LUND UNIVERSITY

MASTER THESIS

Reinforcement Learning in Industrial Applications

Author:
Niklas KOTARSKY
Eric BERGVALL

Supervisor:
Dr. Johan GRÖNQVIST
Examiner:
Prof. Bo Bernhardsson

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Engineering, Engineering Physics

in the

Intersection of Reinforcement Learning and Automatic Control
Department of Automatic Control

LUNDS UNIVERSITET

October 14, 2020

MSc Thesis
TFRT-6111
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2020

Declaration of Authorship

We, Niklas Kotarsky, Eric Bergvall, declare that this thesis titled, "Reinforcement Learning in Industrial Applications" and the work presented in it are our own. We confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourself jointly with others, we have made clear exactly what was done by others and what we have contributed ourself.

Signed: Eric Bergvall, Niklas Kotarsky

Date: 23/9-2020

"People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world."

Pedro Domingos

LUND UNIVERSITY

Abstract

Automatic Control
Department of Automatic Control

Master of Science in Engineering, Engineering Physics

Reinforcement Learning in Industrial Applications

by Niklas KOTARSKY
Eric BERGVALL

Although reinforcement learning has gained great success in computer games, there are only few yet known implementations in industrial applications. This despite the fact that reinforcement learning offers interesting methods to optimise the control of nonlinear processes. In this thesis we have used two model free reinforcement learning algorithms (PPO and DDPG) to control three different simulations of industrial processes, the simplified Tennessee Eastman, original Tennessee Eastman and the Haldex brake valve. Both reinforcement learning algorithms could in almost all cases learn to reach a set point. In addition, hyperparameters were found to have a high impact on training performance. In conclusion, our tests indicate that the model free reinforcement learning algorithms are basically capable of controlling industrial processes. Python code for the PPO algorithm applied to the Original Tennesse Eastman process can be found at Github. [1]

[1] https://github.com/Heigke/Reinforcement-Learning-In-Industrial-Applications

Acknowledgements

Our sincerest thanks and well wishes to,
Johan Grönqvist at the Automatic Control department at Lund University for the many hours of advice and discussions,
Johan Roos, Adrian Sahlman and Johan Ullén at Sentian.AI for their never ending stream of curiosity and help in the facing of new challenges,
Pontus Fyhr and Edo Drenth at Haldex Brake Products AB for letting us explore the world of RL with the help of their model.

Statement of Work

In this thesis Niklas Kotarsky and Eric Bergvall have together investigated model free reinforcement learning on industrial processes. Although the work has been done as a team it shall be clearly stated that Niklas Kotarsky has worked with the DDPG algorithm and Eric Bergvall with the PPO algorithm. The STE process was implemented by Sentian.AI, the Haldex brake was created by Haldex Brake Products AB, the OTE process was created by J. J. DOWNS and E. F. VOGEL. The wrapping of the OTE process was to a big extent done by Niklas Kotarsky and Johan Grönqvist. The wrapping of the Brake, so that it behaved like the OpenAI gym, was done by Niklas Kotarsky.

During a major revision of the thesis, done by Niklas Kotarsky during an intensive week, the following was rewritten.

Eric Bergvall's original formulation of the abstract was rewritten to explain what has been done, what we have done and the conclusions. In chapter 1, introduction sections written by Eric Bergvall about our perspective of reinforcement learning (section 1.1.1), the reinforcement learning today (section 1.1.2) and in the future (sections 1.1.3) was distilled into a compact introduction (section 1.1). The technical introduction to model free reinforcement learning (section 1.2.1) was jointly written by Eric Bergvall and Niklas Kotarsky at first but Eric's contribution of On-Off-policy RL was removed at the revision since this difference is not discussed, to any great length, in the thesis. Section 1.3 The different environments was originally written by Eric Bergvall but the detailed descriptions of the environments were reduced to the existing compact explanation. This also due to the fact that environment details are discussed to a very small extent in the thesis. Section 1.4 Quantitative Problem Formulation was to a beginning written by Eric Bergvall but was rewritten by Niklas Kotarsky during the revision. In chapter 2 the section 2.2 Method was first written by Eric Bergvall and was rewritten by Niklas Kotarsky. The introduction to Chapter 3 LQP Environment was written by Eric Bergvall but removed to make the thesis easier to read. Two sections 3.1.2 and 3.1.3 with the titles "Can the algorithms handle larger systems?" and "Can the algorithms handle oscillating systems?" were written by Eric Bergvall but were removed due to their sparse information. In chapter 4 Haldex Brake, section 4.3 Discussion was originally written by Eric Bergvall but was rewritten by Niklas Kotarsky. In chapter 6 Original Tennessee Eastman chemical plant the introduction was in the beginning written by Eric Bergvall but later revised by Niklas Kotarsky. Chapter 8 Discussion from an Industrial Perspective was jointly written by Eric Bergvall and Niklas Kotarsky and later revised by Niklas Kotarsky. Also, chapter 9 Conclusion was at the beginning written by Eric Bergvall and Niklas Kotarsky together and later rewritten by Niklas Kotarsky.

Thus the sections 1.2.3, 1.2.4, 2.1, 3.1, 5.2, 6.1 and 7.2 were solely written by Eric Bergvall and the rest was written or rewritten by Niklas Kotarsky. The implementation details of the PPO algorithm can be found in section 1.2.4 and for the DDPG algorithm it can be found in section 1.2.7.

Contents

List of Abbreviations

STE	**S**implified **T**ennessee **E**astman
OTE	**O**riginal **T**ennessee **E**astman
LQR	**L**inear **Q**uadratic **R**egulator
PPO	**P**roximal **P**olicy **O**ptimisation
DDPG	**D**eep **D**eterministic **P**olicy **G**radient
RL	**R**einforcement **L**earning

To infinity and beyond!...

Chapter 1

Introduction

1.1 Background and Motivation

Model free reinforcement learning has lately achieved great success by learning how to play video games. Thus, algorithms can achieve higher scores than humans under the same conditions. These algorithms are also often based on deep neural networks, which successfully map the video game frames to actions. This makes these methods promising candidates for controlling highly nonlinear tasks. These tasks often propose great challenges in the field of control theory. Despite these facts, reinforcement learning has not yet seen widespread implementation in the industry. The reason for that might be a yet unknown behaviour of the algorithms in real industrial settings. Therefore it is of importance to test the algorithms on problems which are as close to the real industrial processes as possible.

In this thesis we have studied how model free reinforcement learning algorithms behave in three different simulations of industrial processes.

1.1.1 General Problem Formulation

In this thesis we have investigated how RL algorithms can handle complex industrial processes. In the beginning, the RL algorithms were tested on the Linear Quadratic Problem (LQP) environment. Thereafter, the complexity was increased gradually to see the capabilities and failure modes of our chosen algorithms.

The next step was to implement the algorithms in more complex environments like the Simplified Tennesee Eastman process (STE) implemented by Sentian.AI, Original Tennesee Eastman challenge (OTE), and the Haldex brake valve. These models are described in more detail in section 1.3.

The hypothesis of this thesis was that:

Hypothesis 1 *Model-free Reinforcement Learning algorithms are able to learn from an industrial application process just like in the previous successes of Reinforcement Learning in video games.*

In order to test this hypothesis we wanted to answer the, following general questions:

Question 1 *How can model free Reinforcement Learning algorithms manage complex processes in an increasing complexity ladder represented by LQP→STE→Haldex Brake Valve→OTE?*

Question 2 *If the Reinforcement Learning algorithms succeed, what might be the key characteristics of the architecture or interface with it that make it achieve its goals?*

Question 3 *If the Reinforcement Learning algorithms fail, what might be the key characteristics of the architecture that cause the problem?*

Question 4 *Is model-free Reinforcement Learning, by itself, suitable for industrial applications?*

After initial experimentation and results we adjusted the original problem formulation see 1.4.

1.1.2 Previous Work

Before diving into the basic theory of reinforcement learning we take a look at what already has been done in this discipline. An excellent introduction to the perspective we try to take on in this thesis is given by Steven Spielberg, 2020. He goes through the impressive work done by previous automatic control researchers in their quest to identify and update the control of complex industrial processes, both discrete and continuous. In a classical setting, some critical problems arise when one need to reidentify the system for a process. This is because one often need to stop the process and also expose it to external excitations.

A comparison between the fundamentals of model predictive control and reinforcement learning is presented in the paper by Joohyun Shin, 2019 in section 3 table 1. This might be of interest if one comes from a classical automatic control perspective and want to know the pitfalls and strengths of the two methods.

Furthermore if one wants to know how (online) adaptive control strategies inspire by reinforcement learning to find optimal solutions the paper by Lewis, Vrabie, and Vamvoudakis, 2012 is most rewarding.

There are many attempts to close the gap between reinforcement learning and classical automatic control with the vision to learn from each other's disciplines but there is still some way to go, hence we write this thesis.

1.2 The Basic of model free Reinforcement Learning

1.2.1 Technical introduction to model free reinforcement learning

In the most general setting of reinforcement learning, we have an agent that interacts with its environment. The environment changes when the agent performs an action and also provides the agent with an observation and a reward as feedback. Loosely speaking, the reward measures how good the action given the state was. The agent's goal is to maximise the cumulative reward.

The agent's behavior is summarised in the policy, which is a function that maps the environment states to actions. It might be most natural to have a deterministic policy, meaning that each state is mapped to an action. There are also many RL algorithms that use a policy that returns a conditional probability over the actions given the state. From this distribution, one can sample an action to mimic exploration or simply just use expectation. There are multiple reasons why one would want to do this. One such reason is the so called "Policy gradients" algorithm.

1.2.2 Policy Gradients

In order to have the agent learn from it's past experience in the environment we need to be able to maximise the expected reward 1.1 over the game with respect to some set of Policies Π.

$$J_\pi = \mathbb{E}[\sum_{t=0}^{t_{end}} r(s_t, a_t)] \text{ s.t.} a_t = \pi(s_t); s_{t+1} = Env(s_t, a_t) \tag{1.1}$$

The simplest way of maximising the expected reward 1.1 would be to use stochastic gradient descent, but this requires us to take the derivative of the environment and the reward function. Both of these derivatives are often unknown or intractable to calculate. Policy gradients circumvents this problem by using a stochastic policy combined with a few clever tricks. First of all using stochastic policy lets us define the probability of getting a certain trajectory τ if we follow the policy π. With these definitions it is possible to derive the policy gradient to be 1.3 as done in *An Outsider's Tour of Reinforcement Learning*. Were τ_t denotes the trajectory up to time t.

$$log(\pi(\tau)) = \sum_{t=0}^{t_{end}} log(\pi(a_t|s_t)) \tag{1.2}$$

$$J_{pg} = \sum_{t=0}^{t_{end}} log(\pi(a_t|s_t))R(a_t, s_t|\tau_t) \tag{1.3}$$

1.2.3 Trust Region and Approximations thereof

A policy gradient step can easily result in an accidental step which decreases performance in the objective-function-landscape. Therefore, to bound the step size a method called trust region was developed. The trust region gives a maximum step size allowed. The trust region creates this by a measurement of change of action distributions. After that the update chooses an optimal step within this region. This is done by maximising the advantage subject to the trust region. In this optimisation there appears a second order matrix derivative called the Hessian matrix, and its inverse which are computationally expensive. The optimisation is instead approximated with soft constraints in the PPO algorithm which is described below. The soft constraint could be a clipping in the objective function which we will go through more thoroughly in section 1.2.4.

1.2.4 Proximal Policy Optimisation

The PPO algorithm works in the following fashion, say the algorithm has taken an action which has resulted in a better discounted reward than it expected with the value function. Then the update of the network simply increases the probability of sampling that action. However, the update is limited in such a way that it does not get to over-optimistic. The same thing goes for the opposite direction, if the PPO has taken an action which has resulted in a worse discounted reward than it expected then it decreases this probability. The update is not too over-pessimistic so the update is limited. The objective function which should be maximized is given in equation 1.4.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{1.4}$$

Beginning from the outer part of the equation the $\hat{\mathbb{E}}_t$ means the update will take a mean over a batch of actions. The $r_t(\theta)$ is described in equation 1.5 where θ refers to

the neural networks parameters.

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \tag{1.5}$$

This fraction corresponds to the ratio of probability between current policy and old policy for the action the PPO has taken. PPO is based on the policy gradient described above. The update then needs to be according to the current policy, in other words, on-policy because the update is conditioned on the current policy. In a practical sense, when implementing PPO, mini-batches are used. Therefore the update also compensates for the fact that the current policy changes after every update. That is why the $r_t(\theta)$ appears in equation 1.4. It compensates for actions that seem to come from another distribution than the current one. The ratio in equation 1.5, the denominator becomes big if the action has a high probability for the old policy, and the nominator becomes small if the action is unlikely for the new policy and thus the fraction becomes small, weighting this action less in the update. So the PPO algorithm does a small amount of off-policy updates but tries to compensate for this by reducing the size of updates which are more off-policy.

The advantage, $\hat{A}_t$ in equation 1.4 is basically the difference between how much discounted reward the PPO got and how much it thought it would get, given by the value function, see equation 1.6.

$$\hat{A}_t := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k - V\left(s_t\right) \tag{1.6}$$

R_k is the reward at time step k and the γ is a discount factor. The ϵ in equation 1.4 is a hyperparameter which was set to 0.2 in this thesis.

Figure 1.1 shows the clipping of the objective function in the PPO. In equation 1.4 this is refered to as $clip$, an operator which limits the function to the given limits. As described in the section about trust regions, the approximation of the update limitation is done by applying a soft constraint. The soft constraint which plateau the objective function limits the gradient in the update of the policy networks weights.

To make sure that the PPO algorithm keeps exploring, an entropy term is intro-

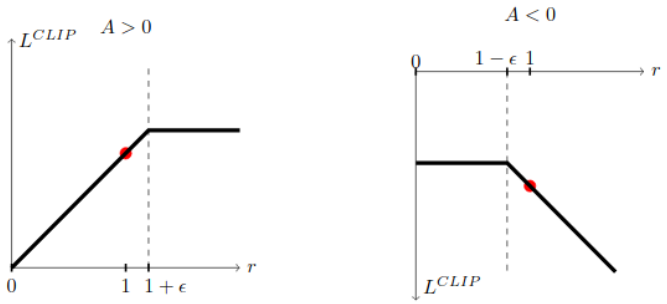


FIGURE 1.1: A positive advantage for the PPO means that it went better than the value function expected and a negative advantage means it went worse than the value function expected. We see how the gradient of the objective function is clipped when moving too much in the desired direction (upwards). The figure is taken from an article by John Schulman, 2017

duced in the objective function. Recall that entropy is just a negative sum of probabilities times the logarithm of the probability which in layman terms is a metric of "information", "surprise" or "uncertainty" as explained by Markowsky, 2017. More specifically the entropy $h(x)$ for a density function $f(x)$ with support X can be written as equation 1.7.

$$h(X) = -\int_{\mathcal{X}} f(x) \log f(x) dx \tag{1.7}$$

So the objective function becomes as follows, see equation 1.8, where α is a hyperparameter mostly set to around 10^{-4}.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t + \alpha \cdot Entropy \right) \right] \tag{1.8}$$

The implementation of this algorithm was to a big extent done by Goodger, 2020, which we used. The alterations made were mostly concerning the application of the algorithm to new environments and the hyperparameters were, to a large extent, kept as in Nikolaj's setting for "BipedalWalkerHardcore-v2".

An interesting note in the case of doing a novel implementation is that Nikolaj's implementation uses something called a recurrent sequence. The recurrent sequence is a fixed amount of timesteps that the update roll through the recurrent networks before correcting its action with backpropagation. This means it only corrects the last action in the so-called recurrent sequence. So when updating the PPO algorithm it only corrects the action at certain increments in time and not every time step.

1.2.5 Bellman's Equation and Q learning

Policy gradients are just one solution to the problem that the environment and the reward function in general are non differentiable. Another way to solve this problem would be to find an approximation to the Q function, which maps every state and action pair to their expected discounted reward 1.9. This effectively estimates how good an action is. The equation that defines the Q function is often referred to as the Bellman equation. If the Q function is given then the task of maximising the reward turns into a maximisation over the actions space instead.

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1}|s_t,a_t}\left[r(s_t, a_t, s_{t+1}) + \gamma max_{a \in A} Q(s_{t+1}, a)\right] \tag{1.9}$$

In most RL settings the Q function is not known, but there are cases were it is easy to estimate. This estimation is normally done through minimising the temporal difference error of the bellman equation 1.10, were $\hat{Q}$ is the estimate of the Q function.

$$TD_{error} = \left(\hat{Q}(s,a) - \mathbb{E}\left[r(s,a) + \sum_{s' \in S} P(s'|s,a)\gamma max_{a \in A}(\hat{Q}(s',a)) \right] \right)^2 \tag{1.10}$$

The function approximators used in Q learning are often neural networks. This gives rise to a well known failure mode referred to as "Catastrophic value approximation", which also affects some of Q learning's extensions. This failure mode arises due to that the neural networks learn unrealistically high estimations for seldom or unseen state and action pair.

1.2.6 Deep Deterministic Policy Gradient

The algorithm deep deterministic policy gradient (DDPG) can best be explained as a mixture of the actor critic algorithm and deep Q learning Timothy P. Lillicrap, 2015. Introduction to the corresponding algorithm is probably easiest from the perspective of deep Q-learning. In deep Q learning, the policy is usually defined as the maximum over the action space. If the action space is continuous this maximisation turns into a costly non-convex optimisation. To reduce the computational cost, DDPG uses a policy network π instead, which task it is to maximise the Q-network 1.11 over the actions given the state.

$$L_\pi = -\frac{1}{N}\sum_{t=0}^{N} Q(s_t, \pi(s_t)) \tag{1.11}$$

The loss term for the Q-network is very similar to the loss term of deep Q learning. Like in deep Q learning, target networks are used in order to stabilise the algorithm. A target networks is defined as the geometric mean of the normal network. The big difference is that the policy network is used instead of the maximum operator. This gives the loss for the Q-function as 1.12.

$$L_Q = \frac{1}{N}\sum_{t=0}^{N}(Q(s_t, a_t) - r_t - \gamma Q_{target}(s_{t+1}, \pi_{target}(s_{t+1})))^2 \tag{1.12}$$

To keep the training behaviour as similar as possible to deep Q learning the policy network should be trained to convergence for every Q network update. This training scheme is incredibly inefficient, since the Q network is untrained at the start of training. To improve training times both the policy and the Q network are updated every training iteration.

To increase training efficiency further the DDPG algorithm is normally also trained on previously collected trajectories. This is achieved by storing all collected trajectories in a replay buffer. Furthermore it is also possible to add trajectories from other controllers to the replay buffer, which raises the question if trajectories generated from other types of controllers or randomly generated trajectories could speed up the algorithms convergence. This problem was studied in Mel Vecerik, 2017, which train the DDPG algorithm to control a robot arm to perform different tasks such as inserting a cable into a connector. They show that the training times can be shortened by adding successfully completed trajectories to the replay buffer.

To get an adequate amount of exploration in the environment the actions are also often purposefully perturbed by noise. The original authors of the DDPG paper Timothy P. Lillicrap, 2015 recommended noise generated by the Ohrnstein Uhlenbeck process, but Gaussian distributed noise has also been proven to work, *Open AI spinning up*.

The discrete Ohrnstein Uhlenbeck process (OU) is an AR(1) process that always decays towards its mean. This gives the update equation for the next noise term x_t as 1.13.

$$a_{t+1} = (1-)a_t + \mu + \sigma e_t \tag{1.13}$$

Were is one minus the decay speed, μ is the mean, σ the standard deviation and e_t N(0,I) distributed parameter. The noise a_t is then added as a disturbance to the policy's actions.

1.2.7 Implementation details of the DDPG algorithm in this thesis

In this study, the DDPG algorithm was implemented stepwise and the multiple modifications are described in the section below. The different algorithms are all implemented in Python with the machine learning library Pytorch. The first DDPG implementation without any extensions is largely based on the GitHub user Higgsfield's implementation, *DDPG implementation*. This code was then modified to be able to solve the partially observable case with a prioritised replay buffer. To handle the partially observable case, recurrent neural networks were included in both the policy and the value networks. Due to the fact that we could not find any good authoritative best practices for how to train the neural networks in the partially observable case we implemented three different variants.

The first variant splits each trajectory into multiple smaller training examples. The algorithm is then trained on all time steps in the training example. We also included a sequence of burn in steps in the beginning of each training example to allow the hidden states of the policy and Q networks to converge, as suggested by Steven Kapturowski, 2019. We also used the old hidden states of the policy and Q network for initialisation during training, in a similar fashion as Steven Kapturowski, 2019.

We also implemented a training scheme were training samples were independently drawn from the replay buffer. The simplest way to implement this is to sample the training points from the replay buffer. To be able to back propagate the hidden states they were rolled up from the beginning of the trajectory. This forces the algorithm to iterate through the batch as well, since the different samples in each batch all have different roll up lengths.

The last update method we created is a combination of the previous two. It still only trains on one time step at a time, but utilises a fixed length burn in and roll up lengths.

The last implementation detail to be discussed is the exploration method. We designed 3 different noise types for exploration. The probably simplest was the normal distributed noise. To make this exploration strategy a little less susceptible to a bad standard deviation the standard deviation was randomised in a reasonable region.

We also used noise generated from the Ohrnstein Uhlenbeck process. To minimise the risk of bad hyperparameter tuning the different parameters of the noise were distributed uniformly . Another reason why the standard deviation was randomised is that some DDPG algorithms decay the standard deviation during training. We did not implement this, since it greatly interfered with Python's multiprocessing.

Finally, we also used an AR(2) process of the form 1.2.7, were a is bounded between 0-1, σ is the standard deviation, e_t is a N(0,I) randomly distributed variable and x_t the noise that is added to the policy's action.

$$x_{t+1} = (1-a)x_t + ax_{t-1} + \sigma e_t \quad (1.14)$$

1.3 The different environments

In this section we introduce the different process environments used in this master thesis. The following three environments have been investigated: Haldex brake, Simplified Tennessee eastman and the original Tenessee eastman. Every environment used has been wrapped in classes to represent an OpenAI gym environment as much as possible. Thus, every environment gets a step function that takes the policy's actions as input and advances the process to the next time step. Finally, the step function returns the observation, reward and if the environment has terminated. The wrapper classes also have a reset function thereby restarting the environment, see documentation of the *OpenAI gym*.

To test our algorithms, we start off with a toy environment. This environment is based on the Linear Quadratic Regulator (LQR), but with some extensions which we refer to as Linear Quadratic Problem (LQP). The environments allow us to vary the degree of complexity on the problem, we can for example vary the number of inputs and outputs as well as the degree of nonlinearity. After that, we will go through a more complex industrial process called "Simplified Tennessee Eastman" (STE) which is a scaled down version of the "Original Tennessee Eastman" (OTE) chemical plant.

1.3.1 Linear Quadratic Problem

The LQR and Linear Quadratic Gaussian-Control (LQG) problems are famous in the automatic control community and act as good jump-off point for understanding the RL algorithms. They are based around a linear system 1.15. Note that this equation actually is a LQG formulation because of the partial observability induced by C and the added noise ϵ. Our LQR-with-extension environment will further on be referred to as LQP.

$$\begin{cases} x_{k+1} = Ax_k + Bu_k \\ y_{k+1} = Cx_{k+1} + \epsilon \\ J_{k+1} = x_k^T Q x_k + u_k^T R u_k + 2x_k^T N u_k \end{cases} \tag{1.15}$$

Were the system matrix is called A, the inputs u, the internal states x and the observables y. Furthermore the reward J is the quadratic distance to origin of the input and states weighted by the matrices Q, R and N.

This linear system can be minimized analytically by the Riccati difference equation if the variables A, B, C, Q, R, N and x_n are known, as shown in equation 1.16. Note that the formulation we have used is only valid for a fully observable problem.

$$\begin{cases} \mathbf{u}_i = -K_i \mathbf{x}_i \\ K_i = \left(B_i^{\mathrm{T}} S_{i+1} B + R_i\right)^{-1} B^{\mathrm{T}} S_{i+1} A_i \\ S_i = A^{\mathrm{T}} \left(S_{i+1} - S_{i+1} B \left(B^{\mathrm{T}} S_{i+1} B + R_i\right)^{-1} B^{\mathrm{T}} S_{i+1}\right) A + Q_i, \quad S_N = Q \end{cases} \tag{1.16}$$

To allow for the study of set point changes the x-term in the reward function J as shown below in equation 1.17 was modified. In equaton 1.17 the desired setpoint at timestep k is s_k.

$$x_k^T Q x_k \rightarrow (x_k - s_k)^T Q (x_k - s_k) \tag{1.17}$$

The following method was used to generate system matrices, see equation 1.18.

$$A_{oscillate} = e^{A_{random} - A_{random}^T} \tag{1.18}$$

The matrix A_{random} is created with normally distributed entries. This ensures that the system matrix is unitary, thus keeping the energy in the system constant.

1.3.2 Simplified Tennessee Eastman

Both the Original- and the simplified Tennessee Eastman were created as benchmarks for model predictive control proposed in the articles DOWNS and VOGEL, 1992 and Ricker, 1993. The original Tennessee Eastman is based on processes in a typical chemical plant. Downs and Vogel proposed it as a challenge to create a controller without looking at either the code or the internal states. Ricker later developed the simplified TE, a simplified process that is still difficult to control with model predictive control.

The STE process is an irreversible chemical reaction in a pressure vessel. More precisely, chemicals A and C, in vapour phase, become liquid D, ($A + C \rightarrow D$). Below is a schematic sketch of the STE process, see figure 1.2.

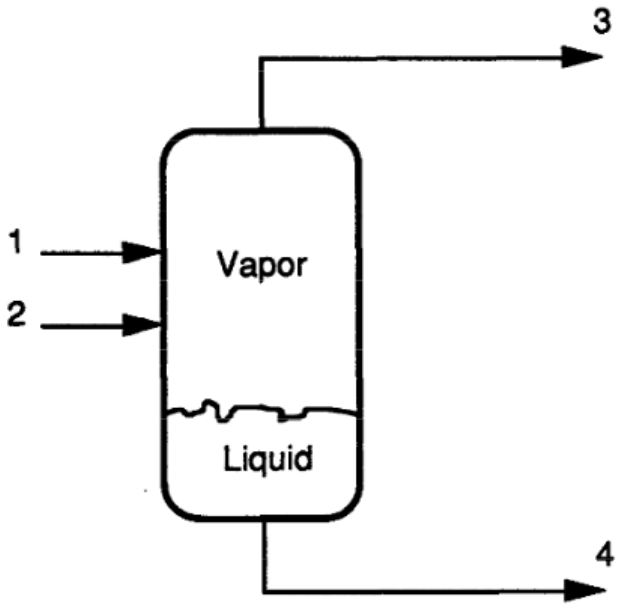


FIGURE 1.2: A schematic sketch of the STE process with two inputs and two direct outputs. Input 1 consists of chemicals A, C and trace amounts of an inert B. Input 2 consists of pure A. Output 3 is the purge loss and output 4 is the product. The product flow is adjusted by a built-in proportional feedback controller (not shown) in response to variations of the liquid inventory. The sketch is reprinted from an article by Ricker, 1993.

The process is regulated by a controller changing the valve position of both the inputs and purge loss to the reactor vessel. The controller also receives 10 measurements from different sensors on the pressure vessel as feedback which are shown in table 1.1.

For the reward function we used an absolute distance to the desired set point in pressure (y_5), purge A (y_7) and product flow (y_4). For some of our tests the cost of

Variable name	Description
y_1	Feed 1 flow measurement
y_2	Feed 2 flow measurement
y_3	Purge flow measurement
y_4	Product flow measurement
y_5	Pressure
y_6	Liquid inventory
y_7	Amount of A in purge (mol %)
y_8	Amount of B in purge (mol %)
y_9	Amount of C in purge (mol %)
y_{10}	Instantaneous cost ($ per kmol product)

TABLE 1.1: The different observations from the STE process.

running the reactor was also included in the reward. In addition, the set point from scenario 2 in the original article Ricker, 1993 was used. In this scenario the pressure (y_5) has to be increased from 2700 to 2850 pascal, the product flow (y_4) from 100 to 130 and the purge fraction A (y_7) from 47 to 63.

The STE environment also has an existing multi loop controller (MLC) built in, which can be used to generate trajectories for the algorithms to train on. The MLC is working well at scenario 2 mentioned above.

1.3.3 Original Tennessee Eastman

The original Tennessee Eastman process is a more complete process that takes more processes relevant for a chemical plant into account. In this study, we will not go into any process details, please see the original article for more information DOWNS and VOGEL, 1992. For our application it is sufficient to note a few things. First the process has 12 inputs and 41 measurements available for control. Furthermore some of the observables are only sampled every 6^{th} or 15^{th} minute, while others are sampled continuously. We choose to discretize the TE process with a sampling distance of 1 minute. The quadratic distance of the constraint variables (reactor level, pressure, temperature, separator level and stripper level) to their initial state were used as the reward function.

1.3.4 Haldex Brake Valve

This model was kindly provided by Haldex. The brake is given as a Functional Mock-Up Unit, which we can interface by using the python package FMPy. For our purposes it is sufficient to know that it takes two different currents as input and has also two outputs called "Brake torque" and "Apply pressure". The two input currents are controlled by specifying the time for "turn off" or "turn on". Furthermore both the input and output was scaled to be between about -1 to 1. Our goal was to control the output "Apply pressure" and get it to follow set points.

1.4 Quantitative Problem Formulation

Knowing more about the environments described above leads to some of the following questions;

Question 5 *How do terminal states affect the performance on the LQP environment?*

Question 6 *How well can model free RL maximise the reward function in STE scenario 2?*

Question 7 *How does the sampling period affect the ability to learn set points in Haldex brake valve?*

Question 8 *Can model free RL learn to reach a set point in the OTE process?*

Chapter 2

General Methods

2.1 Hardware

2.1.1 Laptops

For computer work, two laptops were provided by Sentian.AI with the specifications given in table 2.1.

	Laptop 1	Laptop2
RAM (GB)	16	24
CPU spec.	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz	Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz
Cores	4	4
Threads	8	8

TABLE 2.1: The laptops provided by Sentian.AIs specifications.

2.1.2 Servers

Most of the more complex computations were run on Sentian server, see table 2.2. In addition some of the tests were performed on Lunds supercluster LUNARC. Especially, the Aurora cluster consisting of nodes specified in table 2.3 was used.

	Sentian server
RAM (GB)	48
CPU spec.	2 Intel(R) Xeon(R) CPU X5550 @ 2.67GHz
Cores	8
Threads	16

TABLE 2.2: Sentians server

	Aurora Node
RAM (GB)	64 (3.2 GB/core)
CPU spec.	2 Intel Xeon E5-2650 v3 (2.3 Ghz, 10-core)
Cores	20
Threads	20

TABLE 2.3: LUNARCs Aurora node specification.

2.2 Test set up

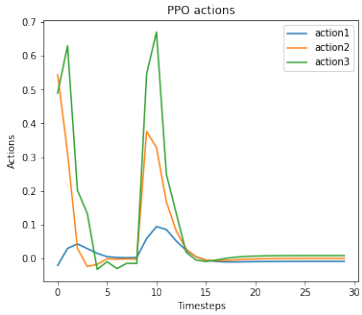
In order to test our hypothesis in our three environments we choose to design more specific tests for each environment. These tests and their results are presented and discussed below. Thereafter, the results are summarized before their relation to our hypotheses is analyzed in more detail in the final discussion.

Chapter 3

LQP environment:

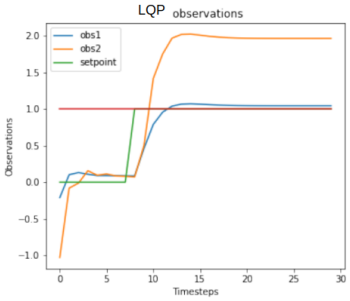
3.1 Can the algorithms learn simple setpoint changes?

The first tests for the algorithms were to see how well they could handle setpoint changes on a partially observable LQP problem. Both DDPG and PPO managed setpoint changes without any problems. An example for PPO algorithm is shown in figure 3.1.



(A)

The PPO algorithm's action when playing one LQP game. The setpoint requests are shown in the right plot (B) as a green line.



(B)

The observations of the LQP problem. There are three observations, setpoint (green), internal state (blue), and a linear combination of the remaining two internal states (orange). The red line marks the setpoint level for this particular run.

FIGURE 3.1: The PPO algorithm can handle different set-point changes on a partially observable problem. The setpoint is shown in green in the rightmost figure and the action response is shown in the leftmost figure.

3.2 Can a roll up length be used that is shorter than the systems time dependency?

As described in 1.2.7, both DDPG and PPO use a short sequence to roll up the hidden state when training. This raises the question if we can learn a state representation that has a longer delay than the roll up length. To test this, an LQP environment with the following matrices 3.1, 3.2,3.3 was used. This results in a partial observable system with a 3 time step delay.

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.1}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.2}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{3.3}$$

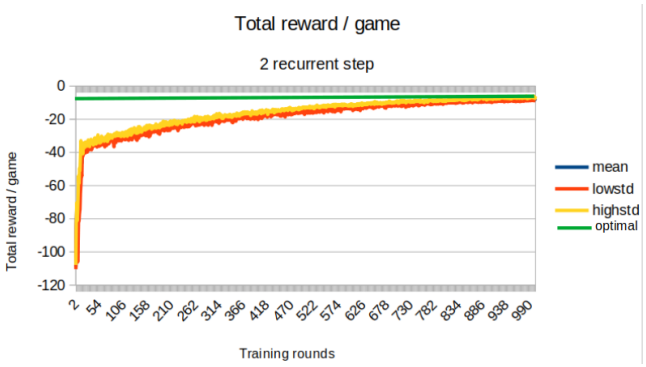


FIGURE 3.2: PPO average training performance on the 3 delay system with 3 roll up steps. It can clearly be seen that the training converges against the optimal solution (green).

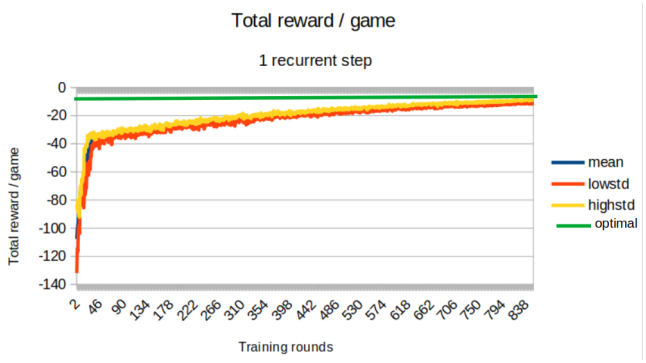


FIGURE 3.3: PPO average training performance on the 3 delay system with 1 roll up step. It can clearly be seen that the training converges against the optimal solution (green).

This shows that both algorithms can learn time dependencies longer than the roll up length. From 3.3 and 3.2 it is clear that the PPO algorithm required about the same amount of training data. This suggests that the use of fewer roll up steps in our other environments might be successful.

Figure 3.4,3.7,3.5 show that the DDPG algorithm converges for both cases. However the training times were longer for the one with 2 roll up steps. In addition, the training is also more unstable for the 2 roll up step case. One reasonable explanation for this effect is that the hidden states might not be as accurate compared to the 6 roll up step case, since the hidden states had a shorter distance to converge. The optimal reward for the infinite game when starting with the vector [1,1,1] is -9, which both algorithms seem to converge to.

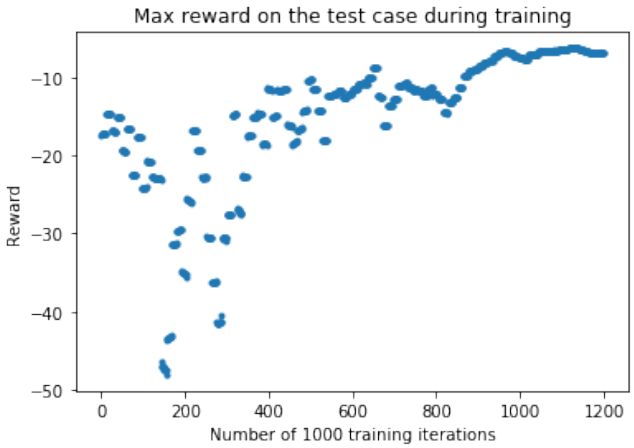


FIGURE 3.4: Maximum reward over 5 full training's. The DDPG algorithm was trained with 2 roll up and 3 burn in steps.

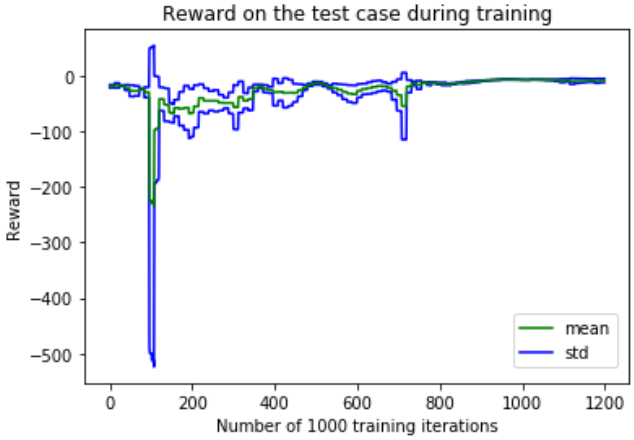


FIGURE 3.5: Average reward over 5 full training's. The DDPG algorithm was trained with 2 roll up and 3 burn in steps.

FIGURE 3.6: Minimum and maximum performance over 24 full training's. The DDPG algorithm was trained on 6 roll up and 4 burn in steps.

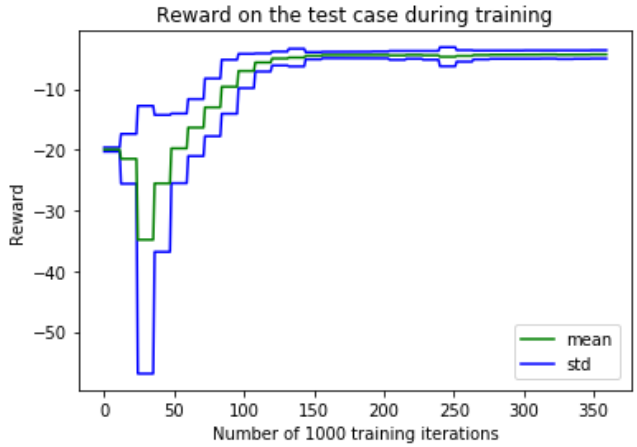


FIGURE 3.7: Average training performance over 24 full training's. The algorithm was trained on 6 roll up and 4 burn in steps.

3.2.1 Reward design on LQP

Both OTE and STE have specific termination conditions, which interfere with the reward functions design. It is therefore of interest to study the interaction between termination conditions and the reward function on the LQP problem. For the LQP problem to be as similar to the OTE and STE a partially observable problem was chosen. The games are also terminated once the absolute value of one state gets bigger than three. The initial reward from the LQP problem was also translated such that it always was positive. To keep the training times short the system dimension was chosen to be 5. The A matrix was initialised according to 1.18 and multiplied by a factor of 1.1. The B matrix was set to the identity matrix and the C matrix had 2 output dimensions and was randomly initialised. The performance of the following reward designs was analyzed. The initial state of the linear system was randomised during training mainly to avoid overfitting to a certain starting state. For the test case, the unit vector was used for initialisation in order to reduce variance of the performance.

- Positive reward with 0 as terminal reward
- Positive reward with -1 as terminal reward
- Positive reward with -10^6 as terminal reward
- Mixed reward positive and negative reward combined

In order to give each design good converging conditions the reward functions was divided by a real number until one game converged. The drawback of this strategy is that one can't compare the scores in absolute terms. The results of 10 trainings are shown in 3.8, 3.9, 3.10.

The DDPG algorithm performance on the test case during training is summarised in figure 3.8, 3.9,3.10. Some of the matrices in the linear system are randomly initialised and therefore the standard deviation was included as a measure of performances distribution. In the mixed reward case 3.9 and in the small terminal reward case 3.10 the mean test reward rises towards the end. In contrast, the increase in the positive reward case is not as pronounced implying that there were very few games that converged except the first.

FIGURE 3.8: Positive reward: Statistics of the performance on the test case during training. The average $\pm$ 1 standard deviation of 10 games is reported.

FIGURE 3.9: Mixed reward: Statistics of the performance on the test case during training. The average $\pm$ 1 standard deviation of 10 games is reported.

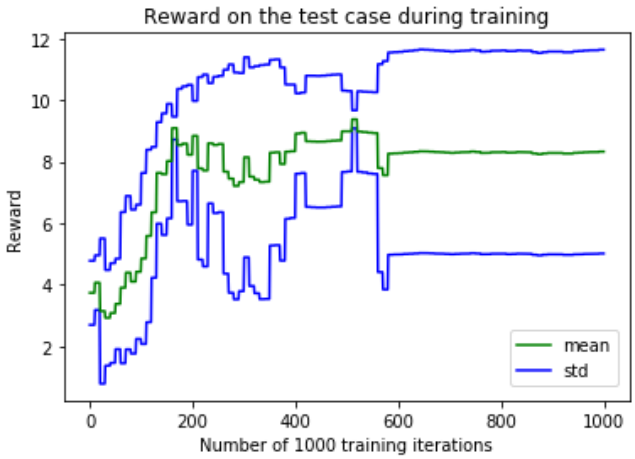


FIGURE 3.10: Small terminal reward: Statistics of the performance on the test case during training. The average $\pm$ 1 standard deviation of 10 games is reported.

All training's for the large terminal reward case diverged, resulting in termination of the game on the first step in the test case [data not shown]. A likely explanation for the observed problem is that the Q function has difficulties in learning the Q value of game ending action and state pairs. This also influences the convergence of the rest of the Q values, since the temporal difference error of the terminal state and action pairs is an order of magnitude larger.

The fact that few trainings converged in the positive reward case may be caused by difficulties in the design of the reward function. The problem here was to design and scale the reward function without having a positive reward in the terminal states. Due to the design of the reward function and termination conditions the terminal states have a higher reward than some of the non terminal states. It is very possible that this could have an effect on the learnability of the Q function.

The mixed reward case performs better than the positive reward case. The main difference between the cases is that the mixed reward has states with a negative reward. This allows for a design with a larger difference between states, whilst keeping the same maximum of the absolute reward thereby aiding in the learnability of the Q function. This reward design is however highly dependent on tuning. Improper tuning of the reward function can thus cause the algorithm to get get stuck in a local minimum trying to end the game as fast as possible since the sum of the discounted reward is higher for terminating the game than to receive another negative reward. This is also the reason for not trying out the negative reward case.

In similarity with the mixed reward case, the small negative reward case performance converged most of the time. In addition, it also improves the learnability of the Q function.

In conclusion, this small test illustrates that the reward design has a big impact on the algorithm's performance. However, the fact that none of the three reward designs managed to always converge shows the difficulty of this test. To get better result one would need to decrease the difficulty of the linear system, such that one of the three reward designs always converged. In addition, running more trainings could also increase the difference between the reward designs and further reduce the standard deviation.

Chapter 4

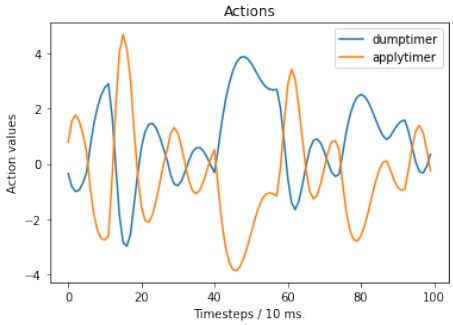
Haldex Brake

4.1 Experiments

The Haldex brake is a simulation of an actual valve in a truck brake. The big challenge with this environment is the lack of a good baseline, hence it is impossible to asses if our algorithms are better than the existing controller. This model gives us the opportunity to study how the different parameters in the algorithm impact performance on multiple set point changes in a non linear process. Of special interest here is how the sampling distance, the discount factor γ and the probability of switching the set point interact with each other.

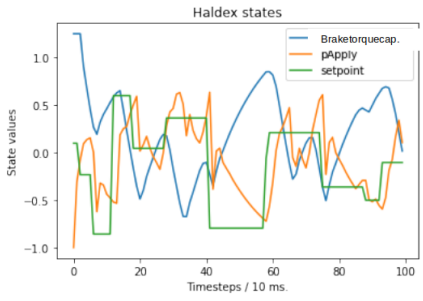
The average absolute distance to the setpoint was used as a performance metric to enable comparisons between the two different sampling distances. All algorithms were trained with quadratic distance to the setpoint as the reward function. To reduce the number of tests, the probability of switching setpoint was kept fixed at 0.1 and 0.01 for the 10 ms and 1 ms sampling distance, respectively. For 1 ms sampling distance the discount factors 0.99, 0.9 and 0.1 were used. For 10 ms the different discount factors used were 0.9, 0.3 and 0.1.

4.2 Results and discussion



(A)

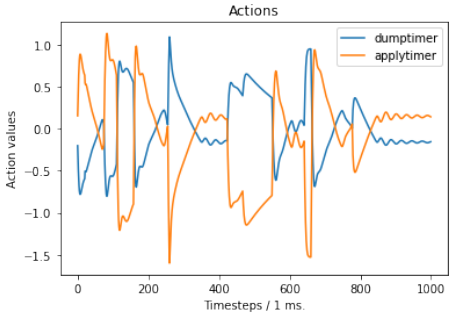
The PPO algorithm's action response to the setpoint changes. The inputs are labeled "applytimer" and "dumptimer".



(B)

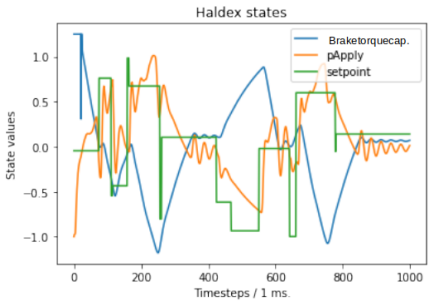
The Haldex brake valve's scaled states including the varying setpoint for the variable Apply pressure.

FIGURE 4.1: Sampling distance 10 ms: The trained PPO algorithm controlling the brake valve.



(A)

The PPO algorithms action in response to the setpoint changes. The inputs are labeled "applytimer" and "dumptimer".



(B)

The Haldex brake valve's scaled states including the varying setpoint for the variable Apply pressure.

FIGURE 4.2: Sampling distance 1 ms: The trained PPO algorithm controlling the brake valve.

Figures 4.1,4.2 illustrate that the algorithm almost always reaches the set point and thus can learn setpoint changes. Thereafter, it was studied how different sampling distances and discount factors γ affect performance.

Sampling distance (ms)	discount factor γ	Average distance per step
10	0.9	0.43
10	0.5	0.27
10	0.1	0.26
1	0.99	0.73
1	0.9	0.32
1	0.1	0.32

TABLE 4.1: Average distance to the setpoint of the trained PPO algorithm varies with respect to the discount factor γ and sampling distance. To clarify an absolute distance to the setpoint for each time step was used and then an average of these distances was taken, this is shown in the rightmost column.

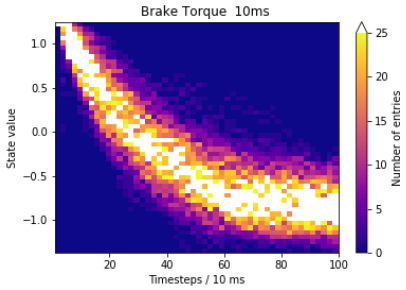
A comparison of the results in table 4.1 reveals that the performance drops drastically when the discount factor is too high. A higher discount factor presses the value network in the algorithm towards taking future rewards more into account. Setpoint changes are impossible to predict causing a degradation of the accuracy of the value network.

This performance degradation can also be seen for 1 ms sampling distance 4.1. In fact, this degradation occurs for a higher discount factor γ than in the 10 ms case. Due to that lower probability of changing setpoint in the 1 ms case, this result is expected.

Furthermore, comparison between 10 ms and 1 ms sampling distance in table 4.1 reveals that the mean distance to the setpoint is higher for the 1 ms case. The lower performance persisted even after optimisation of the algorithm's hyperparameters [data not shown]. This is probably due to the large standard deviation of the policy in the PPO algorithm.

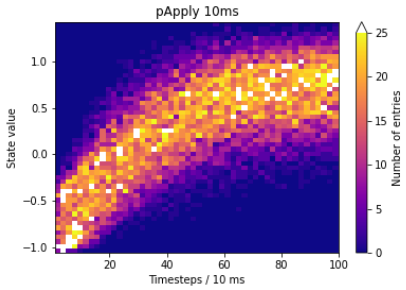
The figures 4.3 and 4.4 present the difference in observed states for the untrained PPO algorithm for 10ms and 1ms, respectively. The main difference between the

heat maps is that the observables from the environment are Gaussian distributed for a sampling distance of 10 ms 4.3, which they definitively are not for 1 ms 4.4. This suggests lowering the standard deviation of the policy should be tested in further investigations.



(A)

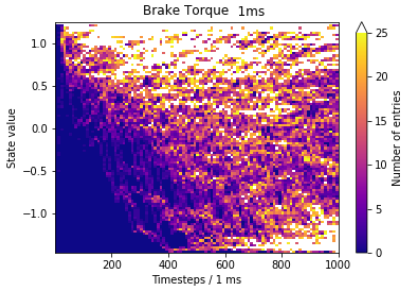
A heatmap for one of the Haldex brake valves variables, braketorquecap.



(B)

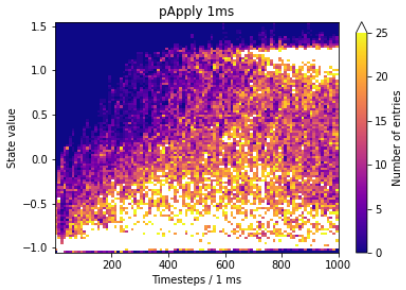
A heatmap for one of the Haldex brake valves variables, pApply.

FIGURE 4.3: Sampling distance 10 ms: Heat map of Brake torque capacity and Pressure Apply when sampling the actions from a normal distribution. This is of interest because the state exploration is of key importance for the algorithms and it seem like they are differing between sampling distances. This figure shows an expected Gaussian exploration around some varying mean.



(A)

A heatmap for one of the Haldex brake valves variables, Braketorquecap.



(B)

A heatmap for one of the Haldex brake valves variables, pApply.

FIGURE 4.4: Sampling distance 1 ms: Heat map of Brake torque capacity and Pressure Apply when sampling the actions from a normal distribution. This is of interest because the state exploration is of key importance for the algorithms and it seem like they are differing between sampling distances. This figure does not show the expected Gaussian exploration.

Chapter 5

Simplified Tennessee Eastman

The STE environment proposed by Ricker, 1993 models a chemical reactor. This environment is less complex than the OTE, but it still contains the same features including terminal states. Therefore it is a good starting point for the evaluation of our algorithms'. Experiments were conducted with both the DDPG algorithm and the PPO algorithm on the STE environment and their results are presented and discussed below.

5.1 Experiments, results and discussion for the DDPG on STE

The following hyperparameters were used in all tests in this section. Learning rates of 10^{-4} and 10^{-5} were used for the value network and policy network, respectively. To avoid oversampling of the first few trajectories in the beginning of training the replay buffer was filled with randomly sampled trajectories. The set point that was used was the same as in scenario 2, with a of pressure 2850 pascal, product flow of 130 and purge fraction A of 63.

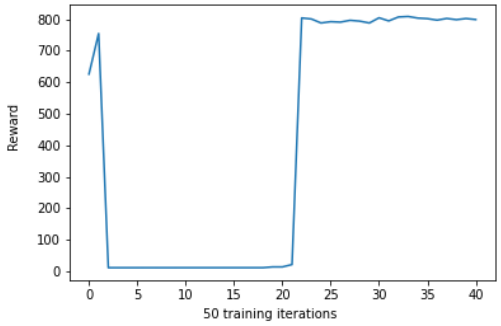
In order to closely monitor the performance of the algorithm the start state of scenario 2 was used for the test case. This has the advantage of not being stochastic, since performance variation is minimized naturally. In addition, the algorithms noise is also turned off for the test case.

5.1.1 How does γ affect training performance

The effect of the discount factor on the training and test result was studied. To get a good overview the algorithm was trained with following discounted factor: 0.2 ,0.9 ,0.99.

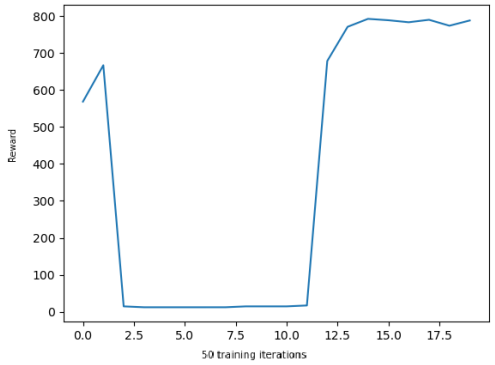
The reward plots during training se figure 5.1 imply that different values of γ have a comparable performance. In all three of these performance plots the total reward is 0 for some episodes. One reasonable explanation for this unexpected behaviour is that the policy increases the pressure too fast, thereby terminating the environment. This dip is shorter for higher discount values. This is probably an effect of the higher back propagation of bad state and action pairs.

Furthermore, the total reward starts to increase before suddenly dropping to zero around episode 5. This is probably best explained by good policy updates due to choice of learning rates.



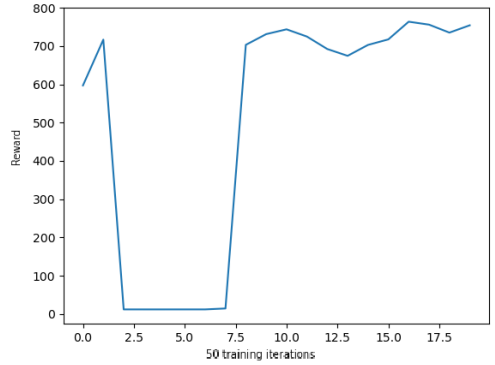
(A)

$\gamma = 0.2$: Test performance during training.



(B)

$\gamma = 0.9$: Test performance during training.

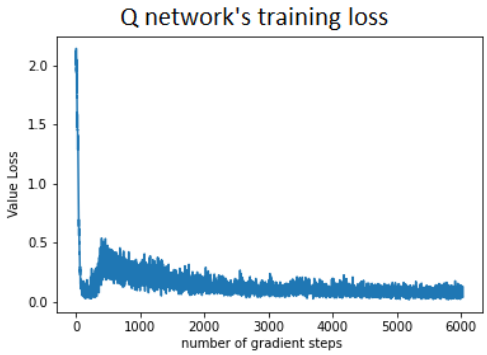


(C)

γ 0.99: Test performance during training.

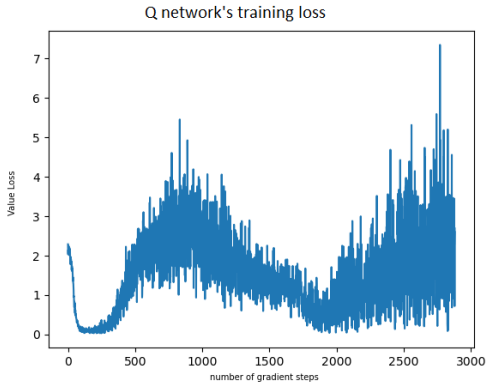
FIGURE 5.1: The test performance during training for different γ.

The training loss of the Q network can also give some insight into testing performance. The only γ for which the Q networks training loss could have converged to 0 is $\gamma = 0.2$. For $\gamma = 0.9$ and $\gamma = 0.99$ the training loss of the Q network should be larger because their discounted Q values are much larger, which is demonstrated in the figures 5.2. However, the Q network's training loss still increases at the end



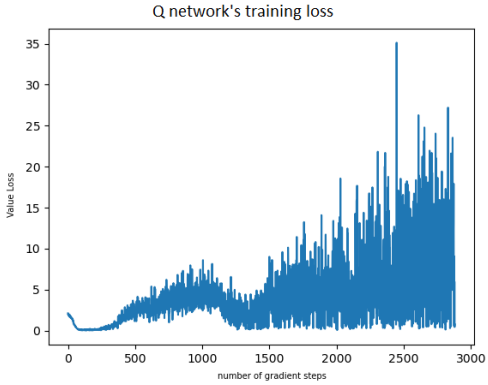
(A)

γ = 0.2:Value loss during training.



(B)

γ = 0.9: Value loss during training with.



(C)

γ = 0.99:Value loss during training.

FIGURE 5.2: The effect of different γ on the value loss.

of training. This could probably be solved by better tuning of the hyperparameters to aid the convergence of the Q networks training loss. The main parameters to be tuned are the batch size, learning rate, and different scaling of the reward. In addition, there are also two extensions to the DDPG algorithm that could mitigate this issue. First, a mixture of n-step and 1-step temporal difference learning could

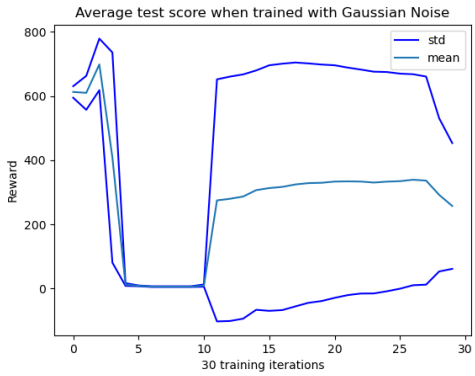


FIGURE 5.3: Gaussian noise: Statistics of the test performance. The average and standard deviation of 4 games is presented.

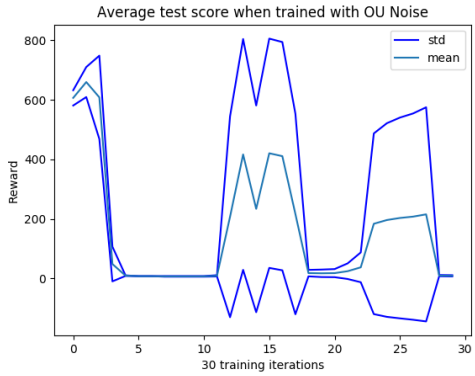


FIGURE 5.4: OU noise: Statistics of the test performance. The average and standard deviation of 4 games is presented.

improve performance. Second, the DDPG extension called TD3 could be of use.

5.1.2 How different types of noise affect training performance

In this section of the study the impact of the noise type on algorithm performance was analysed. Thus, the algorithm was run with noise generated from a normal Gaussian distribution, Ohrnstein Uhlenbeck process and a AR(2) process. See section 1.2.7 for details.The algorithm was trained with the different noise types 4 times to give a good understanding of the variation in performance.

The figures 5.3,5.4,5.5 illustrate the testing performance during training. As presented in these figures, a large portion of the training's with OU noise and Gaussian noise did not manage to increase the reward at the end. Comparing the three different noise types reveals that the algorithm consistently got the highest reward when trained with AR noise. However, this changed if maximal number of sample in the replay buffer was reduced by a factor of two 5.6.

Comparison of the test performance during training with reduced replay buffer size 5.6 to 5.3, 5.4, 5.5, reveals that there is an improvement in performance for both Gaussian and OU noise case. Thus, whilst the different types of noise have an impact on training performance. They also significantly impact the optimal hyperparameters. Therefore, it would have been interesting to compare the test performances of different types of noise with optimised hyperparameters. However, this requires

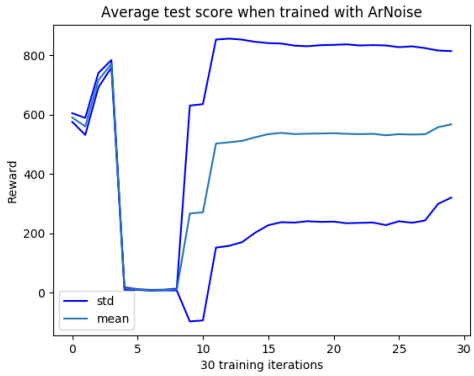


FIGURE 5.5: Ar noise: Statistics of the test performance. The average and standard deviation of 4 games is presented.

orders of magnitude more compute resources and is therefore beyond feasibility of this study.

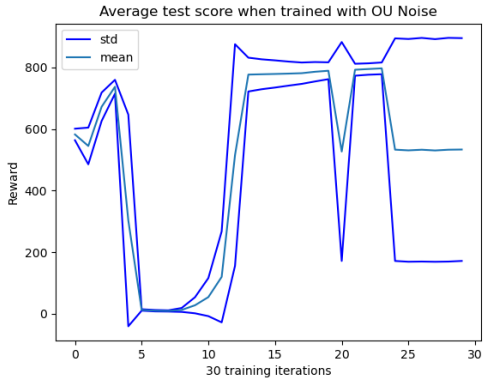
5.1.3 Do expert trajectories help performance?

For the Simplified Tennessee Eastman we have had access to an already existing controller that can control the reactor. The MLC receives a reward of 860 on second scenario. This makes it possible to train the DDPG algorithm on data generated from the MLC. To test this, the algorithm was trained on the following types of trajectories:

- Only played trajectories
- Only expert trajectories
- A mixture of 50% expert and 50% random trajectories

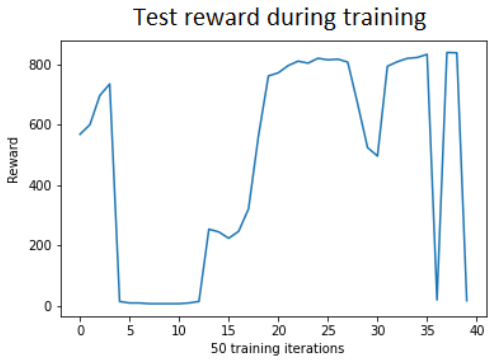
The results from the DDPG algorithm training on data generated from the MLC are presented in figures 5.7,5.8,5.9,5.10 and 5.11. Due to the similarity between figure 5.11 and 5.8 we conclude that the random trajectories do not affect testing performance. This shows that the random initialised multi loop controller probably shows enough of the state and action space to avoid the Q functions divergence.

Comparing testing performance in figures 5.11, 5.8 and 5.7 indicates that different trajectories result in similar performance. However, training on random and expert trajectories 5.11 and only expert trajectories 5.8 find the termination condition faster than training on generated trajectories 5.7. This is probably an effect of that the algorithm sees very few trajectories that terminate if training on random and expert trajectories. The observation of Q networks training loss displayed in figure 5.10 compared to figure 5.9 strengthens this idea. In figure 5.9, the Q networks training loss gets larger at the end of training when trained with generated trajectories. This might be due to the fact that the policy increases the pressure too fast and shuts down the plant. The reason why we do not see an immediate effect is that the likelihood of sampling these trajectories is very low at the beginning of training. However, in figure 5.10 the increase of the Q networks loss is missing when trained on expert and generated trajectories thus indicating that the number of those trajectories is rather low in the replay buffer.



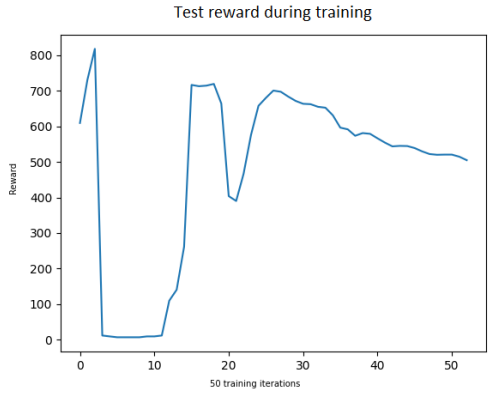
(A)

OU noise: Statistics of the test performance. The average and standard deviation of 4 games is presented.



(B)

Gaussian noise: The test reward during 1 training.



(C)

Ar noise: The test reward during 1 training.

FIGURE 5.6: The test performances during training when the replay buffer size was reduced by a factor of two compared to 5.3, 5.4,5.5.

5.2 Experiments with PPO

For the four different multi-loop controller baselines we get the performance:

FIGURE 5.7: DDPG's test performance at the start of training when trained on its own trajectories

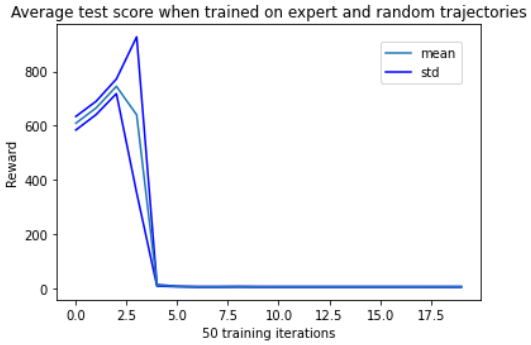


FIGURE 5.8: DDPG's test performance at the start of training when trained on expert and random trajectories

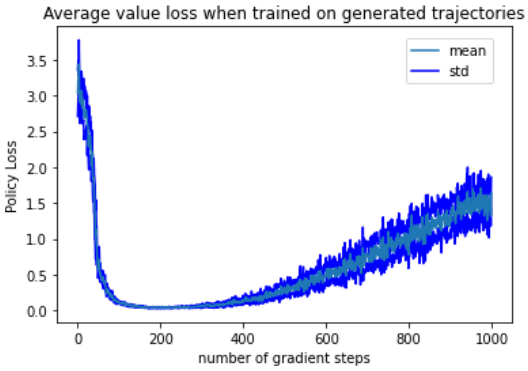


FIGURE 5.9: DDPG's value loss when trained on generated trajectories.

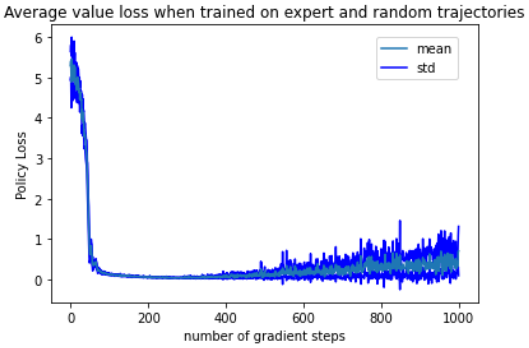
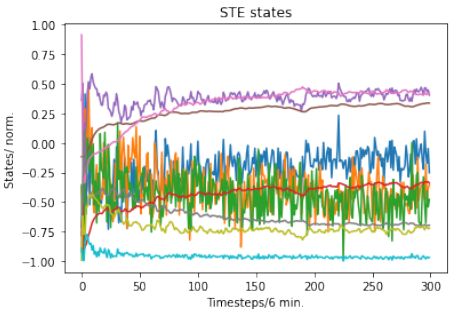
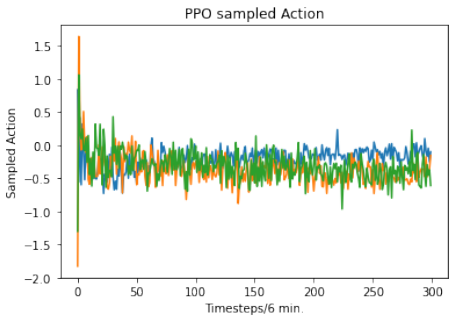


FIGURE 5.10: DDPG's value loss when trained on expert and random trajectories.

FIGURE 5.11: DDPG's performance when trained on only expert trajectories.



(A) The STE states when PPO algorithm is trying to reach the setpoint 2850 Pa, 63 % of A in purge and 130 kmolh^{-1} in product flow.



(B) The PPO algorithms actions when trying to reach the setpoints 2850 Pa, 63 % of A in purge and 130 kmolh^{-1} in product flow.

FIGURE 5.12: In this result we see the PPO algorithms ability to control the STE process. This is after around 6000 failures. The performance metric pure quadratic reward was 853 which was slightly worse than the multiloopcontroller baseline at 860.

We can see that the PPO algorithm, with its performance in pure quadratic reward of 853 does not manage to beat the baseline of the multiloop controller which scores 860. It is however interesting to see the characteristics of the control which, in the beginning, does some oscillation and then keep the actions rather constant and

Reward type	MLC	PPO
Pressure, Product flow, Purge A	860	853
Pressure, Product flow, Purge A and Instantaneous Cost	794	745

TABLE 5.1: This table shows the pure quadratic reward metric between the PPO and the multiloop controller. We can clearly see that the PPO algorithm performs worse than expected and does not manage to beat the multiloop controller. We believe that the discount factor gamma γ has a large impact on the performance of the PPO algorithm in environments where there is risk for a shutdown. The discount factor gamma effects the farsightedness of the algorithms.

still manages to get close to the multiloop controller's performance.

Chapter 6

Original Tennessee Eastman chemical plant

The OTE was originally developed by DOWNS and VOGEL, 1992 as a benchmark in control theory, which is representative for the chemical industry. It is therefore of interest to study how well model free RL algorithms can control the process.

6.1 PPO algorithm

The OTE process is started in its base case, see the paper by Ricker, 1994, with a quadratic loss distance to the base case on the constraint variables given in the article by DOWNS and VOGEL, 1992.

The first investigation was to see if we could get the algorithm to survive in the OTE environment since it is an unstable process. The results from this can be seen in figures 6.2 and 6.6. The interpretation of these results is that a very large exploration is successful in terms of making the algorithm learn how to survive in the OTE environment.

Secondly, it was investigated whether the PPO could get closer to the setpoints and actually do what is expected of a regulator. This was done by lowering the standard deviation of the exploration to try and push it towards the setpoints. This can be seen in figure 6.12 where the training reward is shown and how it is effected by the change of standard deviation. The increased performance in distance to the reactor level and reactor pressure is shown in figure 6.13.

Thirdly, the performance in connection to the planning horizon was investigated by altering the discount factor gamma. It was lowered from 0.99 to 0.96 which roughly effects the planning horizon from 4 hours to 1 hour. The training performance and how it is effected by a change in standard deviation can be seen in figure 6.11.

6.1.1 Results

The PPO algorithm on the OTE process after around 22 hours of training and 16000 shutdowns of the OTE. Laptop 1 was used, see table 2.1 for specifications.

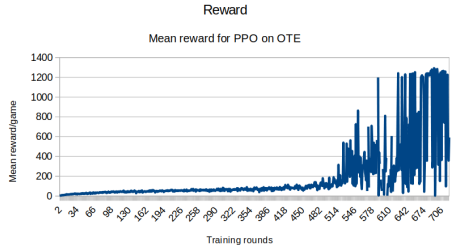
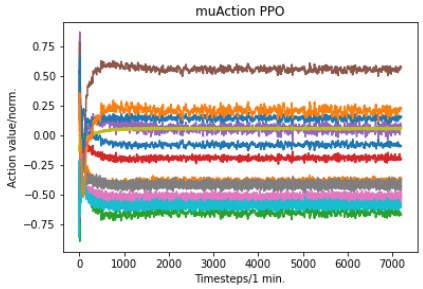
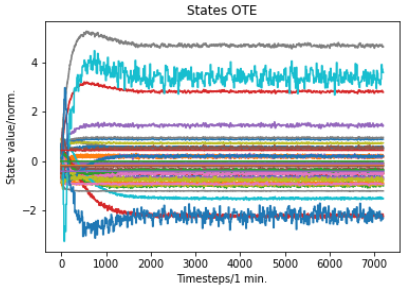


FIGURE 6.1: This result shows the mean reward per game during training performance, the training has not converged. The variance is high and it explores early shutdowns often. This is after around 22 hours of training (1600 shutdowns).



(A)

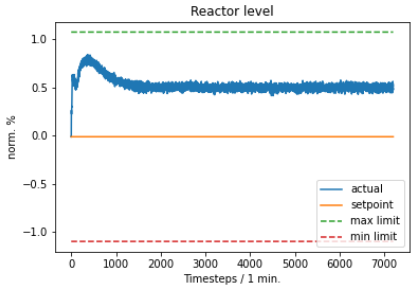
The PPO algorithms mean action response when trying to keep the setpoints shown in figures 6.3a, 6.3b, 6.4a,6.4b and 6.5a.



(B)

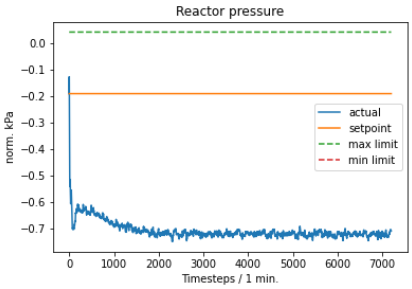
The 41 observable variables and the 5 desired setpoints shown in figures 6.3a, 6.3b, 6.4a,6.4b and 6.5a.

FIGURE 6.2: These plots show how the PPO algorithm manages to survive in the OTE environment with, just like in the STE environment, a quick oscillation in the beginning and then constant actions with small adjustments. The states do not reach the desired setpoints as shown below. This is after around 22 hours of training (1600 shutdowns) on the OTE process where it played at most 24 hours in game. It manages to survive for more than 5 days in the OTE environment when tested. The algorithm uses the mean of its actions whereas in figure 6.6 it uses a sample.



(A)

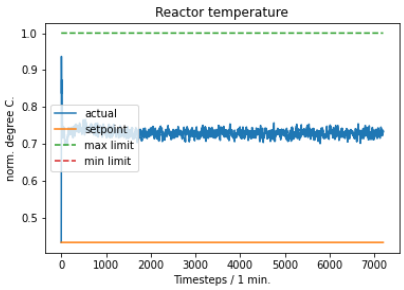
The normalized reactor level together with max, min, mean and the actual state.



(B)

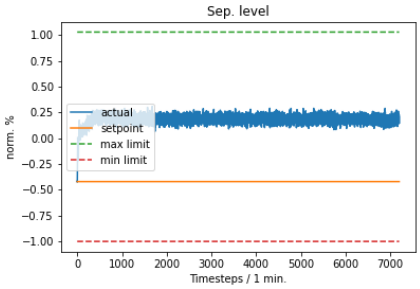
The normalized reactor pressure together with max, (there is no min for this variable), mean and the actual state.

FIGURE 6.3: This figure shows the reactor level and reactor pressure on the OTE process for a PPO algorithm with a discount factor gamma of 0.99 and a log standard deviation of -1. It does not reach the setpoints and just survives.



(A)

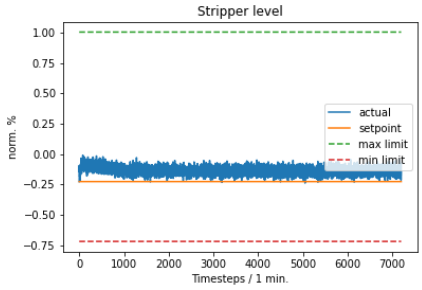
The normalized reactor temperature together with max, (there is no min for this variable), mean and the actual state.



(B)

The normalized separator level together with max, min, mean and the actual state.

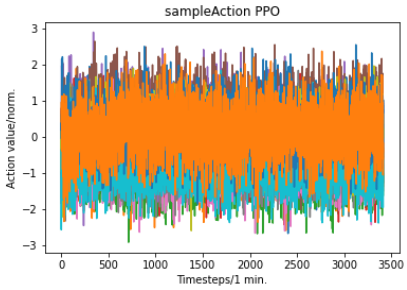
FIGURE 6.4: This figure shows the reactor temperature and separator level on the OTE process for a PPO algorithm with a discount factor gamma of 0.99 and a log standard deviation of -1. It does not reach the setpoints and just survives.



(A)

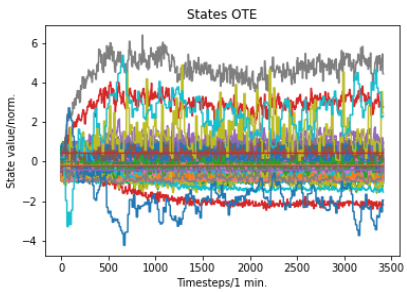
The normalized stripper level together with max, min, mean and the actual state.

FIGURE 6.5: This figure shows the stripper level on the OTE process for a PPO algorithm with a discount factor gamma of 0.99 and a log standard deviation of -1. It does reach the setpoint but with a slight offset and this was the easiest one to reach considering the others are off to a greater exent.



(A)

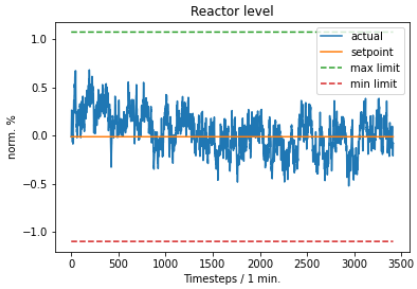
The PPO algorithms sampled action response when trying to keep the setpoints shown in figures 6.7a, 6.7b, 6.8a,6.8b and 6.9a.



(B)

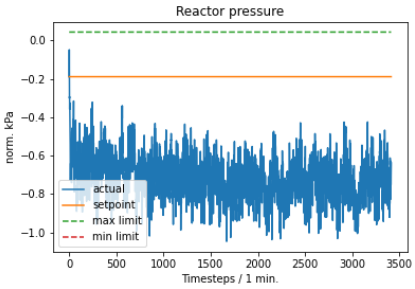
The 41 observable variables and the 5 desired setpoints shown in figures 6.7a, 6.7b, 6.8a,6.8b and 6.9a.

FIGURE 6.6: This figure shows the PPO algorithm with a gamma of 0.99 and a log standard deviation of -1. The sampled action goes outside the interval of -1 and 1 which are the limits of the actions in the OTE process. This type of noise creates a very large exploration which effects the convergence to the setpoints. These figures represents a state of the algorithm after around 22 hours (1600 shutdowns) of training on the OTE process.



(A)

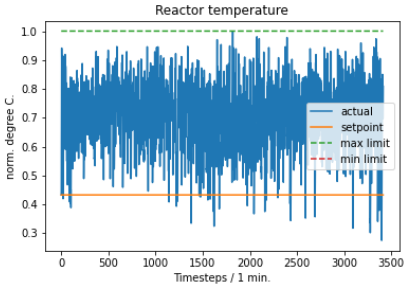
The normalized reactor level together with max, min, mean and the actual state.



(B)

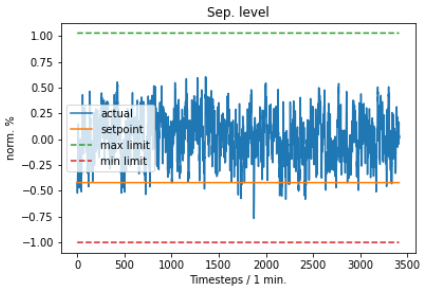
The normalized reactor pressure together with max, (there is no min for this variable), mean and the actual state.

FIGURE 6.7: This figure shows the exploration of the states when the PPO trains with a log standard deviation of -1. The exploration makes the algorithm see a large part of the state space of the reactor level and reactor pressure. The exploration varies a lot between variables which will effect optimisation.



(A)

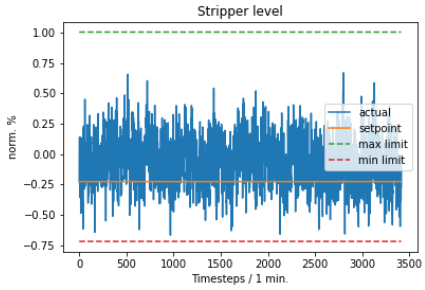
The normalized reactor temperature together with max, (there is no min for this variable), mean and the actual state.



(B)

The normalized separator level together with max, min, mean and the actual state.

FIGURE 6.8: This figure shows the exploration of the states when the PPO trains with a log standard deviation of -1. The exploration makes the algorithm see a large part of the state space of the reactor temperature and separator level. Notice however that the exploration varies a lot between variables which will effect optimisation.



(A)

The normalized stripper level together with max, min, mean and the actual state.

FIGURE 6.9: This figure shows the exploration of the states when the PPO trains with a log standard deviation of -1. The exploration makes the algorithm see a large part of the state space of the stripper level.

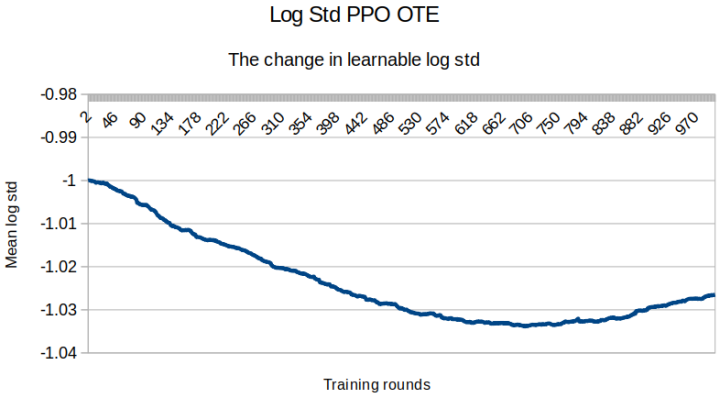


FIGURE 6.10: This result shows the miniscule variation of log standard deviation by the learnable standard deviation for the PPO algorithm. The PPO algorithm has the option to let the standard deviation of the action distribution be learnable in a sense that back propagation will try to adjust it during training. The figure shows the mean over the 12 input variables learnable logarithm of standard deviations. This is over the full training of 22 hours.

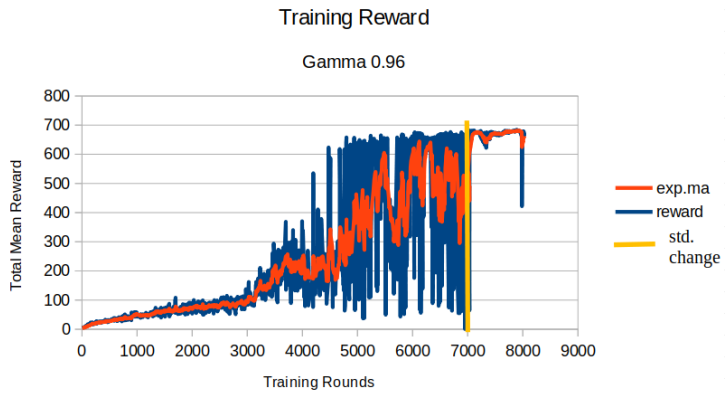


FIGURE 6.11: This figure shows the training performance of the PPO algorithm when a gamma of 0.96 was used (˜1 hour horison). At training round 7000 the log standard deviation was decreased from -1 to -3. the change was made at a stage where the algorithm could survive but not reach the setpoints. This made the algorithm get closer to the setpoints and survive more in the training.

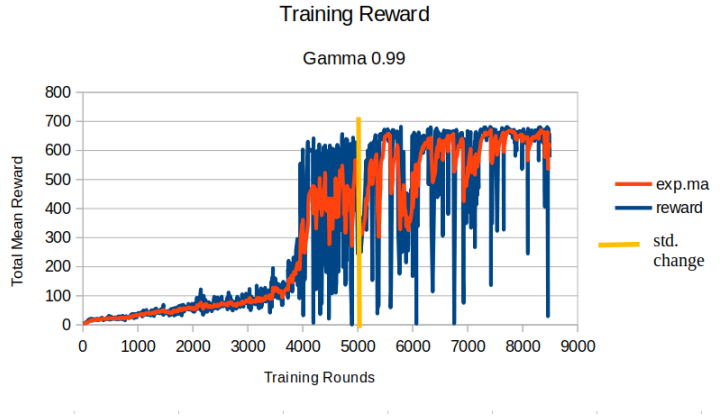
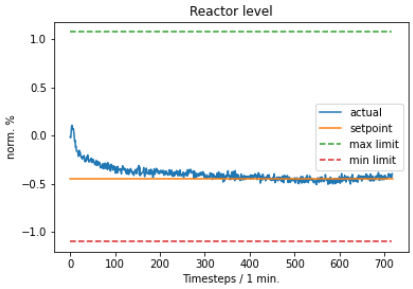
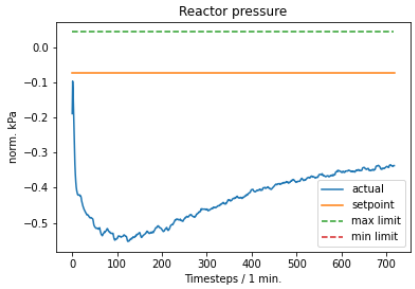


FIGURE 6.12: This figure shows the training performance of the PPO algorithm when a gamma of 0.99 was used (˜4 hours horizon). At training round 4900 the log standard deviation was decreased from -1 to -3. The change was made at a stage where the algorithm could survive but not reach the setpoints. This made the algorithm get closer to the setpoints and survive more in the training. It can be seen that its reward goes down more often than for gamma 0.96.



(A)

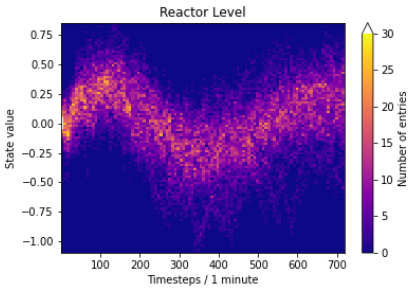
The normalized reactor level together with max, min, mean and the actual state.



(B)

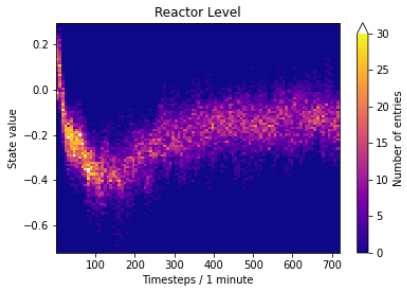
The normalized reactor pressure together with max, (there is no min for this variable), mean and the actual state.

FIGURE 6.13: This result shows the increased performance for the PPO algorithm in terms of distance to the setpoint for the reactor pressure and level when the log standard deviation has been decreased from -1 to -3. To get a comparison see figure 6.8



(A)

The state exploration of the PPO in the variable reactor level with gamma 0.99 before the change in log standard deviation from -1 to -3.



(B)

The state exploration of the PPO in the variable reactor pressure with gamma 0.99 after the change in log standard deviation from -1 to -3.

FIGURE 6.14: This figure shows the PPO algorithms state exploration in the variable reactor level with a gamma of 0.99. This investigates if the state exploration differs before (A) and after (B) the change in log standard deviation from -1 to -3. The slow oscillation decreases as we decrease the standard deviation.

The findings shows how the PPO algorithm learns how to survive in the OTE process with a lot of exploration. It can also be shown that the decrease of standard deviation decreases the distance to the setpoints both for discount factor at 0.99 and 0.96. A difference in state exploration could be seen as well when lowering the discount factor.

6.1.2 Discussion

It can be concluded that the PPO algorithm manages to survive in the OTE process while not paying much attention to the setpoints, see figure 6.2. It is thus concluded that the PPO algorithm can survive in the OTE process and disregards the setpoints when too much exploration is used, see 6.6. When trying to help the algorithm on

its way by manually decreasing the standard deviation one can notice that it is possible to push the PPO algorithm towards the setpoints, see figure 6.11. This is what can be expected to happen when decreasing the exploration. However, it shall be emphasised that the manual decrease in standard deviation does not always work. Reinforcement learning algorithms are known for sometimes deviating in training and this happened in a few tests when altering the exploration. It shall be added that a very reliable first part of training was noticed when the PPO algorithm mainly focuses on surviving longer, in all the tests this seemed to succeed. It is probably a very clear signal to the optimiser that if the PPO survives a longer amount of time it gets more reward. This poses an interesting question which is: if the algorithm firstly focuses on surviving and not to reach the setpoints, how long training in terms of minutes should one then do to both make sure the algorithm learns how to survive and to reach the setpoints? One would probably need to train long enough for the algorithm to see the full dynamics of the states, for example if one variable has the slowest oscillation period of around 12 hours then this might be a good choice as training time.

A change in the characteristics of the training reward could be seen when trying to increase the performance by lowering the discount factor gamma from 0.99 (a horison around 4 hours) to a value of 0.96 (a horison of around 1 hour), see 6.12 and 6.11. Although this is just one training and more statistics was difficult to gather due to limited computational power it can be concluded that it would be interesting to investigate further the relation between changing standard deviation and the discount factor gamma.

The change in state exploration is not drastically changed when decreasing the exploration as can be seen in figure 6.14. This is what we expect and it means the algorithm can keep exploring and at the same time reach the setpoints.

Another interesting observation is that slow dynamics, which is not altered by the quick random oscillation in the sampling of an action, could be seen as a part of the system in which the feedback comes long after a certain choice of policy. For example it is clearly written in the original Tennessee Eastman article, by DOWNS and VOGEL, 1992 that the process is sensitive to rapid changes, in the region 3-7 minutes, in product flow (stream 11). This creates an interesting analogy to the well studied computer games where the rewards are sparse and sometimes deceptive.
One famous example is Montezuma's Revenge (if googling add the term "game" unless you want to find information concerning stomach problems for tourists in Mexico) which basically is a 2D game where the player tries to find their way through an underground maze avoiding dangers by jumping, running, sliding, climbing and collecting items. Sparse rewards comes into play when the character needs to explore several rooms to finally find the desired item. Different exploring techniques have been developed to address this issue as shown in works by Adrien Ecoffet and Clune, 2018, based on the known problems of catastrophic forgetting which Roby Velez, 2017 writes about. It might be interesting to investigate the more hand engineered way of isolating neuron groups to create more of a memory. However, it is difficult to know how far along the path of model-free vs model-based approaches one shall go to potentially imitate the human way of learning.

6.2 DDPG algorithm on Original Tennessee Eastman process

To study the DDPG algorithm behaviour on the OTE all three training methods discribed in 1.2.7 were used with a wide variety of hyperparameters. Details of the

different strategies are explained in the implementation details 1.2.7. Unfortunately, the computational power and time frame of this study was not enough to use automated hyperparameter tuning. Therefore, only general observations and trends are presented and discussed below.

The first update strategy trained was the method were the gradients always are rolled up from the beginning of the trajectory. Unfortunately, this update method took too long time to ever be trained to completion [data not shown]. One reason for that is the need of a small batch size, putting heavy constraints on the learning rates. To avoid this problem, the training of the algorithm should also utilise multiprocessing. However, the longer the policy survives, the further back the gradients get rolled up slowing down training speed heavily. In comparison, the other two training methods do not suffer from this problem, since they use a fixed number of roll up steps making them better suited for longer games.

The second strategy tested, was the training method with biased batches. This method performed better than the first, almost trained to completion see figure 6.16b. An advantage of this strategy is that it can use extremely large batches effectively. The drawback is that it needs to use large batches, thereby being data inefficient.

The third strategy tested was the training method with a fixed number of roll up steps, but with unbiased batches. This method has the drawback that the policy must survive at least the number of burn in and roll up steps combined. To achieve this, actions for the first number of burn in and roll up steps were sampled from a Gaussian distribution with a standard deviation of 0.01. The computational resources and time frame of this study did not allow for enough tuning of this method [data not shown]. For further study, it would have been interesting to sample the first actions from the policy instead combined with fewer burn in and roll up steps.

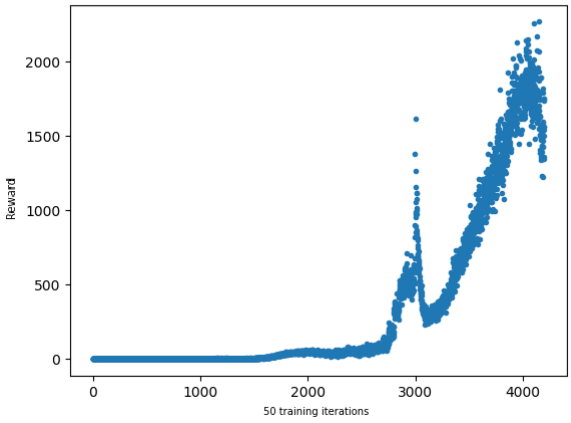
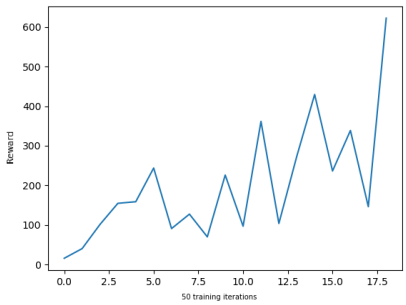
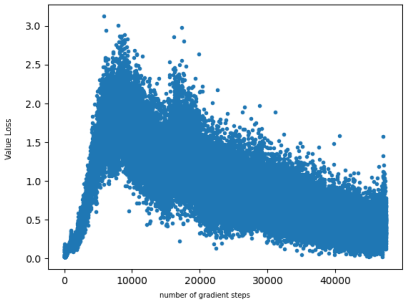


FIGURE 6.15: Example of the Q networks training loss diverging. This specific example was created with a γ of 0.98 and with the unbiased batch method.

Figure 6.15 displays the diverging training loss of the Q network. This is the main failure mode that was encountered when training the DDPG algorithm on OTE. On all three methods it was found that increasing batch size, lower learning rates and lower discount factor γ mitigated this failure mode. The main cause of this failure mode is probably the phenomenon called catastrophic value approximation. For further studies it would be interesting to investigate the Q values of certain action and state pairs in more detail.



(A) Test performance of the most successful training of the DDPG algorithm on the TE environment. In the end the DDPG algorithm survived 10 hours before terminating the game.



(B) Q network's training loss on the most successful training of the DDPG algorithm on the TE environment.

Chapter 7

General discussion

The performance of the PPO and DDPG algorithms in LQP ,STE, OTE and Haldex brake environments reveal general similarities in algorithm behaviour. Thus, testing the algorithms in our experiments shows that the neural networks have a higher difficulty learning when the input and output is not bounded around 1 in magnitude. This proved most challenging for the Q network and value network. A reason for that might be that a well scaled reward does not necessarily lead to a well scaled true value or Q function.

An additional similarity of our tests was the trade off between the discount factor γ and the rewards scaling. As observed in our tests, lower absolute rewards enables the use of a higher γ. This controls how far the algorithm takes future rewards into account versus how much it maximises the next reward.

Another interesting aspect is that Gaussian noise provided enough state space exploration such that the PPO could learn to control the tested environments to some extent. Regarding the DDPG it was also shown that the type of noise also affects which hyperparameters give good performance. Thus, making a direct comparison between the different noise types for the DDPG difficult.

For both, PPO and DDPG algorithms the training time could be improved by using more parallelisation. Fully taking advantage of the distributed computation power on clusters would certainly decrease the training time.

In addition to the similarities described above, there are some algorithm specific behaviours observed in the different environments. For the PPO algorithm it is shown in the LQP chapter that the roll up sequence length does not impact performance. Furthermore, fewer roll up steps allowed for a faster training. This was due to fact that algorithm only trains on samples that are the number of roll up steps apart. Another big advantage for fewer roll up steps is that the policy needs to at least survive the number of roll up steps in the environment.

For the DDPG algorithm, we propose that the hidden states are rolled up on a fixed distance and unbiased batches are used. This strategy performed well for the LQP, STE and Haldex Brake environments whereas it failed for the more complex OTE environments probably due to bad hyperparameter tuning. Given the computational power at our disposal we choose to focus on the PPO algorithm for the Haldex Brake and OTE experiments.

In conclusion, the algorithms managed to control our environments to some extent. Furthermore we could reach setpoints in both the STE and the Haldex Brake indicating the use of model free reinforcement learning are basically capable of controlling industrial processes.

7.1 Model free reinforcement learning in gaming and industrial processes

In this thesis we tested to apply model free reinforcement learning to industrial processes by treating them as games. Two model free algorithms, DDPG and PPO, were used to control three different simulations of industrial processes.

In this thesis we have shown that DDPE and PPO algorithms were able to reach setpoints in the Haldex Brake and STE environments. The PPO algorithm also managed to learn a policy that does not crash the OTE environment in the first 5000 steps. This would correspond to around 12 hours on laptop 1. Thus, from a game perspective, the algorithms could learn to complete the industrial processes.

However, there are few key differences between a game approach and an industrial approach, which are beyond the feasibility of this thesis. The main difference is that in a game environment the algorithms can play millions of games with the only drawback being the need for more computing resources. This does not transfer to an industrial setting were it is expensive to run with sub-optimal performance and environment termination often results in a catastrophically high cost.

These challenges need to be solved in order for model free RL to succeed in industrial environments other than in niche applications. There are some emerging approaches that try to solve these challenges. For example, research is going on to integrate model free reinforcement learning with classic control theory. These combined approaches could provide performance and stability guarantees, which does not exist in today's RL methods.

Another solution would be to pre-train the actor before deployment in the real environment. This is studied in the field of inverse reinforcement learning.

7.2 Project Reflections

In this short section, we go through some thoughts about whether we think we have succeeded or not with what we planned to do in this thesis. The grand vision was all along to be able to successfully make our algorithms control the environments to our liking. In almost all of the environments, we managed to give our directions in the form of setpoints to the algorithms which they managed to follow. Thus we feel that the main hypothesis 1 is fulfilled to some extent. To some extent because the algorithms have a lot of potentials to improve their control. For example, we would like the algorithms to get closer to the setpoints, handle more variables as setpoints and be able to handle more complex variations of setpoints instead of just step changes.

We have realised that the questions about characteristics for success and failure, question 2 and 3, are very difficult to answer scientifically but we have done our best to do so. In retrospect, a more discrete or quantitative version of these questions could have been formulated where the word "success" and "characteristics" would have been more clear. We can say that there is some way to go, either in the further development of algorithm architecture or hyperparameter tuning, before model free RL makes it out to the industry. Despite these insights question 4 which asks whether model-free RL is suitable for industrial applications could also have been answered better if it were more clear what we define as "suitable".

Concerning the more quantitative problem formulations, we could easier address them and say if they were fulfilled or not.

All in all, we feel that model free RL has great potential if not as a single lonely controller then definitely as a small cog in a greater wheel of the control architecture.

Chapter 8

Conclusion

Our results show hyperparameters have a great impact on the algorithm's training performance. In order to automatical tune the hyperparameters running time of the algorithms needs to be shortened. However, even with automatical tuning of the hyperparameters we would still need a basic understanding of how those affect the learning.

A major challenge for model free RL algorithms is to reach a baseline performance of existing controllers. We have had some preliminary success with adding trajectories from a controller to the replay buffer.

Taken together, reinforcement learning is a fast evolving field opening with high potential for optimization of industrial processes. With this potential in mind more effort will be needed to lift RL algorithms from games to real industrial processes.

Bibliography

Adrien Ecoffet Joost Huizinga, Joel Lehman Kenneth O. Stanley and Jeff Clune (2018). "Montezuma's Revenge Solved by Go-Explore, a New Algorithm for Hard-Exploration Problems (Sets Records on Pitfall, Too)". In: URL: https://eng.uber.com/go-explore/.

DOWNS, J. J. and E. F. VOGEL (1992). "A plant-wide industrial process problem". In: URL: http://users.abo.fi/khaggblo/RS/Downs.pdf.

Goodger, Nikolaj (2020). "Proximal Policy Optimisation with PyTorch using Recurrent models". In: URL: https://medium.com/@ngoodger_7766/proximal-policy-optimisation-in-pytorch-with-recurrent-models-edefb8a72180.

Higgsfield. *DDPG implementation*. URL: https://github.com/higgsfield/RL-Adventure-2. (accessed: 22.07.2020).

John Schulman Filip Wolski, Prafulla Dhariwal Alec Radford Oleg Klimov (2017). "Proximal Policy Optimization Algorithms". In: URL: https://arxiv.org/abs/1707.06347.

Joohyun Shin Thomas A. Badgwell, Kuang-Hung Liu Jay H. Lee (2019). "Reinforcement Learning – Overview of recent progress and implications for process control". In: URL: http://www.sciencedirect.com/science/article/pii/S0098135419300754. "(accessed: 09.24.2020)".

Lewis, F. L., D. Vrabie, and K. G. Vamvoudakis (2012). "Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers". In: *IEEE Control Systems Magazine* 32.6, pp. 76–105.

Markowsky, George (2017). "Information theory". In: URL: https://www.britannica.com/science/information-theory. (accessed: 24.06.2020).

Mel Vecerik Todd Hester, Jonathan Scholz-Fumin WangOlivier Pietquin Bilal Piot Nicolas HeessThomas Rothörl Thomas Lampe Martin Riedmiller (2017). "Leveraging Demonstrations for Deep ReinforcementLearning on Robotics Problems with Sparse Rewards". In: URL: https://arxiv.org/pdf/1707.08817.pdf. "(accessed: 09.08.2020)".

Open AI spinning up. URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html. (accessed: 22.07.2020).

OpenAI. *OpenAI gym*. URL: https://gym.openai.com/. (accessed: 24.06.2020).

Recht, Ben. *An Outsider's Tour of Reinforcement Learning*. URL: http://www.argmin.net/2018/06/25/outsider-rl/. (accessed: 22.09.2020).

Ricker, N. Lawrence (1993). "Model predictive control of a continuous, nonlinear, two-phase reactor". In: URL: https://www.sciencedirect.com/science/article/pii/095915249380006W.

Ricker, N.L. (1994). "Optimal steady-state operation of the Tennessee Eastman challenge process". In: URL: https://www.sciencedirect.com/science/article/pii/009813549400043N.

Roby Velez, Jeff Clune (2017). "Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks". In: URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0187736.

Steven Kapturowski Georg Ostrovski, John Quan-R emi Munos Will Dabney (2019). “Recurrent experience replay in distributed reinforcement learning”. In: URL: https://openreview.net/pdf?id=r1lyTjAqYX. "(accessed: 20.06.2020)".

Steven Spielberg Aditya Tulsyan, Nathan P. Lawrence Philip D Loewen R. Bhushan Gopaluni (2020). “Deep Reinforcement Learning for Process Control: A Primer for Beginners”. In: URL: https://arxiv.org/pdf/2004.05490.pdf. "(accessed: 09.24.2020)".

Timothy P. Lillicrap Jonathan J. Hunt, Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra (2015). “Continuous control with deep reinforcement learning”. In: URL: https://arxiv.org/abs/1509.02971. "(accessed: 28.06.2020)".

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
October 2020

Document Number
TFRT-6111

Author(s)
Niklas Kotarsky
Eric Bergvall

Supervisor
Johan Grönqvist, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Reinforcement Learning in Industrial Applications

Abstract

Although reinforcement learning has gained great success in computer games, there are only few yet known implementations in industrial applications. This despite the fact that reinforcement learning offers interesting methods to optimise the control of nonlinear processes. In this thesis we have used two model free reinforcement learning algorithms (PPO and DDPG) to control three different simulations of industrial processes, the simplified Tennessee Eastman, original Tennessee Eastman and the Haldex brake valve. Both reinforcement learning algorithms could in almost all cases learn to reach a set point. In addition, hyperparameters were found to have a high impact on training performance. In conclusion, our tests indicate that the model free reinforcement learning algorithms are basically capable of controlling industrial processes. Python code for the PPO algorithm applied to the Original Tennesse Eastman process can be found at Github. 1

1 https://github.com/Heigke/Reinforcement-Learning-In-Industrial-Applications

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-54

Recipient's notes

Security classification

http://www.control.lth.se/publications/