From Data To Control: Learning an HVAC Control Policy

Bram de Wit [a], *Lambert* Schomaker [b], *Jaap* Broekhuizen [c]

[a] Advanced Climate Systems, Groningen, the Netherlands, b.dewit@acs-buildings.com

[b] Faculty of Science and Engineering, University of Groningen, Groningen, the Netherlands, l.r.b.schomaker@rug.nl

[c] Advanced Climate Systems, Groningen, the Netherlands, j.broekhuizen@acs-buildings.com

Abstract. This study introduced a framework for smart HVAC controllers that can be used at scale. The proposed controllers derive their control policy solely from data. First a simulator of the process is learned, which we call the Neural Twin. The results showed 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 several hours ahead. Then, the Neural Twin was used to develop two different control algorithms. The first algorithm learned a control policy for a process using a neural network. The network was trained using Proximal Policy Optimization by gathering experience from the simulated environment provided by the Neural Twin. The second algorithm performed Model Predictive Control using the Neural Twin at real time during control. It used the Neural Twin to choose an optimal control sequence, given a set of possible control sequences. It selects the optimal control sequence based on a horizon, which is usually a few hours ahead. Both control algorithms were inspected in several environments and for one of those environments the best controller was tested on a physical room. The results show that the control algorithms were able to handle a wide variety of different processes, without manual tuning. The controllers achieved 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 was concluded that the control methods pose an attractive alternative compared to conventional controllers.

Keywords. Deep Learning, Reinforcement Learning, HVAC control, system identification.
DOI: https://doi.org/10.34641/clima.2022.362

1. Introduction

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 [2]. 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. The International Energy Agency claimed the following in their report of 2018: "The buildings and construction sector should be a primary target for greenhouse gases (GHG) emissions mitigation efforts, as it accounted for 36% of final energy use and 39% of energy- and process-related emissions in 2018." ([3] page 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.

The goals above pose a conflict, as regulating the indoor climate 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.

Much research has already been done with respect to machine learning and smart building controllers. An extensive survey of AI-assisted HVAC control is given in [4]. One of the difficult parts of HVAC control is that often the underlying dynamics of a building are unknown, and these dynamics can be very complex. When finding optimal HVAC control policies, it can be useful to try and understand the underlying dynamics of the system one is trying to control. This knowledge can then be exploited to create a good control policy. The process of using observed data of the process to build mathematical models of that dynamical system is called *system identification*.

An important part of system identification for building control is indoor temperature prediction. Many machine learning methods have been used for this purpose and have proven to be successful [5]. In [6], a Long Short-Term Memory-based (LSTM-based) sequence-to-sequence framework was proposed, achieving good results in predicting indoor temperatures 6 hours ahead at a 10 minute interval. The Neural Twin framework presented in this study was heavily based upon this work.

The recent developments in the field of Deep Reinforcement Learning (DRL) have also been applied to HVAC control. A recent survey showed the potential of these methods [7]. 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, making online deployment simply not desirable. Another survey showed that of the 77 articles that were studied [8]:

1. The majority of the DRL agents in the studies did not control the actuators of the building directly. Instead, they controlled for example the setpoint that a controller needs to regulate. Classic controllers were still needed to track the setpoints determined by the DRL agent;
2. 91% of the studies did not include previous states as inputs;
3. 90% of the studies used data simulated by simulation software such as EnergyPlus;
4. 83% of the studies did not include predictions as inputs;
5. Only 11% of the studies implemented and tested their approaches in an actual building.

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 reinforcement learning.

Reinforcement Learning has been applied onto many different parts of the building's control. In [9], 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 [10]. There, temperature violations were kept approximately constant but the cost of regulating was decreased significantly. In [11], 5% - 12% energy savings were achieved, also by controlling 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 authors' knowledge, only one study tried learning a DRL using an environment based on a model which learned the system dynamics of the building [12] (hence, it did not need a simulator). In that study, 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 retaining thermal comfort.

This study contributes to the literature presented above by:

1. Introducing a scalable approach for learning a HVAC control policy, solely using sensor data gathered from real buildings (instead of a simulated buildings);
2. Controlling the actuators of a building, rather than the setpoints. Therefore, it completely substitutes classical control methods;
3. Testing the approach on a variety of environments, with different physical properties and different HVAC controls, showing the scalability of the approach;
4. Testing the obtained controller on a real building, validating its performance in the physical world.

2. Research Methods

2.1 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.

The data was retrieved from Climatics, the building management system of the buildings used in this study. 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 does not yield the full system state, as a state 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 time step consists of several components.

1. Values for all sensors of the building relevant for the specific process. These can be splitted into three types of sensors: the sensors to model (room temperatures), actuator sensors and remaining/external sensors that cannot be modelled or controlled, but are relevant for the process.
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).
3. Encoded cyclic time information. Three cycles can be identified for a data set of a building: a day cycle, a week cycle and a year cycle. All cycles are mapped to a unit circle, and the corresponding x and y coordinates are fed to the network as input.

In general, the data was used to simulate short episodes of historical data. In order to do this, a certain start time t is considered. 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.

2.2 Environments

This study tested on four different environments. The environments all vary in some factor, either in size, geographic location or HVAC control system. The four environments are:

1. Sports hall, Slochteren. A relatively large room heated by a radiator. A schematic of the control system for this room is given in Figure 1.
2. Sports hall, Hoogezand. A comparable room heated by a radiator, located in Hoogezand.
3. Changing rooms, Hoogezand. A relatively small room, heated by a radiator.
4. Central hall, Kalckwijck. A large entry hall heated by an air handling unit.

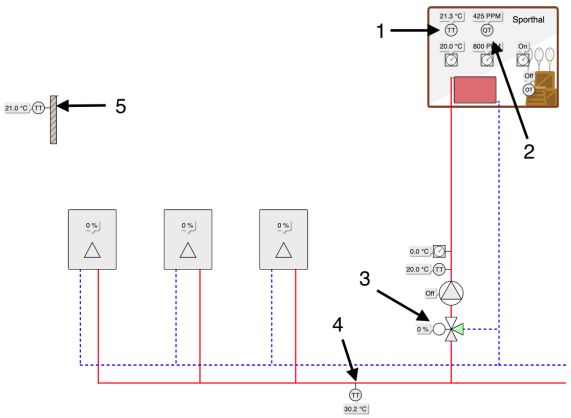

Fig. 1 - This figure was taken from Climatics. It shows the schematic for environment 1: the sports hall in Slochteren. The schematic shows the relevant sensors for this environment. Sensor 1: air temperature of the sports hall. Sensor 2: CO2 sensor, measuring parts per million (PPM), i.e. the concentration of the air. Sensor 3: valve position for the radiator. Sensor 4: central supply temperature. Sensor 5: outside air temperature.

2.3 Neural Twin

The Neural Twin is an Encoder-Decoder model based on Gated Recurrent Units (GRUs). A GRU retains its state over time, which is often called the hidden state. The sensor data fed to the network modifies this hidden state over time. 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 the same hidden size). The encoder will encode all past observations $\{\mathbf{o}_{t-h}, \ldots, \mathbf{o}_{t-2}, \mathbf{o}_{t-1}\}$ in order to yield a compact representation of the state of the process. The hidden state of the encoder is then used as the initial hidden state of the decoder. Then, when feeding a new observation to the decoder, it predicts the process values one timestep in the future. This can be realised by stacking a Multilayer Perceptron on the decoder, which interprets the hidden state at that time. 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 weather variables) depending on the use case of the Neural Twin. For Model-Predictive Control 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 accurately as possible, one can choose to let the input of the decoder consist of the full observation. In this case, one can use historical data for these external variables concatenated with the output of the decoder and generated control values as input for the decoder. This might be desirable when one wants to train a Reinforcement Learning agent in a simulation

setting. Then, the Neural Twin will not be needed during inference, as in that case the trained agent will take over the control, which does not require future values of external variables.

A schematic view of an Encoder-Decoder network in the context of a control system is given in Figure 2. 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.

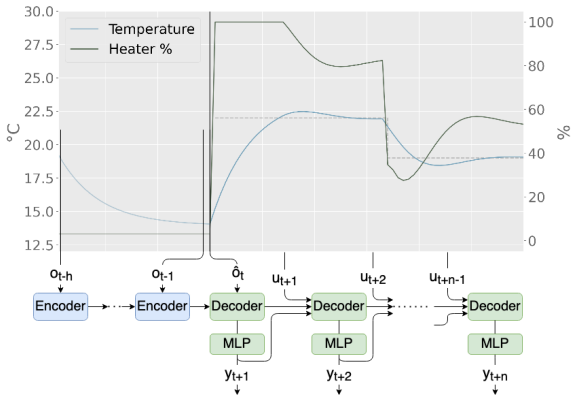

Figure 2 - A schematic view of a Neural Twin model where the decoder model expects a different number of inputs then the encoder model. The black vertical line represents the split between past observations and future observations. Past observations $\{\mathbf{o}_{t-h},\ldots,\mathbf{o}_{t-2},\mathbf{o}_{t-1}\}$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

The Neural Twin is trained using gradient descent. 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 the decoder receives the target value of the previous timestep as input, instead of its own output from the previous timestep. 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.

2.4 DRL 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 a simulator. Historical data was used for the external variables during simulation.

In this study, a Multilayer Perceptron 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 [13], the past *n* observations will be concatenated. In addition to this, the agent will need to know the desired setpoint of the room. The setpoint will be concatenated to the past observations and these will serve as inputs for the policy network.

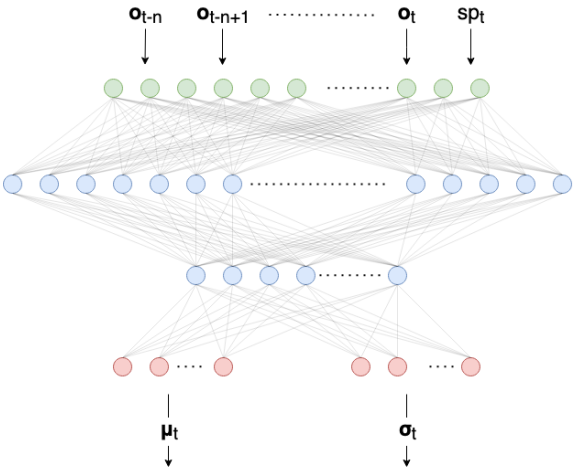

Figure 3 - A schematic view of the policy network used by the agent.

In order to perform exploration during training, the network is designed to 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. $\mathbf{a}_t \sim N(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t)$. During inference, the means will be used as actions. The action eventually produced by the agent will be offsets for the current control values. Hence, if there is a single actuator that has a current value of 50 and the output of the network is -2, the actuator's value will become 48. A schematic view of the policy network is given in Figure 3.

Proximal Policy Optimization [14] is used to train the agent. The Neural Twin of the environment is used to simulate episodes. It will take historical start moments, but then use the control values generated by the agent to simulate short episodes and gather experience to learn from.

Reinforcement Learning algorithms are designed to optimize a reward function. The reward function

describes quantitatively how 'good' the current state is. The reward function in this study consists of three weighted parts. The weights allow one to prioritize certain factors of control more compared to others. The reward function account for the following three factors:

1. Thermal comfort: the agent is rewarded by being close to the setpoint.
2. Stability: the agent is rewarded when its control is considered stable. The first order movements of the control values are used to measure this.
3. Energy consumption: the agent is rewarded more when using less energy.

This leads to the total reward:

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

Here, α and β can be used to weigh the different parts of the reward. The aim of the agent is to find a policy that maximizes this reward function.

2.5 Model-Predictive Control

Another way to utilize a trained Neural Twin is by using it for Model-Predictive Control (MPC). Note that for MPC the decoder of the Neural Twin can only accept the control values and the values it predicts as input. At run time future values for other sensors will not be available.

In this study, the random-sampling shooting method was used for MPC. 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 many trajectories $\mathbf{y}$ for each of these control sequences. For all the trajectories a cost can be calculated, and the first action of the control sequence that yielded the lowest-cost trajectory will be executed. This is repeated at every timestep. Also here, the actions that are denote offsets for the current control value. The actions were within the range of -50% and 50%.

The cost function plays the key role in MPC, as it determines what is the best trajectory, and hence what will be the next control value. The cost function used will be the conjugate of the reward function used for DRL, hence both algorithms will optimize for the same objective: DRL optimizes the reward, whereas MPC minimizes the cost.

Note that after training a Neural Twin, no additional training is needed, and it can be used directly for MPC. The algorithm's performance is heavily dependent on the hyperparameters h and K. The horizon h must be long enough to be able control the process accurately, but short enough in order for the Neural Twin to remain accurate. The number of sequences K must be large enough in order to generate enough sequences such that a good control sequence will be found, but small enough such that it remains computationally feasible.

2.6 Experimental Setup

The ultimate goal of the study was 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. The error of the predictions of the Neural Twin was reported.
2. One in which the validity of training a DRL agent on a Neural Twin of an environment is verified. The reward of the trained agents was reported.
3. One that reveals the influence of the hyperparameters for MPC control. The cost of control was reported for various combinations of hyperparameters.
4. One in which the two control methods are compared, based on control on the Neural Twin. The cost/reward of the control algorithms were compared when controlling on the Neural Twin.
5. One in which the two control methods are verified by trying them out in a physical environment. This was done only for the sports hall in the Duursweldhal, because testing on buildings can be costly and the authors got permission for this location.

3. Results

Figure 4 shows the result of the training of a Neural Twin for environment 1. The figure shows the mean and standard deviation of the training and validation loss over time for ten Neural Twins. As can be seen from the figure, the training and validation loss converge after approximately 25

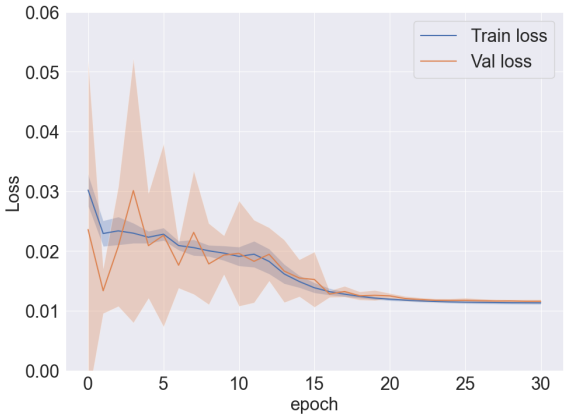

Figure 4 - Neural Twin training results for environment 1.

epochs around the same value, with low variance.

The same was observed for the other environments, but the graphs are excluded for brevity. Table 1 shows the error of the Neural Twins on the test set measured in ◦C for all environments.

Tab. 1 - Neural Twin results on the test data. The mean and the standard deviation of the ◦C error on the test data for the 10 training procedures are given.

Env	Error in ◦C
1	0.24 ± 0.21
2	0.23 ± 0.32
3	0.16 ± 0.13
4	0.20 ± 0.16

Figure 5 shows the results of training an DRL agent 10 times on the Neural Twin for environment 1. The graphs for the other environments are excluded for brevity. It can be observed that the training and validation reward converges over time, both to approximately the same value with low variance. Other environments showed the same behaviour.

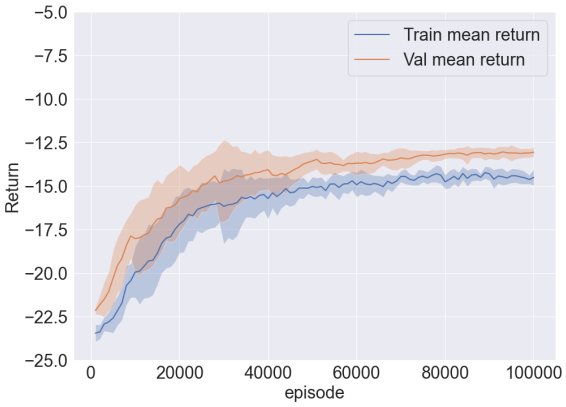

Figure 5 - DRL agent training results for environment 1.

Figure 6 shows the results for the hyperparameter experiment for environment 1. The graphs for the other environments are excluded for brevity. It is observed that the cost decreases for longer horizons and for a larger number of random sequences. The optimal combination of hyper parameters lies somewhere around the horizon of nine or ten timesteps ahead with at least 200 sequences, as there the cost is the lowest.

Table 2 shows the results when comparing the DRL agents to the conventional controller used when collecting the data, based on simulations with their corresponding Neural Twin. A negative percentage denotes an improvement, whereas a positive percentage denotes a decrease in performance. It can be observed that the DRL agents were able to decrease the energy usage for all environments while approximately retaining the same amount of comfort and stability. Table 3 shows the same but for the MPC algorithm.

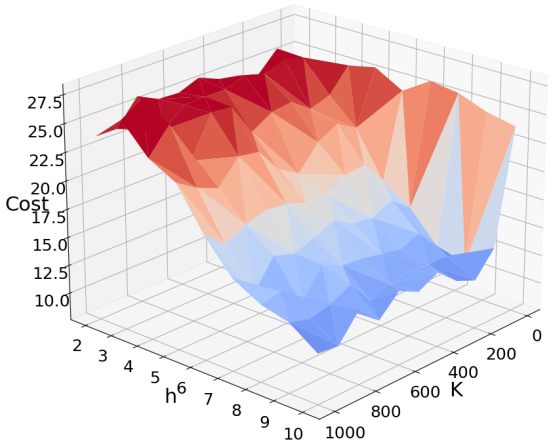

Figure 6 - MPC hyperparameter experiment results for environment 1.

Tab. 2 - Comparison of DRL control agent with the conventional controller. Negative values denote an increase in performance.

Env	Energy usage	Setpoint deviation	Roughness
1	-15.4%	-2.8%	-2.4%
2	-7.46%	-2.7%	-1.9%
3	-5.3%	-3.0%	+1.5%
4	-46.9%	-3.7%	+1.7%

Tab. 3 - Comparison of MPC algorithm with the conventional controller. Negative values denote an increase in performance.

Env	Energy usage	Setpoint deviation	Roughness
1	-17.5%	-3.3%	-2.5%
2	-5.7%	-3.2%	-2.4%
3	+25%	-4.8%	-1.0%
4	+37.2%	-6.1%	+2.3%

It was observed that in some cases the MPC algorithm decreased the energy usage, but in other cases also significantly increased the energy usage compared to the conventional controller. The deviation from the setpoint and the stability of the controller remained approximately equal.

Figure 7 shows the performance of the DRL agent on the physical room, i.e. the agent controlled the physical room in real time. It should be noted that the setpoint was enhanced by Climatics to warm up the room before the actual setpoint jump around

06:00 AM, which is not visible in the figure. This was done automatically to make sure the occupants did not enter a cold room in the morning.

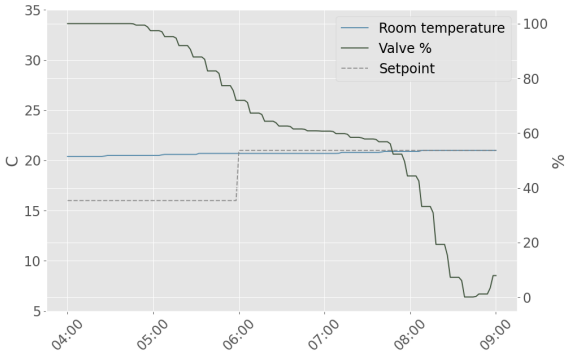

Figure 7 - Performance of the DRL agent controller on the physical room, i.e. the sports hall in Slochteren.

4. Discussion

The results of Table 1 show that all Neural Twins were able to converge to an error of approximately 0.2 ◦C. Furthermore it should be noted that there is some degree of variance in the error of the Neural Twins across environments. This can be explained due to the fact that all processes are different, and some can simply be more complex to model. The changing room environment consisted of a small room with almost no windows, making it easier to model. This is confirmed by the small error of 0.16◦C. Figure 4 shows how the loss converged over time when training the Neural Twin for environment 1. It can be observed that in the first 10 epochs the loss does not decrease much, and there is a high variance for the validation loss. This is because during these epochs teacher forcing was used with a decreasing chance the more epochs had passed. Hence, the training gradually became more difficult during these epochs. After 10 epochs, no teacher forcing was used and from there it can be seen that the training loss and validation loss converge to approximately the same value with low variance.

Figure 5 shows the learning scheme of the DRL agent on the Neural Twin of environment 1. It can be seen that the return converges over time. The return on the validation set is slightly higher than the return on the training set. This is because when validating the agent, the agent acted deterministically, whereas during training the agent acted stochastically. From the figure it can be concluded that a Neural Twin can serve as a valid simulator for a DRL agent to learn a policy on, and a DRL agent is able to control the simulator (note that it does not yet say anything about the validity of this policy on the physical process).

In Figure 6 the influence of the two hyperparameters for MPC can be clearly seen. It is observed that the cost decreases when the horizon is increased. This makes sense, as a large room like a sports hall often has a large time delay when reacting to changes in control. Hence, when taking a larger horizon into consideration when determining the optimal control values, it is expected that the MPC algorithm performs better. The number of random sequences generated at every time step does not seem to have a big influence on the cost, as long as it is larger than 100 sequences. This makes sense, as too few sequences will not cover the control space well enough to get good sequences to begin with, hence the algorithm will never be able to pick a good one. However, it seems that as soon as 200 or more sequences are generated, the cost does not decrease anymore, hence the extra sequences only introduce extra computational cost.

Table 2 and Table 3 compare the performance of the controllers with the conventional controller, i.e. the one that currently controls the process. This controller was manually tuned during the time it operated. From these results it can be concluded that the DRL agent improved upon the performance of the conventional controller for all four environments. It decreased the energy usage for all four environments while retaining the thermal comfort and stability of control. It also performed better compared to the MPC algorithm, which in 2 environments led to an increase in energy usage. For example in environment 4 the energy usage increased 37%. This can be explained by the fact that this environment was controlled by multiple control values, as opposed to the other environments. As the number of control values increases, the space of possible control values grows exponentially. This highlights an important bottleneck for the MPC algorithm, as a large number of control values means generating exponentially more sequences before the algorithm starts to work. The DRL agent decreased the energy usage for this environment with 47%. After consideration this is considered as optimistic, because this environment was controlled by an air handling unit, where the current controller controls for both the CO2 level and the temperature level. The DRL agent was only focussed on the temperature level, although this can be easily extended to other factors as well in future research. As a result, the comparison might not be completely valid. It *does* show that the DRL agent was able to control a more complex environment with an air handling unit and multiple control values. Furthermore, the reward function can easily be modified to include the CO2 level as well.

Figure 7 shows that the policy learned on the simulator transfers to the physical world as well. The figure shows how the process value reaches its setpoint and how the control value is adjusted correspondingly. Unfortunately, due to time constraints it was not possible to gather enough information about the comparison with the conventional controller, as the agent would have had to control the room for several weeks or months in order to obtain a quantitative measure. Hence, these

results only show that the gap from the simulator to the physical world is small enough to be able to train policies on the simulator, i.e. the Neural Twin.

5. Conclusion

To summarize:

- The Neural Twin framework can be used to accurately simulate the dynamic processes of various environments. The average prediction error was approximately 0.2 °C.
- The Neural Twin was used to train/develop two different control policies, one based on DRL and one based on MPC.
- The DRL agent showed the best control performance, outperforming the conventional controller in all four different environments. It was able to reduce the energy usage significantly between 5% and 40%, while retaining thermal comfort and stability.
- The policies learned on the Neural Twin transferred to the physical world, as the controller was able to control the physical process in real time as well.
- The simulators and policies were learned without manual tuning of hyperparameters, showing the scalability of the approach.
- Control using MPC degraded as the action space growed. DRL still handled this well, but it remains unknown whether the approach will work for large action spaces (i.e. a complex air handling unit with 7 actions).

6. Acknowledgements

The authors would like to thank Advanced Climate Systems (https://acs-buildings.com/en/) for letting us use their high resolution data and their platform: Climatics. Furthermore the authors would like to thank the building owners for allowing us to publish the results based on their data and letting us test the controller on the physical room.

The datasets generated during and/or analysed during the current study are not publicly available because the data is private but can be made available when requested in order to continue further research in cooperation with ACS. In that case, please contact the first author!

7. References

[1] Klepeis, Neil E., et al. The National Human Activity Pattern Survey (NHAPS): a resource for assessing exposure to environmental pollutants. Journal of Exposure Science & Environmental Epidemiology, 2001, 11(3): 231-252.

[2] Mayowa, M. O., Rowland, A. G., Babatunde, H. T., & Mumuni, A. 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, 2015, 3(4): 552-558.

[3] IEA, I. E. A. Global status report for buildings and construction. 2019.

[4] Cheng, C. C.; Lee, D. Artificial intelligence assisted heating ventilation and air conditioning control and the unmet demand for sensors: Part 1. Problem formulation and the hypothesis. *Sensors*, 2019, 19(5): 1131.

[5] Alawadi, S., et al. A comparison of machine learning algorithms for forecasting indoor temperature in smart buildings. *Energy Systems*, 2020, 1-17.

[6] Mtibaa, F., et al. LSTM-based indoor air temperature prediction framework for HVAC systems in smart buildings. *Neural Computing and Applications*, 2020, 32(23): 17569-17585.

[7] Yu, L., et al. Deep reinforcement learning for smart building energy management: A survey. *arXiv preprint arXiv:2008.05074*, 2020.

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

[9] Jia, R., et al. Advanced building control via deep reinforcement learning. *Energy Procedia*, 2019, 158: 6158-6163.

[10] Wei, T., Wang, Y., Zhu, Q. Deep reinforcement learning for building HVAC control. *Proceedings of the 54th annual design automation conference 2017*. 2017. p. 1-6.

[11] Brandi, S., et al. Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings. *Energy and Buildings*, 2020, 224: 110225.

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

[13] Mnih, V., et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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