled. REINFORCE derives the policy gradient in a manner such that it can be sampled:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] && (33)\\
&= \nabla_\theta \int_\tau P(\tau|\pi_\theta)R(\tau) && (34)\\
&= \int_\tau \nabla_\theta P(\tau|\pi_\theta)R(\tau) && (35)\\
&= \int_\tau \frac{P(\tau|\pi_\theta)\nabla_\theta P(\tau|\pi_\theta)}{P(\tau|\pi_\theta)}R(\tau) \qquad \text{multiply by } \frac{P(\tau|\pi_\theta)}{P(\tau|\pi_\theta)} && (36)\\
&= \int_\tau P(\tau|\pi_\theta)\nabla_\theta \log P(\tau|\pi_\theta)R(\tau) \qquad \nabla \log x = \frac{1}{x} && (37)\\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\pi_\theta)R(\tau)] && (38)
\end{aligned}$$

This form allows us to sample experience and compute the gradient with respect to θ in order to perform policy updates. REINFORCE then uses this form to iteratively sample an episode after which it performs an policy update. In Equation 38, instead of the $R(\tau)$, sometimes a baseline is used, such as $R(\tau) - V^{\pi}(s_0)$. This reduces the variance in the updates as it centers the returns, and thus high rewards do not yield larger updates in comparison to small rewards.

2.8.8 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [58] is based on Trust Region Policy Optimization (TRPO) [60], which optimizes the policy with one major difference in comparison to other algorithms: it constrains the size of the policy update in terms of the Kullback-Leibler divergence between the current and the next policy. This shields the algorithm against making steps that change a policy drastically. It changes the objective to maximize into (this is TRPO):

$$J(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right],$$
$$\text{subject to } \mathbb{E}_t\left[KL(\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t))\right] \leq \delta$$

where θ_{old} are the current parameters, A_t is the advantage function: $Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t)$, $KL(\cdot,\cdot)$ specifies the Kullback-Leibler divergence between the new and the old policy and δ is the upper bound for the difference between the two policies during an update (the authors of [60] used $\delta = 0.01$). Intuitively, it makes sure no policy updates are done that change the policy too much in a single update.

PPO strives to achieve the same goal as TRPO, but using a penalty. Instead, they propose the following objective, introducing a clip term on how much the policy can change. Following the notation of the authors in [58], let us define the ratio $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. The clip term then limits the ratio by which the policy can change as follows:

$$J_{clip}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)\right] \tag{39}$$

The intuitive interpretation of the new objective can be given by considering two cases. First, when A_t is positive, in order to maximize the objective, $\pi_\theta(a_t|s_t)$ should become more likely, which increases r_t. The clip term then limits how much it can increase. When A_t is negative, in order to maximize the objective, $\pi_\theta(a_t|s_t)$ should become less likely. Again, the clip term then limits how much this will be. Furthermore, in order to encourage exploration the entropy of the policy can be added, using a weight, to the objective. This prevents the algorithm from setting very high probabilities on certain actions early on in training.

2.9 Artificial Intelligence Applied To Buildings

Using ANNs for system identification was already suggested in 1990 in [61]. Through the years, as Deep Learning techniques and more computing power came available, more research was devoted to this subject. Some examples of system identification using applied to buildings can be found in [62] [63] [64].

An important part of system identification for building control is indoor temperature prediction. Although many machine learning methods have been used for this purpose (for a recent survey see [65]), RNNs have proven to be good candidates for predicting indoor temperatures. In [66], a novel LSTM-based approach was able to predict indoor temperatures both 5 minutes and 30 minutes in advance, based on 2 months of data. In [67], an LSTM-based sequence-to-sequence framework was proposed, achieving good results in predicting indoor temperatures 6 hours ahead at a 10 minute interval.

Many research has already been done with respect to machine learning and smart building controllers. Many different types of controllers were proposed under which fuzzy controllers [68] [69] [70] and PID neural networks [71] [72]. An extensive survey of AI-assisted HVAC control is given in [73]. Although research has shown that fuzzy controllers achieve good performance, they are based on rules, which may be difficult to come by and vary for every system, which makes it difficult to scale the approach.

Another way to incorporate AI into the control of buildings is by estimating heating or cooling load in order to predict energy usage of buildings. Being able to predict the energy load of a building allows for Demand-Response control, meaning one can adjust its control scheme to use energy when it is least expensive. This is especially relevant for smart grid solutions. Load prediction has been done with for example ANNs [74], Support-Vector Machines (SVMs) [75] [76] and ensemble learning (regression trees) [77] [78]. An RL approach to this problem studied in [79] trained an agent to tune setpoints for all zones simultaneously, based the expected energy loads. Although tested in a simulation, they found that their approach was able to reduce energy significantly. A recent study used an LSTM network to predict energy consumption in combination with MPC using Particle-Swarm Optimization (PSO) to optimize the energy consumption in a demand response scenario, decreasing the energy need by approximately 35% [80].

The recent developments in the field of Deep Reinforcement Learning (DRL) have also been applied to HVAC control. A recent survey showed the promise of these methods with respect to improvement of performance [81]. Although promising, the survey concluded that most DRL-based methods are studied within a simulation setting, and gives as reason that DRL-based agents require much time to train and real-time interactions, making online deployment hard and not desirable. Furthermore it argued that model-based DRL approaches are preferred over model-free DRL approaches, due to the related fact that model-based methods are much more sample efficient, and hence require less training time. Another survey [82] showed that of the 77 articles that were studied:

- The majority of the DRL agents in the studies did not control the actuators of the building directly. Instead, for example the setpoint is modelled towards a controller needs to regulate. classic controllers still needed to track the setpoints determined by the DRL agent.

- 91% of the studies did not include previous states as inputs.

- 90% of the studies used data simulated by simulation software such as EnergyPlus.

- 83% of the studies did not include predictions as inputs.

- Only 11% of the studies implemented and tested their approaches in an actual building. Again it is argued that the demanding training process is one of the main causes for this.

Both surveys agree on the fact that no standardized benchmark exists for this problem, which have advanced the research in computer vision, speech recognition and RL.

Reinforcement Learning has been applied onto many different parts of the buildings control. In [83], the main supply temperature is determined and compared to the baseline as provided by ASHRAE, which is based on the outside air temperature. Another study trained different Q-networks for all zones in the building, where temperature violations were kept approximately constant, but cost of regulating was decreased significantly [84]. In [85], 5% - 12% energy savings were achieved, also by setting supply temperature setpoints. It made the action space discrete by letting the agent choose between five different supply temperatures. All three studies mentioned above used simulated data and evaluated the performance using a simulator. To the best of the author's knowledge, only one study tried learning a DRL using an environment based on a model which learned the system dynamics of the building [86] (hence, it did not need a simulator). Here, an LSTM was used to approximate state transitions from state action pairs, on which an agent was trained. The agent learned to determine optimal setpoints for three air handling units (trained separately), decreasing the energy consumption with 27%-30% while maintaining thermal comfort.

Differences This study differentiates itself from the literature presented above by:

1. Using real data gathered from real buildings to train and evaluate on. This is more realistic compared to data generated by a simulation program;

2. Controlling the actual actuators of the building, rather than the setpoints. Therefore, it completely substitutes classical control methods;

3. Requiring no extra metadata on the building such as location, construction or information about the physical properties of the room. It only uses the sensor data gathered from the building, in combination with weather data;

4. Testing the obtained controller on the actual building, validating its performance in the physical world.

3 Methods

This section will introduce the details of the application of the methods used to achieve the goal as described in Section 1, using the concepts introduced in Section 2.

3.1 Data

In order to use the approach described in this study, a time series data set is needed which reveals the dynamics of the system at interest. A constant interval is assumed, which can and should be varied for different processes, i.e. for fast processes this interval can be 2 minutes, whereas for a slow process (regulating a swimming pool for example) the interval can be 10 minutes. Data from Climatics is always available at a 2 minute interval, hence it should be re-sampled when training for relatively slow processes.

At every timestep t, all sensor values are observed, which will be denoted as the observation vector $\mathbf{o}_t$. From the day a building starts logging, up until now, it will collect these observation vectors at a fixed time interval (every 2 minutes), creating a sequence of observations $\{\mathbf{o}_0, \mathbf{o}_1, \ldots, \mathbf{o}_T\}$, where T is the current time. It is important to note that a single observation $\mathbf{o}_t$ does not yield the full system state, as it will consist of temperatures and values moving in a particular direction and this direction is not included in $\mathbf{o}_t$.

An observation at a specific timestep consists of the following components:

1. Values for all sensors of the building relevant for the specific process. These can be splitted into three types of sensors:
   - The sensor(s) to model. For a room this will be the sensor that measures the temperature of the room.
   - Actuator sensors. These are all the sensors which can be controlled by the system in order to achieve the control objective. For a room this might be the valve which determines how many hot water is let into the radiator.
   - Remaining sensors. These sensors cannot be controlled and will not be modelled, but they are relevant for the control process. As an example, this might be a supply temperature of the water leaving the boiler, before it reaches the radiator.
2. Values for weather data. In this study, weather data is retrieved using the DarkSky API, which provides hourly weather data. In the data set, weather data is padded (thus during an hour the weather data remains constant). The following weather data is used:
   - cloud cover, a number in the range $0-1$
   - humidity in %
   - UV index
   - wind bearing in degrees (0-360)
   - wind gust
   - wind speed (in m/s)
   - outside air temperature

   The DarkSky API allows us to retrieve forecasts for an hour ahead for all of these values.

3. Encoded cyclic time information. Three cycles can be identified for a data set of a building:

    - Day cycle, starting at 00:00 and ending at 23:59. This cycle allows the model to learn an occupancy schedule of the building, in combination with the week cycle.
    - Week cycle, starting on Monday and ending on Sunday. This allows the model to distinguish between weekdays and weekends.
    - Year cycle, using the week number. This allows the model to learn patterns occurring in different seasons.

    All cycles are mapped to a unit circle, with the corresponding frequency. As there are 86400 seconds in a day, the data will be mapped to the unit circle using

    $$y_{minutes} = \sin(2\pi x/86400)$$
    $$x_{minutes} = \cos(2\pi x/86400)$$

    Here, x is the relative second of that day. When using 2 minute intervals these will be multiplies of 120. This will yield a point on the unit circle relative to the current position in the cycle. For the week and year cycles the denominators will be 7 and 53 respectively.

3.1.1 Data format

In general, the data will be used to simulate short episodes of historical data. In order to do this, a certain start time t is considered, which should be somewhere within the available period. Then the *input sequence* will be several hours leading up to t. The *target sequence* will be several hours after time t. During real-time control, t will be the current time. Note that during real-time control, the Neural Twin can only be used to predict/simulate ahead when the network is trained in such a way that the decoder accepts only control values and process values. This is because during control it is unknown what the feature values will be of the other factors (room occupation or weather conditions for example). This matters only when using the Neural Twin for MPC. When using a DRL agent, the Neural Twin is not needed during inference, and the agent will only use values from the past timesteps to directly produce an action. In this case, historical values can be used as inputs for the decoder, next to the simulated control values and process values.

3.1.2 Case study: sports hall at Duurswoldhal Slochteren

The data used in this study for the real world scenario was extracted from the Duurswoldhal at Slochteren. All sensors relevant for the sports hall were selected. This consisted of the following sensors:

- Room air temperature in °C
- CO2 sensor that measures the parts per million (PPM) in the air, i.e. the concentration of the air
- Valve position in %
- Central supply temperature °C
- Outside air temperature in °C

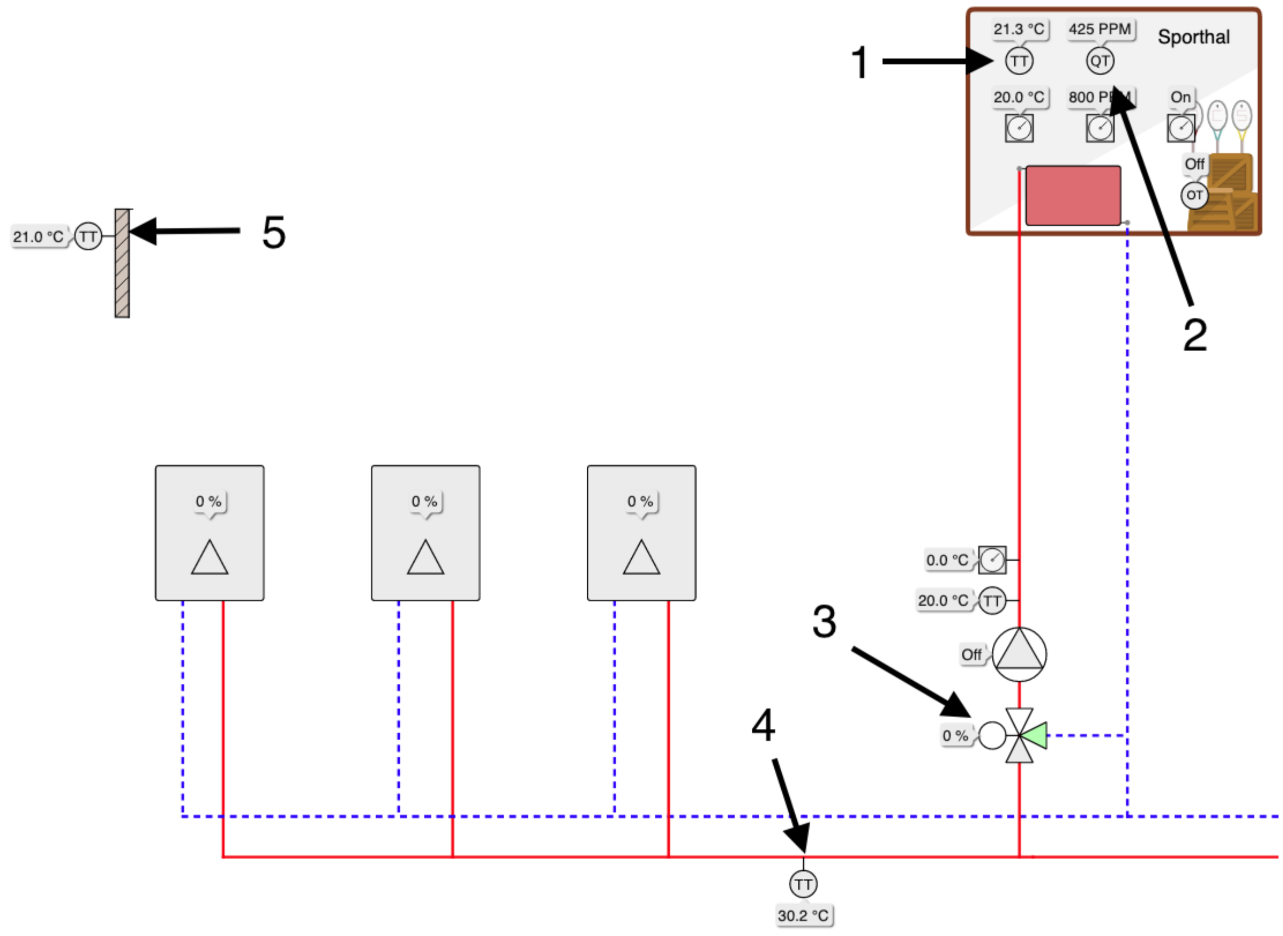


Figure 11: Part of the schematic for the Duurswoldhal Slochteren. The schematic shows the relevant sensors for the 'Sporthal' (Dutch for sports hall). Sensor 1: air temperature of the sports hall. Sensor 2: CO2 sensor, measuring parts per million (PPM), i.e. the concentration of the air. Sensor 3: valve position for the radiator. Sensor 4: central supply temperature. Sensor 5: outside air temperature.

Data was retrieved between the period 2019-11-01 until 2021-06-01. Hourly weather data was retrieved and aligned (using forward padding) with this data set and cyclic information was computed and added. The data was re-sampled to have 10 minute intervals, as the process of heating a sports hall with a radiator is fairly slow. The data set consisted of 578 days, yielding approximately 80000 observations, where every observation consisted of 17 values. A schematic of the control system can be found in Figure 11.

3.1.3 Train-validation-test split

In order to evaluate the trained models, the data was split into three parts. 60% of the data was used for training the Neural Twin and the DRL agent, 20% for validation and 20% for testing. In order to make sure that no overlapping data was used in the validation or test set, Algorithm 1 was used for sampling validation and test sets from the full data set. At first, the training set is initialized to be the full data set. Then, it samples a random time in the train data set which will mark the beginning of a target sequence. The complete target sequence will then be removed from the train set. This sampling repeats until the train set is 20% smaller. Therefore, while there may be overlap between training examples, there will never be overlap between validation or test target sequences, either with themselves or with examples from the training set. In total, this yielded approximately 8300 hours of data for training, 2700 hours for validation and another 2700 hours for testing.

Algorithm 1: Sampling a subset of non-overlapping examples

```
Result: $\mathcal{D}_{new}$, non-overlapping examples for 20% of the data
Input: data set $\mathcal{D}$;
initialize empty data set $\mathcal{D}_{new}$;
determine number of samples $N$ of $\mathcal{D}$;
while remaining samples in $\mathcal{D} > 80\%$ of $N$ do
    sample a random start time $t$ from $\mathcal{D}$;
    extract input and target sequence around $t$ and add to $\mathcal{D}_{new}$;
    remove all data in the target sequence from $\mathcal{D}$;
end
```

3.1.4 Preprocessing

Before the data can be fed to a Neural Twin, it needs to be preprocessed. This is because all sensors contain values in different ranges and their variance might vary due to its underlying nature (a valve position can change much more quickly than a room temperature). This can make it difficult for a neural network to adjust it weights properly. Therefore, in this study all data is normalized to the range $0-1$.

3.2 Neural Twin

The Neural Twin will be an Encoder-Decoder model based on GRUs.

In the context of control systems, Encoder-Decoder architectures can be interpreted as follows. Both the encoder and the decoder share the same memory structure (the GRUs have same hidden size). At every timestep the encoder receives the observation (sensor values) at that time. Over time, the encoder will then encode the current state of the system, based on these past observations:

$$\{\mathbf{o}_{t-h}, \dots, \mathbf{o}_{t-2}, \mathbf{o}_{t-1}\}$$

The hidden state of the encoder then contains a compact representation of the state of the system. The hidden state is used as initial hidden state of the decoder. The output of the decoder will consist of the process values of interest. This can be realised by placing one or several extra fully connected hidden layers interpreting the hidden state of the decoder, which outputs the process values. However, as opposed to the encoder input, the input of the decoder might consist of only the process and control values of the system (excluding any external variables such as central supply temperature) depending on the use case of the complete model.

For MPC the latter is desirable because at inference time the model will not have access to the future values of external variables. When constructed like this, one can use the outputs of the decoder in combination with newly generated control values to simulate a trajectory over a short horizon.

If, however, the goal of the model is to simulate the system as good as possible, one can choose to let the input of the decoder consist of the full observation. In this case, as input for the decoder, historical data values of these external variables can be concatenated with the output of the decoder

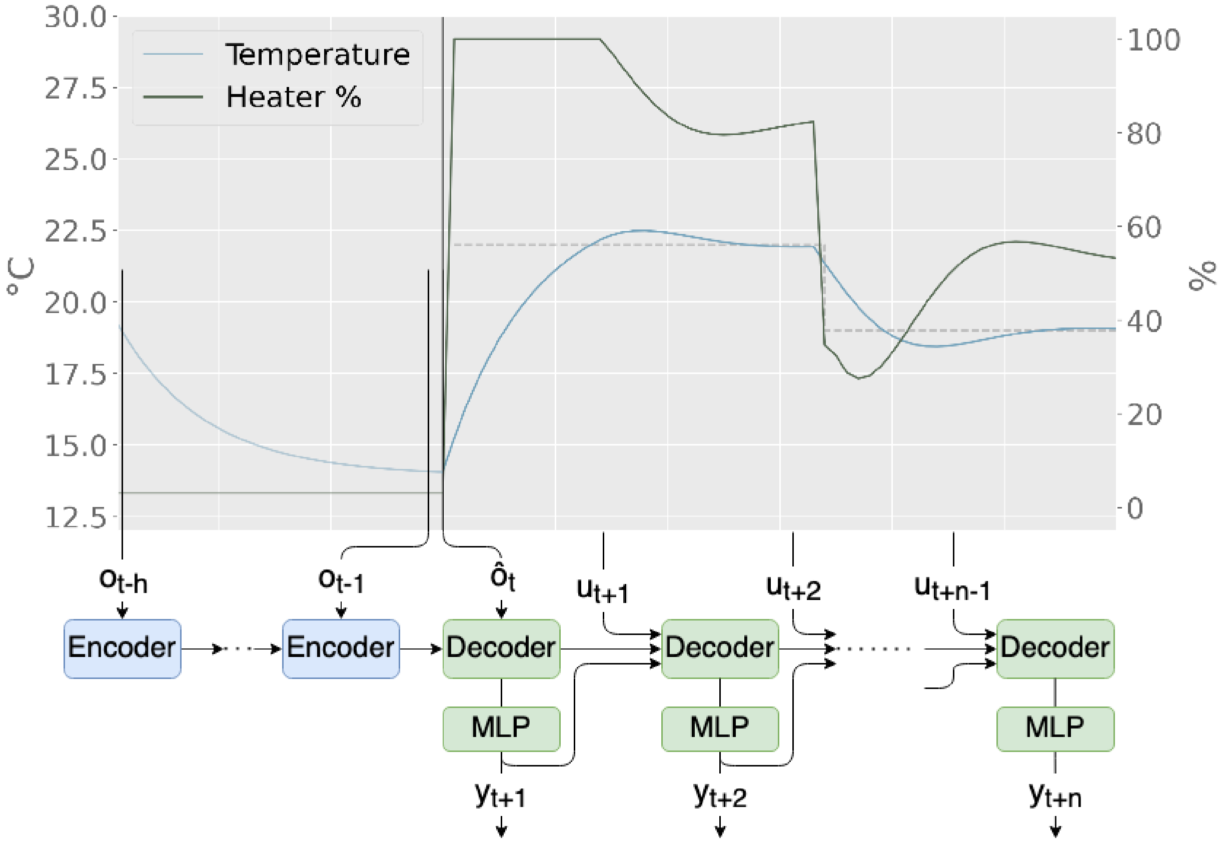


Figure 12: A schematic view of an Encoder-Decoder model where the decoder model expects different inputs then the encoder model. The black vertical line represents the split between past observations and future observations. Past observations $\{\mathbf{o}_{t-h}, \ldots, \mathbf{o}_{t-2}, \mathbf{o}_{t-1}\}$ are fed into the encoder. Then an initial observation $\hat{\mathbf{o}}_t$ is fed to the decoder and from there the decoder simulates using its previous output $\mathbf{y}_{t+1}$ and newly generated control values $\mathbf{u}_t$ to generate output $\mathbf{y}_{t+2}$, and so on.

and generated control values. This might be desirable when one wants to train a Reinforcement Learning agent in a simulation setting. Then, the Encoder-Decoder model will not be needed during inference, as in that case the trained agent will take over the control (which does not require the simulating ahead).

A schematic view of an Encoder-Decoder network in the context of a control system is given in Figure 12. This figure shows the case where the decoder network omits external variables in its input. If this would be added, it would mean that at every timestep historical values for these external variables should be added.

Training scheme The Neural Twin is trained using gradient descent as described in Section 2. It is trained for several epochs using mini batches. An important concept called *teacher forcing* is used during training. Teacher forcing means that during training instead of its own output from the previous timestep, the decoder receives the target value of the previous timestep as input. This helps speed up convergence at the beginning of training, as in the beginning it is likely that at the end of a target sequence the inputs for the decoder will be far from the realistic inputs when using its own outputs. During the course of training, less teacher forcing will be used and eventually the decoder will only use its own outputs. Then, when teacher forcing is no longer used, the learning rate is decreased exponentially over the remaining epochs. In order to enforce regularization, simple weight decay is used. This forces the weights to remain smaller, which helps against overfitting. The general training scheme can be viewed in Algorithm 2.

Algorithm 2: General training scheme of a Neural Twin

```
Input: data set $\mathcal{D}$;
init encoder and decoder network parameterized by $\theta$;
set number of epochs $E$;
set learning rate $\eta$ to initial value;
for epoch in $E$ do
    for minibatch in $\mathcal{D}$ do
        if teacher forcing then
            predict sequence $\hat{\mathbf{y}}$ for the minibatch using original data for the decoder;
        else
            predict sequence $\hat{\mathbf{y}}$ for the minibatch using the decoder's own output;
        end
        calculate the loss $L(\hat{\mathbf{y}}, \mathbf{y})$;
        perform parameter $\theta$ update using the Adam optimizer;
    end
    update learning rate;
end
```

Simulating behaviour of the Neural Twin In this study inputs of 30 timesteps are used. For 10 minute intervals this means the encoder is fed 6 hours of data. The decoder outputs 30 timesteps as well, meaning a the Neural Twin learns to predict 6 hours ahead. As the Neural Twin uses its own predictions as inputs for the next prediction, errors accumulate over time, hence the Neural Twin should not be used to simulate a process indefinitely. Instead, it is advised to use the Neural Twin for no longer than the prediction window that was trained on (the 30 timesteps). After this, a new start point should be determined from which the process can be simulated again.

3.3 Control Agent

The trained Neural Twin can be used as a simulator for a DRL agent that will try to learn an optimal control policy. In this case it is desirable to provide full input to the decoder of the Neural Twin, as it will only be used as simulator, meaning historical data can be used for the sensors other than control actions and simulated values.

In order to learn the control of a process the following aspects should be determined:

1. What will be the observation space for the agent to learn? What will an observation consist of?
2. What will be the action space of the agent? How will the action of an agent affect the environment?
3. What will be the reward function that is used by the agent to evaluate experiences?
4. Which algorithm will be used for learning and what does the main training scheme look like?

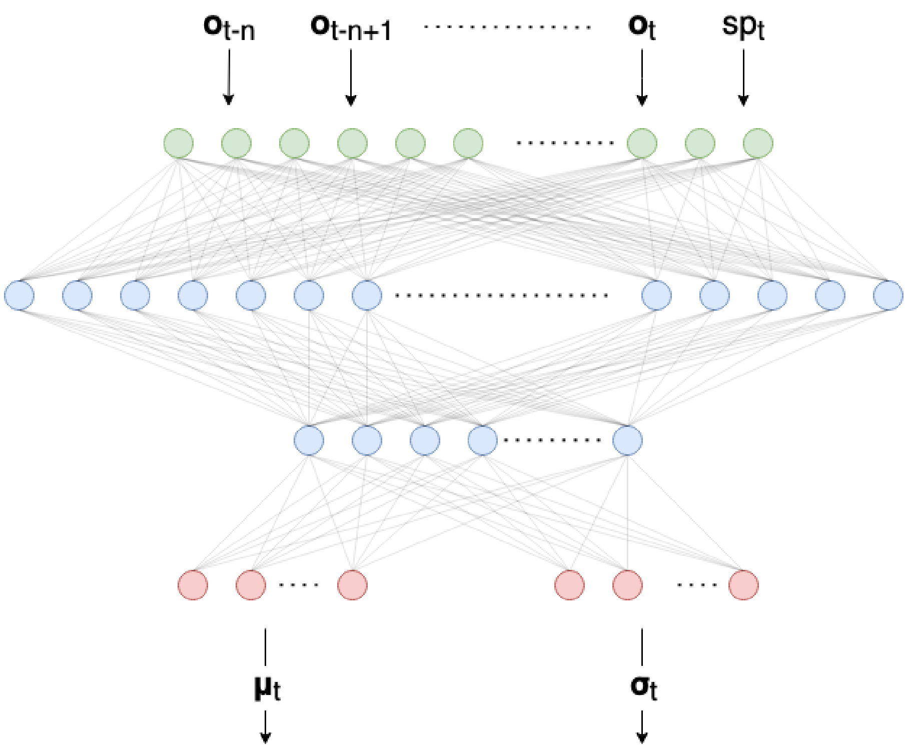


Figure 13: A schematic view of the policy network used by the agent. It receives as input a number of past observations and the current setpoint sp_t, and outputs means $\boldsymbol{\mu}_t$ and standard deviations $\boldsymbol{\sigma}_t$ for every actuator of the system. The distributions describe offsets for the current actuator values. Actions can then sampled according to $\mathbf{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t)$.

Observations In this study, an MLP is used to parameterize the control policy learned by the agent. These types of networks expect only a single vector as input. However, giving only the current observation $\mathbf{o}_t$ as input for the network will not be enough, as the network will not be able to extract higher order movements of the system (for example is the temperature increasing or decreasing?). Inspired by [55], where multiple frames of an Atari game were concatenated, instead of only the current observation, the past n observations will be concatenated. In addition to this, the agent will need to know the (preprocessed) current setpoint of the room. The setpoint will be concatenated to the past observations and these will serve as inputs for the policy network.

Actions The output of the policy network will be an offset for the current control values. Hence, if there is a single actuator, its current value is 50 and the output of the network is -2, the actuator's value will become 48. In order to perform exploration during training, the network will output a mean and a standard deviation for every actuator of the control problem, i.e. $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$. During training, a value will be sampled from the distributions formed by these outputs, i.e. $\mathbf{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t)$. During inference, the means will be used as actions. A schematic view of the policy network is given in Figure 13.

Reward In order for the agent to know how well it performs, it needs a reward function. The reward function describes quantitatively how 'good' the current state is. The reward function consists of three parts, which need to be weighted accordingly. They account for the following three goals:

1. Thermal comfort: this part is represented by the difference in temperature and setpoint. Hence,

the reward will be the negative of the absolute difference between these two, as the closer the temperature is to the setpoint, the higher the reward should be:

$$R_{thermal} = -|T_{room} - SP_{room}| \tag{40}$$

2. Stability: this part is concerned with the stability of control. The more stable the control is, the fewer physical actions an actuator has to make, increasing the durability of the physical controllers in the building. Furthermore, this will make it easier to predict the behaviour of the building, which helps when optimizing the building from the supply side. Inspired by the Hamiltonian, the first order movement of the control values will be used. This yields the following reward:

$$R_{stability} = -\frac{1}{M}\sum_{i=0}^{M} |a_i| \tag{41}$$

This makes intuitive sense: if the offsets are small, the control sequence will be more stable. Furthermore, it allows the control of the speed of the control policy. The higher η, the slower the control will be, as large offsets will be discouraged.

3. Energy consumption: this part is concerned with decreasing energy usage. If two control sequences both achieve the same reward for thermal comfort, obviously the control sequence using the lowest amount of energy is preferred. Although at the locations energy sensors were not present, the energy consumption is proportional to the integral of the control sequences (in a heating scenario). Hence, keeping this integral as low as possible will yield lower energy consumption. Assuming control values are modular and can be within the range $0\% - 100\%$ (although easily modifiable towards other control ranges), this can be calculated by taking the mean of the current control values:

$$R_{energy} = -\frac{1}{M}\sum_{i=0}^{M} u_i \tag{42}$$

The total reward function then will be:

$$R_{total} = R_{thermal} + \alpha R_{stability} + \beta R_{energy} \tag{43}$$

Here, α and β can be used to weigh the different parts of the reward. The maximum value this reward can be is 0, as all rewards on themselves can at most be zero. The aim of the agent is to find a policy that maximizes this reward function.

Algorithm and Training Scheme The PPO algorithm is used to train the agent. The algorithm expects episodes, which will be simulated using the Neural Twin. Although the control problem is in theory infinite, the Neural Twin becomes less accurate as more time passes due to the accumulation of error. Therefore, episodes of 6 hours are used. At the start of an episode, a random time available in the train set is selected which acts as the start of the episode. Historical data from this point on is used as input for the decoder during the episode. Random setpoints are generated for the episode, within reasonable bounds of the starting point. The agent then gathers experience for a number of episodes, and then uses these experiences to perform an update for its value and policy network according to the PPO clip objective. The training takes place according to Algorithm 3. After training, this yields a network that can be used to control the process. During training, the agent is evaluated by

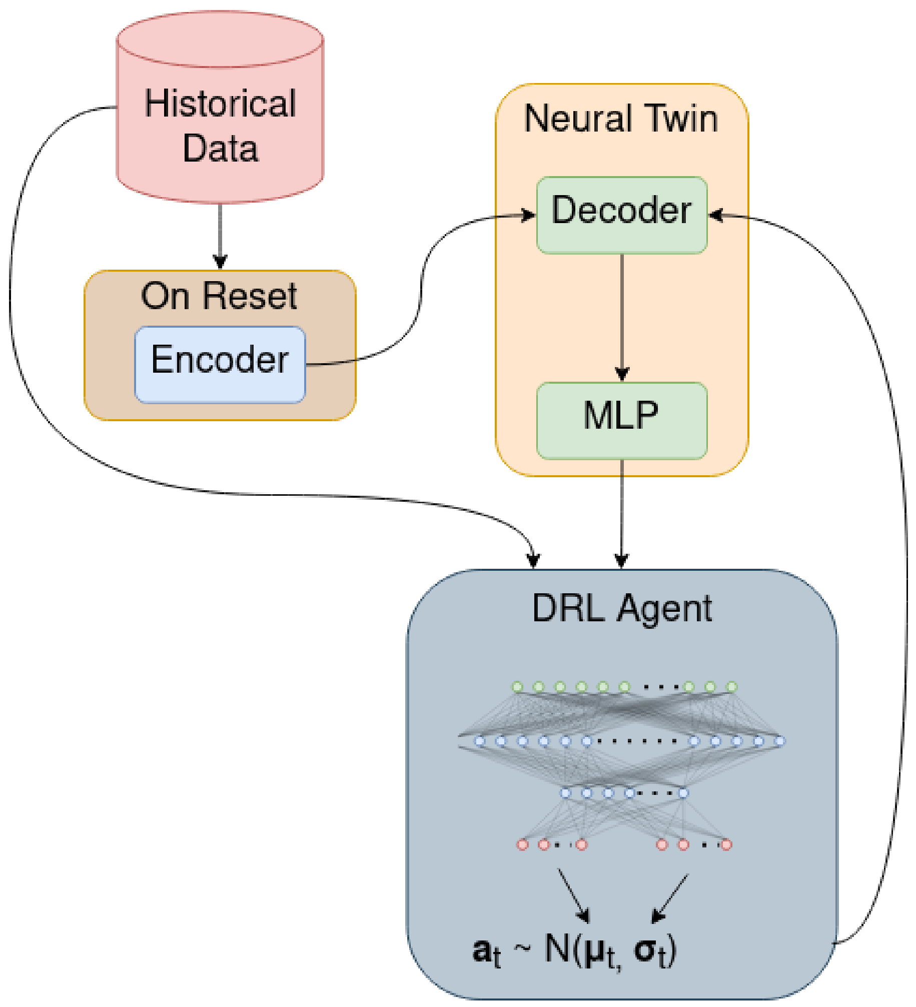


Figure 14: A schematic view of the training scheme of the DRL agent. On reset (at the start of an episode) historical data is fed to the encoder. The state of the encoder is passed to the decoder, which starts simulating. The output of the decoder, together with historical data (weather conditions or external factors) are fed to the agent which generates control values. These are then fed back to the decoder which simulates the next timestep.

performing a number of episodes that start at a time which is in the validation set. The agent will act deterministically and the mean return of the test episodes are remembered and logged over time in order to track the progress during training.

The agent can then easily be used to control the process according to Algorithm 4. A schematic view is given in Figure 14.

Algorithm 3: Training a control agent using PPO

```
Input: Neural Twin $NT$;
init policy network $\theta_\pi$;
init value network $\theta_V$;
set number of episodes per train step $N$;
set number of batches per train step $B$;
set number of updates per batch $U$;
while episodes seen $<$ MAX_EPISODES do
    for episode in range $N$ do
        reset environment;
        while episode not done do
            get exploration action $\mathbf{a}_t$ using $\theta_\pi$;
            perform action in the environment using the Neural Twin;
            store experience tuple $\langle \mathbf{o}_t, \mathbf{a}_t, r_t, \mathbf{o}_{t+1} \rangle$;
        end
    end
    set $\theta_{\pi_{old}} \leftarrow \theta_\pi$;
    calculate advantages $A_t$;
    for update in range $U$ do
        update $\theta_\pi$ by maximizing $J_{clip}(\theta_\pi)$ using gradient descent;
        update $\theta_V$ by minimizing $J(\theta_V)$ using gradient descent;
        if $KL(\theta_{\pi_{old}}, \theta_\pi) > \delta$ then
            break;
        end
    end
    reset memory;
end
```

Algorithm 4: Controlling a process with a trained agent

```
Input: Control policy network parameterized by $\theta_\pi$;
while controlling do
    observe $\mathbf{o}_t$ from the environment;
    deterministically calculate action $\mathbf{a}_t$ using $\theta_\pi$ and $\mathbf{o}_t$;
    perform action in the environment;
end
```

3.4 Model-Predictive Control

Once a Neural Twin is trained, it can be used to perform Model-Predictive Control (MPC). Note that for MPC the decoder of the Neural Twin can only accept the sensors it predicts and control values as input, as at run time future values for other sensors will not be available yet. It is advised to set the length of the target sequences shorter in this case as fewer information is available to make good predictions over time (for example only 1 hour ahead instead of 6 hours).

In this study, the random-sampling shooting method is used for MPC as described in the previous section. When applied to the process of controlling a room, it generates K random action sequences for a specific horizon h. The Neural Twin is now used to simulate ahead the process value trajectory $\mathbf{y}$ for each these control sequences. For all of the trajectories the cost can be calculated, and the first action of the action sequence $\mathbf{a}$ that yielded the lowest-cost trajectory will be executed. At the next timestep, this is repeated. The actions that will be generated are same actions as for the agent, hence the actions will denote offsets for the current control value. In this study the actions were within the range of -50% and 50%.

Cost function The cost function plays the key role during control, as it determines what is the best trajectory, and hence what will be the best control sequence. The cost function consists of the exact same parts as the reward function for the DRL agent: optimizing for comfort, optimizing for stability and optimizing for energy consumption. MPC however aims to minimize a cost function, thus instead of multiplying the rewards with -1, as was done for the reward function for the DRL agent, the rewards here will be kept strictly positive. Hence, the lower bound of the cost is simply 0. This yields the simple cost function:

$$cost(\mathbf{u},\mathbf{y}) = \sum_{t=0}^{h} |y_t - SP_t| + \alpha|a_t| + \beta u_t \tag{44}$$

where SP_t denotes the setpoint, y_t represents the predicted temperature, a_t represents the action taken and u_t represents the control value. The index t here represents the offset in time, where $t = 0$ is the current timestep.

The total cost of an episode will be defined as the sum of the costs obtained during that episode:

$$cost_{episode} = \sum_{t=0}^{T} cost_t(\dots) \tag{45}$$

Note that after training a Neural Twin, no additional training is needed, and it can be used directly for control. The complete control loop is given in Algorithm 5. However, the algorithm's performance is heavily dependent on the three hyperparameters λ, h and K:

1. When λ is too large, the control algorithm will simply never spend energy, which will result in the valve always being shut. When λ is too small, the algorithm will simply not pay enough attention to energy usage, unnecessarily using more energy than might be needed.

2. When h is too small, the controller might not foresee the influence its actions will have over a longer time, hurting its objective in the long run. h should also not be too large, as then the predictions of the NT might be too much off, hurting the performance (as actions do not achieve what is expected).

3. When K is too small, only a small portion of the action space will be covered. In this case a poor control sequence, relative to the optimal control sequence embedded somewhere in the action

Algorithm 5: MPC with random-sampling shooting method using a Neural Twin

```
Input: Neural Twin $NT$;
set energy cost weight $\lambda$;
set horizon $h$;
set number of random control sequences $K$;
while controlling do
    generate $K$ random control sequences $\mathbf{u}$ with length $h$;
    use $NT$ to predict trajectories $\mathbf{y}$ for all $\mathbf{u}$;
    determine the best control sequence $\mathbf{u}^* = \min_{\mathbf{u}}(cost(\mathbf{u},\mathbf{y}))$;
    execute first control action from $\mathbf{u}^*$;
end
```

space, might be selected during control. For larger K the probability of generating a control sequence close to the optimal sequence increases. However, also the larger K is the more computationally expensive the control algorithm becomes. Especially for high dimensional action spaces this might pose a challenge due to the real time constraints of control systems.

3.5 Reward vs. Cost Function

Although mentioned before, it should be noted that the reward function of the agent is the negative of the cost function used for MPC (given that α and β are set to the same values in both equations). Hence:

$$R_{total} = -1 * cost(\mathbf{u},\mathbf{y}) \tag{46}$$

Therefore, both methods are optimizing for the same objective, which makes it possible to compare their performance.

For both control algorithms, the thermal comfort part of the cost function is calculated in °C. The stability part is calculated in %, hence the values for both algorithms will lie in the range of -50% and 50%. The energy part is also calculated in %, but the range for these values lie between 0% and 100% for both algorithms.

In this study, α was set to 0.3 and β was set to 0.1. The parameters α and β could be used to enable different 'modes' for the control algorithms, as it lets the control algorithms focus more heavily on certain parts of the reward. For example, when β is increased, this means that the algorithms will focus more on lower energy consumption, and care less about the temperature violation as a result of this.

4 Experimental Setup

4.1 General outline

The ultimate goal of the study is to obtain a control policy that maximizes performance. Before such a policy is obtained, several steps need to be taken. For each of these steps, experiments were conducted for a total of 4 different environments. For every environment, the following experiments were performed:

1. One in which the validity of Neural Twins of an environment is verified.
2. One in which the validity of training a DRL agent on a Neural Twin of an environment is verified.
3. One that reveals the influence of the hyperparameters for MPC control.
4. One in which the two control methods are verified, based on control on the Neural Twin.
5. One in which the two control methods are verified by trying them out in a physical environment. This is done only for one environment.

Figure 15 shows the general flow of the experiments for all environments.

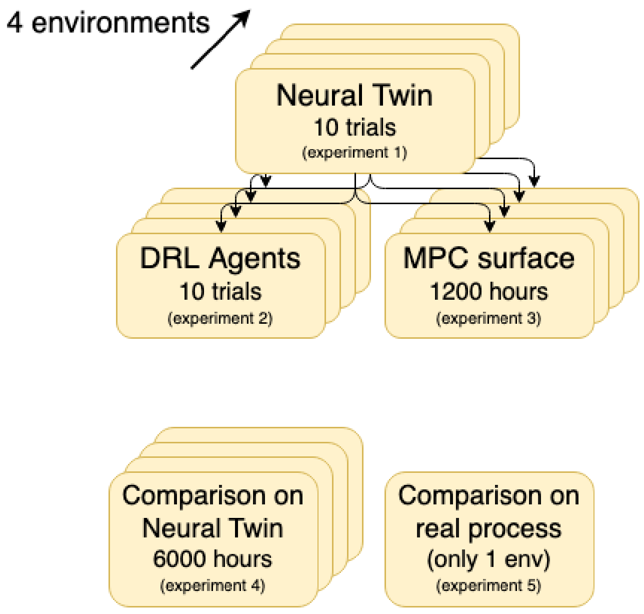


Figure 15: The general workflow of the experiments for all the locations.

4.2 Environments

A total of four environments is used, and every experiment except for the real world is done on each of the environments separately. This shows that the presented methods are generalizable to different environments with even different physical equipment without further manual tuning. The following four environments are used in the experiments:

1. Sports hall of Duurswoldshal in Slochteren. This is the environment already described in the previous section, and its sensors and schematic was already shown in Figure 11. Table 1 gives an overview of the sensors used for this environment. This environment is also used for the real world (physical) test of the controllers. This environment was chosen simply because it was not possible to test in the real world for all environments, but the building's owners of the Duurswoldhal were kind enough to allow us to test this use case there.

2. Sports hall of a gym in Hoogezand (as far as the author is aware, the building has no name). It consists of almost exactly the same components as the Sportshal in Slochteren, except there is no CO2 sensor in this setup. In Figure 16, the sports hall corresponds to room on the right. Table 2 gives an overview of the sensors used for this environment. Note that the this table is also used for the changing room environment (described next), only a different room air temperature and different valve is used. Data was available from 2019-10-01 until 2021-06-01.

3. Changing room of the same gym in Hoogezand. Although it consists of the same sensor setup as the sports hall, it is a completely different room, behaving differently. The changing room corresponds to the left room in Figure 16. Table 2 gives an overview of the sensors used for this environment, with a different room air temperature and a different valve compared to the model for the sports hall in Hoogezand. Data was available from 2019-10-01 until 2021-06-01.

4. Central hall of the sports center Kalkwijck. It is a relatively large room, and its main difference with the other environments is that it is regulated using an air handling unit (AHU). The total AHU is controlled by two actuators, instead of only one (as in the other environments): one controls how much air is let through from outside, the other controls how much the air is heated. Figure 17 shows the schematic of the room. It includes another room (a sauna), which is crossed out in the figure, as this room will be left out of consideration for this environment. The schematic does *not* include the flow sensor and the water temperature sensor of the main water supply (the red line leaving the figure in the bottom left corner), although these are included within the data. Table 3 gives an overview of the sensors used for this environment. Data was available from 2019-11-01 until 2021-06-01.

4.3 Experiments

Experiment 1: training Neural Twins

In order to inspect the performance of Neural Twins, several training trials were conducted for every environment. Every Neural Twin consisted of the same model structure: both encoder and decoder consisted of a single GRU layer consisting of 100 nodes. A single fully connected layer from size 100 to size 1 (as for all models there is a single output) is then stacked on top of the GRU layer of the decoder which interprets the hidden state. The models were trained with the mean absolute error (MAE or L1) loss function. For all models, an interval of 10 minutes was used between subsequent timesteps. For every environment, a Neural Twin was trained 10 times, and the results show the

Role	Sensor	Unit	#
Sensor to predict	room air temperature	°C	1
Sensor to control	valve position	%	2
Other sensors	outside air temperature	°C	3
	central supply temperature	°C	4
	CO2 concentration	PPM	5
Weather data	(described previously)	-	6 . . . 13

Table 1: All sensors used in the model for the sports hall of the Duurswoldhal.

Role	**Sensor**	**Unit**	**#**
Sensor to predict	room air temperature	°C	1
Sensor to control	valve position	%	2
Other sensors	outside air temperature	°C	3
	central supply temperature	°C	4
Weather data	(described previously)	-	5 . . . 12

Table 2: All sensors used in the model for the sports hall/changing room of the gym in Hoogezand. Both models use the same conceptual sensors, but for the actual model the room air temperature and valve differs for these models.

Role	**Sensor**	**Unit**	**#**
Sensor to predict	room air temperature	°C	1
Sensor to control	outside air suction valve	%	2
	heater valve	%	3
Other sensors	outside air temperature	°C	4
	supply fan	binary on/off	5
	exhaust fan 1	binary on/off	6
	exhaust fan 2	binary on/off	7
	flow main supply water	m^3/s	8
	main supply water temperature	°C	9
Weather data	(described previously)	-	10 . . . 17

Table 3: All sensors used in the model for the central hall in Kalkwijck.

Hyperparameter	**Initial value**	**Final value**	**Scheme used**
Batch size	32	-	-
Epochs	30	-	-
Gradient clipping	1.0	-	-
Weight decay	0.01	-	-
Teacher forcing chance	90%	0 %	$0.9-(0.1*epoch)$
Learning rate	0.01	0.00001	exponential decay

Table 4: Hyperparameters used when training Neural Twins.

Hyperparameter	**Initial value**	**Final value**	**Scheme used**
Total number of episodes	100000	-	-
Training frequency	every 100 episodes	-	-
Number of updates per step	80	-	-
γ (discount factor)	0.95	-	-
ε for clipping	0.2	-	-
Maximum KL divergence	0.01	-	-
Entropy weight	0.01	-	-
Gradient clipping	0.5	-	-
Learning rate	0.01	0.0001	linear decay

Table 5: Hyperparameters used when training DRL agents.

average loss and standard deviation for these 10 trials. Ultimately, the performance on the test set will be compared with the performance on the validation set. Table 4 shows the hyperparameters used for all trials.

Experiment 2: training RL agents

In the same spirit, 10 trials will be carried out for every environment on which an agent is trained using the Neural Twin (obtained earlier) of that environment. All agents have the same MLP structure $[in, 128, 64, 32, out]$, hence 3 hidden layers. All layers have ReLU activation functions except for the last layer, which has no activation function.

The mean return (Equation 25) is calculated over time, i.e. during training. This is done for both episodes from the train set (which happens automatically during training) and episodes from the validation set. For the episodes of the train set, the mean return of the episodes since the last evaluation point is used. For the episodes of the validation set, 20 episodes are selected at random, and the agent acts on them deterministically. During training the performance will suffer from the exploration the agent performs (by selecting actions stochastically) and therefore it is expected that the train performance can be lower than the validation performance.

Table 5 shows the hyperparameters used for all trials. The maximum KL divergence is realized by checking during the 80 updates done every train step whether or not the new policy is diverged more then the given maximum. If this is the case, the rest of the updates for that train step are cancelled.

Experiment 3: MPC performance

The goal of this experiment is to inspect the influence of the hyperparameters on control performance. The random-sampling shooting method (RSSM) is used for MPC. Recall that the two hyperparameters for RSSM are the horizon h and the number of random sequences K generated at every timestep. Hence, these two hyperparameters generate a plane of possible combinations. In order to measure the 'cost' of a combination of hyperparameters, the performance of the RSSM algorithm is measured throughout 200 episodes (1200 hours) of control on the Neural Twin of that environment. The total cost (according to Equation 45) of all episodes is averaged and remembered as 'cost' for that specific combination of hyperparameters. Doing this for a grid of combinations yields a surface, yielding a visual interpretation on the effect of the two hyperparameters.

Experiment 4: Control performance experiment

The goal of this experiment is to compare the two control methods on their environment. The two control methods were compared using the Neural Twin of that environment. For this experiment, 1000 episodes were sampled (6000 hours of control) at random from the data. All episodes were 'played' twice: once with the trained DRL agent, once with the RSSM control method with the best control settings (both were obtained from the previous experiments).

The performance of a control algorithm is calculated as

$$performance = \frac{1}{N}\sum_{n=0}^{N}\sum_{t=0}^{T} r_t^{(N)}, \tag{47}$$

where N is the number of episodes, T is the number of timesteps in an episode and $r_t^{(N)}$ is the reward at timestep t of episode N. In words, the performance was calculated as the average over all episodes of the sum of the rewards for an episode. This basically yields a negative version of the total cost of an episode as described in Equation 45, which was used for RSSM. For DRL, this corresponds to the undiscounted return (undiscounted version of Equation 25) of an episode. As both control algorithms optimized for this objective, they can be compared using this performance measure.

The average performance of the 1000 episodes of both control algorithms is reported for every environment.

Furthermore the controllers are compared to the conventional controllers. For both the DRL agent and the MPC controller, three comparisons are made with respect to the original controller (the results will be averaged over 1000 episodes, i.e. 6000 hours):

1. Comparison in tracking of setpoint, i.e. what is the average deviation from the setpoint. For one episode this boils down to $\frac{1}{T}\sum_{t=0}^{T}|SP_t - y_t|$.

2. Comparison in energy consumption. For this comparison an estimate will be used based on the integral of the valve over time. For one episode the calculation will be $\frac{1}{T}\sum_{t=0}^{T}u_t$, where u_t denotes in % the control value at time t. On itself it does not say much about absolute energy usage, but in comparison between controllers the difference between the control values can roughly denote the change in energy usage.

3. Comparison in stability, i.e. how 'smooth' are the control sequences? For one episode the calculation will be $\frac{1}{T}\sum_{t=1}^{T}|u_t - u_{t-1}|$.

In addition to this, the multiple spectra will be compared to each other in order to inspect both the validity of the Neural Twin as a simulator as well as the performance of the different controllers.

Experiment 5: Real world experiment

The goal of this experiment is to inspect the gap between simulation and the real world. Both control methods were implemented in the real world for one environment: the sports hall of the Duurswoldhal in Slochteren. Both methods were tested for a week on the real system, and their performance is compared. Although the two weeks were subsequent, the comparison might still be skewed as it is not possible to control for weather conditions. However, this experiment does yield valuable insights about the gap from simulation to the real world for both control methods separately.

4.4 Implementation details

Data was retrieved using the Climatics API, provided by Advanced Climate Systems (ACS). This returned logs for every sensor with a 2 minute interval. Weather data was retrieved using the DarkSky[2] API.

The algorithms were implemented using Python and the open source packages PyTorch [87] and OpenAI Gym [88]. A custom gym environment was created from the Neural Twin trained with PyTorch. The experiments were run on a computer with an NVIDIA GeForce GTX 1060 6GB video card. With this equipment, training a single Neural Twin took about 30 minutes and training a DRL agent took approximately 75 minutes.

ONNX was used to transfer Neural Networks to an ARM64v8 environment which was required in order to connect the networks with the control system of the buildings.

[2]`https://darksky.net/forecast/40.7127,-74.0059/si12/en`

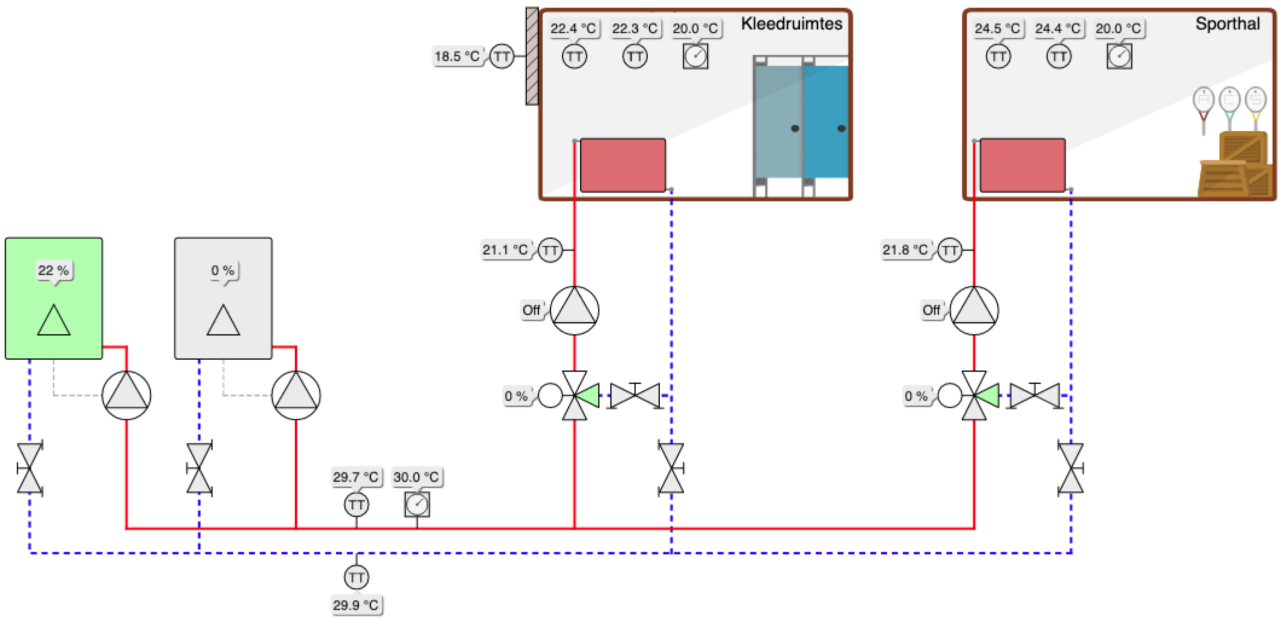


Figure 16: Schematic of the Gymzaal Jan building in Hoogezand. It consists of two rooms, which are both modelled. The structure is fairly similar to that of the Duurswoldhal, presented in Figure 11. In this schematic, the left room is the changing room and the room on the right is the sports hall.

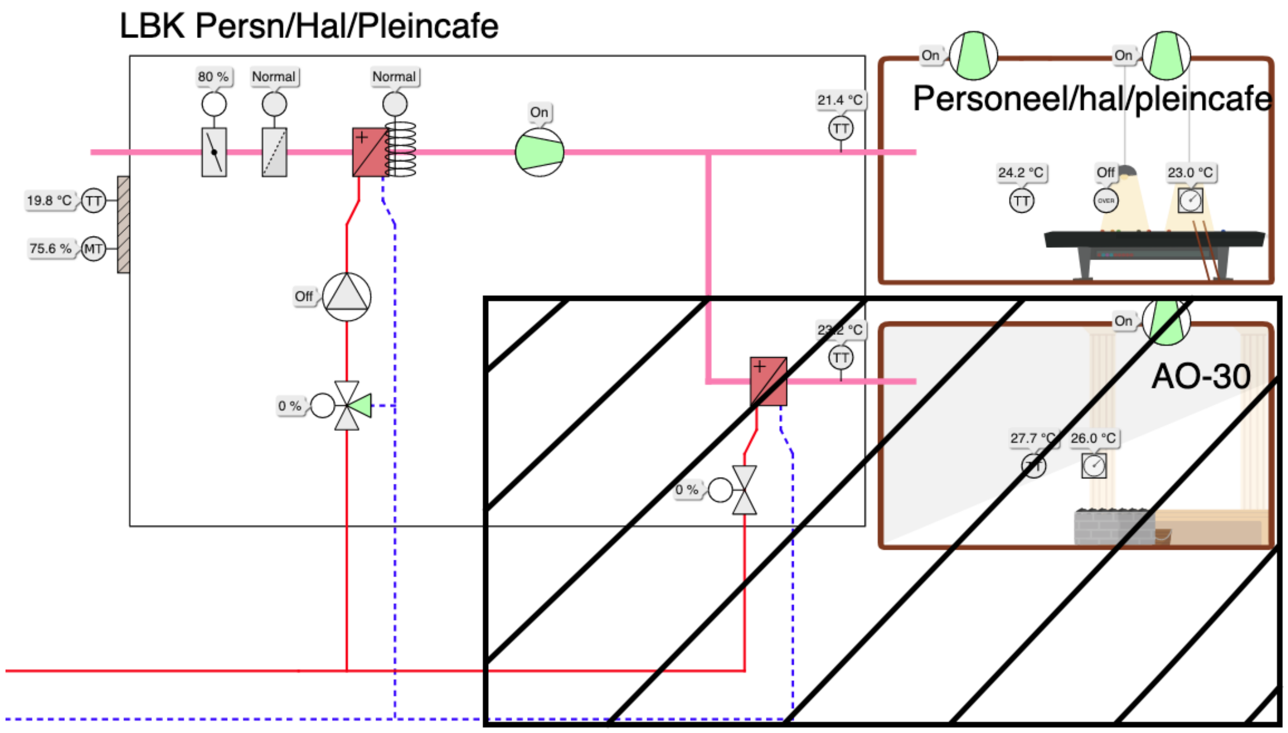
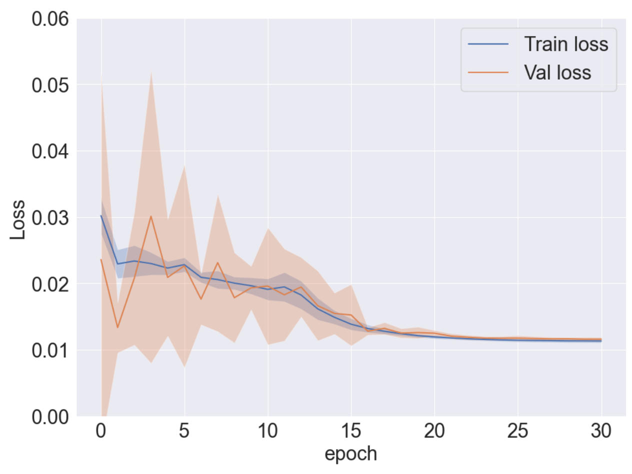


Figure 17: Schematic for the central hall in Kalkwijck. It shows the AHU and the room, as well as a crossed out room (bottom right) which is left out of consideration for this environment. The AHU unit combines outside air with a heater which is heated using the supply temperature coming from boilers. Then this heated air will be blown into the room. Two ventilators on top of the room can be used to extract air from the room. The water supply temperature and flow sensors are not present in this figure, but are used when modeling the environment.

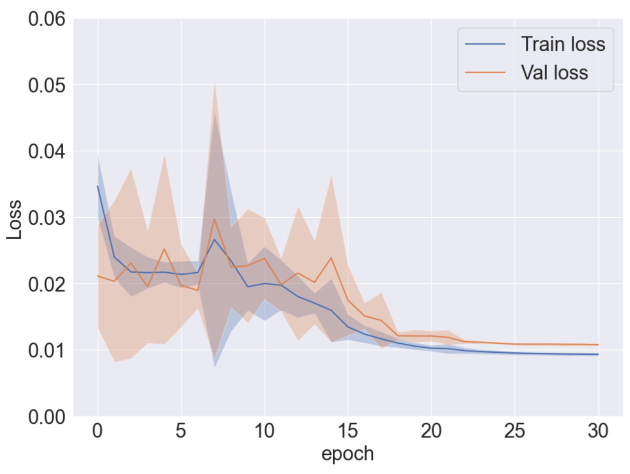
5 Results

5.1 Experiment 1: Neural Twins

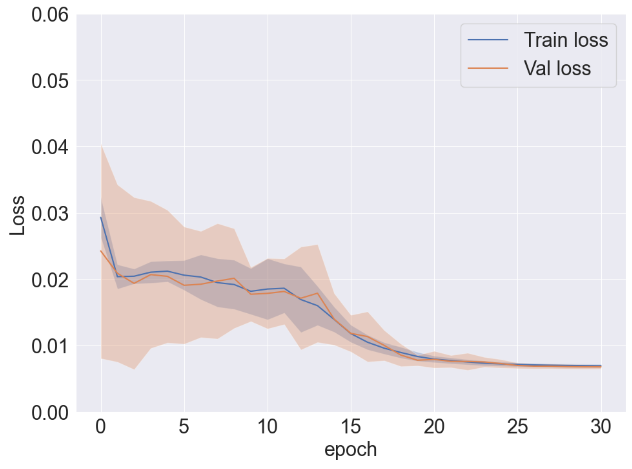
Figure 18 show the results for experiment 1 for all four environments. It shows the mean loss and the deviation of the loss over 10 trials for every environment. Table 6 shows the loss on the test set (not shown in the figure) when compared to the validation loss. It can be observed that all Neural Twins converge to a stable point after 30 epochs. In the table it is observed that the test loss is approximately equal to the validation loss, and in the plots it is observed that the validation loss is approximately equal to the training loss, indicating that the Neural Twins did not overfit during training.



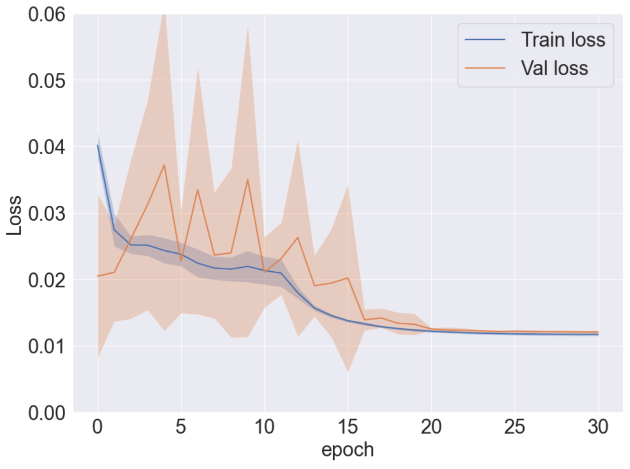
(a) Sports hall, Duurswoldhal.



(b) Sports hall, Hoogezand.



(c) Changing room, Hoogezand.



(d) Central hall, Kalkwijck.

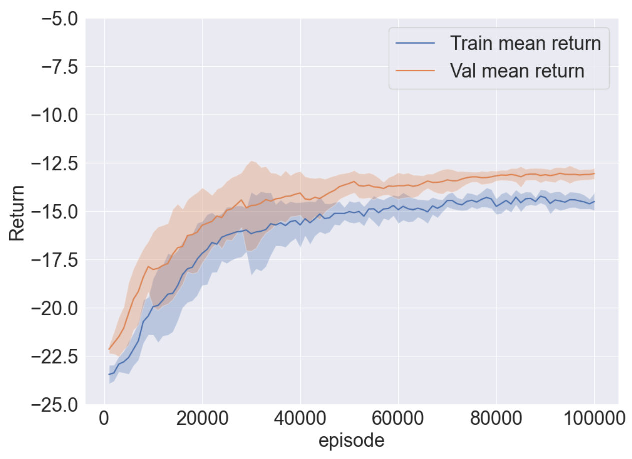
Figure 18: Results of the experiments for training Neural Twins for the four environments. Each plot is constructed based on 10 trials. The solid line represents the mean and the shaded area represents the standard deviation.

Environment	**Validation loss**	**Test loss**	**Test error in °C**
Sports hall, Duurswoldhal	0.0116 ± 0.0003	0.0115 ± 0.0003	0.24 ± 0.21
Sports hall, Hoogezand	0.011 ± 0.0002	0.009 ± 0.0003	0.23 ± 0.32
Changing room, Hoogezand	0.0067 ± 0.0003	0.007 ± 0.0003	0.16 ± 0.13
Central hall, Kalkwijck	0.01205 ± 0.0002	0.0118 ± 0.0003	0.20 ± 0.16

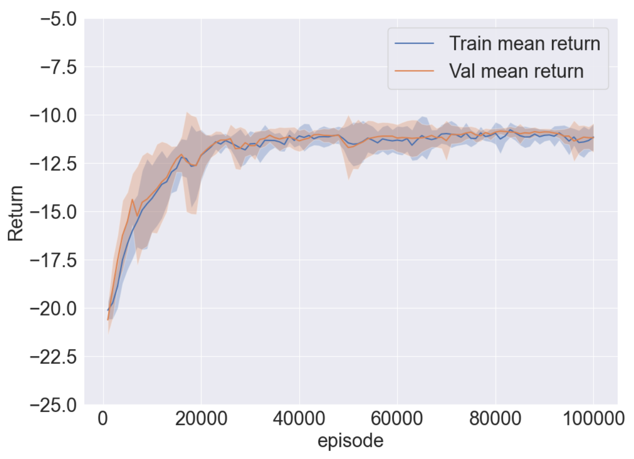
Table 6: Comparison of performance of the Neural Twin on the test set relative to the validation set for all environments.

5.2 Experiment 2: DRL agents

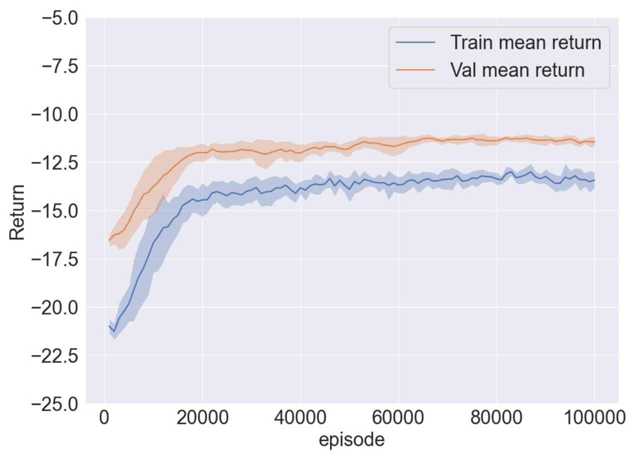
Figure 19 show the results of experiment 2, where an DRL agent is trained on a Neural Twin of an environment. The plots show the average and standard deviation of the return over time. It should be noted that the return obtained from episodes on the validation set are in general higher compared to the return obtained episodes on the training set. This can be explained by the fact that the returns from the validation set do not suffer from exploration, as is the case with the returns during training. Exploration causes the agent to take random actions based on their probability, hence sometimes taking bad actions. It is observed that all agents converge within 100000 episodes. Furthermore it can be observed that all agents converge at a different magnitude in terms of returns. For example, the agent for the changing room in Hoogezand converges with a reward close to 10, whereas the agent for the sports hall in the Duurswoldhal converges with a reward close to 12.



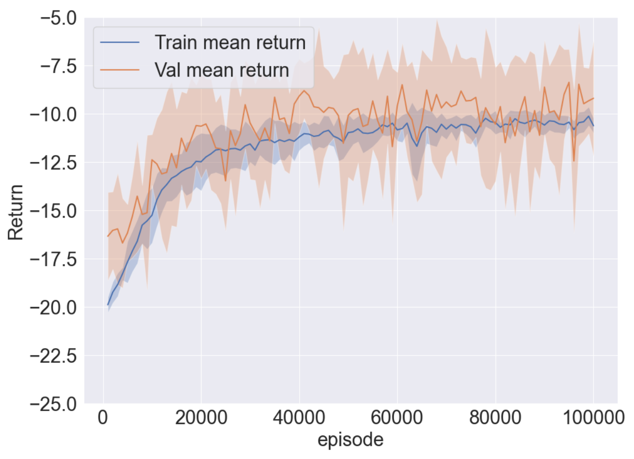
(a) Sports hall, Duurswoldhal.



(b) Sports hall, Hoogezand.



(c) Changing room, Hoogezand.

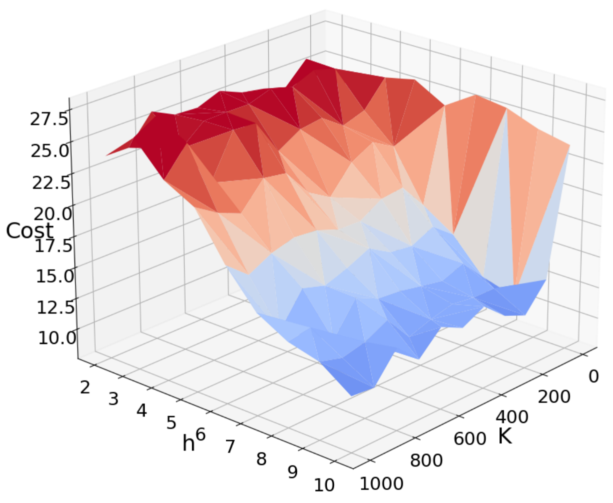


(d) Central hall, Kalkwijck.

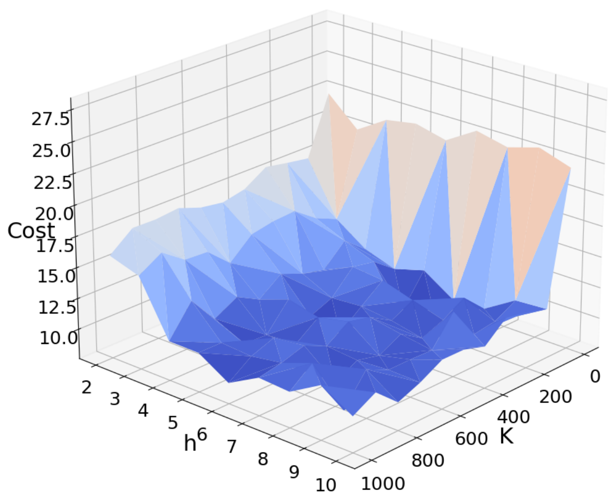
Figure 19: Results of the training of DRL agents for all environments. Every plot shows the average of the return of the 10 trials, as well as the standard deviation.

5.3 Experiment 3: MPC

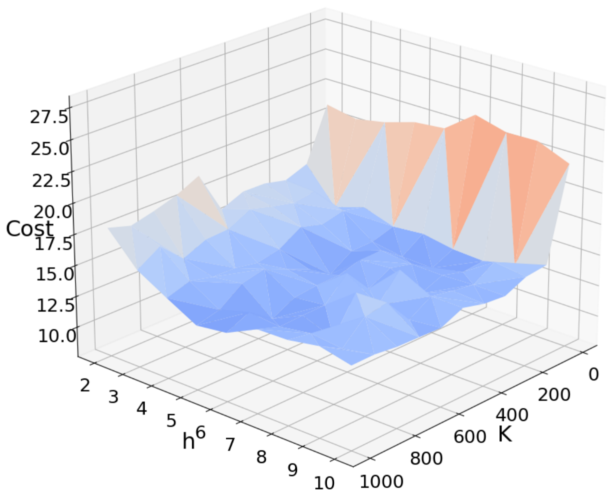
Figure 20 show the effects of the hyperparameters (h, the horizon, and K, the number of random control sequences generated every timestep) on the performance of MPC when using the random sampling shooting method. A large set of combinations of hyperparameters are all evaluated on 200 episodes, i.e. 1200 hours of control. The average episode cost (which is the same as the absolute value of the return of an episode) is calculated and plotted. Hence, the result is a surface where low values of the surface indicate relatively good combinations of parameters. As observed, in the figure, all four environments yield different surfaces, and different combinations are required for all environments. However, in general a too small horizon h leads to significant increase in the cost. The same goes for the number of sequences K generated every timestep: if this number is too small, the performance is relatively worse compared to larger values of K.



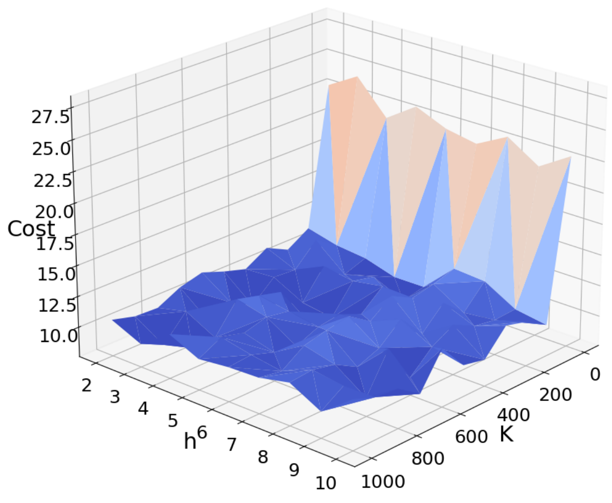
(a) Sports hall, Duurswoldhal.



(b) Sports hall, Hoogezand.



(c) Changing room, Hoogezand.



(d) Central hall, Kalkwijck.

Figure 20: Results for the MPC experiment. The plots show the cost of the combinations of the two hyperparameters relevant to MPC using the random sampling shooting method: the horizon h and the number of random control sequences generated every timestep, K. The plots show the average cost of control according to Equation 44, averaged over 1200 of hours of control, simulated using a Neural Twin of the process.

5.4 Experiment 4: Comparison of Control

Table 7 shows the results when comparing the two methods of control. Based on Experiment 3 and Figure 20, the following hyperparameters were used for MPC:

- Sports hall, Duurswoldhal: $h = 10$, $K = 1000$
- Sports hall, Hoogezand: $h = 6$, $K = 700$

- Changing room, Hoogezand: $h = 5$, $K = 1000$
- Central hall, Kalkwijck: $h = 3$, $K = 500$

Figure 21 shows an example of an episode of control using both methods. The upper figure shows the control using the DRL agent whereas the bottom figure shows the control using the MPC algorithm.

Environment	DRL control return	MPC return
Sports hall, Duurswoldhal	-14.65	**-13.89**
Sports hall, Hoogezand	**-9.99**	-10.76
Changing room, Hoogezand	**-12.38**	-13.92
Central hall, Kalkwijck	**-10.09**	-11.17

Table 7: Comparison of control between the DRL agent and MPC using the random sampling shooting method. 1000 episodes (6000 hours) of control were simulated using the corresponding Neural Twin, and the mean return of these episodes are shown in this table.

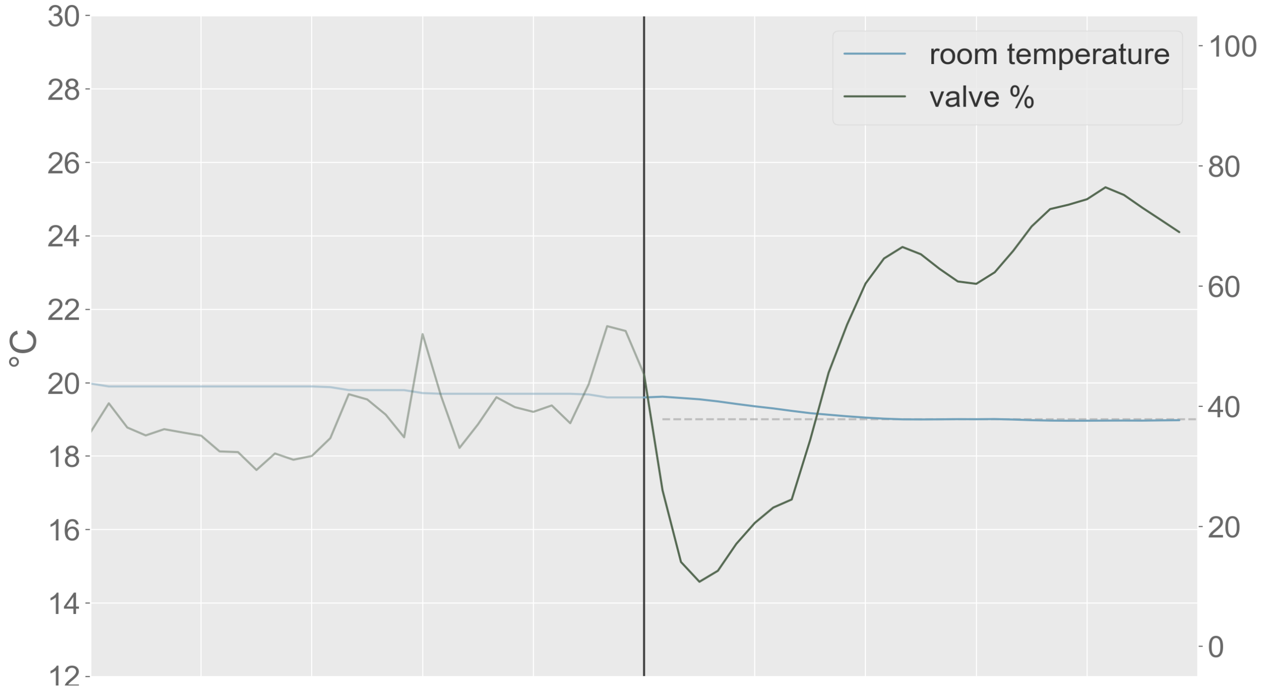
Table 8 and Table 9 compare the DRL agent and the MPC controller (respectively) with the conventional controller. The results are generated by comparing data from 1000 episodes with the original data (which represents the conventional controller). For the DRL controller Table 8 shows that for all environments the DRL controller was able to improve on energy performance, while remaining thermal comfort (even improving it slightly). For some environments it increases the roughness of control by a small amount ($< 2\%$).

Environment	Setpoint deviation	Energy usage	Roughness of control
Sports hall, Duurswoldhal	-2.8%	-15.4%	-2.4%
Sports hall, Hoogezand	-2.7%	-7.46%	-1.9%
Changing room, Hoogezand	-3.0%	-5.3%	+1.5%
Central hall, Kalkwijck	-3.7%	-46.9%	+1.7%

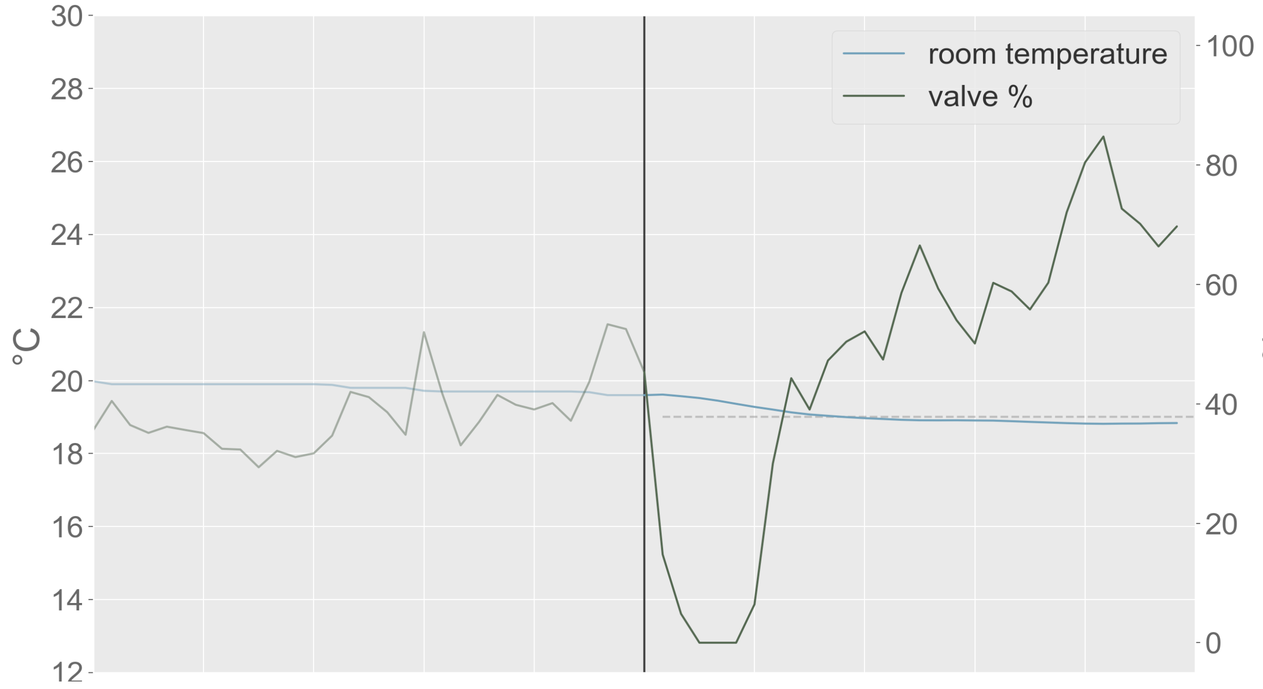
Table 8: Comparison of control between the DRL agent and the conventional controller for all environments. Values denote the change in % compared to the conventional controller, i.e. based on the historical data.

Environment	Setpoint deviation	Energy usage	Roughness of control
Sports hall, Duurswoldhal	-3.3%	-17.5%	-2.5%
Sports hall, Hoogezand	-3.2%	-5.7%	-2.4%
Changing room, Hoogezand	-4.8%	+25.4%	-1.0%
Central hall, Kalkwijck	-6.1%	+37.2%	+2.3%

Table 9: Comparison of control between the MPC controller and the conventional controller for all environments. Values denote the change in % compared to the conventional controller, i.e. based on the historical data.



(a) Example of the performance of the DRL agent during an episode in the sports hall, Duurswoldhal.



(b) Example of the performance of the MPC algorithm during an episode in the sports hall, Duurswoldhal.

Figure 21: Comparison of the performance of both control algorithm when both start in the same situation. The upper figure shows the performance of the DRL agent, the bottom figure shows the performance of the MPC algorithm. Values on the left of the black vertical line are historical values, values on the right represent simulated values.

Figure 22 shows four different spectrums for the sports hall, Duurswoldhal. All spectrums are normalized in the sense that the area under the spectrum sums to 1. The spectrums describe the following four scenarios:

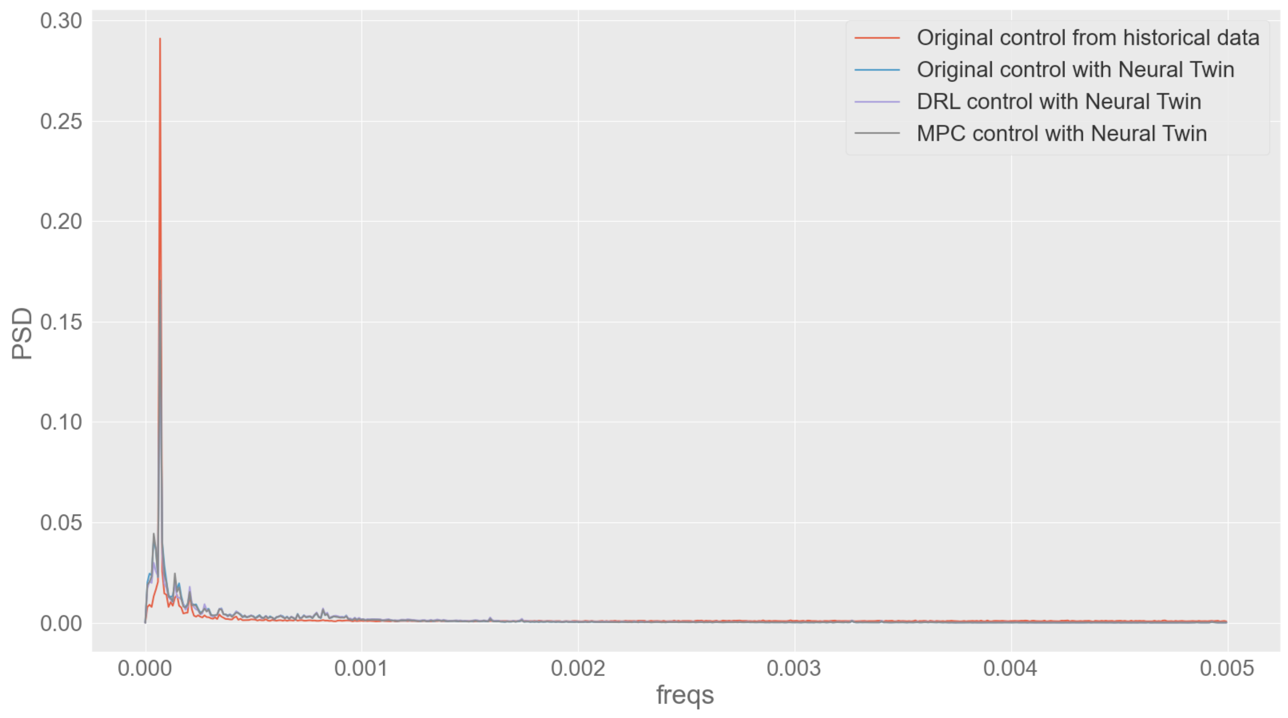
1. The spectrum of the historical data (hence with the classic controller). This spectrum shows the periodicity and the characteristics of the original controller.
2. The spectrum of data simulated using the Neural Twin using historical control values. The expectation is that this spectrum matches the previous one as closely as possible.
3. The spectrum of data simulated using the Neural Twin with the DRL controller.
4. The spectrum of data simulated using the Neural Twin with the MPC controller.

These spectrums show the characteristics of the different controllers, and the comparison with real world data. From the spectrums it is observed that the historical data shows one very clear peak, and this peak is captured when comparing it to the spectrum of the Neural Twin with the historical control values. In general, the two spectrums look very much alike. This shows that the Neural Twin is able to extract the periodicity and the characteristics of the process. Furthermore it is observed that the other scenarios all have the same peak, but this peak is less intense compared to the spectrum of the historical data. This means that the periodicity is redistributed somewhere else. When looking at a zoomed in version of the spectrum, it can be observed that the other spectrums show a small peak before the large peak. Furthermore they show some more noise in the higher frequencies, which confirms the redistribution.

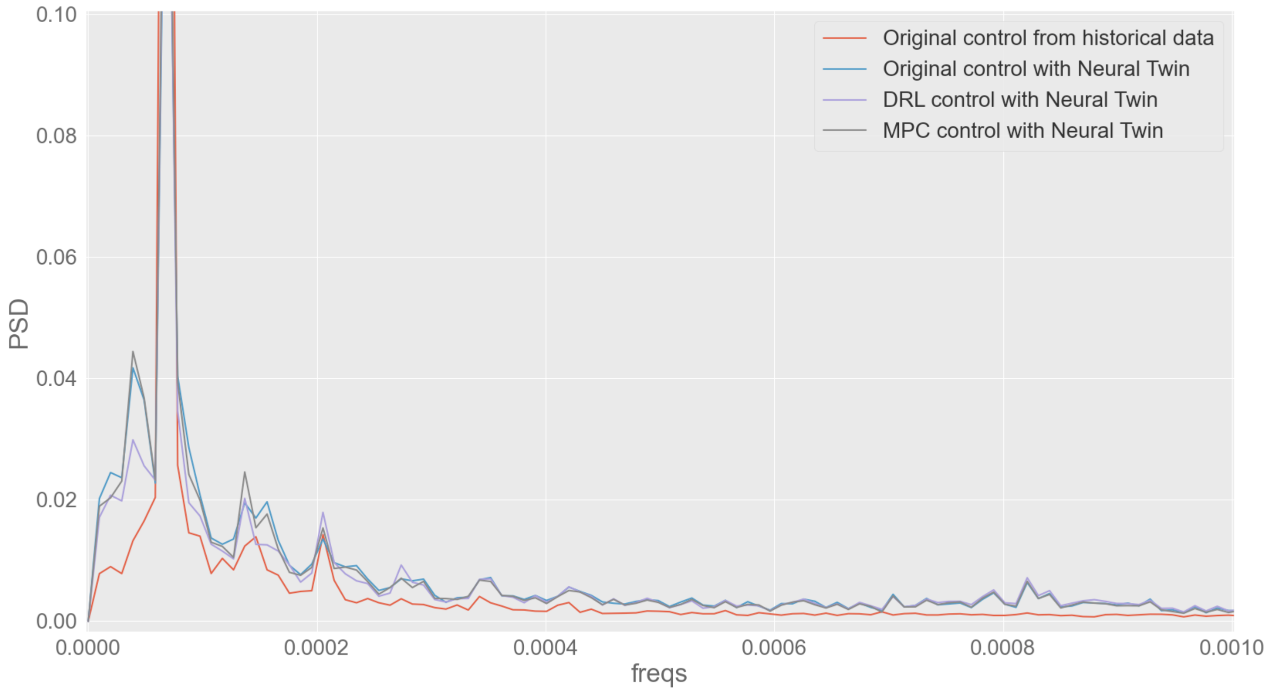
5.5 Experiment 5: Real World Control

Figure 23 shows the performance of the controllers on the physical environment, the sports hall in the Duurswoldhal. It is important to note that the figures cannot be compared one on one as opposed to earlier (Figure 21), as now the environmental variables differed (the control happened on different days). However, it does show that both controllers were able to reach and maintain the setpoint of during the day. The deviation of control was around 0.2 degrees Celsius, a difference which is barely noticeable in the sports hall.

Based on the figure, it would be safe to say that the control of the DRL agent is smoother than the control using MPC. The MPC control values are more spiked and inconsistent (looking at and comparing the first few hours of both days). Especially the rate of change of the valve when the room temperature approaches the setpoint shows that the DRL controller slowly closes the valve, whereas the MPC controller very quickly closes the valve.

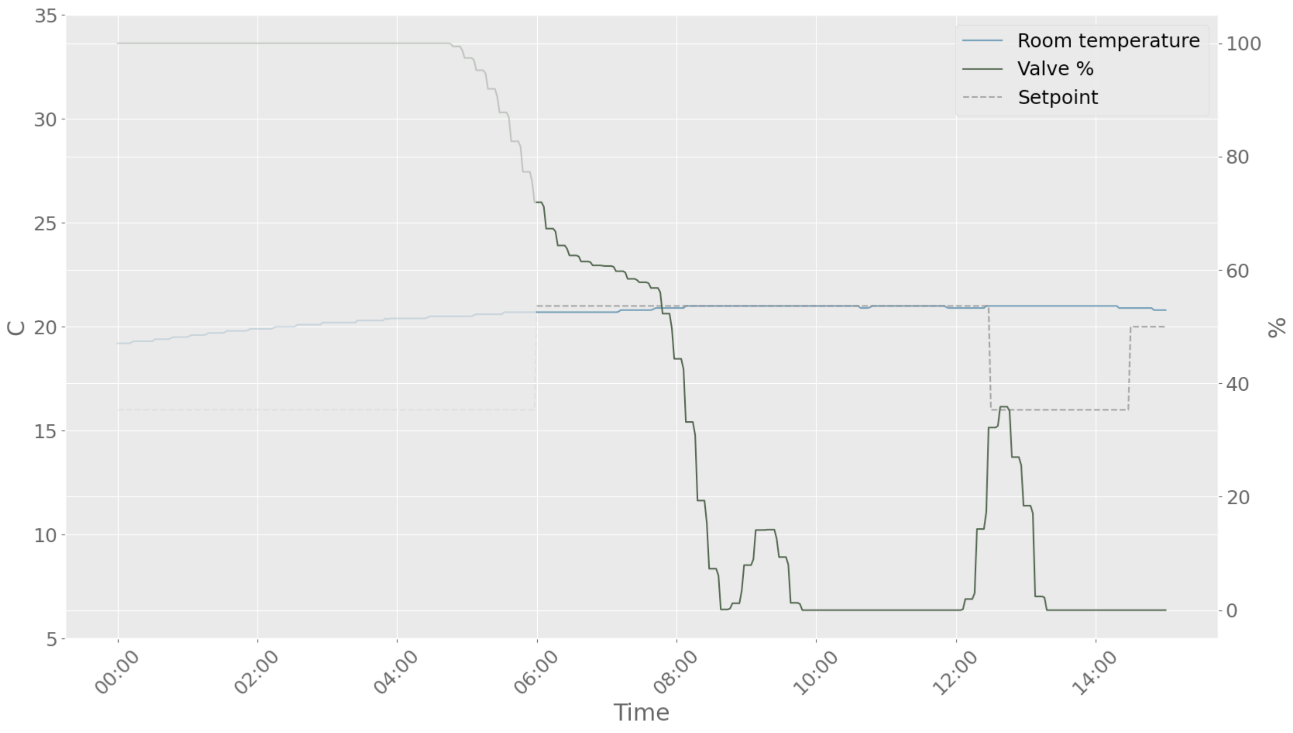


(a) Spectral analysis of the sports hall of the Duurswoldhal, for different scenarios.

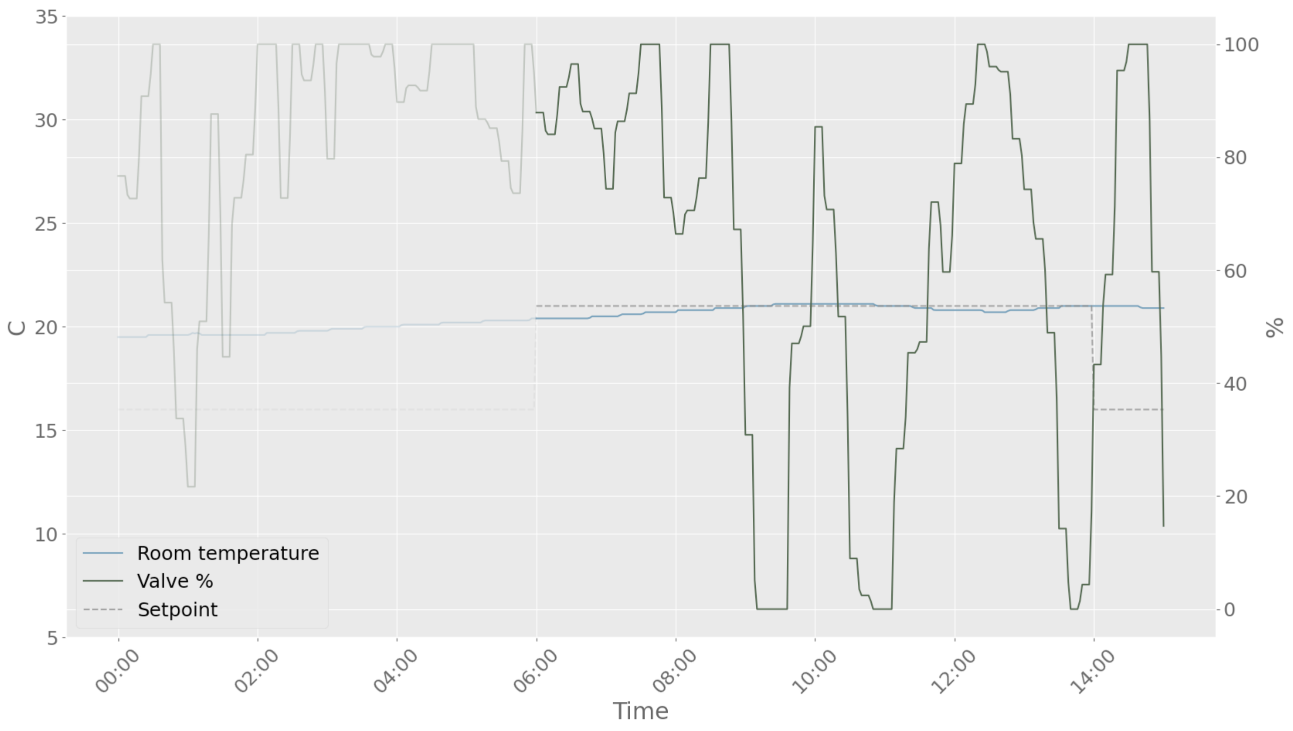


(b) Spectral analysis of the sports hall of the Duurswoldhal, for different scenarios, zoomed in.

Figure 22: Spectral analysis of 4 different scenarios of the sports hall, Duurswoldhal. It shows the spectrum for the historical data, simulated data with the Neural Twin using the original controller, simulated data with the Neural Twin using the DRL controller and simulated with the Neural Twin using the MPC controller.



(a) Performance of the DRL agent on the physical room on October 14th 2021.



(b) Performance of MPC on the physical room on October 22th 2021.

Figure 23: This figure shows the performance of the two controllers on the physical environment. These figures are *not* meant as a comparison between each other, as it were different days with different circumstances and environmental variables. The figure *does* show that the controllers learned using the Neural Twin transfer to real world control quite well.

6 Discussion

6.1 Neural Twin training

The results show that the Neural Twin is able to act as a simulator and it can be learned from data in a stable manner. On average the prediction error is 0.2 °C. This is acceptable as many of the controllers and sensors in the HVAC sector do not operate in a more precise manner. The plots in Figure 18 show that the validation loss converges to approximately the train loss, and stabilizes across multiple runs. At the start the deviation of losses across runs is fairly high and the loss does not decrease much. This can be explained by the chance for teacher forcing which degrades slowly over the first 10 epochs. After this a more steep decrease in loss is observed for all plots. Furthermore the test loss is comparable to the validation loss for all Neural Twins, as observed in Table 6. This shows that they are all generalizable to unseen data.

It is interesting to note that the Neural Twins do not all converge to the same loss. Although all achieve a low enough loss to be usable as simulator, it is dependent on the process how precise the Neural Twin is. This also means that for harder processes, the Neural Twin will be less reliable compared to relatively simple processes. When the Neural Twin is used in an end-to-end setting, it is advised to use some form of validation before using the Neural Twin as the basis for one of the controllers. It might be interesting to find out what the minimal required loss is before a Neural Twin is usable (this might be dependent on the type of process as well).

Furthermore, when inspecting two of the spectrums shown in Figure 22a representing the historical data and the Neural Twin using the historical control values, it can be concluded that the Neural Twin captures the periodicity and the characteristics of the original data. This ensures that the Neural serves as a valid simulator for the process. It should be noted that the spectrum has not a perfect overlap, and the peak in the spectrum of the Neural Twin is lower than the same peak in the spectrum of the historical data. This means that the Neural Twin emphasizes another periodicity or noise somewhere else in the spectrum.

6.2 DRL agent training

From Figure 19 it is shown that the Neural Twin can act as a simulator which an DRL agent can learn patterns from. For each of the plots it can be seen that the agent improves itself compared to its initial performance, and each of the agents converge. For almost all the plots the reward of the validation episodes is higher than the reward of the train episodes. As described earlier, this can be explained by the fact that during validation the agent is not exploring and is choosing its actions deterministically. This logically leads to better performance compared to the reward of the training episodes, as here the agent chooses actions based on chance, and hence might pick a bad action for the sake of exploration.

The deviation in reward across runs is minimal for all plots except for the Central hall in Kalkwijck. One distinguishing factor of this environment is that it requires two control values (as opposed to one control value as for the rest of the environments). Furthermore, the control system is based on an air handling unit, which is known to be able to control at a much faster rate than a radiator (this is also confirmed by Figure 20d, where it can be observed that the horizon has minimal influence on the cost). These two facts combined might make the convergence less stable for this environment. This also raises the question whether bigger problems arise when one tries to learn a control policy for more complex control systems with more control values.

Also here it should be noted that different environments converge to different rewards. The sports hall in Duurswoldhal converges to a reward of approximately -12.6, whereas the sports hall in Hoogezand converges to a reward of approximately -11. This again highlights the fact that every process is different, and its characteristics determine how well the agent is able to do. It might be the case that the sports hall in the Duurswoldhal is simply harder to regulate due to its geographic orientation, in a way such that it is highly influenced by weather conditions. Furthermore the room might be insulated poorly in comparison to the other environments, causing the reward to be worse. When looking at control episodes from the Duurswoldhal agent, such as shown in Figure 21a, it can be seen that the agent has managed to control the process, however its reward is simply lower due to the environmental constraints.

6.3 MPC hyperparameters and room characteristics

Although the intention of Figure 20 was to inspect the influence of hyperparameters on the performance of the MPC controller, it also reveals interesting properties of the different environments.

In general it can be observed that the cost decreases when the horizon increases (at least for the first few steps). This makes logical sense, as most processes are fairly slow; one cannot heat a room within 10 minutes using a radiator, and more importantly, one cannot stop the increasing/decreasing temperature in a room within 10 minutes. Hence being able to look ahead is important in order to reduce overshoot/undershoot. One can also imagine that this only applies to a certain length. Being able to look ahead one day might not have that much of an influence on what the controller should do now (for some very slow processes it might though). Hence, what can be seen in the plots gives away something about the nature of the process. For example, the controller for the sports hall in the Duurswoldhal still benefits from increasing the horizon from 7 to 8. However, the controller for the sports hall in Hoogezand does not benefit from the same horizon. This tells us that sports hall in Hoogezand has a shorter regulation interval, and the regulation happens in faster cycles compared to the Duurswoldhal. This can be caused by physical constraints such as insulation, volume of the room and/or capacity of the controller.

Interesting to note is that the Central hall in Kalkwijck does not seem to benefit at all from an increase in horizon at any point. As discussed before, this hall is regulated by an air handling unit, which is able to impact the room air temperature in 10 minutes. This means that at any point during control it can 'switch course' from heating to cooling or the other way around. This means that the controller does not benefit from a large horizon as is the case for the other environments which all heat a room using a radiator.

When looking at the K parameter, it can be observed that after the initial increase to 100 sequences, this parameter does not have much influence on the cost. It is expected that at least some sequences are needed to generate sample space large enough such that one good action is in it, but from the results it can be seen that there is not much difference in performance between 200 and 1000 control sequences.

Also here it can be observed that all processes have their lowest cost at a different level. Again, the sports hall in Hoogezand and the Central hall in Kalkwijck have lower cost compared to the other two environments.

As shortly discussed earlier, the plots also show something about the characteristics of the environments. The plots suggest that the Central hall in Kalkwijck is a process in which the controller has a short regulation interval, i.e. the controller has a direct influence on the process value. The controller in the Sports hall in Duurswoldhal however does have a large regulation interval. More concrete, the steepness of the surface in the h direction says something about the interval at which a controller has influence on the process.

6.4 Comparison of control

In Table 7, the return of both controllers are compared. From this it can be observed that except for the Sports hall in Duurswoldhal, the DRL agent outperforms the MPC controller. When comparing the controllers to the classic controllers of the environments, the same results arise. The first thing to note is that both controllers retain (even slightly improve) the deviation from the setpoint, i.e. the thermal comfort in the environments. The same holds for the roughness of the control, which stays approximately the same compared to the classic controllers. This is important because this represents the physical constraints of the controller. If the roughness would increase this could mean that the lifespan of a controller might decrease significantly.

When looking at the energy usage of the controllers, it can be seen that the DRL agent improves the energy usage for all environments. Only for the Sports hall in Duurswoldhal the decrease in energy usage is less compared to the MPC controller. The MPC controller did not decrease the energy usage for two of the environments, and even increased them by a significant amount.

Figure 22 shows a comparison of the spectrums for the different controllers. The general spectrum still looks the same as the original controller, but it can be seen that the difference lies in the magnitude of the peak. As the controllers improve upon the classic controller (whose spectrum is shown with the historical data), it can be argued that the difference observed in the zoomed in version of the spectrums in Figure 22b explains the improvement, where the shifted frequency distribution of the controllers represent the change in performance.

Furthermore, Figure 21 shows a pattern that was observed by the author many times, where the DRL agent performed 'smoother' control compared to the MPC controller. This suggests that the agent was able to learn an at least locally smooth function (although it can be questioned what happens in unlikely situations with such a network). The control of the MPC controller is less smooth, and especially for low K this will remain the case, because the control sequences from which control values are selected are generated randomly. When K approaches infinity the control will appear more smooth, but this will be computationally infeasible.

From the results it can be concluded that the DRL agent in general outperforms the MPC controller.

6.5 Real-world performance

In the current setup of the physical experiment it is not reliable to say anything about the comparison of energy usage. This is because due to time restrictions only a small sample of control could be taken for both controllers. Because every day is different, this sample will cannot be compared to the same time but one year earlier, and the time in which the DRL agent was controlling might have had very different environmental circumstances compared to the time in which MPC was controlling the process. Therefore, from the results obtained in this study only conclusions will be made about the transfer from simulated control to real world control.

When looking at the performance of both the controllers on the physical room, it can be noted that both controllers are able to control the physical room. From this it can be concluded that the sim-to-real gap of the Neural Twin for this environment was small enough in order to learn patterns that transfer to the real world.

For the same reasons as described earlier, it is observed that the DRL controller shows smoother performance compared to MPC. Smoother control is in favor of the valve, due to the physical endurance of the controller. The more rough control is, the shorter the lifetime of the valve before it breaks down. Therefore, the smoother control of the DRL agent is preferred.

When inspecting the results of the physical experiment, an interesting observation was made. Fig-

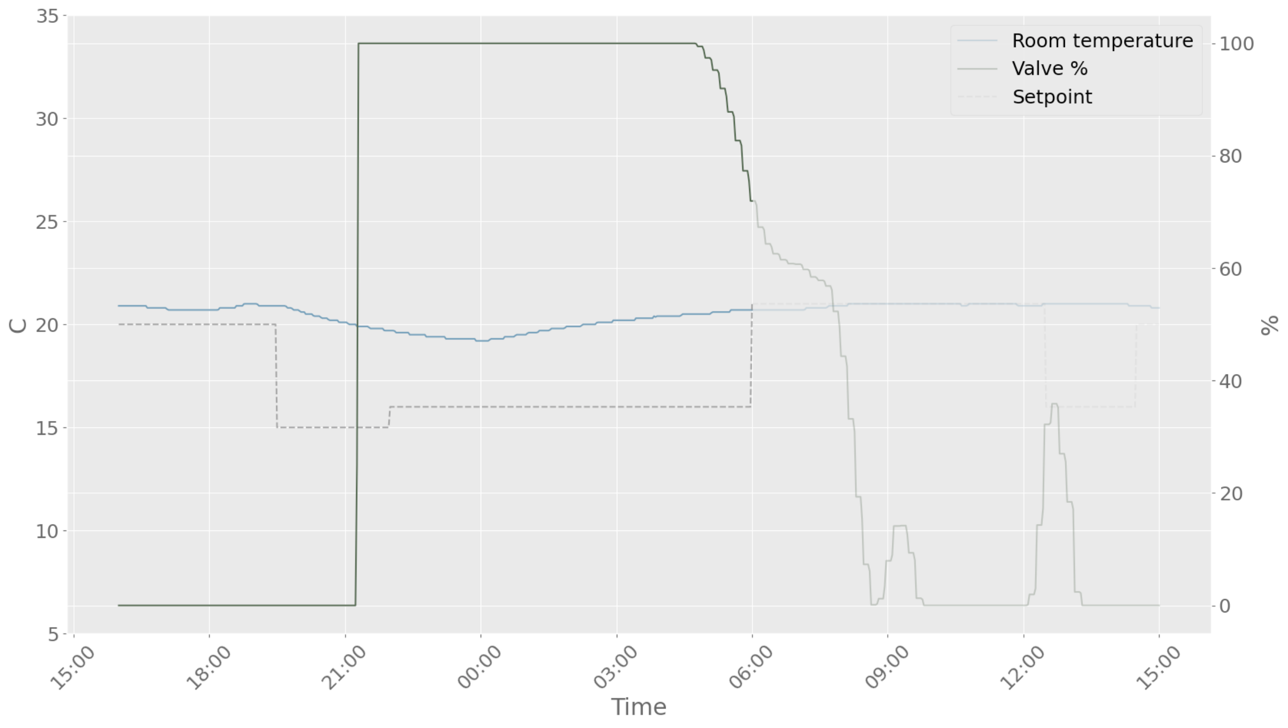


Figure 24: Almost the same plot as Figure 23a, but another part is highlighted and a few extra hours at the left are shown.

ure 24 shows roughly the same time interval as in Figure 23a, but another part is highlighted. From this it is directly observed that the agent is heating the room, but the setpoint officially asks for 16°C. This can be explained by the following: in Climatics (the building's management system used), a user specifies a schedule of desired temperatures. In this case, the user required a temperature of 21°C at 06:00. This concerns a deterministic, planned form of control (cf."open-loop", in the Introduction). However, if the system started heating at 06:00, it would mean the room would be cold for quite some time before the desired temperature is reached. To solve this problem, Climatics provides an 'startup' option, which calculates how early the heating process must begin such that the desired temperature is reached in time. As a result, the setpoint jump is sent to the agent earlier. In the figure, at 00:00 the agent already got a setpoint of 21°C, not 16°C, hence the agent reacts logically. This leads to visually confusing graphs as the one in Figure 24. More importantly, it may occur that more energy is spilled during nightly prewarming than during daytime.

However, this is a part of the system that can be optimized that was not included in this study. Currently, the building is using energy to heat the room mostly in the night, and much less during the day. The agent directly influences the supply side of the building, because when valves are closed, the supply side does not really have to do much to maintain a steady supply temperature. The influence of all rooms in a building on the building as a whole is another study on its own, not further pursued in this study.

Although the solution works (at 06:00 the room temperature setpoint of 21°C is reached), one might question whether this is the most energy efficient approach. Maybe it would be more beneficial to provide a higher supply temperature (in the figure a supply temperature of approximately 40°C was given) and to heat a room quicker, instead of a lower supply temperature and slower heating of the room. This question should deserve a study on its own.

6.6 Conclusion

This study proposed two new approaches of learning control policies for HVAC systems. Both methods are based on the concept of a Neural Twin. A Neural Twin is an encoder-decoder network that tries to capture the underlying dynamics of a building's process. It is trained in a supervised manner on historical data of a process. When the Neural Twin is trained, it can be used as a simulator of the process.

Although the Neural Twin can be used for many purposes, in this study it was used to develop two different control policies of a process. One approach used Reinforcement Learning to learn a policy regulating the process. The other approach used Model-Predictive Control to simulate at real time the process a number of timesteps ahead using many different sequences of control values and selects the best control values among them.

Both approaches have their own advantages. When using Reinforcement Learning, a policy is represented by a simple neural network. This is convenient during real time control, as a new control value can be generated with a single forward pass. This makes it computationally inexpensive compared to the presented MPC algorithm. However, this approach also is prone to exploiting weak spots of the Neural Twin, which might work on the simulator but do not translate to the real world. An advantage of the MPC algorithm is that it can be used as soon as a Neural Twin is obtained, and requires no further training. However, it is dependent on the hyperparameters K and h, for which different settings might be required for different processes. This can hurt the scalability of this approach. Furthermore this approach becomes exponentially more computationally expensive when the action space (like the number of valves) increases.

The approaches were tested on four different processes. From the results it can be concluded that the Neural Twin framework was able to learn a valid simulator for all processes (Figure 18 and Table 6). This speaks in favor of the generalizability and scalability of the Neural Twin. The same conclusion can be drawn from the results of the agents trained on the Neural Twins (Figure 19). All agents show learning and converge within the given number of episodes. It also shows that not all processes can be treated the same, as it might be the case that one process is simply harder to regulate, causing its average return after convergence to be worse compared to another process, while both might have reached 'optimal' performance.

From Table 7 it can be concluded that the DRL agent performed better compared to the MPC algorithm. Furthermore in Figure 21 it can be seen that the control of the DRL agent is more smooth when compared to the control of the MPC controller. The explanation for this is that the MPC controller generates control values at random, whereas the DRL agent actually tries to learn a function, which is more likely to be locally smooth. However, no guarantee can be given that the agent will actually learn a smooth control function (note that this can be controlled to some extent using the Hamiltonian reward). As for the MPC controller, it is expected that as the number of random sequences generates increases, the control will become more smooth, as MPC also tries to optimize based on stability (Equation 44). Compared to the classic controllers, the DRL agent improved the performance for all environments.

Although the results show that the control of a process can be learned automatically, it was also observed during the test of the physical environment that this of course is not the whole story in optimizing building performance. Next steps should include a study of the interconnection between processes within a building in order to optimize the stability and performance of a building as a whole.

6.7 Summary of Main Contributions

1. The study introduced a scalable and generalizable framework for learning a simulator based on historical data only. The concept is called a Neural Twin.
2. The study presented two methods with which one can automatically (learn to) control a process based on a Neural Twin. This study used both Reinforcement Learning and Model-Predictive Control.
3. The study validated the control methods in a physical environment, showing that the sim-to-real gap was small enough in order to transfer the controllers learned using a simulator to their physical environment.

6.8 Limitations and Future Work

Although the current study presents methods on how to automatically obtain control for a process, it still requires much time before it can be applied to a process. One needs to gather data for this process for a variety of seasons before one can train a reliable Neural Twin. In the meantime the building still needs to be regulated which cannot be achieved using the two methods described in this study (both need a Neural Twin before they can be learned). Hence, one would need to use classic control methods until enough data is available. Therefore, the approaches in this study would mainly be applicable to buildings that already have a control system, which can then be changed to use the methods described in this study.

As argued before, a process on its own has an influence on the process of the building as a whole. For example, when opening and closing its valves, it disturbs the process of the boiler trying to provide a steady supply temperature. How these processes interact with each other, and how these processes as a whole can be optimized together, could yield a very interesting study in the future.

6.8.1 Transfer Learning

In order to use the methods on buildings from the start, it is important to transfer knowledge learned in a previous building to the new buildings. When this is possible, one can start controlling using the previously learned knowledge in a new environment. One could realise this by identifying rooms that behave similarly across different buildings, and training an agent that controls multiple rooms, such that the knowledge that is learned is more about controlling a 'type of room' instead of controlling a 'specific room'.

6.8.2 Safety

Both control methods are based on the Neural Twin networks, which means they inherit bias learned by the Neural Twin. Although the results are promising, it does not tell us anything about how stable the networks will be in unexpected conditions. Furthermore, it can be quite hard to determine what the behaviour of Neural Twin and the DRL agent will be as soon as data is encountered which falls outside the expected (training) distribution. It would be interesting to see the behaviour of the control algorithms under extreme conditions.

Bibliography

[1] N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann, "The national human activity pattern survey (NHAPS): a resource for assessing exposure to environmental pollutants," *Journal of Exposure Science & Environmental Epidemiology*, vol. 11, no. 3, pp. 231–252, 2001.

[2] B. Chenari, J. Dias Carrilho, and M. Gameiro da Silva, "Towards sustainable, energy-efficient and healthy ventilation strategies in buildings: A review," *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 1426–1447, 06 2016.

[3] M. Mayowa, "Indoor air quality and perceived health effects experienced by occupants of an office complex in a typical tertiary institution in Nigeria," *Science Journal of Public Health*, vol. 3, p. 552, 01 2015.

[4] H. Ajimotokan and L. Oloyede, "Influence of indoor environment on health and productivity," *New York Science Journal*, vol. 2, 01 2009.

[5] International Energy Agency (IEA), "Global status report for buildings and construction," tech. rep., International Energy Agency & United Nations Environment Programme, Paris, 2019.

[6] S. Sharma, Y. Xu, A. Verma, and B. K. Panigrahi, "Time-coordinated multi-energy management of smart buildings under uncertainties," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 8, pp. 4788–4798, 2019.

[7] G. Gao, J. Li, and Y. Wen, "Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning," 2019.

[8] S. Brandi, M. S. Piscitelli, M. Martellacci, and A. Capozzoli, "Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings," *Energy and Buildings*, vol. 224, p. 110225, 2020.

[9] R. Jia, M. Jin, K. Sun, T. Hong, and C. Spanos, "Advanced building control via deep reinforcement learning," *Energy Procedia*, vol. 158, pp. 6158–6163, 2019.

[10] D. Azuatalam, W.-L. Lee, F. de Nijs, and A. Liebman, "Reinforcement learning for whole-building HVAC control and demand response," *Energy and AI*, vol. 2, p. 100020, 2020.

[11] Y. Kim, "A supervised-learning-based strategy for optimal demand response of an HVAC system," 2019.

[12] P. L. Simona, P. Spiru, and I. V. Ion, "Increasing the energy efficiency of buildings by thermal insulation," *Energy Procedia*, vol. 128, pp. 393–399, 2017. International Scientific Conference "Environmental and Climate Technologies", CONECT 2017, 10-12 May 2017, Riga, Latvia.

[13] B. Drury, "The control techniques drives and controls handbook / bill drury," *SERBIULA (sistema Librum 2.0)*, 01 2009.

[14] ASHRAE, "Standard 135-2016– BACnet–a data communication protocol for building automation and control networks (ANSI approved)," 2016.

[15] D. B. Crawley, C. O. Pedersen, L. K. Lawrie, and F. C. Winkelmann, "EnergyPlus: Energy simulation program," *ASHRAE Journal*, vol. 42, pp. 49–56, 2000.

[16] S. e. a. Klein, "TRNSYS 18: A transient system simulation program, solar energy laboratory." `http://sel.me.wisc.edu/trnsys`, 2017.

[17] WUFI Pro 6.2, Fraunhofer Institute for Building Physics. `www.wufi.de`.

[18] World Health Organization. Environmental Health in Rural and Urban Development and Housing Unit, "Indoor environment : health aspects of air quality, thermal environment, light and noise," 1990.

[19] N. Minorsky., "Directional stability of automatically steered bodies," *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922.

[20] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, pp. 220–222, jun 1993.

[21] P. Woolf, "PID tuning via classical methods." `https://eng.libretexts.org/@go/page/22413`, 2021.

[22] M. T. Coughran, "Lambda tuning - the universal method for PID controllers in process control." `https://www.controlglobal.com/whitepapers/2013/lambda-tuning-the-universal-method-for-pid-controllers-in-process-control/`, 2013.

[23] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[24] A. Nagabandi, G. Yang, T. Asmar, R. Pandya, G. Kahn, S. Levine, and R. S. Fearing, "Learning image-conditioned dynamics models for control of underactuated legged millirobots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4606–4613, IEEE, 2018.

[25] G. Ramos Ruiz, E. Lucas Segarra, and C. Fernández Bandera, "Model predictive control optimization via genetic algorithm using a detailed building energy model," *Energies*, vol. 12, no. 1, p. 34, 2019.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[27] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15, Springer, 1987.

[28] D. Teodorovic, P. Lucic, G. Markovic, and M. Dell'Orco, "Bee colony optimization: principles and applications," in *2006 8th Seminar on Neural Network Applications in Electrical Engineering*, pp. 151–156, IEEE, 2006.

[29] ASHRAE, "Standard 55-2020 thermal environmental conditions for human occupancy," 2020.

[30] P. Fanger, *Thermal Comfort: Analysis and Applications in Environmental Engineering*. Danish Technical Press, 1970.

[31] S. Schiavon and K. H. Lee, “Dynamic predictive clothing insulation models based on outdoor air and indoor operative temperatures,” *Building and Environment*, vol. 59, pp. 250–260, 2013.

[32] J. Sjöberg, H. Hjalmarsson, and L. Ljung, “Neural networks in system identification,” *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 359–382, 1994. IFAC Symposium on System Identification (SYSID’94), Copenhagen, Denmark, 4-6 July.

[33] M. Nechyba and Y. Xu, “Neural network approach to control system identification with variable activation functions,” in *Proceedings of IEEE Int. Symp. on Intelligent Control*, pp. 358–363, August 1994.

[34] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, “Nonlinear systems identification using deep dynamic neural networks,” 2016.

[35] D. Masti and A. Bemporad, “Learning nonlinear state-space models using deep autoencoders,” in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 3862–3867, IEEE, 2018.

[36] Y. Wang, “A new concept using LSTM neural networks for dynamic system identification,” in *2017 American Control Conference (ACC)*, pp. 5324–5329, IEEE, 2017.

[37] J. Gonzalez and W. Yu, “Non-linear system modeling using LSTM neural networks,” *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 485–489, 2018.

[38] M. Forgione and D. Piga, “Continuous-time system identification with neural networks: Model structures and fitting criteria,” *European Journal of Control*, vol. 59, pp. 69–81, 2021.

[39] R. Kumar, S. Srivastava, J. Gupta, and A. Mohindru, “Comparative study of neural networks for dynamic nonlinear systems identification,” *Soft Computing*, vol. 23, no. 1, pp. 101–114, 2019.

[40] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[42] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[43] S. Hochreiter and J. Schmidhuber, “Long short-term memory.,” *Neural Computation*, vol. 9(8), pp. 1735–1780, 1997.

[44] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” 1999.

[45] F. A. Gers and J. Schmidhuber, “Recurrent nets that time and count,” in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, pp. 189–194, IEEE, 2000.

[46] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[47] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[48] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[49] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[50] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[51] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[53] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[54] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[55] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[56] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[57] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO (1)*, pp. 222–229, Citeseer, 2004.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[59] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, pp. 1861–1870, PMLR, 2018.

[60] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[61] S. R. Chu, R. Shoureshi, and M. Tenorio, "Neural networks for system identification," *IEEE Control systems magazine*, vol. 10, no. 3, pp. 31–35, 1990.

[62] M. D. Knudsen, R. E. Hedegaard, T. H. Pedersen, and S. Petersen, "System identification of thermal building models for demand response–a practical approach," *Energy Procedia*, vol. 122, pp. 937–942, 2017.

[63] F. Scotton, L. Huang, S. A. Ahmadi, and B. Wahlberg, "Physics-based modeling and identification for HVAC systems," in *2013 European Control Conference (ECC)*, pp. 1404–1409, IEEE, 2013.

[64] J. Kreider, D. Claridge, P. Curtiss, R. Dodier, J. Haberl, and M. Krarti, "Building energy use prediction and system identification using recurrent neural networks," 1995.

[65] S. Alawadi, D. Mera, M. Fernández-Delgado, F. Alkhabbas, C. M. Olsson, and P. Davidsson, "A comparison of machine learning algorithms for forecasting indoor temperature in smart buildings," *Energy Systems*, pp. 1–17, 2020.

[66] C. Xu, H. Chen, J. Wang, Y. Guo, and Y. Yuan, "Improving prediction performance for indoor temperature in public buildings based on a novel deep learning method," *Building and Environment*, vol. 148, pp. 128–135, 2019.

[67] F. Mtibaa, K.-K. Nguyen, M. Azam, A. Papachristou, J.-S. Venne, and M. Cheriet, "LSTM-based indoor air temperature prediction framework for HVAC systems in smart buildings," *Neural Computing and Applications*, vol. 32, no. 23, pp. 17569–17585, 2020.

[68] D. Kolokotsa, "Comparison of the performance of fuzzy controllers for the management of the indoor environment," *Building and Environment*, vol. 38, no. 12, pp. 1439–1450, 2003.

[69] A. Berouine, E. Akssas, Y. Naitmalek, F. Lachhab, M. Bakhouya, R. Ouladsine, and M. Essaaidi, "A fuzzy logic-based approach for HVAC systems control," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 1510–1515, 2019.

[70] R. Alcalá, J. Alcalá-Fdez, M. J. Gacto, and F. Herrera, "Improving fuzzy logic controllers obtained by experts: a case study in HVAC systems," *Applied Intelligence*, vol. 31, no. 1, p. 15, 2009.

[71] H. Shu and Y. Pi, "PID neural networks for time-delay systems," *Computers & Chemical Engineering*, vol. 24, no. 2-7, pp. 859–862, 2000.

[72] J. Kang, W. Meng, A. Abraham, and H. Liu, "An adaptive PID neural network for complex nonlinear system control," *Neurocomputing*, vol. 135, pp. 79–85, 2014.

[73] C.-C. Cheng and D. Lee, "Artificial intelligence-assisted heating ventilation and air conditioning control and the unmet demand for sensors: Part 1. problem formulation and the hypothesis," *Sensors*, vol. 19, no. 5, p. 1131, 2019.

[74] G. Xue, C. Qi, H. Li, X. Kong, and J. Song, "Heating load prediction based on attention long short term memory: A case study of xingtai," *Energy*, vol. 203, p. 117846, 2020.

[75] D.-S. Kapetanakis, D. Christantoni, E. Mangina, and D. Finn, "Evaluation of machine learning algorithms for demand response potential forecasting," in *The 15th International Building Performance Simulation Association Conference (Building Simulation 2017), San Francisco, United States of America, 7-9 August 2017*, IBPSA, 2017.

[76] Q. Li, Q. Meng, J. Cai, H. Yoshino, and A. Mochida, "Applying support vector machine to predict hourly cooling load in the building," *Applied Energy*, vol. 86, no. 10, pp. 2249–2256, 2009.

[77] M. Gong, J. Wang, Y. Bai, B. Li, and L. Zhang, “Heat load prediction of residential buildings based on discrete wavelet transform and tree-based ensemble learning,” *Journal of Building Engineering*, vol. 32, p. 101455, 2020.

[78] A. Lahouar and J. B. H. Slama, “Random forests model for one day ahead load forecasting,” in *IREC2015 The Sixth International Renewable Energy Congress*, pp. 1–6, IEEE, 2015.

[79] D. Azuatalam, W.-L. Lee, F. de Nijs, and A. Liebman, “Reinforcement learning for whole-building HVAC control and demand response,” *Energy and AI*, vol. 2, p. 100020, 2020.

[80] B.-K. Jeon and E.-J. Kim, “LSTM-based model predictive control for optimal temperature set-point planning,” *Sustainability*, vol. 13, no. 2, p. 894, 2021.

[81] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang, and X. Guan, “Deep reinforcement learning for smart building energy management: A survey,” *arXiv preprint arXiv:2008.05074*, 2020.

[82] Z. Wang and T. Hong, “Reinforcement learning for building controls: The opportunities and challenges,” *Applied Energy*, vol. 269, p. 115036, 2020.

[83] R. Jia, M. Jin, K. Sun, T. Hong, and C. Spanos, “Advanced building control via deep reinforcement learning,” *Energy Procedia*, vol. 158, pp. 6158–6163, 2019.

[84] T. Wei, Y. Wang, and Q. Zhu, “Deep reinforcement learning for building HVAC control,” in *Proceedings of the 54th annual design automation conference 2017*, pp. 1–6, 2017.

[85] S. Brandi, M. S. Piscitelli, M. Martellacci, and A. Capozzoli, “Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings,” *Energy and Buildings*, vol. 224, p. 110225, 2020.

[86] Z. Zou, X. Yu, and S. Ergan, “Towards optimal control of air handling units using deep reinforcement learning and recurrent neural network,” *Building and Environment*, vol. 168, p. 106535, 2020.

[87] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[88] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI gym,” *arXiv preprint arXiv:1606.01540*, 2016.

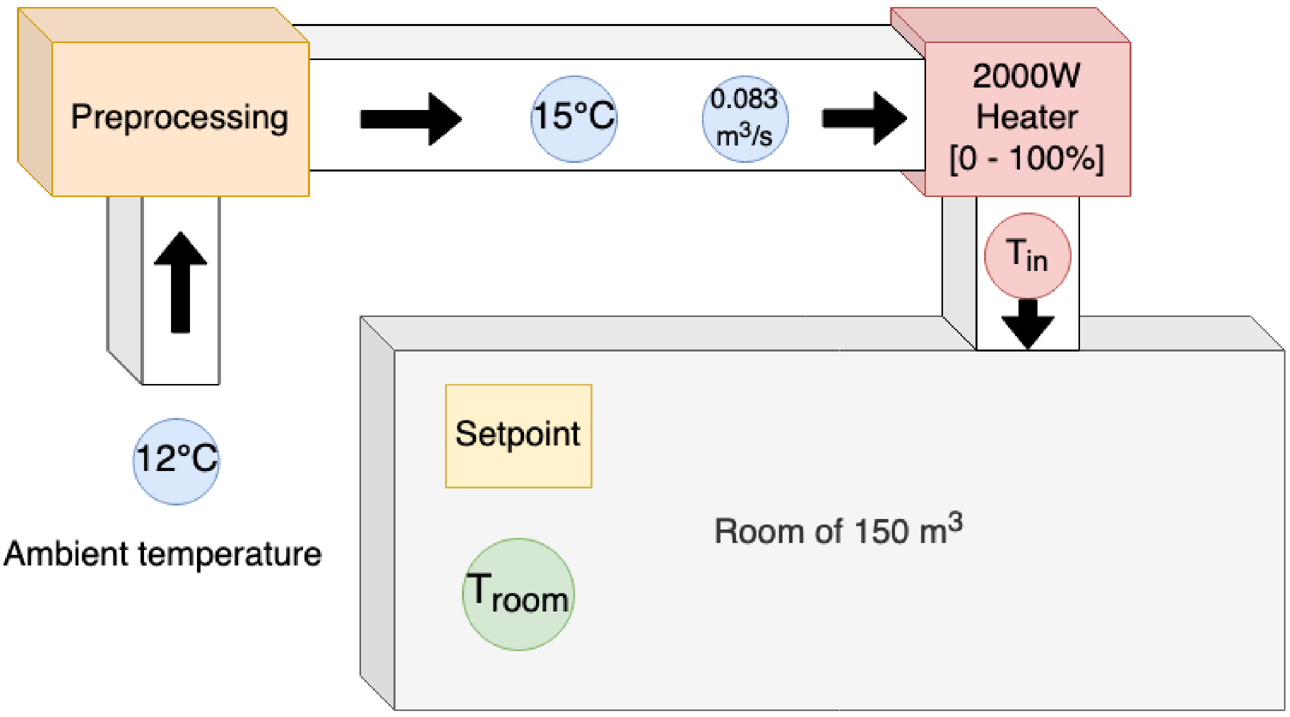


Figure 25: Schematic view of the room in the simulation.

Appendices

A Simulation

During the development of this study, a simple simulation was constructed on which the several methods could be verified individually. The simulation was used to debug the implementation of the DRL training algorithm and to verify the validity of the Neural Twin acting as a simulator.

The simulation room

Figure 25 shows a schematic of the simulation room. The simulation consists of a single room with a volume of 150m^3 (10x5x3). The air temperature of the room is measured with a sensor which logs values every 2 minutes. The air temperature of the room can be manipulated using an air handling unit. The following assumptions are made for the sake of simplicity:

1. The ambient temperature is constant at 12 °C.
2. The supply air temperature of the air handling unit is assumed to be preprocessed such that it is always at a constant temperature $T_{supply} = 15$ °C.
3. The air flow provided by the air handling unit is considered to be constant at 0.083 m^3 per second (300 m^3 per hour).
4. Heat is lost through the walls and the roof of the room (no heat is lost through the floor). This corresponds to an area A of $(10*5)+2*(10*5)+2*(5*3) = 180$ m^2.

5. The thermal transmittance (U-value) for this area is known and fixed at 0.5 W/m^{2}K (which corresponds to a reasonably isolated area).

A heater is placed with a capacity of 2000 W, which can be controlled by sending a control signal between 0% and 100%. This control signal will have a linear influence on the heater. This heater can be used to heat up the air as received from the air handling unit, which will be assumed to be constant in both temperature and flow.

With this information, the temperature of the air after the heater can be calculated, T_{in}, as well as the room air temperature T_{room} that follows from this. These temperatures are described by the following equations:

$$H_{watt} = H_{max} * \frac{H\%}{100} \tag{48}$$

Where H_{watt} represents the energy produced by the heater, H_{max} represents the maximum capacity of the heater and $H\%$ represents the current control signal of the heater. Then,

$$T_{in} = T_{supply} + \frac{H_{watt}}{\phi_v * \rho_{air} * c_{air}} \tag{49}$$

Where T_{in} represents the temperature of the air that is blown into the room, T_{supply} represents the temperature of the air before the heater (15 °C), ϕ_v represents the air flow (0.083 m^3), ρ_{air} represents the density of air and c_{air} represents the thermal capacity of air.

With this, the total energy added to the room can be calculated:

$$E_{added} = \phi_v * \rho_{air} * c_{air} * (T_{in} - T_{room}) \tag{50}$$

Furthermore the room loses energy through its walls according to:

$$E_{lost} = A * U * (T_{ambient} - T_{room}) \tag{51}$$

The total energy is simply the sum of those two:

$$E_{total} = E_{added} + E_{lost} \tag{52}$$

Finally, the new temperature of the room, after 120 seconds (2 minutes) will then be:

$$T_{room} = T_{room} + 120 * \frac{E_{total}}{V * \rho_{air} * c_{air}}, \tag{53}$$

where V is the volume of the room.

Please note that this is an extremely simplified simulation of a room, and its purpose is merely to act as a test bed on which the approaches presented in this study were validated.

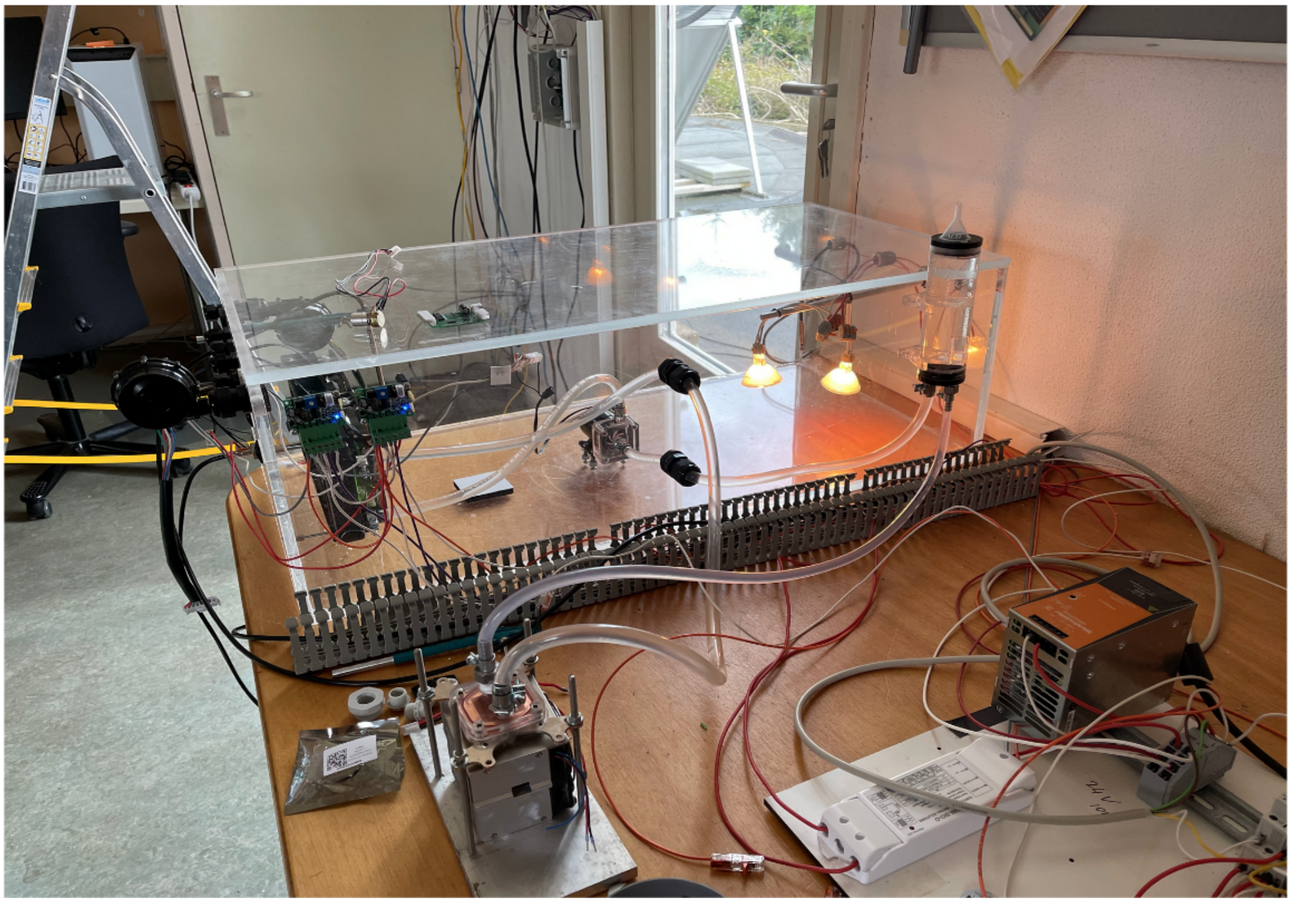


Figure 26: Lab setup used for online learning.

B Online learning

During the early stages of the project, another idea was to perform online learning, i.e. learning along the way. Because this would disrupt the regulation of a room in the early stages of learning (because the agent has not learned anything yet), an lab setup was created on which this concept could be tested. Figure 26 shows the lab setup used to test online learning. It is a mini room made from thick glass. It can be heated by turning on the lamps inside (the mini room can be made approximately 40 degrees Celsius).

Because during online learning the agent acts directly in the environment, it takes time to collect experiences, meaning the experiences are costly. Because of this, the Soft Actor-Critic algorithm was used during online training. The agent was given three inputs:

1. The ambient air temperature (outside the mini room).
2. The inside air temperature.
3. The modular value of how much the lamps were on (0% - 100%).

The agent was learning for approximately two weeks and the results are shown in Figure 27. From these results it can be observed that in the beginning the agent has no control over the temperature. However, after approximately a week the agent starts steering the temperature around the setpoint and the longer it trains, the closer it hovers around the setpoint.

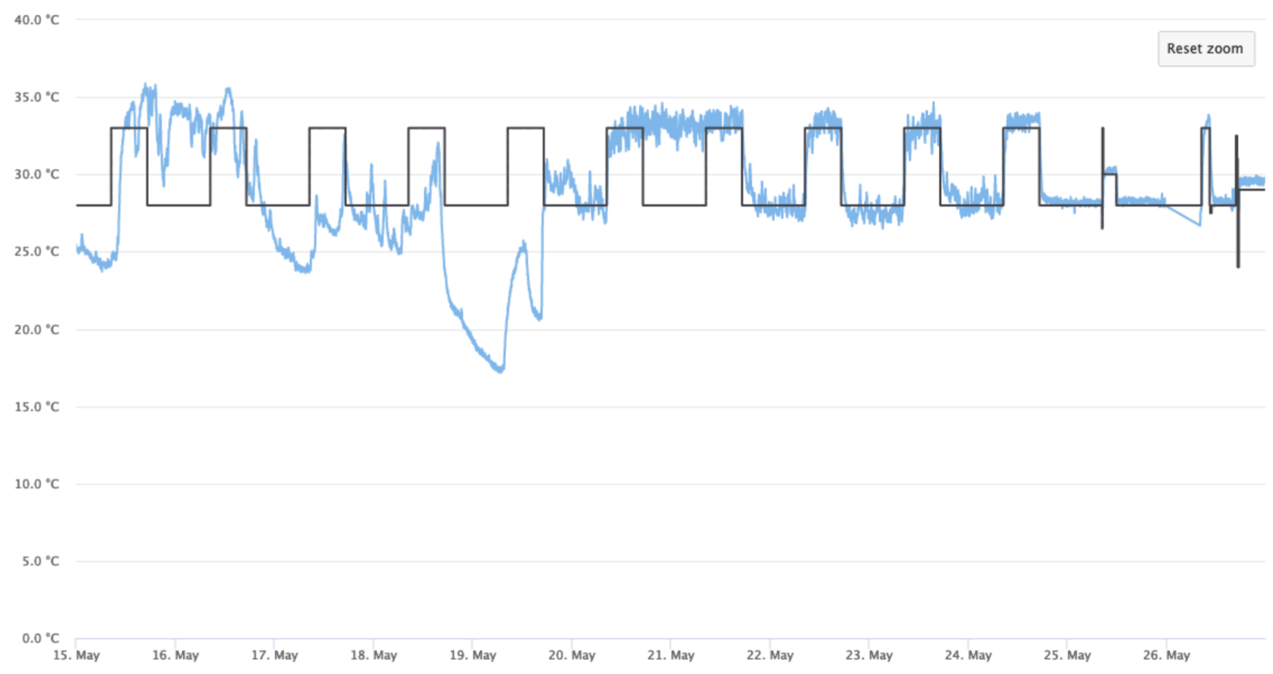


Figure 27: Results of the real time RL algorithm. The blue line denotes the mini room air temperature and the black line denotes the setpoint.

Although technically the experiment was successful, when thinking about the orders of magnitude a real room is more difficult than our mini room, we concluded that using online learning (from scratch at least) is too costly and too time consuming. The mini room reacts within two minutes on a change in control, but a sports hall can take half an hour. Furthermore there is a much wider range of environmental possibilities and disturbances for real rooms. Our lab setup required only 3 inputs, but real rooms require easily 4 times as many.

From this, we estimated that learning control of a room could easily take multiple months. This is infeasible as buildings cannot be out of control for such a long time. This is one of the reasons we were motivated to learn a simulator (a Neural Twin) for control.