Model-free control of compressors in an air network with unknown future air demand

Jeroen Willems* *jeroen.willems@flandersmake.be*
FlandersMake

Denis Steckelmacher* *denis.steckelmacher@vub.be*
Vrije Universiteit Brussel

Bruno Depraetere *bruno.depraetere@flandersmake.be*
FlandersMake

Ann Nowé *ann.nowe@vub.be*
Vrije Universiteit Brussel

Abstract

Optimal control of complex systems often requires access to an exact or almost-exact model, and information about the (future) external stimuli applied to the system (load, demand, ...). This is particularly true in the case of air networks, in which compressors have to fill an air tank, usually proportionally small compared to the production of the compressors and the average downstream demand of air. The demand of air therefore largely impacts the pressure in the tank, and the compressors have to react quickly to changes in demand. In this paper, we propose a method based on Reinforcement Learning to produce a high-quality controller for 3 compressors connected to the same air network. The Reinforcement Learning agent does not assume any model (so the compressors, tubes, losses and demand do not have to be modeled) and does not observe the future demand of air, or an approximation of it. Still, the learned controller performs comparably to a highly-tuned Model Predictive Controller, and largely outperforms MPC when even a small error exists in the predicted future demand. This demonstrates that Reinforcement Learning allows to produce high-quality controllers in challenging industrial contexts.

1 Introduction

In this paper, we consider the optimal control of a dynamical 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 $\dot{x}(t) = f\left[x(t), u(t), t\right]$. 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, an objective function (that depends on the states encountered by the system) is minimized. Instances of optimal control include moving towards and tracking a setpoint (Borase et al., 2021), tracking some (moving) reference signal, or performing a task while minimizing energy consumption. Methods to achieve optimal control range from simple to very complicated, starting from proportional-integral-derivative controllers (Borase et al., 2021), linear programming, general approaches 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 model-predictive control (Camacho and Alba, 2013). Intuitively, MPC observes the current state of the system, then uses a computational model of

it to produce the best-possible control signal, according to the model. This control signal can be produced by solving differential equations (Becker et al., 2000), linear-quadratic methods (Anderson and Moore, 2007) or Monte-Carlo methods (Kantas et al., 2009), that simulate many possible outcomes of many possible control signals and select the best one.

All these methods share a common trait: they are model-based. They assume that $f\left[x(t), u(t), t\right]$ is available, in a representation that can be used to compute control signals. This means that the system must be modelled (friction, how motors rotate, the shape of objects being moved, etc), along with any function of t used by f. In this paper, we focus on a particular example: as time passes, the users of an air distribution network turn machines on and off, leading to a time-dependent demand of air. Conventional optimal control methods need to know that time-dependent demand of air in advance, in order to optimally control the system.

However, in the real world, it is impossible to know future demands (of air, electricity, goods, ...). Methods exist to estimate these 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. Moreover, these two separate components, the estimator and the optimal control algorithm, do not work in tandem. This leads to significantly sub-optimal control signals being produced, as we show in this paper for our air distribution network use-case.

We propose to instead use another method of producing control signals: Reinforcement Learning. Reinforcement Learning is a Machine Learning family of algorithms that allows to learn a controller from experience. The important property of RL, that we otherwise introduce 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 f function.[1]

Our contributions are as follows:

- We introduce an air-distribution network use-case, based on real-world experience and with complex non-linear dynamics. Its source code is available at URL.
- We show that a well-tune Model Predictive Controller controls the system badly when even small errors are introduced in the estimation of the future demand of air. These errors are much smaller than expected in the real world, which shows that MPC cannot be used in practice on our use-case
- We formalize our use-case as a Reinforcement Learning problem, for which we precisely define the observation space, action space and reward function. We then evaluate a state-of-the-art RL algorithm on our use-case. The algorithm has no access to the future demand of air, not even an approximation of it.
- Our agent largely outperforms the MPC with an approximate future demand, and almost matches the MPC controller that has access to the future demand (the gold standard, that is impossible to match in practice). This demonstrates that Reinforcement Learning can be applied to challenging real-world control tasks for which some unobservable external signal influences the system being controlled.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning is a Machine Learning approach that allows to learn a closed-loop controller from experience with the controlled system. In the vast majority of cases, Reinforcement Learning considers a discrete-time Markov Decision Process defined by the tuple $\left\langle S, A, R, T, \mu^0, \gamma \right\rangle$, 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,

[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.

$T : S \times A \times S \to [0, 1]$ 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 $\mathcal{R}^M$.

Reinforcement Learning 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 some initial state according to μ^0, and a new episode starts. The objective of a Reinforcement Learning agent is to learn an optimal (possibly stochastic) policy $\pi(a|s)$ such as, when following that policy, the expected sum of discounted rewards $\sum_t \gamma^t R(s_t, a_t \sim \pi(s_t))$ is as high as possible.

Reinforcement Learning does not assume that the learning agent has access to the reward or transition functions. 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. Interacting, without a model, with a physical plant is possible with Reinforcement Learning.

2.1.1 POMDPs

Partially-Observable Markov Decision Processes (Monahan, 1982) 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 $\Omega : S \to O$ 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 for compressed air 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 modelled. We still observe that the "history of past observations" approach to POMDPs helps the agent in our setup, even if it does not strictly interact with a POMDP.

2.2 Reinforcement Learning Algorithms

Reinforcement Learning algorithms allow an agent to learn a (hopefully) optimal policy in a Markov Decision Process. In this article, we focus on on-line model-free Reinforcement Learning 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.

Such RL algorithms are divided in three main families: value-based, that learn how good an action is in a given state (this is called Q-Values), policy-based, that directly learn how often 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 review 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. Continuous actions, often used in industrial settings, cannot be used with value-based algorithms.

Policy-based algorithms historically started with REINFORCE (Williams, 1992), followed by Policy Gradient (Sutton et al., 1999), still the basis of almost every policy-based or actor-critic algorithms to this day. Most

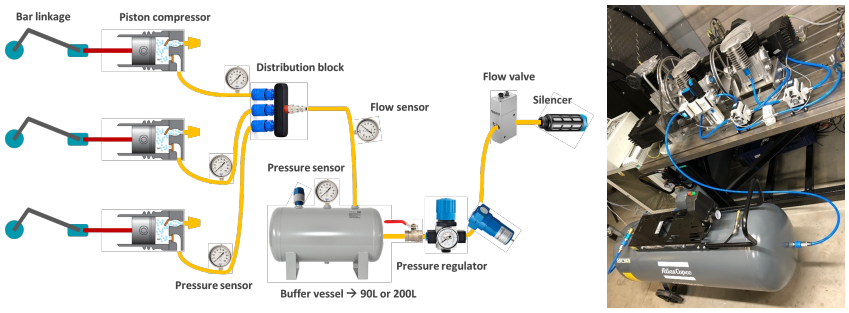

Figure 1: *Left:* graphical depiction of the simulated setup, 3 compressors fill an air tank while air is taken out of it by some varying demand. The pressure in the tank must be maintained between 3 and 5 bars. *Right:* a physical implementation of the setup for evaluation on real hardware.

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) 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 normal). A modern variant of 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 the Soft Actor-Critic in our experiments.

3 Air network setup

The particular use-case we consider in this paper is an air distribution network, with three compressors. We chose this setting because of some of its important properties:

1. This setting is highly complex, with several compressors having different efficiency points, and the air distribution network leading to non-linear dynamics.
2. The use of compressed air is wide-spread in the industry, and improvements in the efficiency of compressors control directly translates to energy and money savings for their operators.
3. Even though complex, this system can be simulated using the Matlab Simulink toolbox, leading to highly reproducible research. Our simulator, Reinforcement Learning agent and experimental scripts are available at this URL.

A simulator for the air network setup has been implemented with Simulink (MathWorks, 2022). For ease of reproducibility, all the files are available at this URL. The simulated setup consists of the following components

- A single compressor is implemented as a piston with valves, driven by a bar-linkage system (a motor driving two shafts to go from a circular to a linear motion). A sensor measures the torque exerted by the motor driving the piston. The mapping from torque to power consumption has been obtained empirically by exciting 3 real compressors and measuring their power consumption for a given torque, thus producing lookup tables used by the simulator.
- The compressor above is instanced 3 times, each time with a different LUT and with different acceptable RPM ranges (50-600, 100-400, 100-400). These 3 compressors feed a single 50-liters air tank, out of which air is extracted through a variable release valve. A safety valve is implemented

and opens whenever the pressure in the tank goes above 5 bars. Finally, losses are modelled with another variable release valve, with the losses (in kilograms per second per bar) being computed from a look-up table obtained by measuring the actual losses of a physical system.

The system accepts three control signals, the rotation speed of the three compressors in RPMs. A fourth signal, not expected to be controlled but still an input to the simulator, is the demand, expressed in kilograms per second. Only one sensor is available in the system: the air pressure in the tank.

The system is controlled once every *minute*. Every minute, the 3 RPMs and the demand flow are set. Then, Simulink runs the system and analytically produces the state of the system after one more minute, given constant RPMs and demand during that minute. One experiment lasts for *250 minutes*. For the Reinforcement Learning agent, this means that an episode lasts for 250 time-steps.

The demand is set according to a *demand curve*. A demand curve is a vector of 250 values for the demand flow. In this paper, we consider 100 distinct demand curves, available at this URL. These 100 demand curves have been generated by randomly perturbing and combining 5 segments of 50 values. For the Reinforcement Learning agent, at the start of each episode, one of the 100 demand curves is randomly selected, and used for the duration of the episode.

A physical setup matching the simulated setup has also been built, using real hardware (and off-the-shelf belt-driven compressors instead of a manually-built bar-linkage). It was used to measure the lookup tables used to map compressor torque to power consumption, and tank pressure to losses. It is however not used in our experiments, that focus on the simulated setup.

4 Reinforcement Learning Environment

Both the physical and simulated setup are modelled as MDPs with the same interface, so the same state space, action space, reward function and (assumed) transition dynamics.

4.1 Action space

The action space is continuous and consists of 3 real values, ranging from -1 to 1. It is considered best practice (Raffin et al., 2021) to have the action and state spaces be centered around 0, and of a range of as close to 1 as possible. Each of the 3 real values allows to set the target RPM of its corresponding compressor, linearly interpolated between 0 (off, for an action value of -1) to 100% (for an action value of 1).

The action space is discontinuous in two places: for every compressor, the actual effect of the action depends on the target RPM value it defines:

- Below 20 RPM: the compressor is turned off completely
- Between 20 RPM and the minimum RPM of the compressor: the target RPM is adjusted to the minimum RPM of the compressor
- Above the minimum RPM of the compressor: the action is left untouched

The effect is that the agent can produce low actions to indicate that a compressor should work at its lowest RPM, and an even lower action to indicate that the compressor should be turned off. This allows the agent to control the on/off status of the compressors, in addition to their target speed, with a single real value.

4.2 Observation space

The observation space consists of several real values that measure past tank pressures and RPMs. For every time-step, 4 real values are logged: the current pressure (in bars) in the tank, and the current RPMs of the 3 compressors. When producing observations, the environment looks N time-steps in the past to produce N real values corresponding to the tank pressures at these past N time-steps (so, this is a history of past tank

pressures). The environment also looks M time-steps in the past to produce $3M$ real values corresponding to RPMs of the compressors during these past M time-steps. N and M can be distinct, and in our experiment, we use $N = 5$ past tank pressures and $M = 5$ past RPMs.

By observing past tank pressures and RPMs, the agent gets a feel of:

- How fast is the tank depleting, which allows it to approximate the derivative of the tank pressure (that only looks at the past), thereby adjusting the future RPMs.
- What is the demand of air in the recent past (by combining how the tank pressure changes and what the RPMs were), which may help the agent guess what the demand will be in the future if there are time-specific patterns in the demand.

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 easy to implement.

4.3 Reward function

The reward function is the change in cost that occurs after a given time-step executes (so, after 60 seconds of simulated time after an action has been applied to the physical or simulated system). More specifically, the reward given after a time-step consists of two components:

1. Minus the amount of kilojoules consumed during the time-step. In the simulated setup, the power consumption of the compressors is approximated in a lookup table (in watts). In simulation, we assume that a compressor instantly reaches its target RPM and its corresponding power consumption, so the amount of kilojoules is $60s \times P \times 0.001$, with P the power (in watts) obtained from the lookup table.
2. A turn on penalty when a compressor turns on. In addition to the point above (the power consumption during the whole time-step), a compressor that goes from off to on at the start of a time-step is considered to incur a cost equal to its maximum (max RPM) power consumption during 60 seconds. For instance, if the third compressor powers on, a reward of $-60s \times 2000W \times 0.001 = -120kJ$ is given to the agent.

5 Baselines

We use as baselines Model-Predictive Controllers. We also implemented a rule-based system (manual conditions on the tank pressures and past RPMs to produce new RPMs), but that system performs poorly even after much tuning, so we omit it in the interest of space.

The MPC is implemented as follows:

1. Two instances of the simulator exist, the one used for execution, and the one used to compute the optimal control signal.
2. Every time-step, the state of the *execution* simulator is copied to the *planning* one. The demand profile (all the future demand flows) of the planning simulator is set to the current and all future demand flows of the execution simulator, followed by zeros (to pad that vector to 250 values).
3. Then, the optimal control is computed in the planning simulator using the IPOpt solver (Biegler and Zavala, 2009), under the constraints that the tank pressure does not fall below 3 bars, and that the compressors are operated in their RPM range. The planning horizon is until the end of the episode, which produces RPMs to apply to the compressors at the current time-step and all the future ones, in addition to the total cost in power until the end of the episode. This is done 8 times, for all

combinations of compressors being on or fully off (zero RPMs are allowed). Then, the combination having the lowest cost is selected, and its RPMs for the current time-step is applied to the *execution* simulator.

4. The execution simulator runs for a single time-step (one minute), then we go back to Point 2.

We observe that two aspects of the MPC controller above are somewhat unrealistic and could potentially be lifted: the fact that the planning and execution simulators are exactly the same (it could be that the execution platform is not exactly modelled by the planning simulator), and the fact that the true future demand curve of the execution simulator is copied to the planning one. We therefore have 4 MPC baselines in this paper:

MPC baseline	Planning = Execution?	Exact future demand?
MPC	✓	✓
MPC no model		✓
MPC no demand	✓	
MPC no model no demand		

5.1 Perturbing the model

The *MPC no model* baselines are produced by making the planning model, used to compute compressor RPMs, slightly different than the execution model, in which the RPMs are actually applied. As part of the model are 2 look-up tables per compressor that matter for the simulation: one that maps RPMs and tank pressures to the compressor's output flow in kilograms per second, and one that maps RPMs and tank pressures to the power consumption of the compressor in watts.

The planning model is perturbed by multiplying all the values in these look-up tables by some number between 1 and 1.5, with an average of 1.25. A single but distinct number is used for every LUT (6 in total, 2 per compressor). The numbers are available in the experiment scripts at this URL, and intuitively translate to making the planning compressors a bit more powerful, or consume a bit more power, than the execution compressors. This represents a coherent (same number for the whole LUT) modelling error of about 25%.

5.2 Perturbing the demand profile

The *MPC no demand* baselines are produced by perturbing the demand profile (list of demand flows for the future time-steps) observed by IPOpt when planning. Because the demand profile intervenes in the computation of the state of the system, it cannot simply be removed. Setting it to all zeroes would work from a software perspective, but would not make sense in the real world (usually, the future demand can at least be somewhat guessed).

In this paper, we instead perturb the demand profile by taking the true demand profile (that will be used in the execution model), and multiply them by $1 + x \sim \mathcal{N}(0, 0.1)$. It is therefore multiplicative noise of average magnitude of 1.1. This represents an average prediction error of 10% on the demand curve. We consider this perturbation as being very small, especially since the normal is centered around zero, leading to no perturbation on average (no bias in the demand profile used for planning). We also expect that any real-world demand estimation mechanism would have a prediction error above 10%.

6 Results

We compare our 4 baselines against a Soft Actor-Critic (Haarnoja et al., 2018) agent obtained through 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: a learning rate of 0.001, a batch size of 512, learning starts after 250 time-steps, gamma is set to 0.98, and the entropy coefficient is set to 0, to allow the agent to learn an almost-deterministic policy (necessary to achieve low power consumption). The Reinforcement Learning agent was allowed to learn in our simulated setup for 2 million time-steps (but a high-quality controller was already learned 400K time-steps as shown in Figure 2).

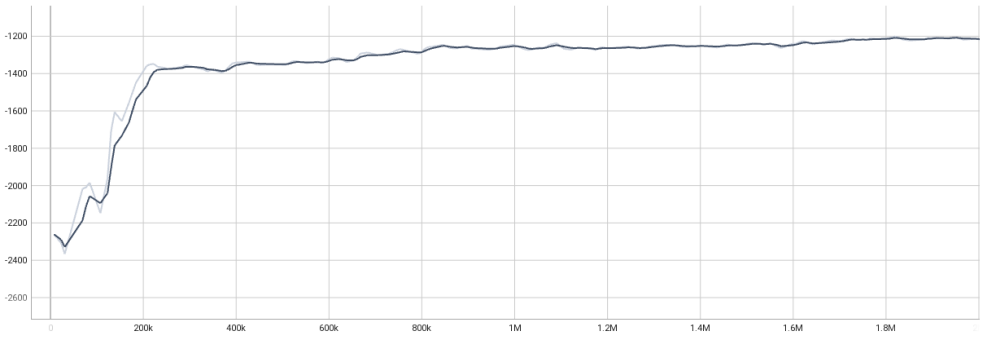

Figure 2: Learning curve for the Soft Actor-Critic Reinforcement Learning agent. The horizontal axis is simulated time-steps (simulated minutes), for a total of 2 million time-steps, or approximately one day of run-time on a cheap laptop. The vertical axis is the reward obtained by the agent per episode, equivalent to minus the average power consumption of the system (in kW). The agent quickly learns a decent controller, and then takes more time to fine-tune it, learning the intricacies of the unobserved demand profiles and how to estimate them.

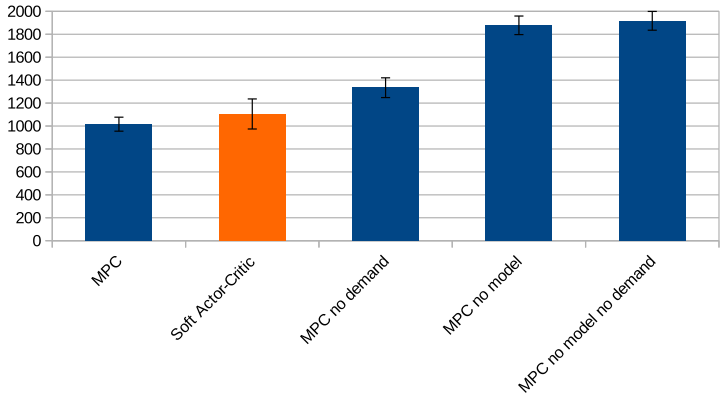

Figure 3: Graphical comparison of the average power consumptions of our 4 baselines, and the proposed Soft Actor-Critic Reinforcement Learning agent. The SAC agent is slightly less optimal than the MPC controllers that observe the true future demand of air (unrealistic), and significantly outperforms the MPC controller that observes a perturbed future demand of air (more realistic). The RL agent has no model of the system and does not observe any future demand at all (fully realistic).

The RL agent and baselines are evaluated on the simulated setup (the *execution* model of MPC) for all 100 demand profiles (see the end of Section 3). The average power consumption over 250 time-steps (entire episodes), in kW, averaged across all 100 demand profiles, is used to measure the performance of the model (lower is better). The standard deviation is computed across the 100 demand profiles. The results are as follows, and are graphically represented in Figure 3.

Algorithm	Average cost (lower is better)
MPC	1016 ± 61
MPC no model	1878 ± 81
MPC no demand	1334 ± 86
MPC no model no demand	1917 ± 82
Soft Actor-Critic RL	1155 ± 131

Our observations are as follows:

- Having access to the exact (planning) model of the (execution) environment, but only observing an approximation of the future demand profile, already reduces the performance of *MPC no demand* 31% compared to *MPC* (with the exact model and exact future demand profile).
- Not having access to the model of the environment largely impairs the performance of MPC in our task, with optimality drops of 85% and 87% with or without access to the exact demand profile, respectively.
- The Soft Actor-Critic does not observe the demand profile (not even a perturbed version of it), and does not use a planning model. It is fully applicable and deployable in the real world. This comes at 14% cost in optimality compared to the perfect MPC planning approach.

By comparing numbers, we can draw our main conclusion. If the future demand profile of an air network system is not observable, two approaches are possible: either we estimate the future demand and use a classical approach such as an MPC, leading to a reduction in optimality of 31%, or we use Reinforcement Learning, leading to a smaller reduction in optimality of 14%. As such, we demonstrate that Reinforcement Learning is not only applicable to a challenging control problem, but that it is also beneficial compared to classical approaches, when the future demand of air is unknown.

If we additionally relax the assumption that an exact model of the system to control is available, then MPC approaches lead to a reduction in optimality of 85% at best, while the Reinforcement Learning approach is only 14% less optimal than the MPC with an exact model.

7 Conclusion

This paper introduces a simulated air network setup, in which 3 compressors have to fill an air vessel while air is extracted from it according to a demand profile. When an oracle provides the future demand profile for the entire duration of an experiment, a model-predictive controller is able to optimally control the system. However, when the future demand profile is not available, as is usually the case in the real world, the MPC quickly degrades. We propose to instead train a Reinforcement Learning agent, that is able to control the system 15% more efficiently than the MPC when the future demand is not known. Additionally, the RL agent does not need a model of the system. It controls the system 67% more efficiently than an MPC that has access to an approximate model of the system. This illustrates, on a realistic industrial task, that Reinforcement Learning allows to lift some assumptions (knowing or approximating the future demand) to increase the range of systems for which high-quality controllers can be produced.

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 has been funded by the Flanders AI Research Program.

References

Anderson, B. D. and Moore, J. B. (2007). *Optimal control: linear quadratic methods*. Courier Corporation.

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

Becker, R., Kapp, H., and Rannacher, R. (2000). Adaptive finite element methods for optimal control of partial differential equations: Basic concept. *SIAM Journal on Control and Optimization*, 39(1):113–132.

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.

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.

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.

Kantas, N., Maciejowski, J., and Lecchini-Visintini, A. (2009). Sequential monte carlo for model predictive control. *Nonlinear model predictive control: Towards new challenging applications*, pages 263–273.

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.

MathWorks (2022). Simulink: Simulation and model-based design.

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 problems. *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.

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.