Reinforcement Learning for Model-Free Control of a Cooling Network with Uncertain Future Demands

Jeroen Willems[1], Denis Steckelmacher[2], Wouter Scholte[1], Bruno Depraetere[1], Edward Kikken[1], Abdellatif Bey-Temsamani[1] and Ann Nowé[2]
[1]*Flanders Make, Lommel, Belgium*
[2]*Vrije Universiteit Brussel, Belgium*
fi *fl* *fi*

Keywords: Predictive Control, Reinforcement Learning, Thermal Systems Control.

Abstract: Optimal control of complex systems often requires access to a high-fidelity model, and information about the (future) external stimuli applied to the system (load, demand, ...). An example of such a system is a cooling network, in which one or more chillers provide cooled liquid to a set of users with a variable demand. In this paper, we propose a Reinforcement Learning (RL) method for such a system with 3 chillers. It does not assume any model, and does not observe the future cooling demand, nor approximations of it. Still, we show that, after a training phase in a simulator, the learned controller achieves a performance better than classical rule-based controllers, and similar to a model predictive controller that does rely on a model and demand predictions. We show that the RL algorithm has learned implicitly how to anticipate, without requiring explicit predictions. This demonstrates that RL can allow to produce high-quality controllers in challenging industrial contexts.

1 INTRODUCTION

Today, reducing the energy consumption of industrial systems is a key challenge. Energy reduction can be achieved by designing more efficient systems, but often improvements can be made by improving the way they are controlled. Academically optimization-based model-based controllers are often proposed when more advanced control is required. While these are effective, they introduce significant complexity due to the need for accurate models and powerful optimizers that run during system operation. Therefore, in industry the adoption is still rather limited. In this work, we explore reinforcement learning (RL) as a simpler, data-driven alternative that requires neither. We benchmark its performance against both model-based and simpler rule-based control (still widely used in industry) by evaluating them on a thermal system comprised of three interconnected chillers.

For the model-based controller in this paper, we consider the optimal control of a dynamic system. As time passes, with time denoted by t, the system changes state. The state of the system at some time t is denoted $x(t)$. Some control signal applied to the system at time t is denoted $u(t)$, and the resulting change of state is denoted $\dot{x}(t) = f[x(t), u(t), t]$. The function f defines how the system reacts to the control signal. It also depends on t, which allows the function to additionally depend on non-controlled information, such as the weather (that we cannot influence), some usage of the system by clients, etc.

The objective of optimal control is to produce control signals $u(t)$ such that, over some time period, a cost function dependent on the system states is minimized. Instances of optimal control include moving towards and tracking a setpoint, tracking some (moving) reference signal, or performing a task while minimizing energy consumption. Methods that attempt to achieve optimal control range from simple to very complicated, starting from proportional-integral-derivative controllers (Borase et al., 2021), over optimal feedback design methods like linear quadratic regularors and H-infinity (Glover, 2021), to function optimization - such as gradient methods (Mehra and Davis, 1972), genetic algorithms (Michalewicz et al., 1992), ant-colony optimization (López-Ibáñez et al., 2008), iterated local search methods (Lourenço et al., 2019) - and especially optimal control, with its continuous and discrete time, direct and indirect, and many more variants (Lewis et al., 2012), and its implementation that adds robustness through feedback called model-predictive control or MPC (Camacho and Alba, 2013). Conceptually, MPC observes the current state of the system, then uses a computational model of it to predict various

Willems, J., Steckelmacher, D., Scholte, W., Depraetere, B., Kikken, E., Bey-Temsamani, A. and Nowé, A.
Reinforcement Learning for Model-Free Control of a Cooling Network with Uncertain Future Demands.
DOI: 10.5220/0013708300003982

In *Proceedings of the 22nd International Conference on Informatics in Control, Automation and Robotics (ICINCO 2025) - Volume 1*, pages 59-70
ISBN: 978-989-758-770-2; ISSN: 2184-2809

possible outcomes, so the best-possible control signal can be chosen by a numerical optimization problem.

Apart from the PID approach, all these methods share a common trait: they are model-based. They assume that $f[x(t),u(t),t]$ is available, in a representation that can be used to design a controller or to compute control signals. This means that the system must be modelled, along with any function of t used by f. When relevant, they further include predictions on expected future events as well, using the model to predict how to best anticipate on those events.

However, in the real world, models are never perfect, and future demands are not always known perfectly. Methods exist to estimate models, or future demands, for instance with Kalman filters (Meinhold and Singpurwalla, 1983) or machine learning approaches such as time-series prediction (Frank et al., 2001). These methods are unfortunately approximate, and prevent *optimal* control from being achieved.

We propose to instead use another method of producing control signals: Reinforcement Learning (RL) is a family of machine learning (ML) algorithms that allows to learn a controller from experience. The important property of RL, that we explain extensively in Section 2, is that it is *model-free*. The RL agent, that learns the controller, makes no assumption about the system being controlled, and does not expect to have access to its state update function f or to the future demands[1].

RL has already been applied to numerous cases in research settings, like video games and (simulated) robotic tasks. Real-life application to industrial applications is more rare, though there are examples that go in that direction, like optimizing combustion in coal-fired power plants (Zhan et al., 2022), cooling for data centers (Luo et al., 2022), application to Domestic Hot Water (DHW) systems (De Somer et al., 2017) and systems of compressors (Chen et al., 2024).

In this paper, we will tackle a cooling problem, somewhat similar to (Luo et al., 2022), with both continuous (power levels) as well as discrete (on/off) control actions, and a non-instantaneous objective to minimize energy with penalties for frequent on/off switches. We will use existing RL techniques, and compare it to other approaches, since like mentioned above, such a problem would typically be tackled with MPC-like controllers that use a model, as well as a prediction of the upcoming demand.

Our contributions are as follows:

- We explain how to set up and formulate the RL problem for this application, relying on existing state-of-the-art RL algorithms for solving it.
- For this application, we present evaluations and comparisons of a rule-based controller or RBC (which is the industrial standard), an MPC, and an RL controller, showing that both MPC and RL outperform the RBC.
- We study the impact of imprecise or limited preview information on upcoming demand for the MPC, and compare it to the RL to show that the RL learns to anticipate even without being given explicit preview information.

[1]Some RL methods are called *model-based* because the RL agent learns an approximate model of f, but the true f is still not observed by the agent.

2 BACKGROUND

2.1 Reinforcement Learning

RL is a machine learning approach that allows to learn a closed-loop controller from experience with the controlled system. In most of the literature, RL considers a discrete-time Markov Decision Process (MDP) defined by the tuple $\langle S,A,R,T,\mu^0,\gamma\rangle$, with S the space of states, A the space of actions, $R : S \times A \to \mathcal{R}$ the reward function that maps a state-action pair to a scalar reward, $T : S \times A \times S \to [0,1]$ the transition function that describes the probability that an action in a state leads to some next state, μ^0 the initial state distribution, and $\gamma < 1$ the discount factor.

In physical applications, the state-space is usually a subspace of $\mathcal{R}^N$, vectors of N real numbers. The action space can either be discrete, with the action being one of some finite set of possible actions, or continuous, with the action space a subspace of $\mathcal{R}^M$.

RL considers multi-step decision making. Time is divided in discrete time-steps, and several time-steps form an episode. When an episode ends, the MDP is reset to an initial state drawn from μ^0, and a new episode starts. The objective of a RL agent is to learn an optimal (possibly stochastic) policy $\pi(a|s)$ such that, when following that policy, the expected sum of discounted rewards $\sum_t \gamma^t R(s_t, a_t \sim \pi(s_t))$ per episode is as high as possible.

For RL the learning agent does not have to have access to the reward or transition functions, it can get by with just evaluations, by e.g. interacting physically with a non-modelled plant. As such, the reward function can be implemented in any way suited for the task, and the transition function does not even have to exist in a computer format.

2.2 POMDPs

Partially-Observable Markov Decision Processes (Monahan, 1982) (POMDP) consider the setting in which the agent does not have access to the state of the process being controlled, but only a (partial) observation. A POMDP extends the MDP with the O observation space, and the $\Omega : S \rightarrow O$ observation function. The reward function still depends on the state: the agent is now rewarded according to information it may not observe.

POMDPs are extremely challenging to tackle, as the framework does not impose any lower bound on what the agent observes. No general solution is therefore available. However, methods that try to infer what the hidden state is (Roy and Gordon, 2002), methods in which the agent reasons about past observations using recurrent neural networks (Bakker, 2001), and methods that give a history of past observations as input to the agent (Mnih et al., 2015; Shang et al., 2021) currently perform the best.

We observe that POMDPs still assume that there is a hidden state, from which the reward function and next state are computed. This is not the case in the setup we introduce later, in which the future demand of cooling is not known to the agent, and not part of the state. It is an external signal that, in the real world, comes from the users and is not modeled. We still observe that the 'history of past observations' approach to POMDPs helps the agent in our setup, even if our setup is even more complicated than a POMDP.

2.3 Reinforcement Learning Algorithms

RL algorithms allow an agent to learn a (near-)optimal policy in a Markov Decision Process. In this article, we focus on on-line model-free RL algorithms: they interact with the environment (or a simulation of it), and learn from these interactions without using a model of the environment, or trying to learn one.[2]

Such RL algorithms are divided in three main families: value-based, that learn how good an action is in a given state, policy-based, that directly learn how much an action should be performed in a state, and actor-critic algorithms, that learn both a value function and a policy.

The best-known value-based algorithms are of the family of Q-Learning (Watkins and Dayan, 1992), with modern implementations being DQN (Mnih et al., 2015) and its extensions, such as Prioritized DQN, Dueling DQN or Quantile DQN, discussed and reviewed by (Hessel et al., 2018). The main properties of value-based algorithms is that they are sample-efficient, they learn with few interactions with the environment, but are in the vast majority of cases only compatible with discrete actions. Despite some contributions in that direction (Van Hasselt and Wiering, 2007; Asadi et al., 2021), value-based algorithms do not scale well to, or do not match policy-based methods in high-dimensional high-complexity continuous-action industrial tasks.

Policy-based algorithms historically started with REINFORCE (Williams, 1992), followed by Policy Gradient (Sutton et al., 1999), still the basis of almost every current policy-based or actor-critic algorithm. Most modern policy-based algorithms are actually actor-critic, because they learn some sort of value function (or Q-Values) in parallel with training the actor with a variation of Policy Gradient. Examples include Trust Region Policy Optimization (Schulman et al., 2015), Proximal Policy Optimization (Schulman et al., 2017) and the Soft Actor-Critic (Haarnoja et al., 2018), the current state of the art. Deterministic Policy Gradient (Silver et al., 2014) (DPG) is worth mentioning because it learns a deterministic policy (given a state, it produces an action), as opposed to the other approaches that are stochastic (given a state, they produce a probability distribution, usually the mean and standard deviation of a Gaussian). A modern variant of deep DPG (DDPG) is TD3 (Fujimoto et al., 2018), that combines DDPG with advanced algorithms for training the critic.

The main advantage of policy-based and actor-critic algorithms is that they are compatible with continuous actions. In this article, we therefore focus on them, and use Proximal Policy Optimization in particular, after we observed that it outperforms the Soft Actor-Critic in our setting. A plausible explanation for this surprising result (SAC is more recent than PPO) is that SAC relies heavily on a state-action critic, itself built on the mathematical assumptions of a Markov Decision Process. Because our task is a *Partially-Observable* Markov Decision Process, the PPO algorithm, that makes less assumptions, is able to learn more efficiently.

3 COOLING NETWORK SETUP

The particular use-case we consider in this paper is a scaled down version of an industrial cooling network. We will study a simulation model of the experimental setup shown in Figure 1. To allow for easier testing on this model and this setup, we consider a scaled

[2]The simulator itself may be considered a model of the environment, but can be approximate, does not need to be differentiable, and is not given to the agent.

cooling task: we perform a shorter cycle (5 or 10 minutes cycles vs. typical 24 or 48 hours cycles), and we have scaled the losses so that typical trade-offs as found in industrial cooling scenarios are present. These rescaled losses ensure the controller faces the typical industrial trade-off when deciding when and how much to use the chillers: alternating between on/off too often causes start/stop losses and wears out the units, while keeping them on for longer durations and building up a thermal buffer by cooling too much will result in extra heat transfer losses to the environment.

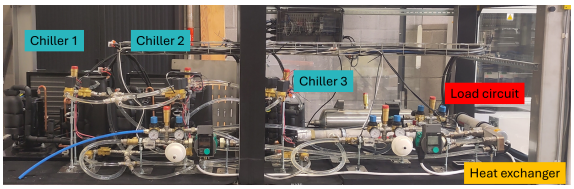

Figure 1: Experimental setup with 3 chillers and a load.

The considered system consists of:

- A cooling circuit with three chillers, which convert input electrical power P_{input} in to cooling power P_{cool}, with a given coefficient of performance (CoP), such that $P_{cool}(T, P_{input}) = CoP(T, P_{input})P_{input}$, see Figure 2. The chillers have varying (electrical) power ranges: 300W-600W, 200W-400W, 130W-260W. When running, they cannot operate between 0W and the respective lower bounds, but alternatively, the chillers can be off (0W).

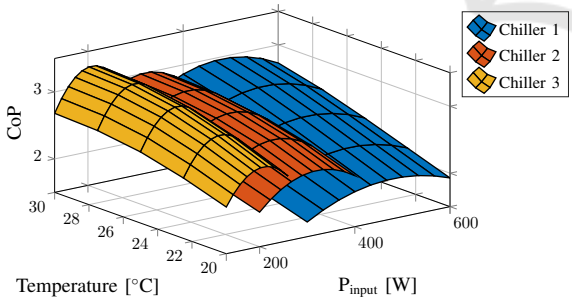

Figure 2: Operating region and CoP of the chillers.

- The load circuit, containing 1.5L water. The relatively small inertia of this load circuit is chosen so testing can be sped up (control results are seen quicker), but makes the system more challenging to control. The load circuit is heated by a programmable heating element according to a given power profile P_{load} detailed later. This circuit aims to mimic industrial cooling demands.
- A heat exchanger and a set of pumps. On the cold side a pump is there to ensure cold liquid flows from the chillers to the heat exchanger, and slightly warmer liquid flows back to them. On the hot side another pump is present to ensure flow from the load circuit to the heat exchanger and back.

The system is modelled in a simplified form using a single temperature T, assumed equal across the system. We thus do not have separate states for cold and hot sides, but simply assume the chillers as well as the heater directly impact T. We also use T as the input for our CoP's. Given this, the dynamics of T can be written as an ordinary differential equation (ODE):

$$\dot{T} = C(-P_{cool,\,total} + P_{load} - P_{loss}(T)), \qquad (1)$$

where $P_{cool,\,total} = \sum_{i=1}^{i=3} P^i_{cool}(T, P^i_{input})$ with i the respective chiller. $C = \frac{1}{mc_p}$ denotes the capacity, with $m = 1.5$ kg the mass of water and $c_p = 4186\ \frac{J}{kgK}$ the specific heat capacity. Second, P_{load} denotes the heat injected on the load side. Last, the power loss in the system is modelled using a lookup table dependent on T, with more power lost to the environment as T becomes lower. As mentioned, to compensate for the small inertia and the short duration of our tests, we have upscaled the losses to correspond to losses as would be faced during typical industrial operation.

The discretized version of Eq. 1 is given by:

$$\begin{aligned} &T(k+1) = T(k) + \\ &T_s(C(-P_{cool,\,total}(k) + P_{load}(k) + P_{loss}(T(k)))), \end{aligned} \qquad (2)$$

where T_s denotes the sampling time and k the respective time sample.

3.1 Control and Sampling

The system is controlled using three signals, P_{input} for each of the 3 chillers. A sampling time of 3 seconds is used, and we simulate tasks with a total duration of 10 minutes, yielding $N = 200$ control intervals. a scaled version of an industrial setting, with smaller inertias and timescales, and upscaled losses to compensate. Real cooling cases would likely need to be evaluated on 24h or 48h cycles.

3.2 Control Goal

The goal of the controller is to minimize the cost, while ensuring a constraint on the temperature is met.

The cost function of the system $\mathcal{J}$ over the total time interval $k \in [1, N]$ is set to:

$$\mathcal{J} = T_s \sum_{k=1}^{k=N} P_{input,\,total}(k) + \sum_{i=1}^{i=3} c^i w^i, \qquad (3)$$

where $P_{input,\,total} = \sum_{i=1}^{i=3} P_{input}^{i}$ with P_{input}^{i} the input power of each respective chiller, each with index i. The first term describes the total energy required to operate the chillers, i.e., the sum of the 3 chiller input powers over time. The goal of the second term is to prevent unnecessary compressor start-up and switching. To do so, it increases each time a chiller turns on in the given interval, w^i, with a given scaling per chiller, c^i, set equivalent to operating each chiller at the highest input power for 4 samples. This corresponds to the power consumption due to startup losses, and has as an additional benefit that it penalizes on/off switching and therefore reduces wear of the chiller's compressor.

Last, the temperature constraint can be written as:

$$T \leq T_{constraint}, \tag{4}$$

where $T_{constraint}$ = 25°C is the maximum allowable temperature, and the cooling has to ensure the temperature remains equal to or below it despite the heating load applied.

3.3 External Heating Profiles

As mentioned previously, the load circuit is heated by an external heating profile P_{load}, 10 minutes long, and consisting of 200 samples. In this paper, we consider 130 different profiles, which are based on a public library for unit commitment, see (IEEE, 2024). 15 of the profiles are shown in Figure 3: it can be seen that repetitive trends occur, but that still significant variations between the profiles are present. In the remainder of this paper, the first 100 profiles are selected as the training set, and the last 30 profiles as the test set.

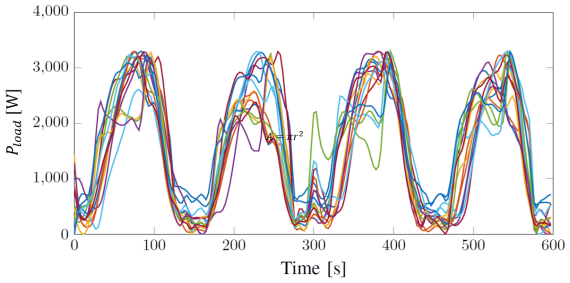

Figure 3: 15 of the 130 heating profiles.

4 REINFORCEMENT LEARNING ENVIRONMENT

We now define the Markov Decision Process to express the cooling case as an RL problem, by defining its action space, observation space and reward function.

4.1 Initial State Distribution

In our setup, episodes have a fixed length of 200 time-steps, equal to the total task duration. When an episode terminates, the simulation is reset (the water temperature in the tank is reset to 25 °C, and the 3 chillers are turned off), and a new simulation can be started.

4.2 Action Space

For the cooling case there are both discrete (on/off) and continuous actions (if on, at what power level). To make it easier for the RL, we transform this into a problem with only continuous actions. Since it is considered best practice (Raffin et al., 2021) to have the action and state spaces centered around 0, we chose a single value to control each chiller, ranging from -1 to 1, which we map to the chiller controls:

- Values below -0.5 are mapped to *off*.
- Values between -0.5 and 0 are mapped to *on*, but at minimum power input.
- Positive values are mapped to *on*, with the range $[0,1]$ corresponding to $[P_{lb}, P_{ub}]$, with P_{lb} and P_{ub} the minimum and maximum chiller power inputs.

This allows the agent to control the on/off status of the chillers, in addition to their power consumption setpoints, with a single real value.

4.3 Observation Space

The observation space consists of several values that measure past tank temperatures and power consumption setpoints. For every time-step, 4 values are logged: the *change* of temperature $(T(k) - T(k-1))$ of the water in the tank during the previous time-step, and the power input setpoints of the 3 chillers. When producing observations, the environment looks N_T time-steps in the past to produce N_T real values corresponding to changes in water temperature at these past N_T time-steps. The environment also looks N_P time-steps in the past to produce $3N_P$ real values corresponding to setpoints of the chillers during these past N_P time-steps. N_T and N_P can be distinct, and in our experiment, we use $N_T = 10$ past temperatures and $N_P = 6$ past setpoints. A final observation is given to the agent, a *running average* (rate of 0.1) of the water temperature.

By observing past changes of water temperatures and chiller setpoints, the agent is able to learn:

- The past demand and losses, by comparing the changes of water temperature with the setpoints.
- The efficiency of the chillers, and how they interact with the system (the agent has no model and has to discover the effect of its actions).

- All this combined allows the agent to build an internal estimate of what the future demand may be, given that the 100 demand profiles are distinct but share some structure.

This information is still not enough for optimal control, as it does not contain information about the future demand, but observing a history of past sensor readings has been shown to be one of the best approaches to learn in Partially Observable MDPs (Bakker, 2001), and is straightforward to implement.

4.4 Reward Function

In this section, the reward function it explained. The reward function is the negation of the change in cost that occurs after a given time-step $k \in [1,N]$. By summing over all time steps, the total reward (of a given episode) is computed.

The reward function is build up similarly to the cost (and temperature constraint) introduced previously in Section 3.2. Hence, the considered reward function (for a given time-step) consists of three components:

1. **Minimizing Energy Consumption.** The first part of the reward function is equal to (minus) the total energy consumed during the considered time-step k, i.e., $T_s P_{input,total}(k)$, with $P_{input,total}(k)$ the power consumed by all the chillers at sample k.
2. **Penalizing Turning-On of Chillers.** The second part of the reward function is a (negative) turn on penalty when a chiller turns on, like in Section 3.2.
3. **Constraint Satisfaction.** If the constraint on the water temperature (in this case, set to 25 °C) is violated, a reward of -2 is given to the agent for this time-step. A backup policy then turns all the chillers ON to maximum power, thereby also providing several turn-on penalties. The episode otherwise continues, and the agent learns over time to meet the constraint to avoid the -2 rewards.

Over an episode (of N samples), after the agent has learned to always meet the constraint, the sum of these rewards perfectly corresponds to the actual cost of running the episode, given by Eq. 3.

4.5 Training Procedure

The RL agent is trained using the Proximal Policy Optimization (Schulman et al., 2017), as implemented in Stable-Baselines 3 (Raffin et al., 2021). Hyperparameters are given in Section 6. The agent is allowed to learn for 40 million time-steps (about 5 hours using 8 AMD Zen2 cores running at 4.5 Ghz).

Every 128000 time-steps, the current policy learned by the agent is saved in a checkpoint file (of about 1.7 MB). After training, all these policies are run on demand profile 1 with exploration disabled, and the run with the smallest cost is selected as *the RL policy*. This evaluation takes about 11 minutes on a single AMD Zen2 core, and finds a policy that:

- Is about 2% better than the last policy produced by the agent (cost of 446 instead of 455);
- Was produced after about 20 million time-steps, so significantly before the end of the training (40 million time-steps).

These two points combined indicate that it is both beneficial and cheap to evaluate a collection of learned RL policies before picking one to use, because agents do not monotonically become better as they learn, but instead oscillate a bit around the optimal policy.

5 BENCHMARK CONTROLLERS

5.1 Model-Predictive Controller

As a first benchmark, we use a model-predictive controller (MPC). It optimizes the state trajectories and controls over a given horizon N_{mpc}. At every time step $k \in [1,N]$, an optimization problem is solved. After solving, the first sample of the computed input sequence is then applied to the system, and we again solve the optimization problem for the next time sample. This is repeated for the entire horizon with length N. The resulting discrete-time optimization problem is set up as follows:

$$\underset{T(\cdot),P_{input}(\cdot)}{\textbf{minimize}} T_s \sum_{k=N_{mpc}}^{k=1} P_{input,total}(k) + \gamma \sum_{i=3}^{i=1} (P^i_{input,\,previous} - P^i_{input}(1))^2, \tag{5a}$$

$$\textbf{s. t.} \quad (2) \qquad \forall\, k \in [1,N_{mpc}], \tag{5b}$$

$$T(k) \leq 25 \qquad \forall\, k \in [1,N_{mpc}], \tag{5c}$$

$$P_{lb} \leq P_{input}(k) \leq P_{ub} \;\forall\, k \in [1,N_{mpc}], \tag{5d}$$

$$T(1) = T_{init}. \tag{5e}$$

Eq. 5a implements the cost function. It is set to minimize (1) the sum of all input power, that is, the total energy consumption over the entire horizon, along with (2) a regularization term which penalizes the difference of the first sample of the current horizon and the last input applied to the system, thereby penalizing switching the control input. The system dynamics are implemented using Eq. 2, using multiple shooting. Eq. 5c implements the constraint on the allowed

temperature, and Eq. 5e ensures each MPC iteration starts from the temperature last measured on the system. Finally, for each chiller, constraints are set on the allowed input power in Eq. 5d.

As mentioned previously, each chiller can be ON (and if so, in a specific input power range), or OFF. This would cause the MPC to require a mixed-integer optimization problem, which are harder to solve. In this paper, we *approximate* the true mixed-integer problem by decomposing it using an outer loop (with only discrete variables) and an inner loop (with only continuous values). Solving the full optimization for a single time instance can be done using the following steps:

- Step 1: Select one of the combinatorial options for ON/OFF status of the 3 chillers e.g., [ON, OFF, OFF]
- Step 2: For the option selected in step 1, solve the inner loop MPC problem that minimizes the total energy.
- Step 3: Go back to step 1 and select another combinatorial option of chiller ON/OFF statuses, until all options are computed
- Step 4: Select the option with the lowest total energy as the solution to the nested MPC problem

In other words, the outer loop selects which chillers are ON or OFF (in the entire time interval). Then, the inner loop solves the above optimization problem (with input powers kept within to their input ranges for chillers that are ON, or kept at zero when OFF). The outer loop iterates over all possible combinations (in this case 8), and computes the resulting cost according to Eq. 3. The solution with the lowest cost (that meets the temperature constraints) is then selected and applied to the system. Note there is no penalty included for switching between ON/OFF states in the inner loop, since the MPC problem solved in the inner loop assumes the same set of chillers remains ON during the entire horizon. In the outer loop we do penalize sets that require a change in ON/OFF states when compared to the solution at the previous time sample.

The MPC problem is implemented using CasADi (Andersson et al., 2019) and is solved using IPOPT (Biegler and Zavala, 2009).

5.2 Rule-Based Controller

As a second benchmark, we use a rule-based controller or RBC. It is implemented in the following way:

- A feedback input $P_{desired} = K_p(T_{setpoint} - T)$ is computed using a proportional controller with gain K_p. The setpoint is given by $T_{setpoint}$, which can be set equal to the constraint on the load temperature (25 °C) minus an offset. This offset can be tuned to ensure that the controller meets the temperature constraint for varying heat load profiles.
- From $P_{desired}$, we can select which chillers to turn ON. This is done by first inspecting the relative efficiencies of combinations of chillers, as shown in Fig. 4. Then, for each $P_{desired}$, we can select the most efficient combination. In order to prevent switching of sets too often, we introduce a simple deadband (i.e., we only switch between combinations if the difference has exceeded this deadband).
- After the set of chillers to be used is chosen in the previous step, we select the input powers of each chiller. The goal is to match $P_{desired}$, while minimizing the total required input power by the chillers. This is done via a simple heuristic.

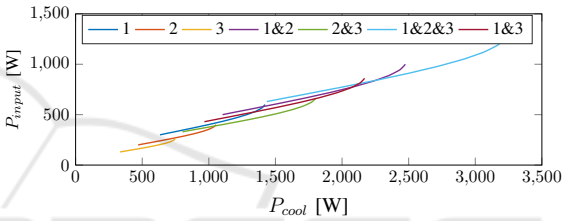

Figure 4: Relative efficiencies of the sets of chillers.

6 RESULTS

6.1 Nominal Case

In this section, we have run each of the controllers for the nominal case: without any perturbations. For each of the controllers, the settings are as follows:

- The RL algorithm is based on the Proximal Policy Optimization algorithm (Schulman et al., 2017), as implemented in the Stable Baselines 3 (Raffin et al., 2021). The hyper-parameters of that agent are the default ones from Stable Baselines 3 (as of January 2023), with the exception of: 128 parallel environments (`n_envs`), a batch size of 200, and both the actor and value networks are feed-forward neural networks with 2 hidden layers of 256 neurons. Given the limited number of neurons the risk of overfitting was deemed low, but more sparse neural networks could be explored in future work to perhaps improve results in regions outside of training. The RL agent was allowed to learn in our simulated setup for 40 million time-steps. These hyper-parameters have been obtained through informed manual tuning, and our results haven been found to be robust to changes in hyper-parameters.

- The MPC has a horizon of $N_{mpc} = 10$ samples, the other parameters are set as outlined earlier.
- For the RBC, we set the temperature setpoint to 23.1 °C and $K_p = 2500$.

We show the outcome of each of the three controllers for a specific example scenario in Figure 5 (the control inputs) and Figure 6 (cooling power: demand and delivered, temperature, cost). As can be seen from the plots, all three make choices that cause the chillers to turn ON and OFF, and choose the corresponding power input, but the solutions differ between the controllers. As can also be seen, the provided cooling for all 3 controllers matches the load profile somewhat, though not perfectly. This can also be seen in the achieved temperature, which goes up or down when the demand isn't matched exactly (which is allowed as long as the temperature remains below the upper bound, though there will be more losses if the temperature is lower). Lastly, it can be seen that the accumulated cost of the three controllers is similar for MPC and RL, but significantly higher for the RBC.

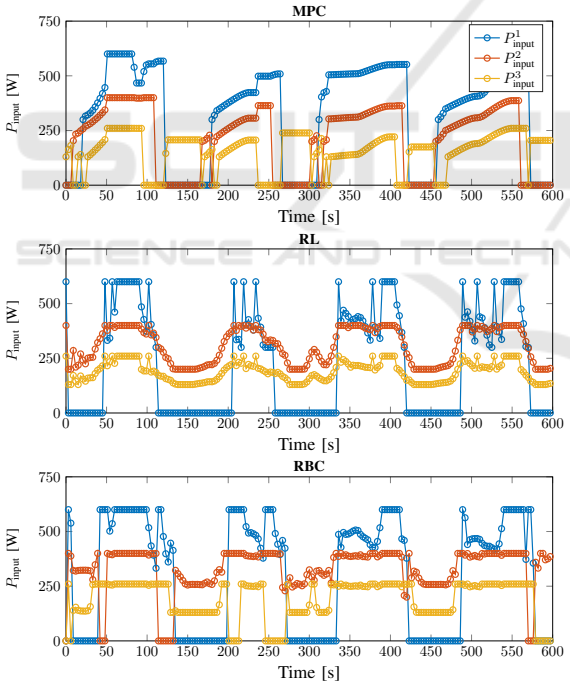

Figure 5: Outcome with each of the 3 controllers on a nominal example scenario. The plots show the electrical powers for all 3 chillers, first for MPC, then for RL, and finally for RBC.

We have also averaged the results, over both training and test set, as shown in Table 1. Similar observations can be made regarding MPC and RL significantly beating the RBC. The RL even appears to have a lower cost than MPC, but an exact comparison is

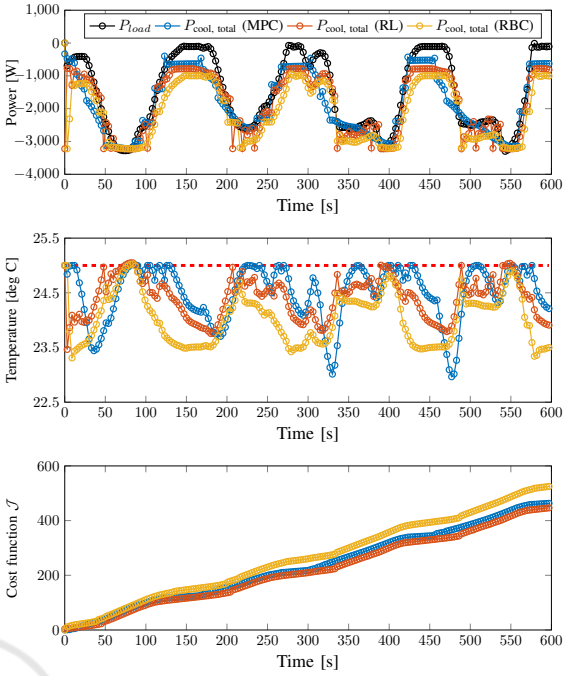

Figure 6: Outcome with each of the 3 controllers on a nominal example scenario. The plots show for all 3 controllers the resulting cooling power versus the demand, the resulting temperature (including the setpoint of 25 °C (red dashed line), and the corresponding accumulated cost.

hard since the RL did slightly violate the upper bound on the temperature (by 0.05 °C), which the MPC did not. Based on rough analysis, this is not expected to yield more than 1% of difference in energy cost, after the RL would be retrained with a small temperature constraints margin. So we conclude the RL achieves nearly the same performance as the MPC does.

The results shown for the RL are those achieved after it has learned how to control the chillers effectively. Figure 7 shows the evolution of the performance from the initial stages of the learning, until the final controller whose results were shown in the graph and the table.

Table 1: Performance for the nominal case.

Algorithm	Average cost (train set)	Average cost (test set)
MPC	454 J	450 J
RL	450 J	448 J
RBC	517 J	512 J

6.2 MPC with Shorter Predictions

In this section, we analyse what happens when the MPC is given shorter prediction windows, in this case

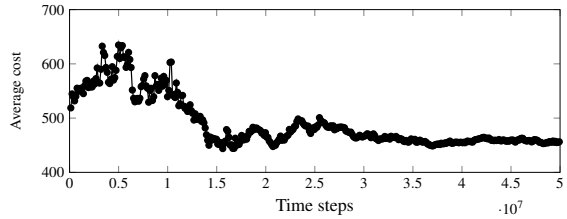

Figure 7: Improvement of the RL policy during the training process (50M time samples, 5:30 hours in simulation). The agent already learns a good policy at around 20M samples, and then further stabilizes.

4 samples (i.e., 12s, so short in comparison with the period time of load variations or the system time constant). The RBC and the RL do not use predictions, so achieve the same results in the previous section.

The results are shown in Table 2. Therein we can see that the MPC with shorter predictions has a performance that falls below that of the RL. We did not include an MPC with even shorter predictions, or none at all, since these often violate the temperature constraints, and therefore would require more extensive tuning.

This shows that the MPC needs to have good predictions, over a sufficiently long window, to be able to control optimally. And since the RL performs equivalently to the MPC controller with good predictions, it furter shows that even though the RL does not use predictions explicitly, it has learned how to implicitly anticipate to the types of profiles in the training set.

Table 2: Impact of shorter predictions.

Algorithm	Average cost (test set)
MPC (4 samples)	475 J
RL	448 J
RBC	512 J

6.3 MPC with Imperfect Predictions

In this section, the MPC is provided with predictions that overestimate or underestimate the demand by 10%, in an attempt to study real-world behavior when predictions of future demand are imperfect. Again, for the RL and the RBC the results are the same. It can be seen in Table 3 that the MPC is sensitive to these predictions, and that in case of such mismatch the RL performs better than the MPC.

6.4 RL with Demands Beyond Training

Finally, we consider the scenario where the demands encountered are different to those considered during the training. This to study the effect on the RL: not of having imperfect predictions since the RL does not use them, but of having a training set that is not complete, or not fully representative of real world behavior.

Table 3: Impact of imperfect predictions.

Algorithm	Average cost (test set)
MPC (demand over-estimated)	490 J
MPC (demand under-estimated)	519 J
RL	448 J
RBC	512 J

Here we have scaled down the demands by a factor 0.8, while the controllers are kept the same as in the previous case. This has an impact on the RL, since the patterns that are implicitly being exploited no longer correspond to those faced. The MPC on the other hand is given this new demand profile, and can therefore adjust very effectively. As a result, the RL now performs noticeably less performant than the MPC controller, as seen in Table 4.

Table 4: Impact with scaled down demands.

Algorithm	Average cost (test set)
MPC	380 J
RL	391 J
RBC	463 J

6.5 Training and Runtime

While the RL has achieved good results, it did require a learning process to do so, running 200000 episodes (40M time samples) of the simulation model. This took around 5 hours using 8 AMD Zen2 cores running at 4.5 Ghz. This is still significantly shorter than the engineering time needed to work out the approaches, set up the environments and debug the codes, for both MPC and RL. Training time is thus not yet the real bottleneck. In future cases, when more variations are considered, or model-plant mismatch, this number might increase further however, so care will need to be taken to keep this manageable.

We also evaluated the runtime needed to evaluate or execute the controllers. As shown in Table 5, the MPC required on average a calculation time of 0.50 seconds, and the RL 0.04 seconds. This is a significant advantage of using the RL over the MPC controller. Note that there are various ways to reduce the computation time of an MPC e.g., reduce its control horizon, use a move-blocking implementation, increase its sampling rate, change low-level optimizer types and settings, but these are not the focus of this work.

Table 5: Calculation times.

MPC	0.5 s
RL	0.04 s

7 DISCUSSION

From the previous, we can see that in nominal conditions RL can match the performance of a nearly-optimal MPC controller. However, RL does so without using predictions of upcoming demands (nor models), which the MPC does use. Furthermore, since the RL does better than an MPC with a reduced prediction horizons, we can see that the RL has implicitly learned the typical demand patterns, and achieves MPC-like performance without needing predictions of the future demands explicitly. This fits with the observation that when we use demands outside of the training set, the performance of the RL does degrade, as then the implicitly learned patterns are no longer correct.

For industrial usage there is then a trade-off to be made. Beyond savings in runtime calculations, using RL would have the benefit of not needing good predictions of future demands, as even with small prediction imperfections the MPC performance drops below the RL. On the other hand, this only holds when the training data for the RL is sufficiently rich to ensure that the usage patterns faced during operation fall within the implicitly learned behavior. In future work we will study this further. We will study if we can train an RL for a more diverse set of usage patterns (e.g., including rescaled versions) and initial system states (e.g., initial tank water temperatures) and how much the performance degrades when the increased variety makes it harder to find patterns to exploit. Techniques such as domain randomization (Tobin et al., 2017) might prove helpful in this case. Furthermore, investigations into possible overfitting due to fully-connected neural networks should be addressed, and possibly resolved with more sparse networks. Another interesting option for study is combining both approaches: let the RL learn patterns implicitly, but also provide it explicit predictions, though possibly indirect or imperfect.

It also has to be noted that while we focused on the impact of imperfect predictions of future demands in this paper, a practical challenge that will be faced in reality is that the models describing the system dynamics will also be imperfect. This will impact the MPC which uses them explicitly, but it will also impact the RL which uses the models to run simulations to train on. Studying the impact of imperfect models is a very interesting challenge, which we will also tackle in future work. We plan to study training RL on sets of variant models, increasing its robustness, and/or approaches wherein we allow a limited fine-tuning of the RL once deployment to the real plant occurs. Next to that, combinations of RL methods with other methods like MPC or RBC controllers will also be considered.

Beyond using RL or combining it with other methods, an alternative (yet CPU intensive) solution is to improve the MPC to better handle imperfect predictions and models, using robust, stochastic or multi-stage formulations. We did not do so or tune the MPC extensively in this paper, since our goal was to show the MPC relies on predictions, whereas the RL did not.

8 CONCLUSION

This paper compares several controllers on a simulated cooling system with 3 chillers. The proposed RL achieves a performance nearly equal to that of an MPC, but without explicit demand predictions nor model, and at a reduced computational cost. It does lose some performance when demands outside the training set are used, but the MPC loses performance when the predicted demands lose accuracy. Future work will therefore focus on making the RL robust to more diverse usage patterns, by e.g. making them rely on both implicitly learned patterns and explicit predictions. We will further study how to train RL on imperfect models, and how to ensure an efficient transfer is possible when there is mismatch between simulation environment and reality.

This paper implemented the proposed approach to a single use-case with a simulation of a scaled-down industrial usage pattern. Nevertheless, losses have been rescaled in order to make the outcomes representative for a wider range of problems: those where complex systems (multiple units, non-linear behavior, mixed integer controls) need to be controlled, while balancing short and long term cost drivers (inability to switch active units all the time requiring to look further ahead and anticipate, while considering the impact of losses as a function of the states encountered). We expect similar outcomes and trade-offs to hold for many industrial cases with similar properties, like unit commitment problems, industrial cooling, water treatment, pumping plants, heating networks, compressed air generation, etc. It is still an open question which of the methods will be more realistic when the number of controllable degrees of freedom increases, e.g. when a pumping plant with 20 pumps is faced rather than one with 3. The challenge will be to let the RL find solutions in an acceptable period of training time. On the other hand, for the MPC mostly the runtime will become a bottleneck, as the MPC with our current formulation scales quadratically in the inner loop (in

ideal case, in practice worse due to non-linearities), but combinatorially in the outer loop. However, several other formulations exist in literature, including heuristic methods, that keep it tractable up to 20+ controllable units.

BROADER IMPACT STATEMENT

Our contributions make more industrial machines controllable in an autonomous way. We mainly envision positive impacts on society, such as reduced energy consumption for the same manufacturing quality, higher manufacturing quality (less waste) and a general improved economy. The machines that would benefit from our contribution are usually not directly controlled by people, so we don't expect jobs to be lost to automation with our contribution. We however acknowledge that any improvement in automation also improves it for sensitive use, such as military equipment production.

ACKNOWLEDGMENTS

This research is done in the framework of the Flanders AI Research Program (https://www.flandersairesearch.be/en) that is financed by EWI (Economie Wetenschap & Innovatie), and Flanders Make (https://www.flandersmake.be/en), the strategic research Centre for the Manufacturing Industry, who owns the Demand Driven Use Case. The authors would like to thank everybody who contributed with any inputs to make this publication.

REFERENCES

Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.

Asadi, K., Parikh, N., Parr, R. E., Konidaris, G. D., and Littman, M. L. (2021). Deep radial-basis value functions for continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6696–6704.

Bakker, B. (2001). Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, 14.

Biegler, L. T. and Zavala, V. M. (2009). Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582.

Borase, R. P., Maghade, D., Sondkar, S., and Pawar, S. (2021). A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, 9:818–827.

Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*. Springer science & business media.

Chen, R., Lan, F., and Wang, J. (2024). Intelligent pressure switching control method for air compressor group control based on multi-agent reinforcement learning. *Journal of Intelligent & Fuzzy Systems*, 46(1):2109–2122.

De Somer, O., Soares, A., Vanthournout, K., Spiessens, F., Kuijpers, T., and Vossen, K. (2017). Using reinforcement learning for demand response of domestic hot water buffers: A real-life demonstration. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–7. IEEE.

Frank, R. J., Davey, N., and Hunt, S. P. (2001). Time series prediction and neural networks. *Journal of intelligent and robotic systems*, 31:91–103.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Glover, K. (2021). H-infinity control. In *Encyclopedia of systems and control*, pages 896–902. Springer.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

IEEE (2024). Ieee pes power grid benchmarks.

Lewis, F. L., Vrabie, D., and Syrmos, V. L. (2012). *Optimal control*. John Wiley & Sons.

López-Ibáñez, M., Prasad, T. D., and Paechter, B. (2008). Ant colony optimization for optimal control of pumps in water distribution networks. *Journal of water resources planning and management*, 134(4):337–346.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2019). Iterated local search: Framework and applications. *Handbook of metaheuristics*, pages 129–168.

Luo, J., Paduraru, C., Voicu, O., Chervonyi, Y., Munns, S., Li, J., Qian, C., Dutta, P., Davis, J. Q., Wu, N., et al. (2022). Controlling commercial cooling systems using reinforcement learning. *arXiv preprint arXiv:2211.07357*.

Mehra, R. and Davis, R. (1972). A generalized gradient method for optimal control problems with inequality constraints and singular arcs. *IEEE Transactions on Automatic Control*, 17(1):69–79.

Meinhold, R. J. and Singpurwalla, N. D. (1983). Understanding the kalman filter. *The American Statistician*, 37(2):123–127.

Michalewicz, Z., Janikow, C. Z., and Krawczyk, J. B. (1992). A modified genetic algorithm for optimal control prob-

lems. *Computers & Mathematics with Applications*, 23(12):83–94.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Monahan, G. E. (1982). State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355.

Roy, N. and Gordon, G. J. (2002). Exponential family pca for belief compression in pomdps. *Advances in Neural Information Processing Systems*, 15.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shang, W., Wang, X., Srinivas, A., Rajeswaran, A., Gao, Y., Abbeel, P., and Laskin, M. (2021). Reinforcement learning with latent flow. *Advances in Neural Information Processing Systems*, 34:22171–22183.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE.

Van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279. IEEE.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32.

Zhan, X., Xu, H., Zhang, Y., Zhu, X., Yin, H., and Zheng, Y. (2022). Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4680–4688.