Norwegian University of
Science and Technology

Reinforcement learning in process control

Eskild Ruud Mageli

June 6th, 2019

Chemical Engineering and Biotechnology
Submission date: June 2019
Supervisor Johannes Jäschke, IKP
Co-supervisor Dinesh Krishnamoorthy, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Preface

This thesis is a Master's thesis for the course TKP4900 - "Chemical Process Technology, Master's Thesis". The work was conducted during the spring of 2019 at the Norwegian University of Science and Technology in the Process System Group at the Department of Chemical Engineering. The written work is meant as an introduction to the concept of reinforcement learning for the researchers at the department along with different implementations of reinforcement learning in process control and how they perform compared to common industry controllers.

I would like to thank my supervisor Johannes Jäschke for allowing me to work with the field of reinforcement learning in control as well as his guidance. I would also like to thank my co-supervisor Dinesh Krishnamoorthy and my father for reading through my thesis and giving extensive feedback on my report. I also would like to thank Professor Keith L. Downing at the Department of Computer and Information Sciences for guidance in reinforcement learning. I would like to thank my friends and family for their encouragement and support. I would especially thank Kristine Øya who has supported me throughout all these years at NTNU.

Abstract

Using reinforcement learning as controllers in the process industries was explored as an alternate path of doing control compared to the regular controllers. Methods such as value-based and policy-based methods were used as controllers for three different cases of tank level regulation. The controllers were compared to a traditional P-controller for evaluation of the controller performance. The reinforcement learning controllers showed promising results as they managed to control the liquid level between the predetermined constraints. However, the P-controllers proved a better performance with smaller input changes compared to the reinforcement learning controllers which had large input changes that resulted in oscillatory liquid level. This thesis shows that the creation of reinforcement learning controllers is complicated and time-consuming and a well-tuned controller would most likely perform better. However, with more research and standardized approaches, there is a huge potential of including this field into the process industries due to its ability to handle nonlinearity and long term evaluations.

Sammendrag

Bruk av forsterkningsinnlæring som kontroller i prosessindustriene ble utforsket som et alternativ for kontroll til sammenligning med vanlige kontrollere. I denne oppgaven var forsterkningsinnlæringsmetodene value based- og policy based brukt som kontrollere for tre forskjellige tilfeller av tanknivåregulering. Kontrollerne ble sammenlignet med en tradisjonell P-kontroller for evaluering av kontrollerens ytelser. forsterkningsinnlæringsregulatorene viste lovende resultater da de klarte å kontrollere væskenivået mellom de forhåndsbestemte begrensningene. Imidlertid viste P-kontrollerne en bedre ytelse med mindre ventilposisjonsendringer sammenlignet med forsterkningsinnlæringsregulatorene som hadde store ventilposisjonsendringer som resulterte i oscillerende væskenivå. Denne oppgaven viser at etableringen av forsterkningsinlæringsregulatorer er komplisert og tidkrevende, og en velinnstilt tradisjonll regulator vil være et bedre valg av regulator. Med mer forskning og standardiserte metoder er det et stort potensial for å inkludere forsterkningsinnlæring inn i prosessindustriene på grunn av dens mulighet til å takle ikke-linearitet og langsiktig evaluering.

Overview of thesis

The structure of the thesis consists of a literature review followed by implementation and evaluation of methods in reinforcement learning in the light of process control. The first section is an introduction to the relevant conventional controllers which are currently used in the process industry. This follows into the general introduction to machine learning and reinforcement learning and their current successful applications. After the introduction, a deep literature review of reinforcement learning is presented. The end of section 2 marks the end of the literature review and the rest of the thesis is about the implementation and evaluation of controllers. Section 3 presents the cases and section 4 explains the implementation, along with its challenges, of the different controllers. The performance of the controllers is evaluated in section 5. In section 6, the controllers performance discussed and further an overall conclusion is then presented in section 7. In the final section potential further work is presented.

In the literature can notation and terminology in machine learning and control theory be different from each other, even though the concepts are the same. For example for states in chemical engineering are x used but denoted as s in ML. This thesis mainly uses the terminology and notation used in machine learning as it is the theory presented in the literature. The fields of control and machine learning often overlap and for this purpose will the terminology within control theory will be presented for better understanding when there is a clear relationship.

Table of Content

List of Figures

List of Tables

List of abbreviations

Abbreviation	Full name
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AC	Actor-Critic
AR	Auto-Regression
ArgMax	Argument of the Maxima
ANN	Artificial Neural Network
APD	Approximate Dynamic Programming
CV	Controlled Variable
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DQN	Deep-Q-Network
DP	Dynamic Programming
FNN	Feedforward Neural Network
GPI	General Policy Iteration
KKT	Karush-Khun-Tucker
LSTM	Long Short-Term Memory
MA	Moving Average
MC	Markov Chain
MC	Monte Carlo
ML	Machine Learning
MDP	Markov Decision Process
MIP	Mixed Integer Programming
MRP	Markov Reward Process
MPC	Model Predictive Control
MV	Manipulative Variable
PG	Policy Gradient
PPO	Proximal Policy Optimization
QP	Quadratic Programming
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTO	Real Time Optimization
SIMC	Simplified Internal Model Control
SL	Supervised Learning
SP	Set Point
TD	Temporal Differences
TRPO	Trust Region Policy Optimization
PID	Proportional-Integral-Derivative
UL	Unsupervised Learning

Table 1: List of abbreviations used throughout the thesis, sorted alphabetically

1 Introduction

Process control is a field within industrial control systems which combines chemical- and control-engineering. For process industries, process control is crucial to maintain safe and optimal production. The first known usage of process control is credited to Ktesibios of Alexandria in the 3rd Century BC, who created a float valve to regulate water levels of water clocks [1]. Since then the field has grown widely and is essential to the economy for a number of nations all around the world. Countries like Norway rely heavily on chemical production from oil and gas [2]. In 2017, the export of oil and gas accounted for 50 % of the countries total international export [3]. Without regulation and control of chemical production, catastrophic failure may happen which will have enormous negative consequences for the society and economy. Process control can prevent such catastrophic failures and is therefore crucial to Norway's Gross Domestic Product, as well as the Gross National Product as most of it is exported.

1.1 Fundamental of process control

Control is a mechanism where the system is driven to a desired state by exerting an external "force". The field of process control uses information about the system to calculate how it responds to disturbances in favor of creating a model that can counteract disturbances for a real process system. Current usage of control strategies reformulates a digital model of the system with mathematical operations to propose the control model. The model is tuned to best counteract disturbances with optimal performance. For a complex system, a stable control model can be difficult to formulate mathematically. The process of creating a control model from the system model can, therefore, be non-trivial. A wrong control model may lead to oscillatory responses, over and undershoots and in the worst case lead to an unstable system [4].

1.2 Current usage of process controllers

In process industries, a huge variety of different controllers are applied to different systems. The choice of controllers is dependent on the complexity of the system as well as the economic potential and risk assessment. There is no universal strategy for choosing controllers and the choice is often done by engineering intuition. However, a system with low economic potential and low risk

often uses a simpler controller than a system with high economic potential and high risk. Common choices of controllers that are widely used are ON-OFF, PID, Model Predictive Control and adaptive controllers which will be briefly discussed in the following sections.

1.2.1 On-Off controller

ON-OFF controllers are simple, inexpensive feedback controllers that are commonly used in noncritical industrial applications [4]. For ideal ON-OFF controllers, the input is only of two potential values:

$$p(t) = \begin{cases} p_{max} \text{ if } e \leq 0 \\ p_{min} \text{ if } e > 0 \end{cases} \tag{1}$$

The p_{max} and p_{min} denotes the on and off values. e is the offset error. The controller has a constant input of either *on* or *off* depending on the systems state, hence the name. ON-OFF controllers are commonly used for systems in which divergence from optimal operation has negligible effect, unless the divergence is large. The simplicity of the controllers makes them suited as thermostat controllers in houses, but not for more complex systems where a state-value needs to be controlled tightly.

1.2.2 PID controller

PID, or a Proportional-Integral-Derivative controller, is a control loop feedback mechanism which are the most commonly applied control technique in process industries [4]. More specifically, the P- and PI-controller are often the default choice. The controller continuously calculates an error value $e(t)$ as the *Controlled Variable's* (CV) divergence from a desired *SetPoint* (SP) value. The error value is corrected by a proportional, integral and a derivative term, hence P,I and D, which calculated how much a *Manipulative Variable* (MV) should be changed. The mathematical ideal form of the overall control function $u(t)$ can be seen from the equation below [5]:

$$u(t) = u_0 + K_c\big[e(t) + \frac{1}{\tau_I}\int_0^t e(t)\,dt + \tau_D\frac{de(t)}{dt}\big] \tag{2}$$

The τ_I is called the integral time constant and τ_D is called derivative time constant. The K_c is the controller gain which indicates the effect of the controller action on the system. Both τ constants and the K_C are parameters which are chosen based on the systems response to changes. There are many strategies to choose the parameters for control tuning. Methods like Ziegler-Nichols, Smiths predictor and SIMC are among popular choices [6].

The beauty with PID controllers, and why they are so widely used in process industries, comes from their simplistic implementation and linearity. For most process systems a linear approximation is good enough for control models and in many of these systems, PID controllers are implemented. PID controllers are linear controllers which means that for nonlinear systems the usage of PID controllers may result in poor control. For nonlinear systems, more complicated controllers like adaptive control and Model Predictive Control are often used.

1.2.3 Model Predictive Control

Model Predictive Control (MPC) is a form of control in which the current control action is obtained by solving, at *each* sampling instant, a finite horizon open-loop optimal control problem [7]. Compared to PID control, the main motivation to use MPC is the multivariable nature and ability to handle constraints. This allows for better performance with non-linear systems. The optimization problem solved at each time steps is a minimization of a cost function. The optimization problems are constructed with the state-space model as equality constraints along with desired constraints for the MV and CV. For a linear case of MPC the general MPC algorithm with state feedback can be seen below in algorithm 1 with a graphical illustration of the procedure in fig. 1.

Algorithm 1: State feedback MPC procedure

---

for *t=0,1,2,...* **do**
  Measure current state x_t.
  Solve a dynamic optimization problem from the measure state x_t.
  on the prediction horizon from t to $t + N$.
  Apply the first u from the predicted input solution to the system.
end

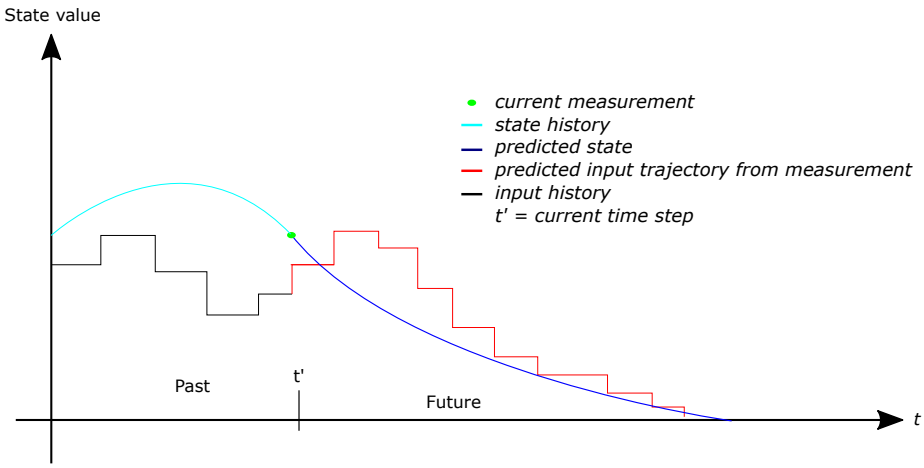

Figure 1: Illustration of the Model Predictive Control procedure at time step t'

The mathematical optimization formulation is presented in eq. (3). This particular formulation is a Quadratic Programming (QP) formulation with a linear model [7]. The following equation is a cost function with the objective being to minimize the deviation between the state x and the setpoint x^{sp}.

$$\begin{aligned} \min_{z \in \mathbb{R}^n} f(x,u) = \sum_{t=0}^{N-1} \frac{1}{2}((x_{t+1} - x^{sp})Q_{t+1}(x_{t+1} - x^{sp}) \\ \text{Subject to} \\ x_{t+1} = A_t x_t + B_t u_t \\ x_0, u_{-1} = \text{given} \\ x^{low} \leq x_t \leq x^{high} \\ u^{low} \leq u_t \leq u^{high} \\ -\Delta u^{low} \leq \Delta u_t \leq \Delta u^{high} \end{aligned} \tag{3}$$

The advantages of using MPC compared to PID are that MPC can work with nonlinear systems and can include constraints. Whereas the PID may change the input from 0 % to 100 % over two-time instances, a constraint to maximal input

changes can be implemented with MPC preventing rapid input changes. MPC provides a lot of flexibility of constraints and implementation. However, tuning of MPC can be difficult and the requirement of solving an optimization problem at each time step can be computational demanding. The time required to solve optimization problems results in MPC often being used for calculating optimal setpoint for controllers in a Real Time Optimization (RTO) layer. However, This optimal setpoint is only as good as the MPC itself and often are adaptive controllers often used to adjust the optimal setpoint when the one calculated is proved to be sub-optimal.

1.2.4 Adaptive control

Adaptive control builds on the concept of *self-optimizing control* where a cost function J is minimized by calculating optimal set values of controllers that are not bounded by active constraints [8]. Adaptive control dates back to the late seventies and has since been in contrast with the regular robust method where it does not need prior information about the control parameters for taking suitable control actions [9]. Adaptive control can generally be divided into two fields dependent on the controller preventing incoming disturbances or reacting to occurred disturbance. The two methods are denoted by feedforward- and feedback adaptive control. There are many different approaches to the concept of adaptive control. One of these is called extremum seeking control which has similar behavior as policy-based reinforcement learning methods.

The optimization on extremum seeking is done by injecting a perturbation to the input value $\hat{u}$, often with a sin function and measuring the shift in J [10]. This allows for evaluation of different $J(\hat{u})$ which are used for shifting the previous predicted $\hat{u}$ towards the local optimum u^*. In other words, the input position SP is adapting to changes in the controlled system for improved performance. An illustration of the extremum seeking controller can be seen in fig. 2.

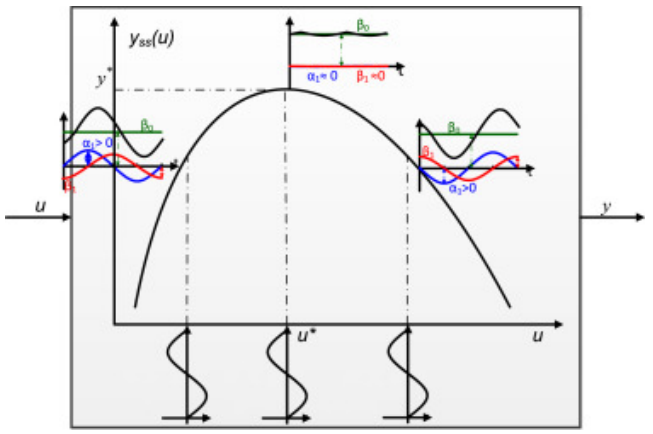

Figure 2: Illustration of the input value adjustment for increased performance for an extremum seeking adaptive controller. Figure aquired from Atta, Khalid Tourkey and Johansson, Andreas and Gustafsson, Thomas Extremum seeking control based on phasor estimation [11]

A common challenge in process control is to find a stable control model for non-linear and discrete systems. Many systems require a lot of knowledge and expertise to implement sufficient control. A question in recent year has been to come up with a general framework which automates this trial and error process of finding a sufficient control model. Recent progress in machine learning have shown promising results in different fields and could potentially be applied in process control as an alternative approach of doing control.

1.3 Machine learning in applications

Machine learning (ML) is a field in computer science which combines mathematical models with algorithms in optimization of specific tasks [12]. The field studies statistical analysis of sets of data with different computational algorithms.

ML is a huge topic with many applications in multiple fields, but generally it can be divided into three different sub-fields: **Supervised Learning** (SL), **Unsupervised Learning** (UL), **Reinforcement Learning** (RL) [13].

- **Supervised learning:** A set of input data is used to predict a set of output data. For each prediction, parameters are adjusted to improve future predictions [12]. In other words, constant feedback from an instructor, indicating right/wrong prediction with the residual between the predicted and correct output. A famous example of SL is to train a machine to correctly classify images of cats and dogs [12].
- **Unsupervised learning:** While supervised learning tries to map a relationship between inputs and outputs, UL has no outputs in the data set [14]. Instead, the method tries to set up a categorical representation of the data with no feedback. In other words, is UL a method for finding relationships within a dataset. This could be the categorizing of individuals based on their personal data for example, for targeted advertisements.
- **Reinforcement learning:** Unlike supervised learning where the data set is assigned before training, RL dynamically collects data during training [15]. The prediction conducted during training affects the future data set and this may result in a lot of predictions without feedback until a criterion is met. RL is the most unexplored sub-field of machine learning, compared to SL and UL, but have seen successful applications in video games and robotics [16].

1.4 Related Work

The concepts of Machine Learning, or ML, dates back to the mid of the 20. century when Arthur L. Samuel first used the terminology in his paper about using machines to play checkers [17]. However, the field ML was not formally used until 1990. In the recent year, ML has surged in popularity due to the improved computational power of computers since ML is computationally demanding. The progress made in the electronic hardware industries and with the storage of large data sets, *Big Data*, has opened up possibilities for companies to use machine learning as a tool for data analysis. Today, ML is mainly applied as an analytic tool for companies with image recognition and statistics.

The field of SL and UL has seen many successful applications in multiple fields. In process control, has especially SL rises as a tool for analysis and optimizing process industries [18]. As for Unsupervised learning have clustering and PCA, which are considered UL methods, been more common approaches in the analysis of process data. However, RL has not seen the same popularity as UL

and SL in the process industries. RL has mostly been explored in niche fields of video-games and cybernetic robotic control. However, some companies like Google DeepMind and OpenAI have experimented with the RL sub-field of ML to create machines that can learn from their own experience.

In 2015 Google's DeepMind presented *AlphaGo* which in 2016 managed to beat the world No. 1 ranked player in the game of Go [19]. This was a benchmark for researchers as Go was considered to be a sufficiently complicated game that no machine would ever beat humans in performance. In other fields like control theory has machine learning improved robotics control. In 2016, the Boson Dynamics group created the walking robot "Atlas" which in 2018 was improved to do human-like parkour stunts [16]. The most recent big application of ML was demonstrated in January 2019 when Google DeepMind released *AlphaStar* which managed to beat top-level players in the computer game *StarCraft* [20]. A game which is considered so difficult that one is required to start practicing from a very young age to compete at the highest level.

In 2009, Jay H. Lee and Weechi Wong published an article where they explored how Approximate Dynamic Programming (ADP) could be implemented as a method for complementing MPC for discrete systems. They argue that ADP has a great potential in the process industries and could potentially for obtaining control policies for stochastic constrained nonlinear systems [21]. An article in the 13th International Symposium on Process Systems Engineering (PSE 2018) touches on the possibilities of using RL in comparison with MPC for future applications [22]. They concluded that RL has the potential to significantly impact the theory and applications in the field of process control.

There are various reasons for not exploring RL as much as SL and UL in process industries. One likely reason is that the implementations of RL require a deep knowledge of computer science since frameworks for RL have not been fully developed as of this date. Tensorflow at the TF Dev summit in March 2019 showcased their first framework which supports simple RL implementations [23]. This thesis explores the possibilities of using RL as a methodology for process controllers and tries to evaluate how RL compares to common controllers in the industry.

Thesis contribution

In this thesis, the field of process control will be combined with RL as a proof of concept for using RL as an alternative way of doing process control. This thesis is a case analysis of different RL algorithms in use for process control. In addition, the theory behind RL will also be presented thoroughly since the field of RL is quite unknown for most process engineers. The RL controllers would be compared to a standard industry controller for evaluation of how the RL-algorithms performs compared to the standard controllers used in the industry. The cases would be created from scratch by mathematical modeling of a tank system and further implemented with a programming language. The simulator, industry controller, and RL-controllers would all be created from generic mass balances and further implemented as a computer software with the Python programming language. This is to have a complete understanding of the dynamic of the system and potentially make changes to the model if needed. However, for the implementation of the RL-algorithm, a deep learning framework would be used for the implementation of the neural networks.

The task of the controllers is to regulate the liquid level in one or multiple tanks and the controller performance would be evaluated with a predefined disturbance. The cases of tank level regulations are more thoroughly introduced in section 3. The aim of the thesis is to evaluate how the trained models control the liquid level to a given SP with incoming disturbance. Due to the size of the code will not the whole code implementation be presented in the thesis. The code used in this thesis consists of $\sim$15 000 lines with only a few third-party dependencies like *numpy*, *pygame* and *pytorch*. The code is made open sourced as future researchers can use this thesis along with the code implementation as a reference for continued research [24].

Terminology and notation

Below is a list of the terminology and notation used in reinforcement learning compared to the termination and notation used in process control. This is due to the terminologies and notations spanning the same concept which can be confusing when coming from process industries.

Terminology used in RL	Explanation
State value (s)	State value (x) or CV
Action value (a)	Input value (u)
Environment	Controlled system, everything which is set by the controller
Agent	Controller, or MV
Reward function (G)	Cost function (J) that also spans positive numbers
Reward (R)	Feedback signal from the controlled system
Hyper-parameters	Parameters which are decided by the engineers
Episode (e)	Defined series of events bounded by an initial state and an end state, in this thesis is the episode a time series from 0 to N time steps.
Training	Automatic altering av parameter for improved system performance
Weights (θ or w)	Parameters in function approximators, e.g. neural networks, which are altered during training
Policy	The predicted input trajectory, given the current state, for the optimal future states
Value function	Metric used to evaluate the goodness of the controlled system being in a state, this is not to be confused with the reward function.
Exploration	Doing sub-optimal input trajectory for sampling new information
Exploitation	Doing optimal input trajectory for fine tuning the parameters for the current optimal trajectory.

Table 2: Explanation of concept used in RL from the perspective of an control engineer. The table is meant as a reference when reading through the thesis

2 Reinforcement learning: A literature review

Reinforcement Learning (RL) concerns the mapping of correct action given the state in an environment [15]. RL mainly consists of two contributors: The *Agent* and the *Environment*. The agent observes a state s from the environment and is able to do an action a. The environment responds to the action and gives back a new state based on the action. Depending on the new state the environment gives feedback to the agent, in this form a reward R, which indicates the goodness of being in the new state. The environment is defined by everything the agent has no control over, including uncertainties. An illustration of the agent acting in an environment can be seen in fig. 3.

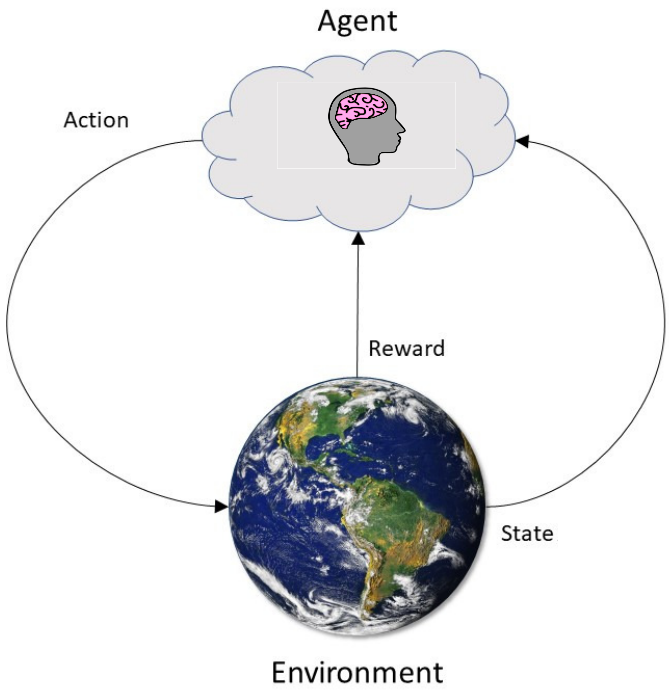

Figure 3: Illustration of the agent doing action a in state s and returned to a new state with reward R

From a control engineering perspective, the agent and environment are the more pedagogic terms for *controller* and *controlled system*. The action a is the controller input u and state s is state x. Sutton and Berto use the terminology *environment* and *agent* because they are meaningful to a wider audience[15]. To be consistent with the literature in RL, this thesis will follow the notation and terminology of Sutton and Berto.

The goal of RL is to evaluate which action is optimal in each state. To be able to find this optimal policy, the agent is required to explore the states and actions. This is called the training of the agent [15]. The agent is acting in the environment and samples up data which it uses to improve its action for future states. The agent improves its policy to learn which action in what state leads to the maximum accumulative rewards. This training process continues until the agent cannot improve its policy which indicates the training is done. This, however, does not indicates that the optimal policy for traversing the environment is found, only that the agent can not improve its performance.

A real-life analogy to RL is the process of teaching a child to throw a basketball into a basket. The agent is the child and the environment is the basketball, the playground, wind and everything the child has no control over. The child is rewarded by if the ball hits the basket. Starting out the child does not know how to throw the ball, let alone how much force he/she should apply when throwing. Exploring different techniques and fine-tuning the shooting arm, the child understands what techniques result in poor performance and which results in a good performance. By letting the child repeatedly shooting the ball and trying to understand the relationship between technique and hitting the basket, the child's performance gradually increases. The same methodology applies to RL where the agent tries out different actions and maps out the maximal expected accumulative reward for possible actions given the current state.

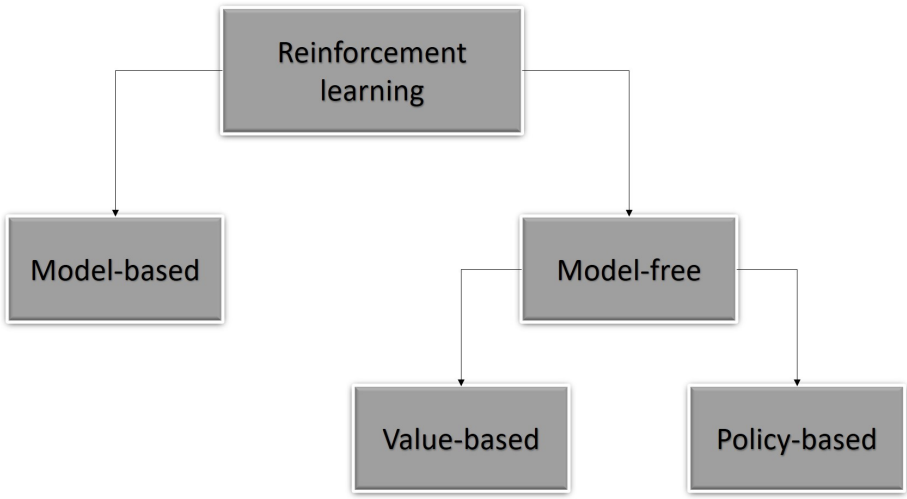

Figure 4: A simple overview of the main methods in Reinforcement learning

Fig. 4 presents a way of separating the different methods in RL. The applied RL methods used in this thesis are only model-free methods as they are best suited for solving the cases presented in section 3. However, to understand model-free RL, knowledge about model-based RL is highly recommended. This main chapter follows the setup in fig. 4 with first an introduction to model-based methods through Markov decision processes. Further, model-free methods will be presented with the main two sub-fields called value-based and policy-based. Finally is a method presented which combines value-based and policy-based.

The methods discussed further in this thesis are all valid options for solving RL problems in theory. The main discussion is often connected to how fast the algorithm solves the RL-problem, as well as the robustness of the application and how feasible they are in practice. There is not one method which is better than all other for all RL problem. Dependent on the dynamics and the goal, different methods may yield equal results. The evaluation of the different methods is often based on how the problems can be formulated as a Markov Decision Process.

2.1 Markov Decision Processes

Markov Decision Processes (MDP) is a discrete stochastic time process often presented as a stochastic model. MDPs are the mathematically idealized form of the RL problem for which precise theoretical statements can be made [15]. It expands on the classic Markov chains which is a statistical model that represents transitions from state s_i to a new state (s_{i+1}) [25]. Where the transition between states in Markov chain models is solely dependant on probability, the transition in MDP is also dependent on the previous action in the given state. In relationship with RL, MDP is the formal problem which RL tries to solve.

In a MDP an agent can interact with an environment at each sequence of discrete time steps $t = 0, 1, 2, 3, ...$ All states in a MDP at time step t is a subset of all possible actions in the MDP, $S_t \in \mathcal{S}$. This basis also applies to an action at each time step $a_t \in \mathcal{A}(s)$, where a in the selected action from the subset $\mathcal{A}$ of possible actions. The transition from state s_t to s_{t+1} can formally be written as:

$$p(s', a) = Pr\{S_t = s', A_{t-1} = a\} \tag{4}$$

The Pr and p is the probability of the transition. This can be reformulated to the *state-transition probability* for all states transitions for state s. A simple finite MDP process can be seen in fig. 5.

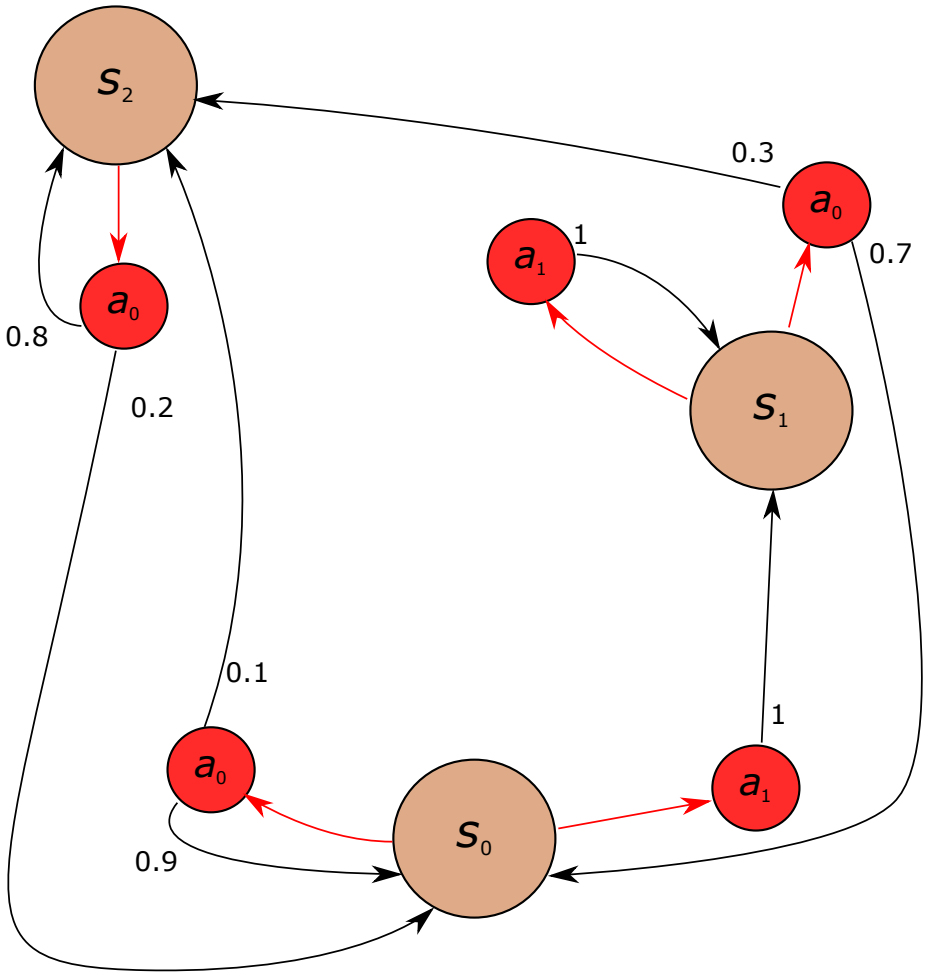

Figure 5: A finite Markov decision process with three possibles states. The numbers besides each action node denotes the probability of the state transition given the action a_i and state s_i

MDP in RL also includes a Markov Reward Process (MRP) which also is an expansion of the Markov chain [25]. Where the Markov chain only feeds back the new state, a MRP also returns a numerical *reward* R_{t+1} for the transition

to the new state S_{t+1}. Introducing the MRP to the MDP, the total probability transition is shown in eq. (5) and eq. (6). The '|' is the conditional probability.

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{5}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{6}$$

The goal in RL is to learn an agent to maximize the future rewards G_t, which is the sum of the collected rewards as seen in the following equation.

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \tag{7}$$

To prevent the agent to not only look at the immediate rewards at the next time step, all rewards for a set of action are summed together to evaluate G_t. This implementation of only looking at the end goal may prompt the agent to do actions that yield low immediate rewards as the agent only maximizes the sum of all rewards. This approach may be suited for many applications but often results with a non-stable training process [15]. This is due to small updates to the G_t may change the behavior of the agent drastically and result in poor behavior. However, a combination of maximizing immediate and long term rewards is desirable, due to long term uncertainty, and a discount factor γ is often introduced. The discount factor is a hyperparameter that weights the importance of future and immediate rewards. The agent may value immediate rewards, but also look ahead. The mathematical formulation of rewards in MRP is usually included with a discount factor from time t to time step T as shown in eq. (8).

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{8}$$

A $\gamma \approx 0$ makes the agent favor immediate rewards close in time while a $\gamma \approx 1$ favors long term rewards. In fig. 6 the discounting of G_t can be seen for three different choices of γ values. The third subfigure with gamma equals 0.9, showcases how the rewards after time step 50 are weighted around zero which results in the agent accounting for only 50 future time steps.

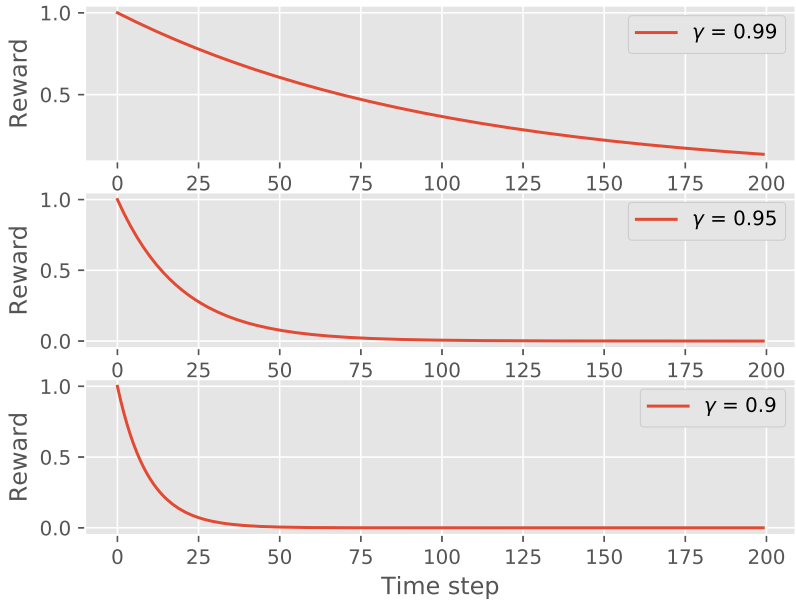

Figure 6: Weights of discounted future rewards from current time step $t = 0$ to time step $t = 200$ with three different discount factors

An assumption of the MDP is that the information from previous states has no effect on the current state. In other words, the process is memoryless between each transition. This property is called the *Markov property* and this assumption lays the foundation of using RL to solve stochastic transition processes like MDP [15]. The reason for this being desired in RL is that the agent only needs to consider the current state when evaluating the action which gives the maximal expected reward. This action which is expected to yield the maximal rewards is also called the optimal *policy* π_*.

2.1.1 Policy, value-function and Bellman equation

Almost all reinforcement learning algorithms involve estimating a *value functions* of a state or an state-action pair that estimates how good it is for the agent to be in a given state or doing a certain action in the given state [15]. The estimation is used to evaluate the *policy* π, the mapping from action a to state s, $\pi(a|s)$, formally denoted as $\pi : A \rightarrow S$.

To evaluate if a policy is good or bad, a *state value-function* v_π or a *state-action value function* q_π is often introduced. The mathematical definition of $v_\pi(s)$ and $q_\pi(s, a)$ are defined as eq. (9) and eq. (10) respectively.

$$v_\pi(s) := \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_t = s\Big] \tag{9}$$

$$q_\pi(s, a) := \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_t = s, a_t = a\Big] \tag{10}$$

$\mathbb{E}$ is the notation for the stochastic expected value. From a control engineering perspective can the value function be seen as a cost function. In comparison computes the cost functions only negative values and of optimizing the cost function is to reduce it towards zero. However, in RL may the value and state-value functions compute positive and negative numbers with the maximum beeing dependent on the architecture of the RL implementation. As stated earlier, the goal of RL is to find the optimal policy q_π for all feasible states. Solving this problems means to find a policy that maximizes the accumulated rewards over a long time series. A new policy $\pi^{'}$ is said to be better than the current policy π if it's expected to return a greater total rewards than the current policy $v_{\pi'}(s) > v_\pi(s)$ [15].

The optimal policy and state value function are the ones where no policy can yield greater rewards. The optimality is denoted the asterisk symbol $*$. The mathematical definition for the optimal policy and state-action function can be described as:

$$v_*(s) = \max_\pi v_\pi(s) \text{ for all } s \in \mathcal{S} \tag{11}$$

$$q_*(s, a) = \max_\pi q_\pi(s, a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{12}$$

An important property of the value functions is whether they satisfy recursive relationships with the policy π [15]. This implicates that a policy π can be evaluated from the value function v_π as shown in equation eq. (13).

$$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \tag{13}$$

Eq 13 is called the *Bellman equation* for v_π and shows the relationship between the value of an action-state and the values of succeeding action-states [15]. This relationship can be exploited to find the optimal policy $q_*(s, a)$ as stated with the *Bellman optimality equation* for $v_*(s)$.

$$\begin{aligned} q_*(s, a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \Big| S_t = s, A_t = a \Big] \\ &= \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big] \end{aligned} \tag{14}$$

In theory, any MDP can be solved with the Bellman equation and the optimal policy can be found with the bellman optimality equation. However, for real life examples this process be can too ambiguous and infeasible in terms of computational efficiency. In the following sections different methods are presented as efficient solutions to MDP. The following methods involves storage of values of tables and later methods with value function approximations will be introduced.

2.2 Tabular methods for solving MDPs

The simplest form for solving MDP with reinforcement learning is called *tabular solution methods*. The idea is to store values of different states and the action taken in tables for all action-state pairs. The method uses one or more tables where each row and column represents the value of taking an action at a given state. The method may also be done by only storing the state values without the associated action. However, action-state value terminology covers the implementation of state values. Therefore, in this section, only state-action values are considered.

As the agent explores all possible action-states, the values in the table will iteratively converge towards the Bellman equation of optimality [15]. Since the MDP is a combination of discrete and continuous values can the optimization problem be categorized as a Mixed Integer Programming (MIP). In theory,

all MDPs can be solved with tabular solutions, but with large action and/or state spaces this method will require large tables to store all possible action-state-values. For most real-life systems this is infeasible in practice. With the requirement of infinite large tables, an approximation is needed. This will be further discussed in section 2.3. but for small finite MDPs however, *dynamic programming* is a common choice for finding the optimal policy.

2.2.1 Dynamic programming

Dynamic Programming (DP) is a method for solving a complex problem by dividing the problem into sub-problems [26]. The Sub-problems are solved individually and their value is stored for future evaluations. This memorization of sub-problems allows for usage of previous experience when solving over-lapping sub-problems. In other words, DP remembers it's past to avoid solving the same problem multiple times.

DP in the context of RL refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP [15]. As discussed earlier, the assumption of a perfect model of the environment is not always feasible in RL, but the fundamental approach DP uses to solve MDP can be generalized for all RL-problems. DP searches for the optimal policy by utilizing the value function $V(s)$. The approach combines *policy iteration* and *value iteration* to find the optimal policy.

Policy iteration consists of two parts, *policy evaluation* and *policy improvement*. The two methods work together as a reliable algorithm for finding the optimal policy for a given MDP. The algorithm explores all states in the MDP and updates the transition probability when the value function converges. The policy evaluates a value function for the given policy based on the Bellman equation. Value iteration, on the other hand, differs from policy iteration by updating the value function before it converges. Policy- and value iteration are popular methods for exploring all possible state transitions for solving finite MDP. The methods are often described by the term *Generalized Policy Iteration* (GPI) which lets the policy improvement and policy evaluation interact with each other. A visualization of GPI can be viewed in fig. 7

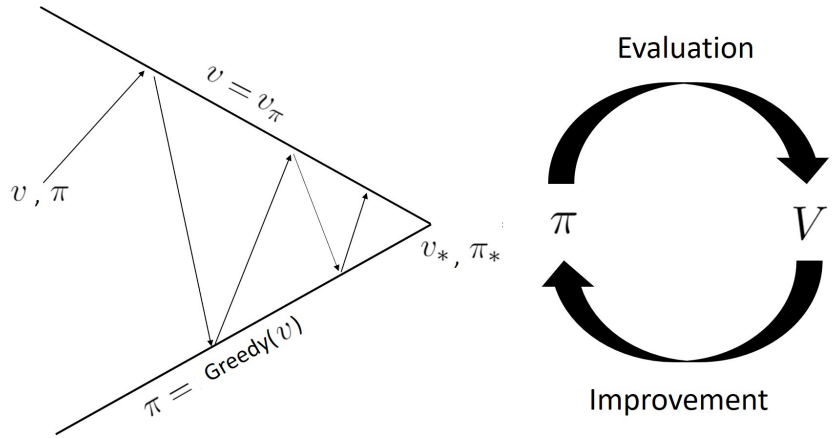

Figure 7: Visulization of the Generalized Policy Iteration (GPI). The two figures illustrated how the policy and value is evaluated and improved iteratively throughout training

The left part of the image represents the convergence of evaluating both the value function and best-found policy (greedy). The evaluations will eventually converge towards the optimal value and policy function v_* π_*. The right part shows the cycle of shifting between policy evaluation and policy improvement.

After evaluating the optimal policy, the agent is considered fully trained and further be used for *control*. Control is a term for when the agent no longer is exploring suboptimal actions and only takes the action which it finds optimal. However, finding the optimal policy requires exploration of the entire MDP state space. The optimal ratio between policy evaluation and policy improvement is usually unique for each problem. It's desirable to only prove the policy but without sufficient evaluation, the best policy will not be discovered. This trade-off is famously called the *Exploration VS Exploitation* dilemma.

If the GPI spans a large MDP, a model based method like DP is not efficient enough as is needs to search through the whole MDP for finding the optimal policy. Alternate methods like *Monte Carlo* or *Temporal differences* are rather the default choice as they do not require to search the whole MDP.

2.2.2 Monte Carlo methods

Monte Carlo methods can arguably be considered the most popular choice of learning method for estimating value functions and discovering optimal policies [15]. Monte Carlo methods use the property of episodic trials to evaluate the value function of a given state. In the fields of mathematics, Monte Carlo is a class of algorithms which repeatedly samples data from a system to evaluate the property of the system [27]. This is often done to evaluate the distribution and probability of different scenarios. In RL, this method implies that the feedback signal of rewards are only prompted when the episode terminates. This means that the transitions of multiple state-actions are made before the training is prompted. The mathematical expression for a simple every-visit Monte Carlo method can be expressed as eq. (15)

$$V(S_t) \leftarrow V(S_t) + \alpha\big[G_t - V(S_t)\big] \tag{15}$$

$V(S_t)$ is the value at state S at time t. G_t is the return at time t and α is a step size parameter. This process of waiting until the episode terminates is often referred to as a *rollout*. For each rollout, the value function is adjusted until the Bellman optimality is satisfied. Monte Carlo methods require a termination of the episode before G_t is can be evaluated. This is convenient for systems where the feedback is only prompted at the end of the episode. Monte Carlo is often used to solve systems like chess and videogames [15]. However, for systems where the rewards are constantly given during the episodes, an alternative approach is to make a prediction before the episode is terminated. A method which is called *bootstrapping* and is utilized in temporal differences.

2.2.3 Temporal differences

Temporal Differences (TD) is a combination of Monte Carlo and dynamic programming ideas [15]. Like DP, the value function is constantly updated as the agent moves through the action-state space. However, in contrast with DP, TD does not require a model of the environment. Like Monte Carlo, TD can learn directly from the exposure to the environment without a MDP model. The value function is updated after each action without waiting for the final outcome. This process is also commonly known as bootstrapping. The initial guess of the value is constantly altered with the experience of the agent. The simplest version of TD is known as TD(0) and can mathematically be shown in eq. (16)

$$V(S_t) \leftarrow V(S_t) + \alpha\big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big] \tag{16}$$

where R_{t+1} is the resulting reward at time step t and γ is the discount factor. Where as the Monte Carlo method waits until the episode terminated and then evalutes the value of the experienced action-states, TD predicts constantly the value as it traverses the action-states and updates the value function based on the TD error δ_t as seen in eq. (17)

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{17}$$

2.2.4 N-step and TD(λ)

The method of temporal differences also expands to other methods know as *n-step* TD and TD (λ). Where the TD(0) only uses the R_{t+1} to update the estimate, n-step and TD(λ) includes n number of rewards in the update. The N-step method can be described as the generalization of TD(0) and Monte Carlo. Where TD(0) updates after each feedback, the N-step method waits for n-steps before updating, hence the name. By setting n to infinite, the n-step method will always wait until termination before updating which is the definition of the Monte Carlo method. An illustration of the difference can be seen in fig. 8.

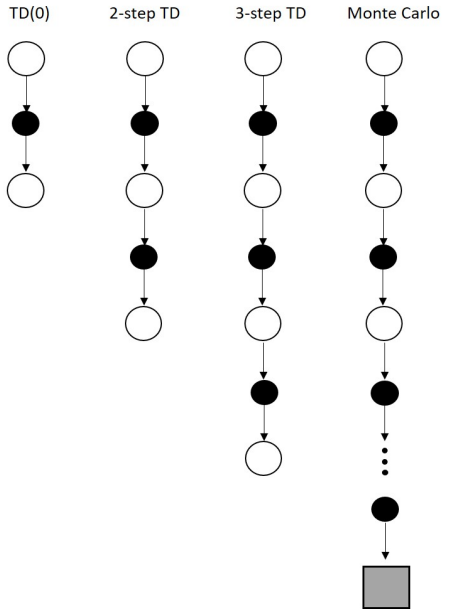

Figure 8: n-step method for temporal differences showing the connection between TD(0) and Monte Carlo with the choices from $n = 0$ to $n = \infty$

Another way of generalizing Monte Carlo and TD is the popular TD(λ). In contrast to N-step methods, TD(λ) has significant computational advantages. This is due to *eligibility traces*, denoted with λ, where λ is a trace-decay parameter $\lambda \in [0, 1]$ [15]. While the n-step method computes n vectors in each episode, utilizes TD(λ) only one trace vector. The long term vector can be split up into multiple shot term vectors which can be computed parallel during training. An eligibility trace of ($\lambda = 1$) is the Monte Carlo method and is TD(0) for ($\lambda = 0$). The usage of eligibility traces has shown to improve efficiency during training when solving RL [15].

2.3 Function approximations for solving MDPs

As presented earlier in section 2.2, the action-state spaces are often too large to be solved using tabular methods. For instance, in the game of chess with 32 pieces and 64 possible positions, the total number of possible states is roughly

$64 \cdot 63 \cdot 62 \ldots 33 \cdot 32 = 1.55 \cdot 10^{55}$, not accounting for illegal states. Creating a table that can store $10 \cdot 10^{55}$ values is intractable. This problem is further increased for continuous action- and state spaces where the real numbers $\mathbb{R}$ cannot be discretized without loss of information. The whole mathematical background for RL builds on the MDP as a discrete stochastic process. To represent the problem as an MDP, a finite number of states is required. However, for continuous state spaces is this not possible. This problem is commonly known as *Curse of dimmentionality*. This curse applies to both the state values as well as the state-action values.

A simple approach for solving this curse is to discrete the action and/or the state space. By lumping the ranges of the possible state space into a finite number, a tabular method can be used. However, this method creates multiple distinct states that generalize subdomains into one state. A discretizing of the state space loses information about the original MDP and creates a new MDP. Since the agent is trained on the new MDP will, this will not necessarily work for the original MDP as the new MDP contains less information than the original.

Another method that is commonly used is to utilize a function to approximate the values of the state given experiences from similar states. This maps the state to an approximation of the state value function. This means that all values from the state value can be used without needing to treat each value as a unique state. In other words, the values from the full state space are used to create a generalization of the state space. To create these approximations a function is required. This function can either be linear or non-linear.

2.3.1 Linear methods

A linear function approximate $f(x)$ can be described by a feature vector $\phi(x)$ and a parameter vector θ, which is often referred to as a weight vector in the context of ML. A general linear function approximator is presented in the following equation.

$$f(x) = \theta^T \phi(x) \tag{18}$$

The function approximator can be described as a predictor that tries to evaluate the observed value of $\hat{f}(x)$. From the field of supervised learning, the variable x is called *features* and can also be described as the input to the function [12]. In RL, the features are often the state which the agent experience. However, the features may consist of multiple states and/or a transformation of the states

which the agent has experienced. This is often done to simplify the agent's task of finding important observations.

Where the tabular methods use a table of all possible state values, a function approximator instead is used to approximate the large table. Instead of updating elements in a table, the weights of the function are adjusted to better fit the true observed state values. Typically, this error is calculated from an error function like mean squared error over a distribution μ of all features as shown in the equation below.

$$MSVE(\theta) = \sum_{s \in S} \mu(s) \big[V^{\pi}(s) - \hat{V}(s, \theta) \big]^2 \tag{19}$$

The distribution μ indicates how much each state is valued in the error function. The adjustment of the weights in eq. (18) is done by minimizing a loss function, like eq. (19), with unconstrained optimization.

The advantage of the linear function approximator is that they are computationally efficient and easy to understand how the weights are adjusted during training. A commonly used linear function approximator in process control is the weighted least squares regression. However, for many non-linear systems, a linear function approximator cannot capture a sufficient generalization of the state values. Instead, a non-linear activation function like Neural Network, rather used. This is further presented in section 2.3.3.

With the minimization of the cost function is the field of mathematical optimization is introduced. Mathematical optimization spans multiple sub-field, but since the goal of the function approximates is to minimize the residual between the predicted and experiences action-state values without any constraints, the sub-field of unconstrained optimization mainly used.

2.3.2 Unconstrained optimization

Unconstrained optimization spans all optimization problems which are not affected by constraints [28]. Meaning that all values θ of the objective function f are valid. In order to improve the function approximator, the difference between the predicted state-action values and the observed state-action values is minimized. The method, like any other optimization method, tries to find the extreme point of a function [28]. Whereas optimization, in general, is formulated with the Karush–Kuhn–Tucker (KKT) conditions, unconstrained optimization

does not include the Lagrange multiplier λ [28]. Consequentially, the following list is the resulting KKT-condition for unconstrained optimization problems [28].

1. If θ^* is a local minimizer and f is cont. differentiable in an open area around θ^*, then $\nabla f(\theta^*) = 0$. This is the first condition.
2. If $\theta^*$is a local minimiser of f and $\nabla^2 f$ exists and is cont. in an area around θ^*, then $\nabla f(\theta^*) = 0$ and $\nabla^2 f(\theta^*)$ is pos. semidef. This is the second condition.
3. Suppose that $\nabla^2 f$is cont. in an area around θ^* and that $\nabla f(\theta^*) = 0$ and $\nabla^2 f(\theta^*)$ is pos. def. Then $\theta^*$is a strict local minimiser of f. This is the third condition.

The conditions presented above simply states that the minimum of a function is the point were a function can't be decrease any further by small adjustments, which is the definition of a *local* minimum. A *global* minimum would be the local minimum which has the lowest objective value compared to all other local minimums. However, there is no guaranty for a local minimum beeing a global if the objective function is non-convex [28]. However, a control engineer is often not interesting in finding the global minimum, only a local minimum which is sufficient for the application.

In comparison with the rewards function in RL, the goal of the function approximator is not to maximize the episodic rewards, but rather predicting the value the agent will earn for each state-action. The error from the loss function is gradually decreased with the use of an optimizer. The function is decreased until at least the first and second KKT conditions are satisfied. The third condition is a sufficient condition and does not always have to be satisfied.

The common choice of optimizer for ML is called *Stochastic Gradient Descent* (SDG) [29]. SDG is a commonly used algorithm in ML which evaluates an approximation of the gradient of the objective function and changes the parameters to decrease the function value [29]. The nature of the algorithm is to gradually decrease the cost function calculated by observed and predicted values. The gradient of the cost function or an approximation of the gradient is calculated from the objective function, indicating how θ must be changed to decrease the cost function. A visualization of the gradient descent method which shows the iterative process used in unconstrained optimization in fig. 9.

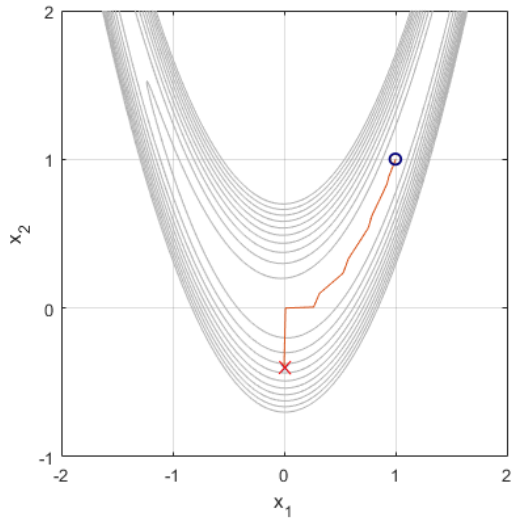

Figure 9: The iterative minimization path for a gradient descent algorithm on the Rosenbrock function [30]

The gradient evaluated from the objective function indicates only in which direction the function decreases. Whereas gradient descent methods often compute the gradient from the objective function, SGD collects randomly selected samples that are used to compute an approximation of the gradient of the objective function.

This means that to satisfy the KKT conditions the values of θ must change in the direction of the calculated gradient. How much the parameters change in each iteration is called the *step length*. This step length is referred to as *learning rate* in the context of ML. Each iteration in optimization is referred to as a *epochs* in the context of ML.

In supervised learning, a phenomenon called *overfitting* may occur. This refers to situations when the trained model "memorizes" the labels of the training data that the model would perform poorly on unseen data [12]. In RL this is prevented by continually having a dynamic set of training data. The agent collects more data as it moves through the state-action space, and in addition, only one epoch is conducted before updating the data collection.

As discussed earlier in section 2.3.1 is not always a linear function approximator sufficient enough for creating a generalization of the environments value function. In these situations are often a non-linear function approximator used. A common non-linear function approximator which is heavily used in ML is called Artificial Neural Network.

2.3.3 Artificial Neural Network

Artificial Neural Network (ANN) or neural network is a biologically inspired framework that is used in multiple applications of ML. The concept of neural networks dates back to the mid 20th century when Warren McCulloch and Walter Pitts created a computational model called threshold logic [31]. ANN consists of multiple algorithms that all can be represented by multiples nodes that are connected in a graph, which is called the network [12]. The Network is a graph of nodes and the connected vertices are the weight parameters. There are many types of ANN used for different ML problems like Feedforward Neural Network (FNN), Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN).

The FNN is the simplest type of a neural network which consists of at least an input layer and an output layer. In addition, may multiple hidden layers between the input and output layers be included. A neural network with multiple hidden layers is called a deep neural network. The field of *Deep Learning* is evolved around this concept and a lot of research is done to understand fully how deep neural networks learn. A side note is that a neural network with only a few hidden layers is called a *shallow network*. An illustration of a shallow complete feed-forward neural network is shown in fig. 10. In the case of a complete FNN, all the nodes in one layer are connected to all the nodes in the next layer with one edge between all nodes.

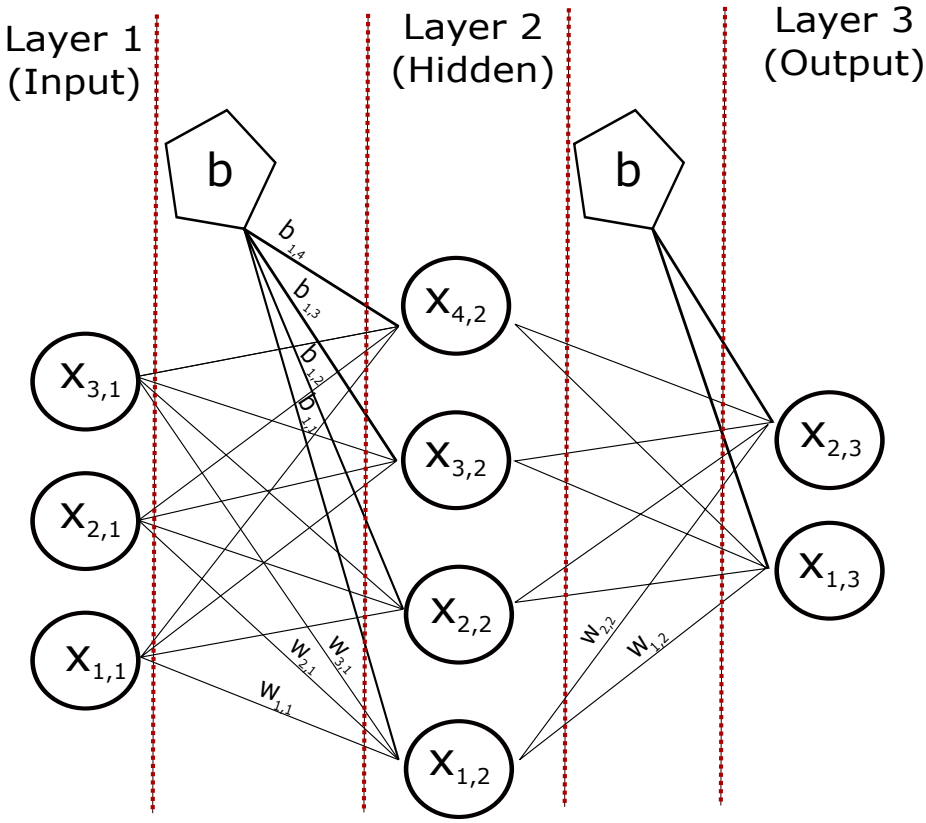

Figure 10: A feedforward artificial neural network with one hidden layer, w and b are the weight and bias parameters

A neural network consists of multiple linear combinations of the nodes in the graph. The value of one node is the sum of all connected nodes, described mathematically in equation eq. (20).

$$x_{i,j+1} = \sum_{i=1}^{N_i} w_{i,j} x_{i,j} \tag{20}$$

Without introducing non-linearity to the network, a neural network will be considered a linear function approximator. This is not preferable for non-linear

systems as discussed earlier in section 2.3.1. Introducing an *activation function* σ transforms the linear equation in eq. (20) into the non-linear equation in eq. (21)

$$x_{i,j+1} = \sum_{i=1}^{N_i} \sigma(w_{i,j} x_{i,j}) + b_{i,j} \tag{21}$$

In which $x_{i,j}$ is the i numbered node value in layer j. N_i is the number of nodes in layer i. w is the weight parameter. With the introduction of an activation function, different nodes will be deactivated, $x \sim 0$, depending on the values of the weights. This may cause that all values of the nodes in a layer are deactivated and the output of the network is zero. With introduction of a bias parameter b can this phenomenon be prevented. The bias parameter is not affected during the weight adjustment of the vertices which are connected in the same layer and may "shift" the activation function left or right to prevent deactivation.

There are multiple activation functions that have been tested and tried in recent years. In particular, the activation function *Rectifier Linear Unit* (ReLU) on hidden layers has proven to yield the best results [32]. Another previous popular activation function is the sigmoid function. The sigmoid and ReLU activation function are both shown in fig. 11 as well as described by the following equations.

$$f_{\text{Sigmoid}} = \frac{1}{1 - e^{-x}} \tag{22}$$

$$f_{\text{Relu}}(x) = x^{+} = \max(0, x) \tag{23}$$

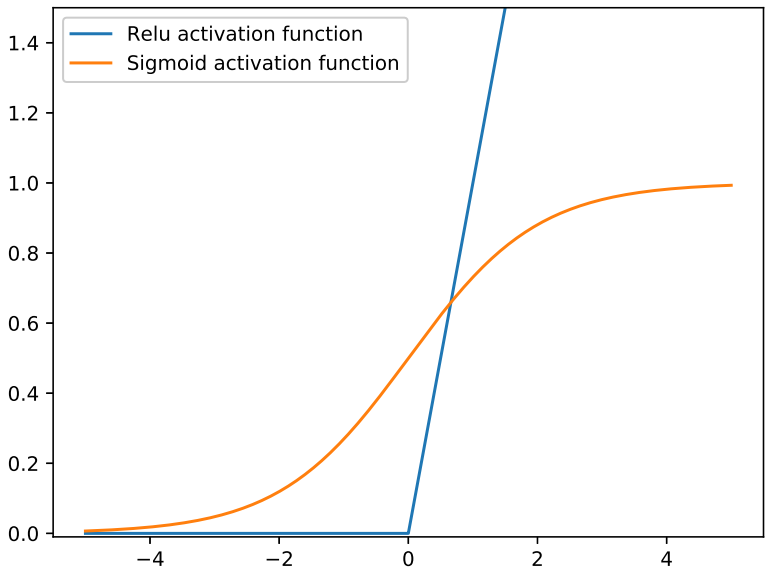

Figure 11: Visual representation of the ReLu and Sigmoid functions

There are many ways to implement the function approximator in RL and in the following section, two methods are emphasized as they are the basis for solving RL in large state-action spaces. The first one being *value-based* and the second one *policy-based*.

2.4 Value-based methods

Value-based methods evolve partially around learning the value function $V^{\pi}(s)$ or a generalization of the -value function [15]. By evaluating the value of the actions at each state, the agent can simply choose the action which yields the maximal value. The policy is therefore defined through the value-function and this is indirectly altered with the value function. The optimal policy for each state is the argument of the maxima of the possible actions as illustrated in

fig. 12 and described by eq. (24). The equation can be categorized as a MIP as the optimization problem is a combination of continuous and discrete values.

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}^{\pi}(s') \tag{24}$$

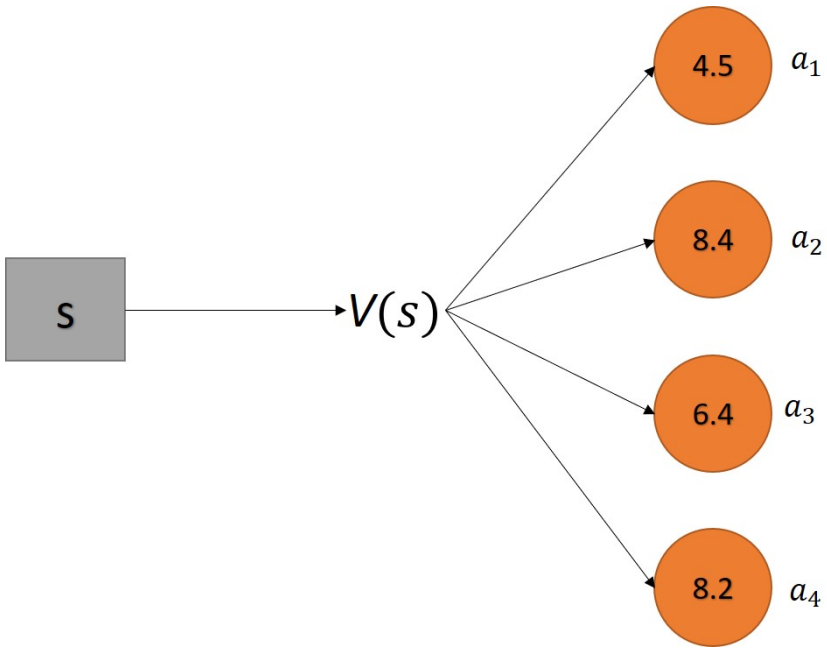

Figure 12: A simplified illustration of the value based method. The value function V calculates the expected value $V(a_i)$ for each action a_i in the current state s. The greedy policy is in this figure a_2

However, to evaluate the policy a model of v^{π} or q^{π} is required. For smaller MDPs can this be solved by using TD and update the table-values in each iteration. However, with large MDPs is this not feasible and this is where the function approximator comes into play as a method for model-free RL. The function approximator calculates a Q-value which is an approximation of the state-action value function $q\pi(s,a)$. The Q-value is the output from the function approximator, called Q-function, that can predict the succeeding states q-value

and evaluate the goodness of different state-action pairs. The Q-function can be seen in the following equation.

$$Q^{\pi}(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) Q^{\pi}(s', \pi(s')) \tag{25}$$

The output of the approximator in value based methods corresponds to the values of the action the agent may choose. The introduction of the Q-function allows for learning without a model of the MDP, called model free methods.

There are mainly two distinctions in value-based methods; *on-policy* and *off-policy*. On-policy covers the traditional TD and Monte Carlo methods which was introduced in section 2.2.2 and section 2.2.3. The idea behind on-policy is to solve the value function of the policy π that the agent is currently following. A commonly used on-policy algorithm for TD-control is called *Sarsa* [15].

Off-policy, on the other hand, is trying to evaluate the policy of one policy while the agent is currently following a different policy. This may seem counter-intuitive, but this allows for the usage of all the policies from the previous iterations to predict the current policy. on-policy however, is based only on the current policy. The advantage of this approach is that off-policy helps to stabilize the performance during training. The off-policy method named *Q-learning* is currently a widely used algorithm for solving large state space MDP.

2.4.1 Q-learning

Q-learning or Deep-Q-Network (DQN), as it is more recently referred to, is an algorithm that dates back to 1989 [33]. The method builds on the Bellman equation shown in eq. (13).

$$Q(s_t, A_t) \leftarrow Q(s_t, A_t) + \alpha \big[R_{t+1} + \gamma \max_a A(s_{t+1}, a) - Q(s_t, A_t) \big] \tag{26}$$

Q-learning became popular when the company DeepMind company published a paper describing DQN for creating human behaviors in multiple Atari games [34]. The team at DeepMind improved the Q-learning with the usage of a deep neural network as Q-function, hence the name Deep-Q-network. The general implementation is shown in the algorithm 2.

Algorithm 2: Psudocode for the Q-learning algorithm

Initialize Q(s,a), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for *episode=0,1,2,...* **do**
  Initialize S
  for *t=0,1,2,...* **do**
    Choose $a_t \in A$ from s_t using policy derived from $Q(s_t)$
    Take action a_t and observe R_{t+1}, s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a A(s_{t+1}, A_{t+1}) - Q(s_t, A_t)\big]$
    if S_{t+1} *is terminate state* **then**
      End current episode
    else
      Continue
    end
  end
end

A tendency Q-learning has during training is a wide variance in performances. This is caused by the argmax function shown in eq. (25). Let's say action a_0 represents an action of going left, actions $a_1 - a_9$ represent the range from left to right and a_{10} represents going right. If both state-action values of a_0 and a_{10} are close to one another in terms of Q-value, a small update to the Q-function however may lead to drastic changes in greedy policy. This behavior may also happen if the agent evaluates a state from which it has no general experience. The agent has no knowledge about how the actions are related and views all as independent actions. Another problem is that value-based methods require a finite number of actions. This means that a continuous action space needs to be discretized before training. This is often not desirable for controlling real systems.

2.5 Policy-based methods

While value-based methods evaluate the policy through the value estimation, policy-methods directly alter the policy without consulting the value function [15]. The method seeks to maximize the performance gradient $J(\theta)$ with respect

to the policy parameters. $\widehat{\nabla J(\theta)}$ is the gradient of the stochastic performance measurement which approximates the directing which increases the likelihood of future rewards, given the policy parameters θ. The update is conducted with a simple gradient accent as shown in the following equation.

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{27}$$

Introducing the stochastic estimator J allows for non-deterministic policies. This is useful for state aliases where multiple states are indistinguishable. The available actions provide a probability given the current state and based on future rewards. The probability actions with high rewards increase while the probability of actions with low rewards decrease. However, this requires a relation between the performance gradient and the effect from the policy parameters which is unknown for the performance gradient. Fortunately, this can be solved theoretically with the policy gradient theorem in eq. (28) [15].

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_s q_\pi(s,a) \nabla_\theta \pi(a|s,\theta) \tag{28}$$

The policy gradient theroem above simply states the relationship between the gradients performance with respect to the policy parameter. $\propto$ means "proportional to" and is the on-policy distribution following the current policy π. For episodic cases can this theorem be presented as an Monte Carlo expectation as seen in eq. (29) where the distribution is substituted out with the gradients expected performance following the current policy.

$$\nabla J(\theta) = \mathbb{E}\big[\sum_s q_\pi(s,a) \nabla_\theta \pi(a|s,\theta)\big] \tag{29}$$

When implementing the episodic case of policy gradient theorem Eq. (29) is more often expressed as a function of the cumulative rewards G_t as seen in eq. (30).

$$\nabla J(\theta) = \mathbb{E}\big[\sum_{t=0}^{T-1} G_t \nabla_\theta \pi(a|s,\theta)\big] \tag{30}$$

The gradient $\nabla J(\theta)$ can be approximated numerically by substituting the gradient ascent equation in eq. (27) to represent a iterative way to change the

parameter θ for improved performance as seen in the policy update functions eq. (31) and eq. (32)

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{31}$$

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \ln \pi(A_t|S_t, \theta_t) \tag{32}$$

In which $\nabla_\theta \ln \pi(A_t|S_t, \theta_t)$ is the eligibility vector explained in section 2.2.4. Eq. (31) and eq. (32) are the same equation with the simple algebraic relation $\nabla \ln x = \frac{\nabla x}{x}$. The reason for reformulating eq. (31) is due to the natural fit into functions approximator as the log probability distribution. By comparing the gradient and the log probability, an increase in $J(\theta)$ is the increase of the probability of the action which yields higher rewards.

The log probability is a maximum likelihood of a set of actions for the agent. However, for a continuous action space, the probability is the expected action with the range between 0 and 1 for higher rewards. By only using the outcome of the log probability, the agent may get stuck doing the same action for different states. This can be solved by implementing a method for exploration. One way is to use an action disturbance which forces the network to explore. One example of action disturbance is to use a distribution around action value with an action variance. Another method is to use the $(\epsilon - greedy)$ method which calculates a probability ϵ doing a random instead of the perceived optimal action.

This implementation has a lot of similarities with the adaptive controller which was presented in section 1.2.4. Both try to shift the cost function towards the optimum. However, where extremum seeking is only bounded by the immediate rewards the policy gradient may look further ahead and do sub-optimal actions for greater cumulative rewards. A common implementation of the policy gradient method is the algorithm called *REINFORCE*.

2.5.1 REINFORCE

The cycle of the REINFORCE algorithm is to collect the rewards from the rollout and shift the probability of the bad actions based on the return value G_t. The algorithm builds on the Monte Carlo methods since the training only occurs between each episode. The general implementation of the REINFORCE algorithm can be seen in algorithm 3.

Algorithm 3: Pseudocode for the REINFORCE with baseline algorithm

---

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$
for *e_i=0,1,2,...* **do**
  Rollout the episode and collect S_t, A_t, R_t following π
  for *t = 0,1,2,... e_i* **do**
    $G_t \leftarrow$ return from step t
    $\theta_{t+1} \leftarrow \theta_t + \alpha^{\theta}\gamma_t\delta\nabla_\theta \ln \pi(A_t|S_t,\theta_t)$
  end
end

γ is the discounting factor, α is the learning rate (step length) and e_i is the episode number. This algorithm can further be improved by using an arbitrary baseline $b(S_t)$ for the experienced episodic returns. This is done by substituting G_t in eq. (32) with $G_t - b(S_t)$. This will often improve performance and decrease the training time [15]. The base line is commonly dependent on the state value $S(S_t, w)$ and is often updated by a step size parameter α^w, where w is denoted the state-value weights.

Policy gradient methods have mainly two weaknesses when solving RL-problems, local minimums, and inefficiency [15]. Since the policy is updated iteratively with eq. (32) the gradient ascent tends to find local minimums for which the agent's expected return cannot be improved. This can be solved by implementing a probability of doing sub-optimal actions for exploration and/or changing the learning rate. However, this requires a large search space which follows into the second weakness that is inefficiency. Policy gradient gradually shifts the expected value based on the return. Since the return from a state has a high variance the process of determining the expected return of a given state requires large numbers of trials. For large systems, this may be computationally unfeasible. By introducing an estimate of the return and by bootstrapping the return value, a new method called *Actor Critic* is introduced.

2.6 Actor-Critic

Actor-Critic (AC) combines value-based and policy-based methods to create a more efficient algorithm for solving large action- and state space MDPs. Whereas the REINFORCE may utilize a state value function as a baseline, the AC takes the baseline one step further by bootstrapping the state value estimate and updating the value estimate from the TD error in eq. (17), an equation which was introduced in section 2.2.3.

An analogy to AC is a child (actor) playing basketball and a teacher (critic) evaluating the child's performance. The child tries different techniques and the teacher gives feedback to the child how to adjust its technique. The teacher also adjusts its perception of the environment for improved feedback to the child. The teacher can learn that climbing upwards would result in falling down and prevent the child from experiencing the fall if he/she sees the child climbing upwards. An overview of the general actor-critic methods is seen in fig. 13.

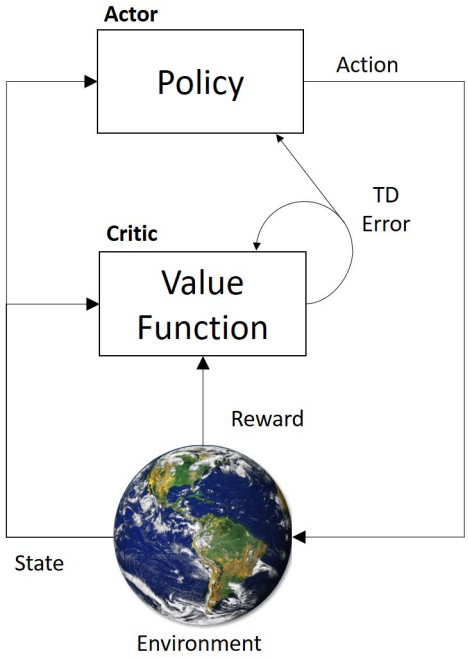

Figure 13: Caption

In the previous chapter a baseline $b(S_t)$ was included in policy gradient theorem eq. (30) for improved performance. Actor-Critic builds on the baseline in policy gradient by switching out the baseline with the Q-function from eq. (25), which was introduced in section 2.4. The introduction of the Q-value is what distinguishes policy gradient with a baseline from AC. This introduction results in a different function for the cost function gradient $\nabla J(\theta)$.

$$\nabla J(\theta) = \mathbb{E}\big[\sum_{t=0}^{T-1} Q_w(s,a)\nabla_\theta \pi(a|s,\theta)\big] \tag{33}$$

The subscript w for the Q-value is notation for using a parameterization of the Q-function, like a function value approximator (often ANN). Using the Q-function instead of the cumulative reward have shown to improve performance [15]. This is due to bootstrapping which have shown to decreases the variance when analyzing dependent observations [35].

AC is considered the general method for all RL algorithms which utilizes both value function and policy update. The generalization spans multiple algorithms that have seen successful applications. Eq.(33) is the general implementation of the AC, and with different parameterizations of the Q-function have many different AC-algorithms in the recent years been created which are now considered state of the art. One example of one of these iterations is called Advantage Actor-Critic (A2C). The method substitutes the Q-function by an advantage value A as seen in the eq. (34). The advantage value A is the difference between the predicted Q-value the experienced one.

$$\nabla J(\theta) = \mathbb{E}\big[\sum_{t=0}^{T-1} A(s,a)\nabla_\theta \pi(a|s,\theta)\big] \tag{34}$$

A2C is considered the basic implementation of the AC in RL. However, many iterations have built on this for improved learning methods. These include Asynchronous Advantage Actor-Critic (A3C), Deep Deterministic Policy Gradients(DDPG), Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) [36]. Actor-Critic method is a field that is rapidly evolving and new implementations are frequently developed.

3 Case description

In this thesis, three cases were created as MDP problems to be solved with reinforcement learning. A tank containing liquid was chosen as the system component for the environment model for all the cases. The container has one inflow and one outflow, where the outflow can be controlled by a valve. An illustration of the system is seen in fig. 14.

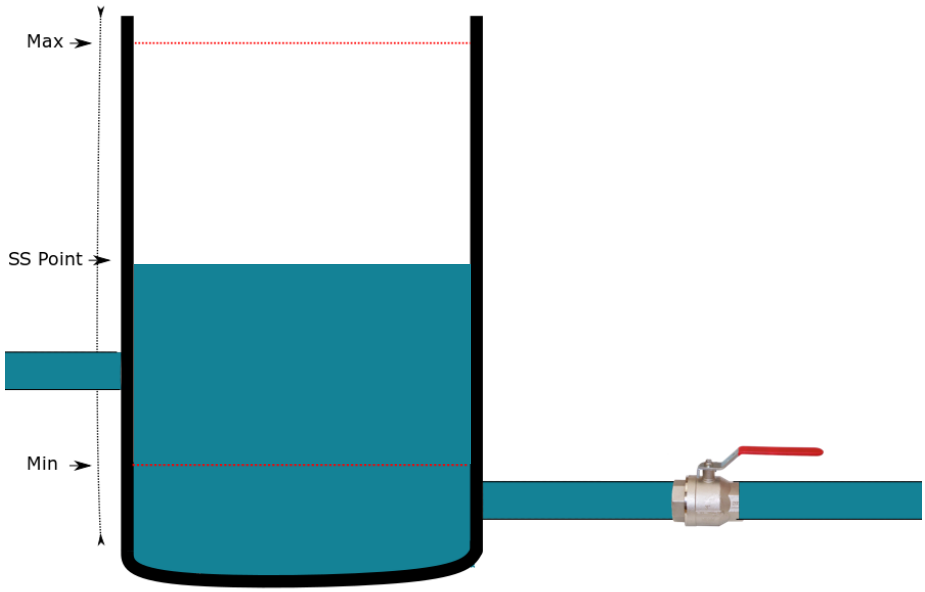

Figure 14: Illustration of a tank containing liquid with one disturbance inflow and one outflow with a choke valve

A model describing the dynamics of the tank was derived from a generic mass balance shown in appendix A. The final model for the tank dynamics with respect to the liquid height is presented in the following equation.

$$\frac{d(h)}{dt} = \frac{1}{\pi r^2}\left(q_{in} - f(z)A_{out}\sqrt{2gh + \frac{\Delta p}{\rho}}\right) \tag{35}$$

The parameters h and r are the height and radius of the tank, ρ is the density of the liquid, q is the mass flow, A_{out} is the cross area of the outflow pipe, Δp is the difference between the tank surface pressure and the pressure at the end of the outflow pipe, and $f(z)$ is the valve opening.

The goal of the RL-implementation in this thesis is to learn an agent to control the liquid level in the tank by opening and closing the valve. The environment is considered the tank level which will be denoted by state value s. The tank level is disturbed by an inflow q_{in} which is calculated by a bounded Gaussian distribution of the inflow for the previous time step as shown in eq. (36). This method for simulating real-world time series is called *Gaussian random walk*. The inflow bounds for the Gaussian flow was set to not exceed a max flow q_{in}^{max} or a min flow q_{in}^{min} as in eq. (36).

$$q_{in,t+1} \sim \mathcal{N}(q_{in,t}, 0.05 \cdot q_{in,0}) \tag{36}$$

The tank has a maximum and a minimum liquid level, h^{max} and h^{min}, which are considered hard constraints. A state value outside the constraint boundaries is considered a system failure or a termination state. The agent's action is the percentage valve opening z ranging between $0 \leq z \leq 1$. The environment is therefore considered all dynamics except the valve position z. The valve equation was set to be linear resulting in the action value as the valve position.

The tank system was not implemented with any system delay. This is because the thesis emphasizes on the RL-methods and not the tank model implementation. In addition would the implementation of system delay results in loss of the Markov property since each state-action would be dependent on the previous state-action. However, this could be solved by using Auto-Regression models (AR), e.g a Recurrent Neural Network (RNN). However, this would increase the complexity as a Long Short-Term Memory (LSTM) architecture in the neural network would be needed.

The first case is to control the liquid level in one tank. The agent should learn to control the liquid level to a given setpoint value between the hard constraints. To utilize episodic rewards, an episode of maximum time t^{max} was introduced. If the state value reached one of the two hard constraints or reach $t = t^{max}$, the

state would be considered a termination state and the episode would end before starting a new. The implementation of controllers and training algorithms can be seen in section 4.

After training the agent should be able to control the liquid level in the tank around the set point regardless of the incoming disturbance. This acceptable deviation from setpoint was chosen to be between two soft constraints. The training would not terminate if the state breached one of the soft constraints but rather considered to perform poorly. After training, a predetermined disturbance with a disturbance step at $t = \frac{t^{max}}{2}$ is used to evaluate the performance of the different controllers. These results are presented in section 5. The predetermined disturbance can be seen in fig. 15

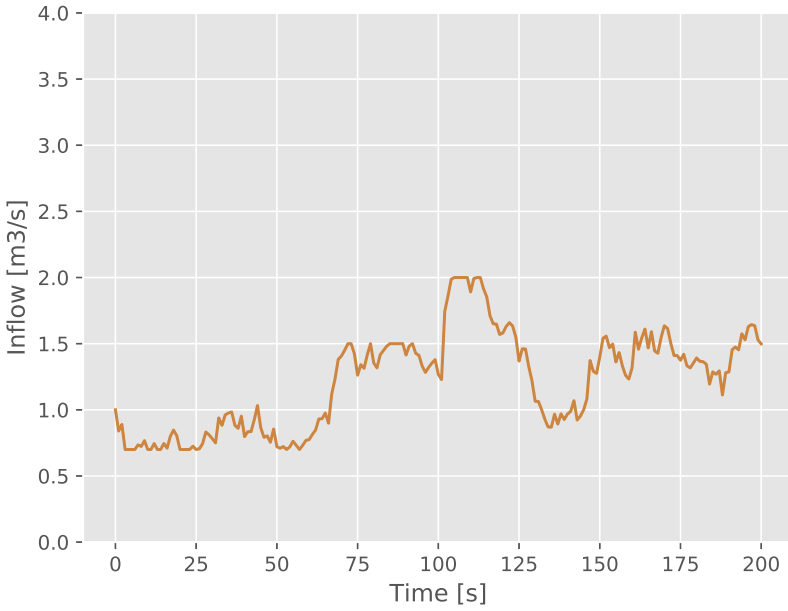

Figure 15: Predetermined inflow disturbance to be used in evaluation of controllers

The second- and third case consists of, respectfully, two and six tanks in series.

Therefore the outflow of the first tank is considered the disturbance to the second tank. Where the state observation in the first case only was the liquid level, in multiple tank series the state also includes the valve position of the previous tank in the series. A visualization of two and six tanks can be seen in fig. 16 and fig. 17. Whereas the disturbance in the first tank will only range between q_{in}^{max} and q_{in}^{min}, the valve position for one tank would determine the disturbance into the succeeding tank in the series.

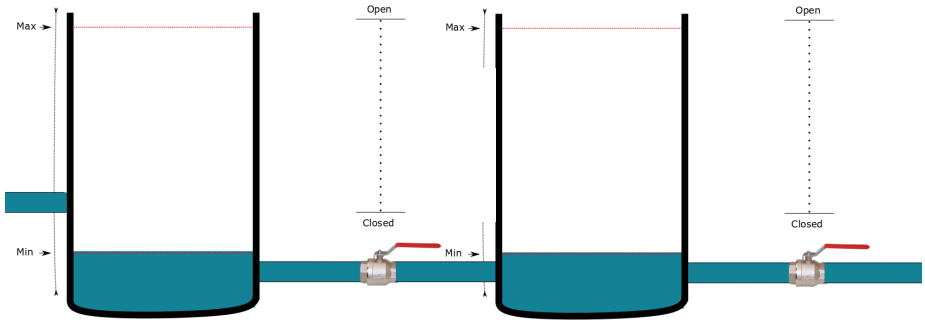

Figure 16: Visualization of the case of two tanks

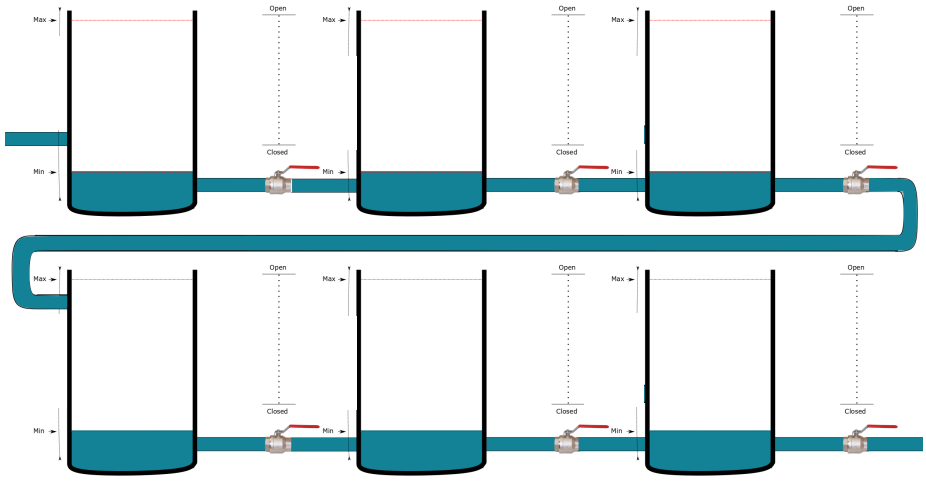

Figure 17: Visualization of the case of six tanks

4 Implementations

Python was chosen as the programming language for the implementation of the simulator and controllers. This is because of the widely documented deep learning frameworks which are compatible with python. A P-controller was chosen as the industry controller since level control is an integrating process and a P-controller is the preferred choice in the industry for these processes. Even though the RL-problem, in theory, could be solved by trial and error on a real tank, the process would require decades of training. Consequently, a simulator was rather chosen for efficient training.

The whole project is presented in the authors Github repository "Reinforcement learning in process control"[24]. The results from the evaluation of the different controllers are presented in section 5.

4.1 Implementation of simulator

The goal of the simulator was to create a digital environment that the agent could train on. The main object of the thesis was the implementation of the RL-methods and not the tank modeling. Consequently, the model was only created to sufficiently simulate a real tank system. The overall structure of the simulator can be seen in fig. 18.

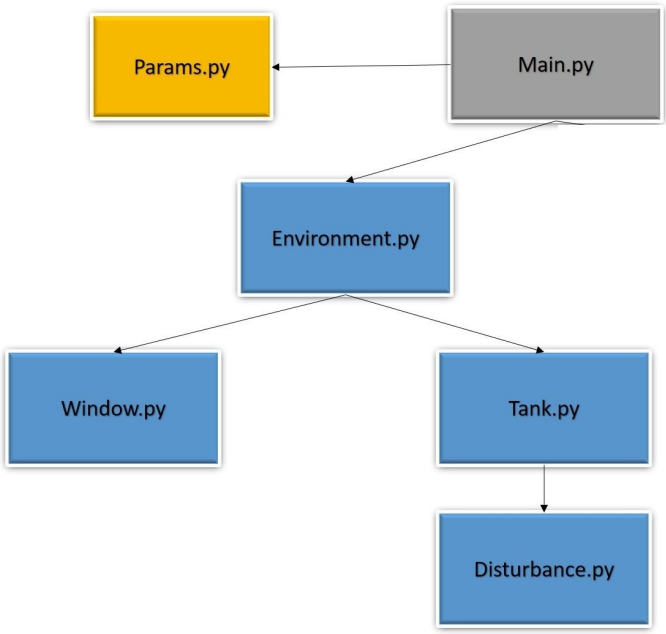

Figure 18: The overall python project structure of the tank system simulator

The simulator was created with implementation of the tank equation eq. (35) as a class in *tank.py*. The disturbance flow q_{in} was not embedded in the tank class, but rather as a separated class in *disturbance.py*. This was done for better code-structure. The tank class was implemented in the *environment.py* which was set up to include an action z from *main.py* and return a new state s along with a reward R. This setup creates the RL-description presented in fig. 3 from section 2.

To better visualize the agent's actions during an episode, a visualization of the simulator was created in *window.py*. The class renders the tank height and the valve position for each time step during training. The simulator used a linear valve equation with no time delay. However, an action delay was implemented for a better graphical illustration of the actions of the agent. The size of the tanks in the series was set to have the same parameters except for the tank

heights. The parameters used in the simulation can be seen in appendix B.

The implication of using a simulator is that the environment is only an approximation of the real environment. This implies that errors in the simulator may affect the performance of the controller if further implemented in process industries. However, as this thesis is about exploring the possibilities of using RL in process control, this concern is not important. If RL is to be applied in real life applications then this becomes important.

To evaluate the different RL-methods, a standard controller from the industries would yield a reference of the agent performance when doing control. A P-controller was chosen as the controller of reference.

4.2 Implementation of P-controller

Level control is an integrating process which means an input-step would lead to a higher new steady-state value, compared to a self-regulating behavior that would converge back to the original steady-state value. P-controllers are commonly used for controlling integrating processes and is why it was chosen as the compared industry controller. Additionally, P-controllers are one of the most commonly used controllers for regulating tank levels. This is because small offsets in the tank level have small to no effects on the overall plant's performance. Besides, a P-controller is inexpensive and easy to tune for good performance.

The controller was modelled from the tank equation, shown in eq. (35) from section 3. The full derivation can be seen in appendix C. The gain, time constant and closed loop gain K_c, were calculated with the Simplified Internal Model Control (SIMC) tuning rule [6] to be the following:

$$K_c = \frac{\tau}{K\tau_c} \tag{37}$$

$$K = \frac{h^{nom}}{f(z^{nom})} \tag{38}$$

$$\tau = \frac{\pi r}{f(z^{nom})A_{out}2g} \tag{39}$$

The controller was created with the model *p-controller.py*, independent of the *environment.py* to best replicate an agent doing actions on the environment. All

hyper-parameters like tank height and nominal disturbance flow were defined in a single file called *params.py*. The implementation of the P-controller with the simulator can be seen in fig. 19.

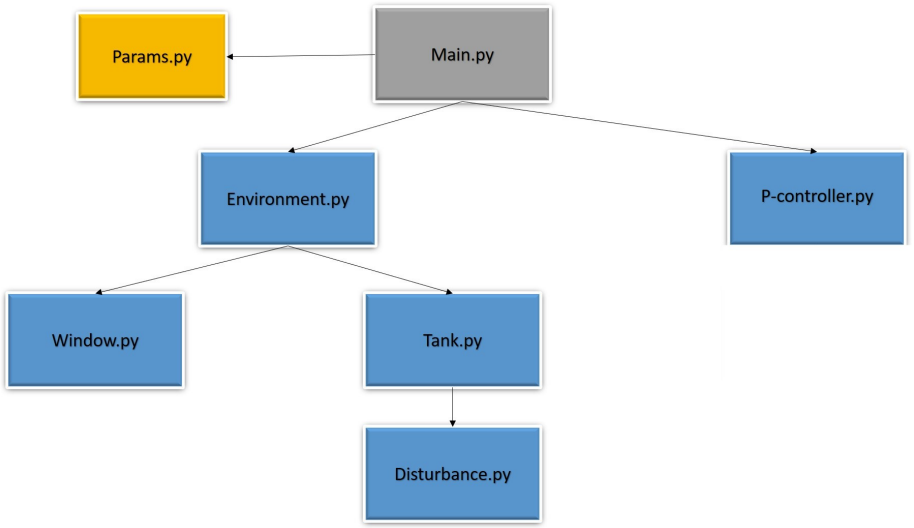

Figure 19: The overall python project structure of the p-controller implementation

The closed-loop constant τ_c is often chosen from a set of different criteria. From eq. (37) it can be observed that a small τ_c results in a high K_c which results in an aggressive controller. Whereas a high τ_c results in smooth control. The choice of τ_c is very dependent on the system requirements and often hard to evaluate. In this thesis, multiple simulations with different τ_c were evaluated of the liquid level's set-point deviation. The deviation was used to calculate the cost function Mean Squared Error (MSE) seen in eq. (40) which was previously introduces as eq. (19) in section 2.3.1.

$$G^{tot} = \frac{1}{N} \sum_{n}^{N} \sum_{t}^{T} (s_t - s^{set})^2$$
$$G^{tot} = \frac{1}{N} \sum_{n}^{N} G_n \tag{40}$$

The equation above sums opp the cumulative cost G as the Sum of Squared Errors (SSE) for N number of episodes. The cost is calculated by the deviations from set-point at each time step t. A script was made for evaluating the response for different τ_c. For each τ_c values, a set of $N = 100$ simulations with $T = 200$ time steps were carried out with random disturbances. Sampling time was set to 5 seconds meaning the inputs were only conducted at each fifth seconds. The reason for not choosing 1 second was for better visualization of the agent's action when compared to the RL-controllers. The τ_c with the lowest MSE was used as the tuning parameter. The set-point was chosen to be 50% of the tank height (5m). The script used for tuning one tank is presented in appendix F.1. The τ_c evaluation for the first tank can be seen in fig. 20.

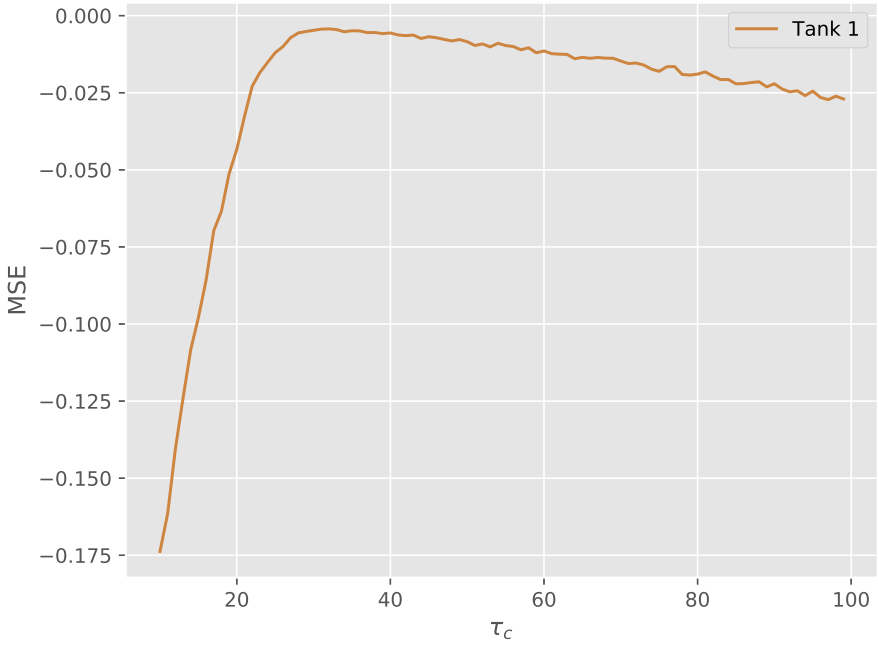

Figure 20: Error function evaluation of the p-controllers tuning parameters τ_c for the first tank

For tuning of multiple tanks, a sequential tuning script was made for tuning of tank_i followed by tuning of tank_{i+1}. This could be done starting from the first until the last tank since all the tanks were only affected by the tuning of the previous tank. The script automatically tuned the first tank and used the best evaluated τ_c for tank_i when tuning tank_{i+1}. The plotted evaluation of the tanks in series can be seen in appendix D. The best performed τ_c for the different tanks is presented in table 3.

Tank	τ_c	Tank radius	Tank volume
1	31	10 m	3141 m^3
2	109	8 m	2010 m^3
3	97	8 m	2010 m^3
4	156	7 m	1540 m^3
5	98	9 m	2545 m^3
6	211	8 m	2010 m^3

Table 3: Best performed τ_c parameters evaluated from the tuning script along with the tank parameters. The heights of the tanks are all set to 10 m

These best performed τ_c parameters were evaluated with the predetermined disturbance. However, the evaluation plot showed an aggressive controller. The initial evaluation of the 1 tank controller can be seen in fig. 33.

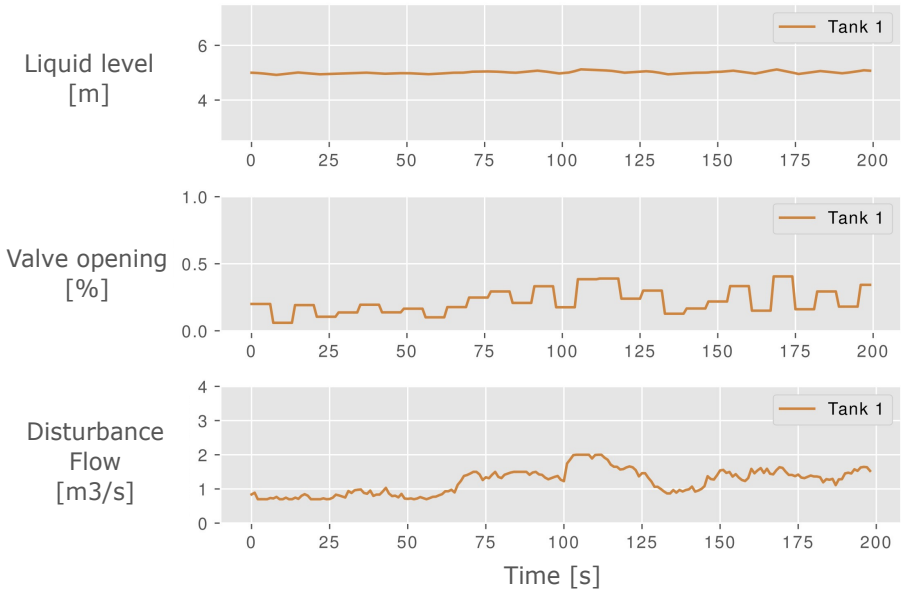

Figure 21: P-controller control for one tank evaluation plot

From the valve opening plot, the input trajectory can be observed to shift quite often. This is not desirable as succeeding components would observe constant disturbance steps which could lead to instability. This can be further seen for the evaluation plot of the two and six tank case in appendix D. The τ_c parameter were increased to have more smooth p-controller and is presented in table 4.

Tank	τ_c	Tank radius	Tank volume
1	200	10 m	3141 m^3
2	250	8 m	2010 m^3
3	250	8 m	2010 m^3
4	250	7 m	1540 m^3
5	250	9 m	2545 m^3
6	250	8 m	2010 m^3

Table 4: Adjusted τ_c parameters for the p-controllers along with the tank parameters. The heights of the tanks are all set to 10 m

The choice of τ_c of 200 and 250 is the relatively high performance in MSE around these values as seen in appendix D. The reason for having a more aggressive tank 1 controller than the subsequent controllers is to counteract the initial disturbance so it does not propagate backwards to the succeeding tanks. The evaluated performance of the adjusted p-controllers can be seen in section 5.

4.3 Implementation of RL methods

The value- and policy-based learning algorithms Q-learning and REINFORCE were implemented for all three cases. A2C was implemented for the single tank case to explore the possibility of combining value-based and policy-based learning methods. The A2C was implemented late during the project period and due to time restrictions was this method not thoroughly explored for all the tank cases. For all implementation of RL, a python class of the agent and a network was created, named *agent.py* and *network.py*. The network consists of a function approximator which was chosen to be non-linear as a neural network. A linear function value approximator would probably be sufficient for this MDP, but since the goal of the thesis was to explore the possibilities of using RL in general, a non-linear function approximator was applied. A non-linear function approximator would signify that the general implementation could even

be applied to more complex systems than those explored in this thesis. An illustration of the RL-project overview is shown in fig. 22.

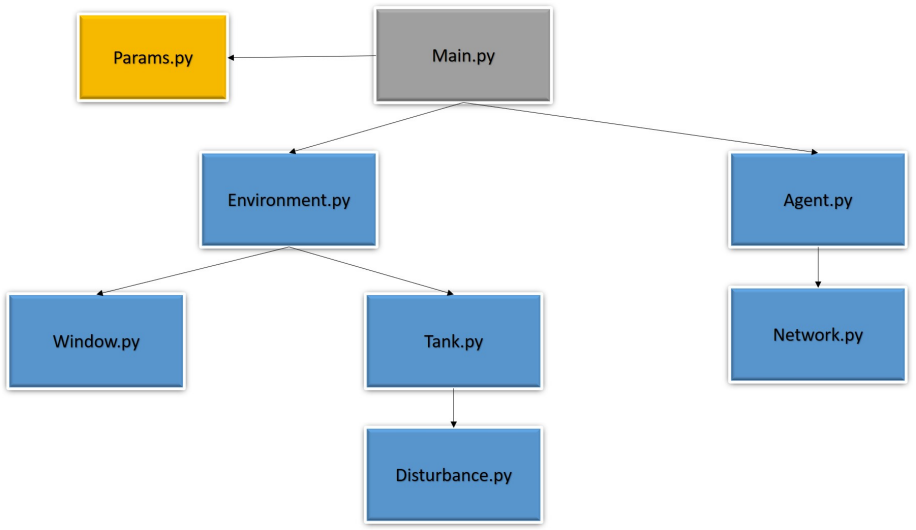

Figure 22: The overall python project structure of the RL implementation

The observation which the environment gave was initially only the liquid height, which corresponds to only one input node for the network. To simplify the agent's task of learning, the meaning of the observation from the environment was transformed into an input state consisting of three sub-observations. This feature engineering consisted of dividing as much information from the observation into multiple independent sub-observations. These sub-observations consisted of the percentage of liquid height, change in liquid level, and a boolean value which indicates if the liquid level is above the setpoint.

The sub-observation of the percentage of liquid height was chosen as it directly dictates the return value from the reward function. The change in the liquid level could be crucial information for the agent in calculating the reward for the succeeding time-steps. The reason for including the boolean value as a sub-observation was to differentiate between the upper and lower constraints when evaluating optimal actions.

One instance of feature engineering was the calculation of the change in liquid

height. This change was calculated from the difference in the last two observed liquid heights. This can be viewed as the D-term in a PID-controller. For the multi-tank system case, the valve position of the previous tank was also added as a state observation for the succeeding tanks. All state inputs used in this thesis for the networks can be seen in table 5

Network inputs	Description
Node 1	Percentage of the liquids tank height
Node 2	Gradient of the liquid height
Node 3	Boolean value determine if the liquid is above 50%
Node 4	Valve position of the previous tank

Table 5: Input nodes as the state observation used during training

In addition to the state from the environment, a reward function must be defined before training. The function takes in an observed and/or a non-observable state and outputs a signal. The signal is a numerical reward and is to be designed for the implementation of the RL-method.

4.3.1 Design of reward functions

The signal from the reward function indicates the goodness of an agents action. Where MPC includes constraints which the controller is not allowed to violate, RL, instead, allows all action-states but rather penalizes them with different magnitude. In many ways is the reward function in RL similar to Merit functions is optimization. Merit functions use a penalty function that allows for violating constraints in order to increase the convergence of finding optimum [28]. However, Merit functions can be implemented in MPC as soft-constraint with the introduction of a penalizing variable. In RL is this penalty a low reward signal which consequentially results in the agent not favoring these actions.

The design of the reward function proved to be difficult in terms of the agent's performance. Initially, the MSE cost function was chosen. The algorithm implemented for the MSE is shown in the algorithm (4) with visualization of the reward function in fig. 23.

Algorithm 4: Algorithm used for MSE reward function

At time step t collect new normalized state s from action a.
if $s > s^{max}$ ***or*** $s < s^{min}$ **then**
| Let Reward = -1
else
| Let Reward = $-(s - s^{set})^2$
end

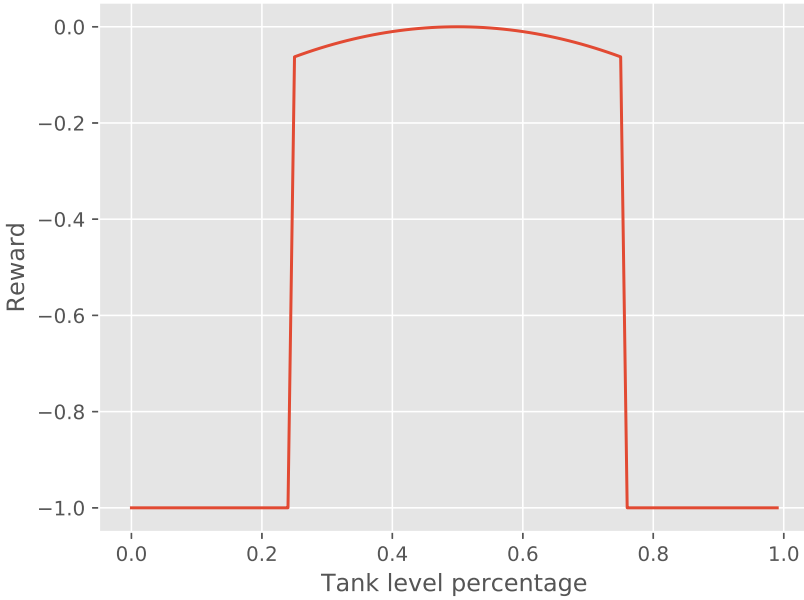

Figure 23: Visualization of the reward function presented in algorithm (4)

The reward function was proven to yield low performance for the value-based

methods. The reason for poor performance for the MSE reward function with Q-learning is probably because the reward between the hard constraints is a continuous function. This could be complex to learn directly as optimal and sub-optimal actions could be almost indistinguishable. Small updates to the function approximator would often result in a change in optimal policy which would prevent the agent from converging towards a decent policy. If the agent did not account for future rewards could potentially this reward function be utilized, but since the agent sums the discounted predicted future rewards when evaluating the current state was this reward function not desirable for DQN-controller.

However, the usage of the MSE reward function did show promising results when applying the REINFORCE and A2C learning methods. The reason for the REINFORCE and A2C showing better performance is most likely due to the usage of directly policy adjustment instead of indirectly policy adjustment with the value function.

A different reward function was rather proposed with only 3 different numerical return values. The reward function algorithm is shown in algorithm (5) with visualization in fig. 24.

Algorithm 5: Algorithm used for reward function

At time step t collect new normalized state s from action a.
if $s > s^{max}$ *or* $s < s^{min}$ **then**
  Let Reward = -10
else
  if $s < 40\%\, s^{min}$ ***or*** $s > 60\%\, s^{max}$ **then**
    Let Reward = 0
  else
    Let Reward = 1
  end
end

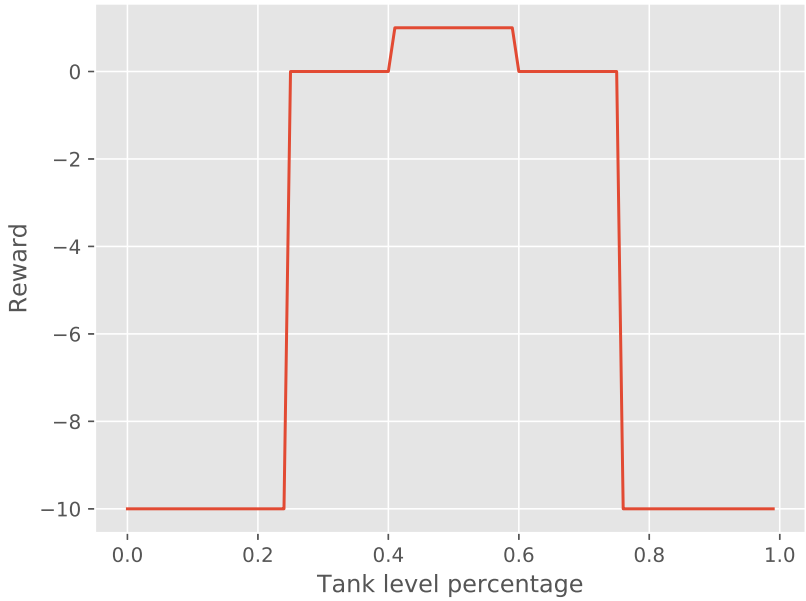

Figure 24: Visualization of the reward function presented in algorithm (5)

The reward function presented above simply states that the agent receives a positive reward if the liquid level is between 40- and 60 %. These values, therefore, creating the desired zone for the liquid level and the goal was therefore to zone control the liquid between these soft constraints. If the agent breaches one of the hard constraints, it is punished with a negative reward and the episode terminates. However, breaching the soft constraints would not result in the termination of the episode but rather giving the agent zero rewards. This reward function proved to be easier for the agent to learn in most of the tank cases and was therefore chosen as the default option throughout the thesis. The setpoint of 50 % would be the point with the largest margin for error and with long term evaluation would the agent hopefully drive the liquid level to this setpoint.

To reduce the variance in reward during training, a Moving Average (MA) was used as the metric for evaluating the training performance of the different im-

plementations. The MA was set to calculate the mean reward of the 50 latest episodic rewards.

The implementation of the different RL-methods was split up into the three different RL methods that were explained earlier in section 1.3. The goal was to train an agent to keep the liquid level between the soft constraints for 90 % of the episode. This would mean reaching a 90 % perfect score before saving the model. Saving the model implies to save the numerical values of the network parameters. Initially, the value-based Q-learning method was explored as it is one of the most explored methods in the field of deep reinforcement learning. Additionally, the learning process is more intuitive to implement and supervise compared to the policy-based and actor-critic methods, which is explained in section 2.5 and section 4.3.4 respectfully.

4.3.2 Value base method implementation: DQN

One tank case was first considered for training and the action range was discretized into a set of 10 positions. The Q-learning method described in eq. (26) was implemented with a neural network as the function approximator. Two hidden layers with 5 nodes and ReLU hidden layer activation function was chosen for the DQN and the reward function in algorithm (5) was used for training. This implies that with an episode of, a reward of 200 is the maximum possible reward to achieve. The exploration vs exploitation problem explained in section 2.2.1, was approached with the epsilon-greedy method [15]. This method sets a hyper-parameter ϵ called exploration rate, which corresponds to a probability of doing a random action. The exploration rate is decreased after each training phase by an exploration discount factor until it reaches a minimum exploration rate ϵ_{min}.

The training performance proved to be very sensitive to different hyper-parameters. The hyper-parameters which had a huge effect on the training performance were the discount factor γ, minimum explorations rate ϵ_{min} and learning rate α. This is to be expected as RL has shown to be very sensitive to these parameters [15]. The training performance for the final DQN one tank implementation is shown in fig. 25.

Figure 25: Experienced rewards for one DQN controller during training. The plots shows the MA episodic reward of the 50 latest episodes

The training performance from fig. 25 showed an initial spike in the learning process as the agent learned quickly not to breach the hard constraints. After the initial spike, however, the agent's performance decreased. This is probably due to the high variance caused by the argmax function explained in section 2.4.1. As seen in the figure, the agent did reach a 90% perfect performance and the model was saved. After the initial training was the saved model improved by training with different learning rates and discount factors for achieving a perfect score of 200 for the MA. This proved not to be difficult, and after achieving a perfect score, the model was saved and used as a pre-trained model for the two and six tank cases.

For the implementation of DQN in a multi-tank system, two options were considered. The first one had only one network for all the controllers with the

output from the network mapped to a unique combination of controller positions. With 10 different positions, this approach would not be scalable as for n number of tanks in series as this would require 10^n output nodes in the network. The chosen approach was, therefore, to implement a multiagent system with one network for each valve.

Initially, all agents would be trained without prior training, but due to low performance, the trained model from the one tank case was loaded as the first agent for the second tank case. The training performance for the two tank case is shown in fig. 26.

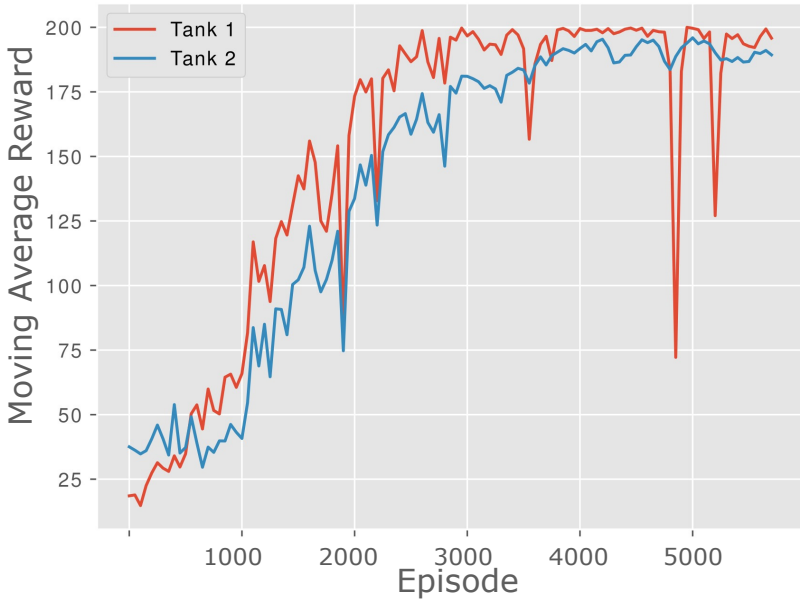

Figure 26: Experienced rewards for two DQN controllers during training. The rewards are the mean rewards of 50 episode for a total of ∼5000 episodes

From the figure, it can be observed that the first agent is doing significantly

better than the second one. This is to be expected as the first agent has already been trained from the one tank case. An important note is that the first agent could not have a perfect score from the first episode without a decent second agent. This is because the episode terminates when one of the two tanks fails. As the second tank fails at an early time step t, the first agent could only collect t number of rewards. The dips in performance from the first tank at the end of the training are most likely due to the high variance of value-based methods. After the initial training was the saved model improved by training with different learning rates and discount factors for achieving a perfect score of 200 for the MA. This proved not to be difficult. However, significantly more trials were needed compared to the optimization done with the first agent model.

For the six tank case, the same approach was used for the two tank case, with only the first agent pre-trained on the one tank system. The training performance for the 6 tank system is shown in fig. 31.

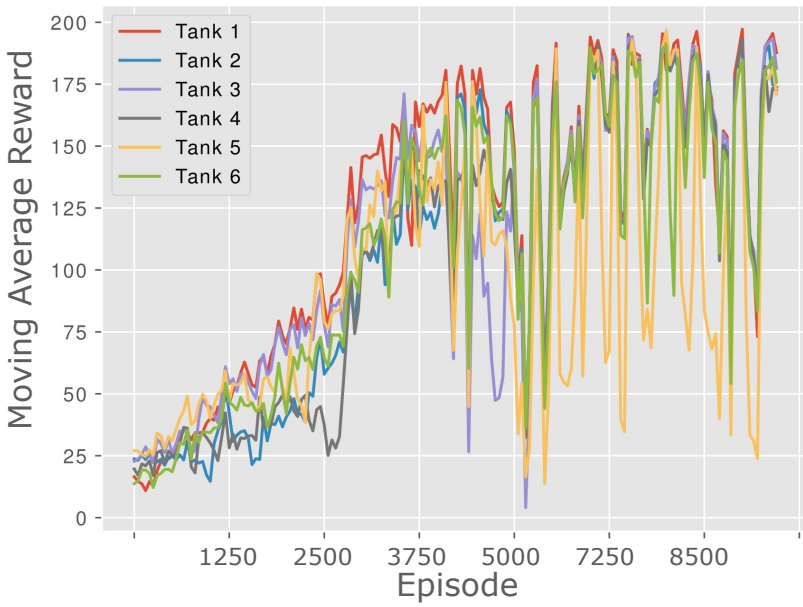

Figure 27: Experienced rewards for 6 DQN controllers during training. The rewards are the mean rewards of 50 episode for a total of ∼8000 episodes

It can be observed from the figure that the agents for tank 5 and 6 have a high variance in performance compared to the earlier agents. This is partially because of the high variance in Q-learning, but also because agent 5 and 6 must learn to adapt to new expected disturbances when the previous agent changes its policy. When tank 4 updates its policy, the expected inflow to tank 5 is changed and the fifth agent must adapt to these changes. An increase in the number of tanks in series would, therefore, increase the hyper-parameter sensitivity of the latter agent. The models were saved and improved gradually for different sets of hyper-parameters until a MA of 200 was achieved for 50 consecutive episodes. This turned out to be more difficult than the previous two tank case, as multiple attempts with different combinations were needed to improve the saved model.

All models from the DQN training were evaluated with the predetermined disturbance explained in section 3. After evaluating the value based method, the policy based method REINFORCE was explored as an alternative to Q-learning.

4.3.3 Policy gradient implementation: REINFORCE

As with Q-learning, initially, one tank case was first considered for training. A neural network was implemented as the function approximator with a sigmoid activation function as the log probability distribution. Whereas the DQN had 10 different output nodes, the policy gradient network only had one output node. The output from the network ranging between 0 to 1 was used as the percentage opening for the valve position z. Two hidden layers with 5 nodes and ReLU hidden layer activation function, was chosen. The reward function in algorithm (5) was used for training. This implies that for an episode of 200 steps, a maximum reward of 200 was possible to achieve.

The epsilon-greedy method was tried as an approach for the exploration vs exploitation problem. However, a solution that proved to yield better performance was the use of an action disturbance with a Gaussian distribution over the action space. The output value from the policy network was used as an expectation value along with a predetermined action variance z_{var} to ensure exploration. This method of exploration is similar to the extremum seeking method explained in section 1.2.4 as the input value is disturbed around its predicted optimal input value. A visualization of the action distribution can be seen in fig. 28.

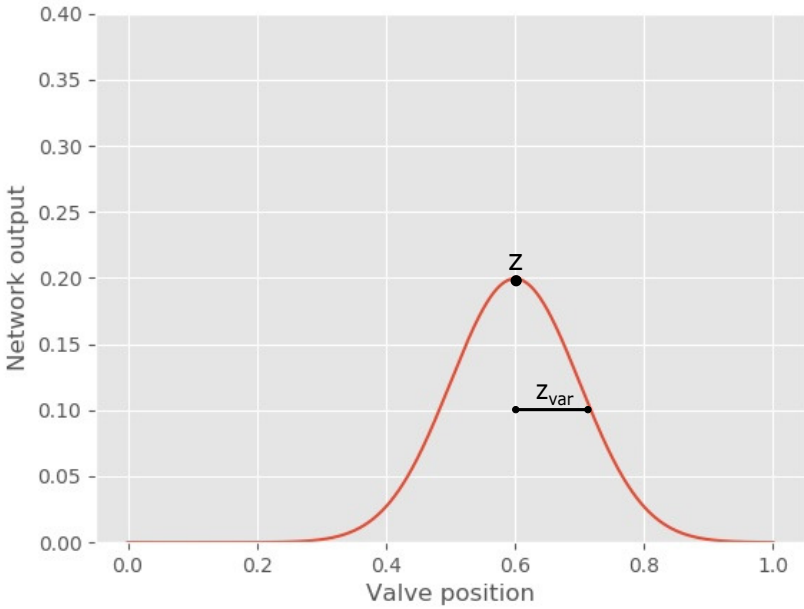

Figure 28: Visualization of the action distribution used for exploration for the REINFORCE policy gradient method

The training performance proved to be very sensitive to different hyper-parameters. The hyper-parameters which had a huge effect on the training performance were the discount factor γ, action variance z_{var} and learning rate α. This is to be expected as the DQN also was sensitive to γ, α and ϵ_{min}. The training performance for the final REINFORCE 1 tank implementation, is shown in fig. 29.

Figure 29: Experienced rewards for one DQN controller during training. The rewards are the mean rewards of 50 episode for a total of ∼3000 episodes

The training performance in fig. 29 shown a steady increase. This is to be expected from the policy gradient method as small incremental changes to the network gradually shift the expected action towards a better performance. However, as seen in the figure the agent did reach a 90% perfect performance and so the model was saved. After the initial training was the saved model improved by training with different learning rates and discount factors for achieving a perfect score of 200 for the MA. This proved not to be difficult, and after achieving a perfect score the model was saved to be used for further evaluations.

For the implementation of REINFORCE in a multi-tank system, the same approach with a multiagent system was chosen. Whereas the DQN showed poor performance without a pre-trained first tank agent, the REINFORCE agents did

manage to show promising results. This is probably due to the policy method's resilience in stochastic environments. Environments which can be described as partially observable MDPs. The training performance for the two tank case is shown in fig. 30.

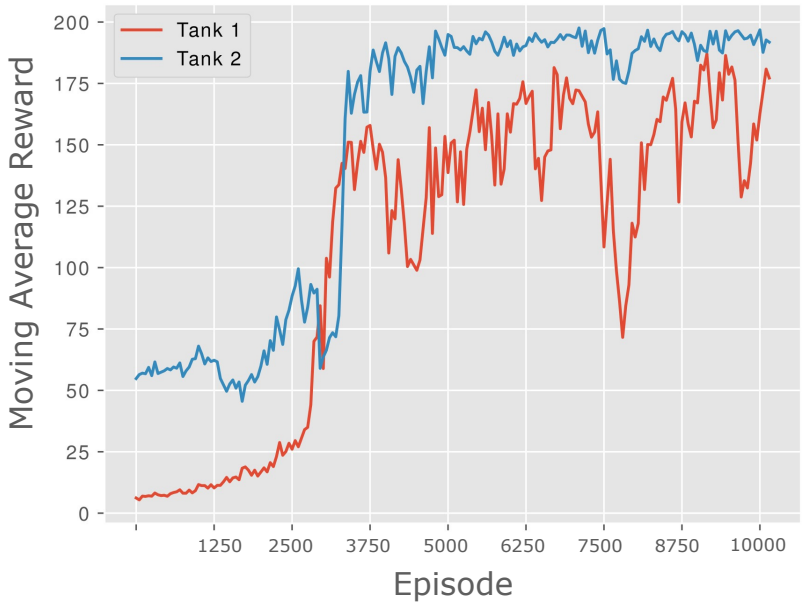

Figure 30: Experienced rewards for two REINFORCE controllers during training. The rewards are the mean rewards of 50 episodes for a total of ∼10 000 episodes

From the figure, it can be observed that the second agent is doing significantly better than the first one. This is most likely due to a local minimum found for the first agent. Compared to the value based method, REINFORCE shows less variance in performance than DQN. The REINFORCE models were saved and loaded with different learning rates and discount factors for achieving a perfect 200 score for 50 consecutive episodes. This was proven not to be difficult. However, notably, a lot more trials were needed compared to the optimization

done with the one tank case.

For the six tank case, the same approach was used for the 2 tank system. No pre-trained models were used. The training performance for the 6 tank system is shown in fig. 31.

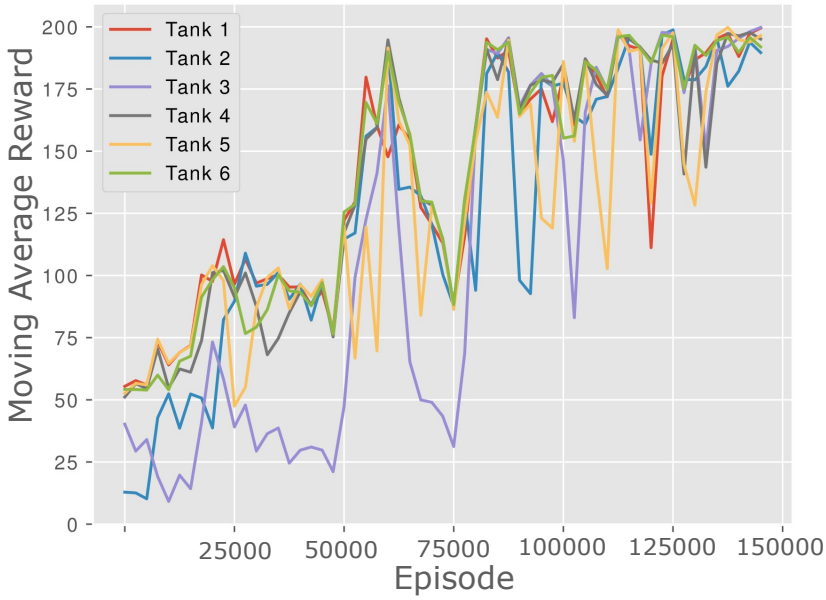

Figure 31: Experienced rewards for one DQN controller during training. The rewards are the mean rewards of 50 episode for a total of ∼150 000 episodes

Comparing the training of 6 tanks training in DQN and REINFORCE shows a smaller variance in training performance. The training is not as smooth as the one tank REINFORCE, but it still manages to converge within the 90% of a perfect score. However, this convergence required 150 000 episodes which are far more than for DQN. The models were saved and improved gradually with different sets of hyper-parameters until a mean of 200 was achieved for 50 consecutive episodes. This proved to be more difficult than the previous

two tank case, as multiple attempts with different combinations were needed to improve the saved model. However, compared to the six tank case in DQN, REINFORCE was easier to improve.

All models from the REINFORCE training were evaluated with the predetermined disturbance explained in section 3. Since the training process for policy gradient methods showed slow convergence, an actor-critic method was briefly explored to evaluate if the training time could be decreased.

4.3.4 Exploration of combining value and policy-based methods

To evaluate how the REINFORCE and DQN training performances could be improved, the actor-critic method A2C, which was explained in section 4.3.4, was explored. Due to limit the scope of this thesis, only one tank case implemented. The goal of the implementation of the actor-critic was to evaluate how the training time potentially could be decreased with AC methods. The training performance for the A2C one tank implementation is shown in fig. 32.

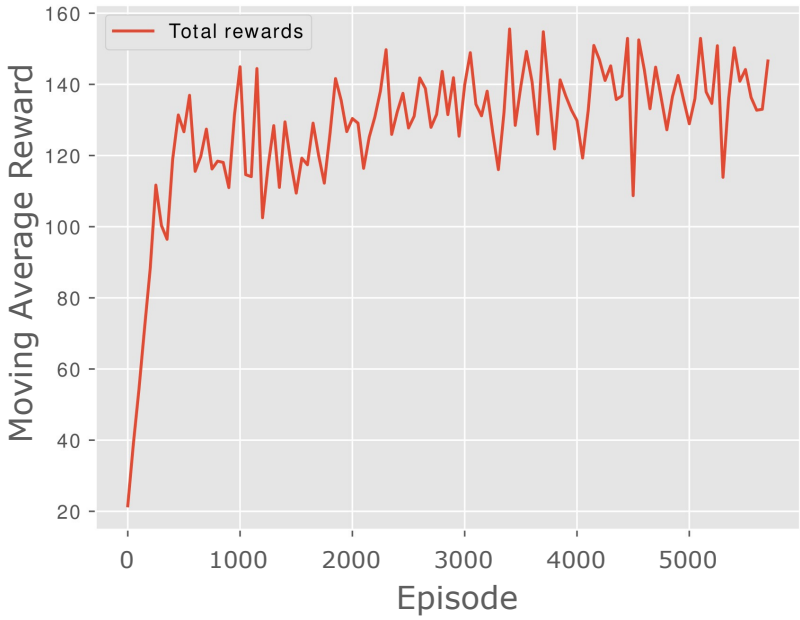

Figure 32: Experienced rewards for one A2C controller during training. The rewards are the MA episodic reward of 50 episode for a total of ∼5500 episodes

The training performance for the A2C method shows a significantly steeper learning curve. This is to be expected as actor-critic methods have shown to perform better than policy gradient methods on continuous state space MDPs [15]. However, the adjustment on the hyper-parameters for a perfect score was proved to be more difficult than expected. During training, the hyper-parameters were more sensitive to the changes in discount factors and learning rates compared to REINFORCE and DQN. This is probably due to the same TD-error being used for both the policy improvement as well as the critic improvement. A2C was also tried implemented on two and six tank systems, but due to the difficulties of finding usable hyper-parameters, this turned out to be a harder task than expected. The training would get stuck at a certain valve position, which the agent would calculate to be the optimal solution for the episode.

Exactly why the agent would calculate this optimum is hard to evaluate and a deep neural network analysis of the weight parameters would be required. This type of analysis is an active research field in deep learning and would require more work than the scope of this theses. However, another approach would be to change the reward function. This could lead to an easier task of training the agent but due to time restriction was this not tried in this thesis.

5 Evaluation of controllers

After training and tuning of controllers were all the controllers evaluated with an evaluation script. The predetermined disturbance presented in fig. 15 was used as inlet disturbance to the first tank. The script evaluated one episode and the dynamics of the tank level, controller input, and disturbance flow throughout the episode was plotted. The general implementation of the evaluation script is presented in appendix F.2. The hard constraints were set to be between 2.5 and 7.5 meters along with the soft constraints between 4 and 6 meters. This means that the controllers were to zone control between the soft constraints and with optimal control being a constant liquid level at 5 meter. Implementation of actor-critic was tried out for the one tank case as an opportunity analysis for future improvements

The evaluation section is divided into three cases, one tank, two tanks, and six tanks. In each case is first the P-controller evaluation presented followed by the evaluation of the RL-controllers.

5.1 One tank case

The evaluation of the one tank case consists of controlling the liquid level in one tank with one degree of freedom. The disturbance is predefined and the tank is cylindrical with 10 meters in height, 10m in radius for a total volume of 3141 m^3.

5.1.1 Evaluation of P-controller

Below is the evaluation plot for the P-controller. The general script for evaluation is presented in appendix F.2.

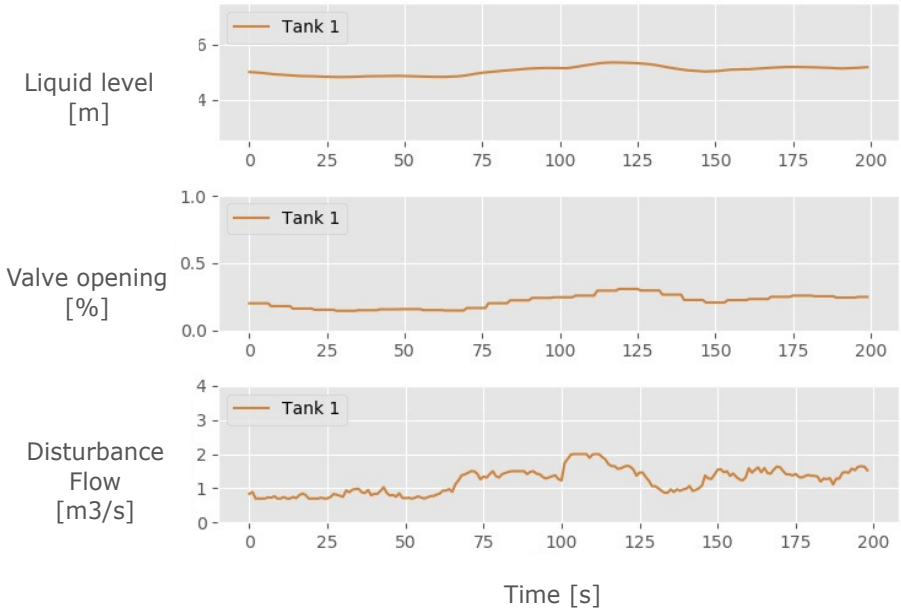

Figure 33: Evaluation of P-controller controller for the one tank case

From figure fig. 33 the P-controller evaluation controls the liquid level close to 5 meter at throughout the episode. The input trajectory consists of small changes which is to be expected with a high τ_c. The disturbance step at time step 100 seems to some effect on the system as the liquid level almost reaches 5.5 m. However, this is far from breaching any soft or hard constraints.

5.1.2 Evaluation of RL-algorithms

Following is the control usage of three RL algorithms, DQN, REINFORCE and A2C. Running the evaluation script, the neural network was loaded and the weights were not altered throughout the episode, and the exploration rate was set to zero. In the case for REINFORCE and A2C, the z_{var} was set to 0.01 to have some uncertainty into the input variable z. This uncertainty is to simulate potential noises that could occur in a real application.

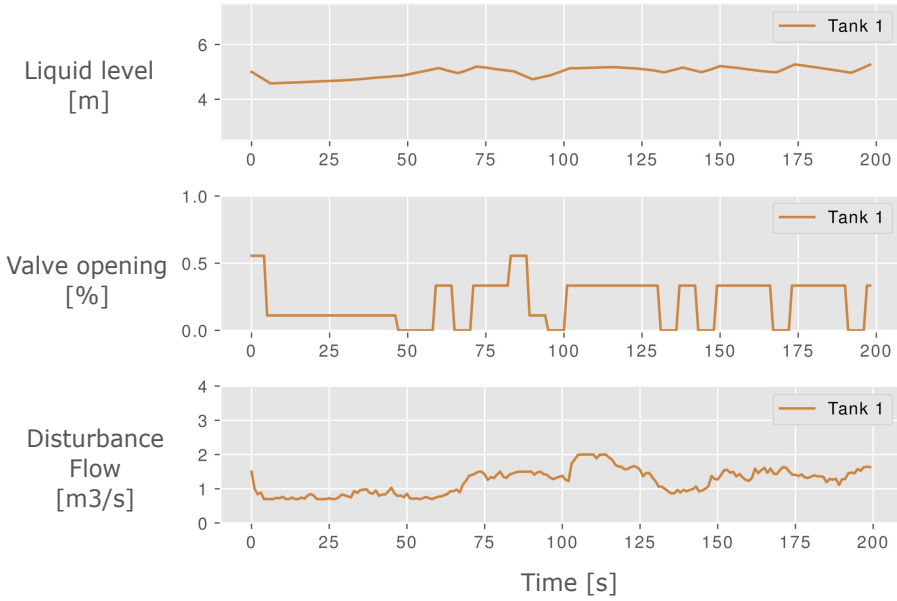

Figure 34: Evaluation of DQN-controller for the one tank case

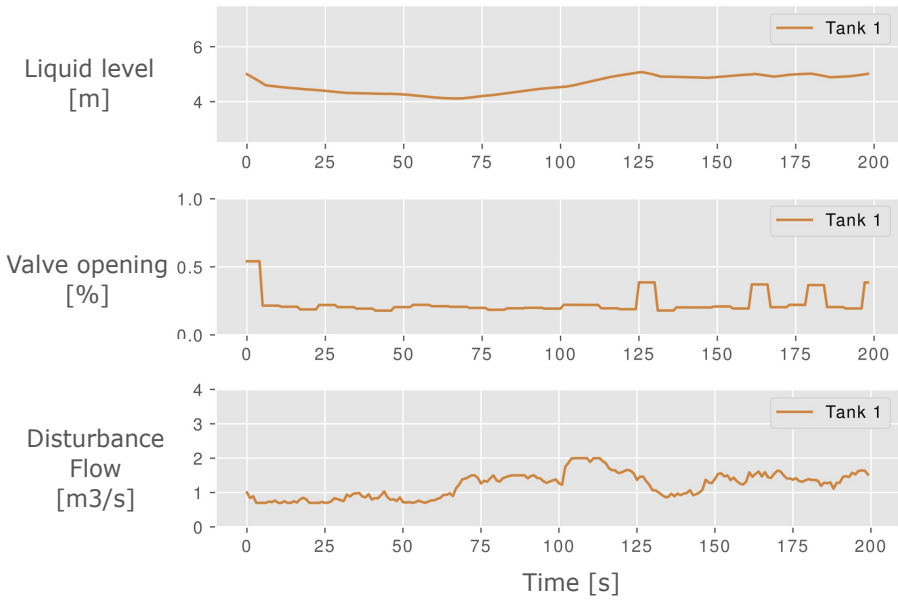

Figure 35: Evaluation of REINFORCE-controller for the one tank case

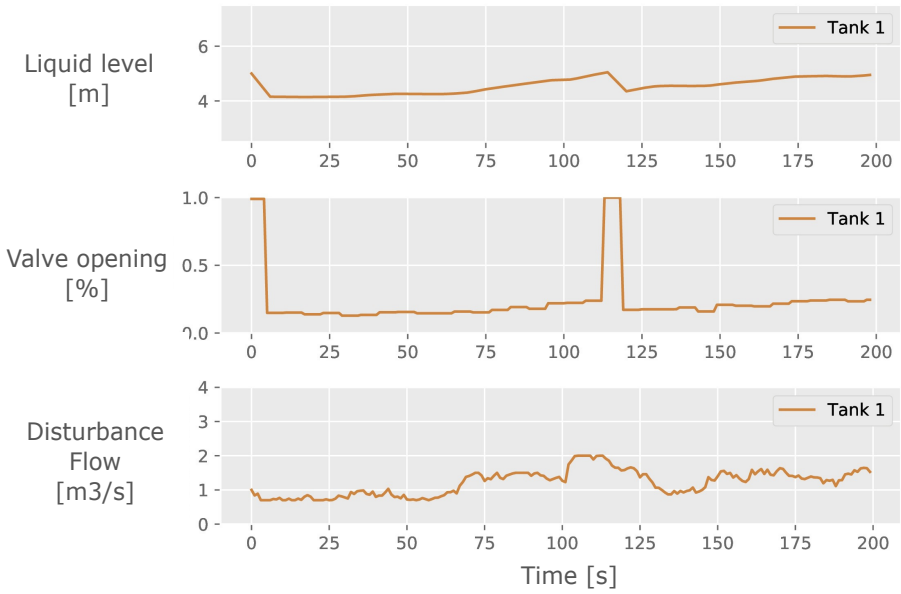

Figure 36: Evaluation of A2C-controller for the one tank case

All RL-controllers manages to keep the liquid level between 4 and 6 meters. This means that they do not break any soft or hard constraint which is crucial for controllers. Looking at the three RL-controllers, the input trajectory of REINFORCE and A2C is more similar to one another compared to the DQN.

The DQN-controller mainly shifting between two/three valve-positions throughout the episode. During training have the agent found out that using a small number of positions is sufficient enough to keep the level between 4 and 6 meters. Consequentially has the agent calculated the Q-value of these output nodes to be superior to the others.

The REINFORCE- and A2C controllers in fig. 35 and fig. 36 can be seen to find the optimal steady state valve position to be $\sim$20 %. This is due to the algorithms resilience to stochastic processes and evaluates an expectation of future rewards. The agent has experienced during training that a valve position of $\sim$20% would yield the highest expected reward. The sudden decrease in liquid

level from 0 to time step 100 for the REINFORCE controller seen in fig. 35 is due to the initial low disturbance inflow. During training would the disturbance in most cases oscillate around of 1 m^3/s which is not the case during evaluation step 0 to 100. The reason for the REINFORCE controller to have a smaller valve position change than the A2C-controller is difficult to explain. As explained for the implementation of RL-methods in section 4.3, would the performance of the RL-controllers vary with different sets of hyper-parameters.

Both the REINFORCE and the A2C can be observed to react with a high input change when the level is 5 meters or above. This can be seen for A2C at time step 0 and 115 as the valve position fully opens to lower the liquid level below 50 %. The same occurs for the REINFORCE controller at time step 0, 125, 175 and 200. This means that input node 3 from table 5 is having an effect on the agent's actions. The reason for this behavior is probably the experienced states the agent has trained on. The nominal valve position is around 30 % which the initial random weights for the agent's network would result in actions that would more often lead to a decrease in the liquid level. Since a level over 5 meters activated input node 3, the agent may interpret this state as unknown and would rather transition to a state below 5 meters which it has visited more often.

The behavior of the controllers in the one tank case is similar to the On-Off controller explained in section 1.2.1. The On-off controller is set to have two actions depending on the observed state. The same can be observed for the REINFORCE and A2C algorithm expect that the "off" setting is the steady state valve position around 20%. The small changes around the 20% opening show a resemblance to the adaptive controllers presented in section 1.2.4. The adaptive-controllers and REINFORCE-controllers trie to make small incremental changes to the valve position in order to lower a cost function (adaptive controllers), or increase the incoming reward (policy gradient controllers).

The P-controller in fig. 33 can be seen to yield better permanence for controlling the liquid level around 5 meters compared to the RL-controller. This is due to the input trajectory for the p-controller is more smooth and with fewer changes in valve position compared to the RL-controllers. A controller which changes the opening controllers is not desirable for many reasons. One reason is that is would lead to unpredictable disturbances which will propagate into succeeding components in the chemical plant, which may cause instability. Another reason is that increasing the number of actions for a choke valve would wear down the equipment faster which leads to a shorter life span for the choke valve.

5.2 Two tank case

The evaluation of the two tank case consists of controlling the liquid level in two tanks with two degrees of freedom. The disturbance is predefined for tank 1 with the given disturbance shown in fig. 15 and no external disturbance for tank 2. The two tanks are cylindrical of 10 and 8 meters in height, 10 meters in radius for a total volume of 3141 m^3 and 2010 m^3 respectfully. A detailed table of the parameters for the different tanks is presented in appendix B.

5.2.1 Evaluation of P-controller

Below is the evaluation plot for the P-controller. The script for evaluation is presented in appendix F.2.

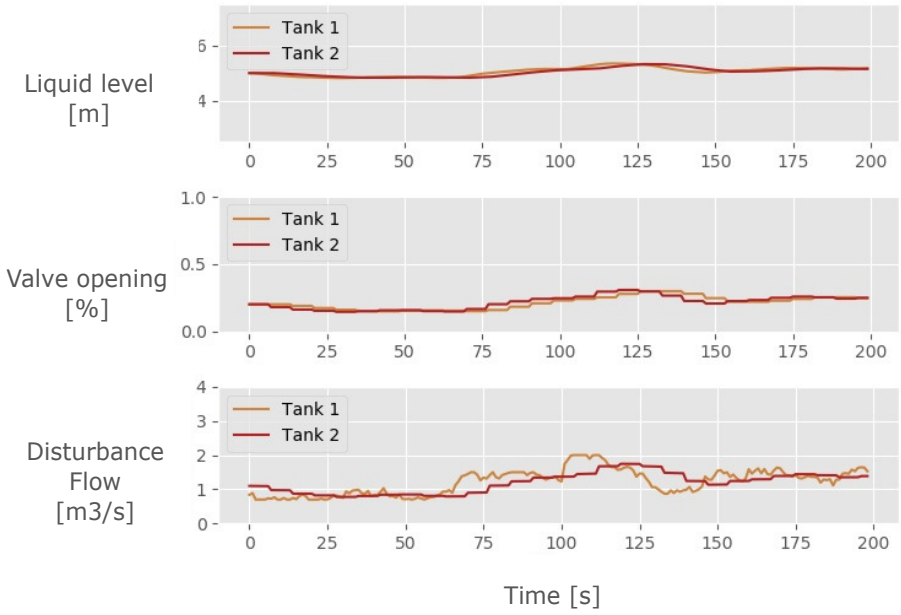

Figure 37: Evaluation of P-controllers for the two tank case

From figure fig. 37 the P-controllers can be seen to control the liquid level close

to 5 meters throughout the episode for both tanks. However, comparing the p-controller for the first and second tank, the valve position trajectory is of similar trends. By comparison between the two valve position trajectory, the second tank positions can be observed as a phase shift forward. This is to be expected as both controllers should behave quite similar and the disturbance to the second tank occurs after the feedback control response in the first tank.

As discussed in section 4 the p-controller can be evaluated as smooth control due to comparison between τ and τ_c. This can be observed in fig. 37 as the change in input values is small for disturbances.

The disturbance step at time step 100 seems to have little effect on the system except for an increase in valve position. However, since the tank one p-controller actions are the resulting disturbance for the second tank, the incoming disturbance for the second's tank can be observed as multiple small disturbance steps. From the figure, it can be observed that this results in a higher change in valve positions for the second controller as it tries to keep the liquid level around 5 meters.

5.2.2 Evaluation of RL-algorithms

Following is the control usage of two RL algorithms, DQN and REINFORCE. Running the evaluation script in appendix F.2, the pre-trained neural network was used with its weights not altered throughout the episode, and the exploration rate set to zero. In the case for REINFORCE was the z_{var} set to 0.01 to have some uncertainty into the input variable z.

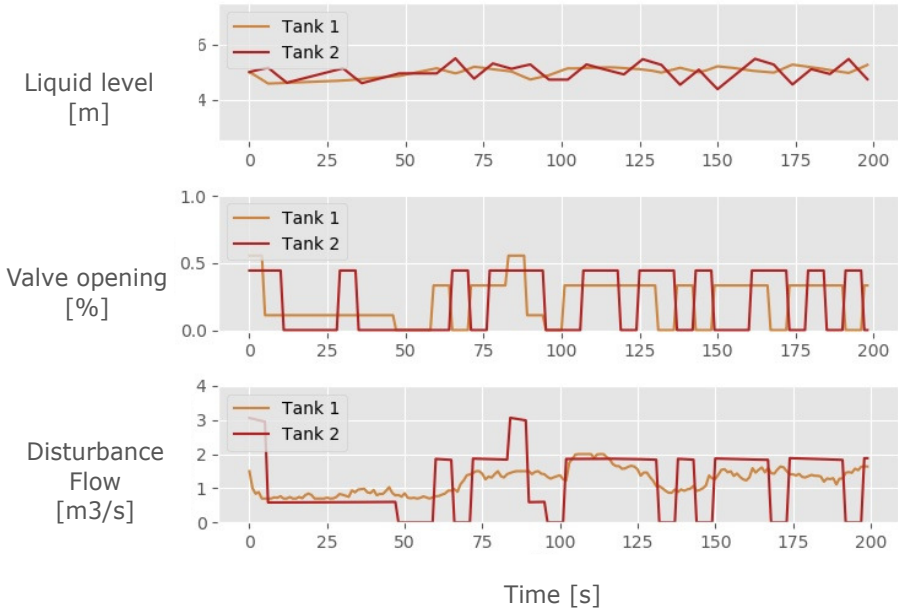

Figure 38: Evaluation of DQN-controllers for the two tank case

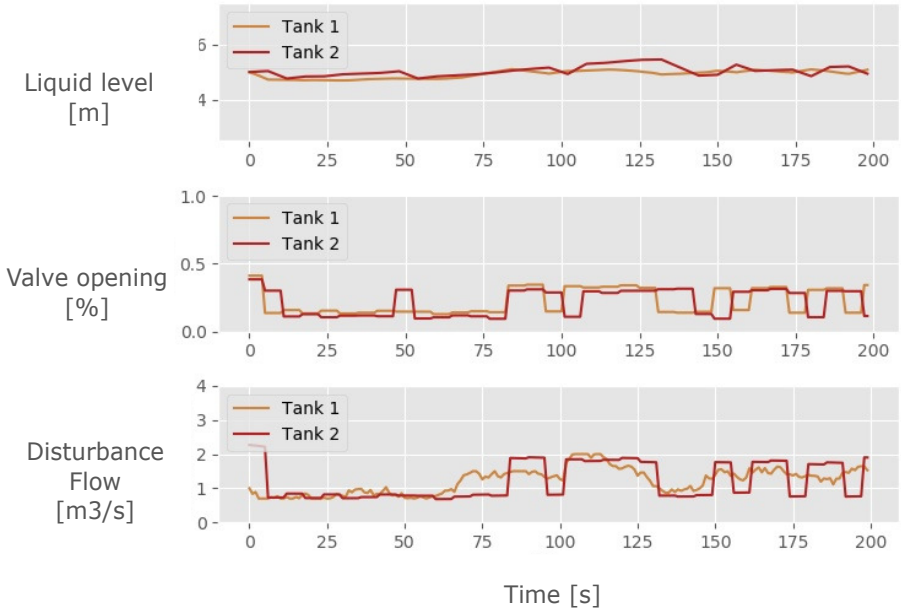

Figure 39: Evaluation of REINFORCE-controllers for the two tank case

All RL-controllers managed to keep the liquid level between 4 and 6 meters. This means that they do not break any soft or hard constraint which is crucial for controllers. However, comparing the liquid level of the first and second tank shows a small oscillatory behavior with a frequency around 25-time steps. This oscillatory behavior is not observed to be unstable, but not desirable as it may lead to instability.

The DQN-controller evaluation for the second tank in fig. 38 uses only two valve positions for control. This is due to the same reasons explained for the DQN for one tank case in section 5.1. However, where the first DQN-controller uses four valve positions, the second controller only uses two positions. This is probably due to the constant disturbance shift which makes it harder for the controller to control the liquid level. Consequentially learned the DQN-controller the policy of only using two valve positions to be the most reliant for achieving a perfect score.

The second REINFORCE-controller in fig. 39 can be sen to find the optimal

steady state valve positions to shift between 20 % and 30 %. Whereas the first REINFORCE-controller has found the optimal position to be around 20% opening. By inspecting the liquid level plot of the second tank the level can be observed to exceed 5m which is most likely the reason for the more frequent of 30 % valve position opening for the second REINFORCE-controller. This indicates that both REINFORCE-controllers have found similar optimal policies as the REINFORCE-controller in the one tank case.

5.3 Six tank case

The evaluation of the six tank case consists of controlling the liquid level in six tanks with six degrees of freedom. The disturbance is predefined and all the tanks are cylindrical with a radius ranging between 7 - 10 meters. A detailed table of the specifics for the different tanks is given in appendix B.

5.3.1 Evaluation of P-controller

Below is the evaluation plot for the P-controller.

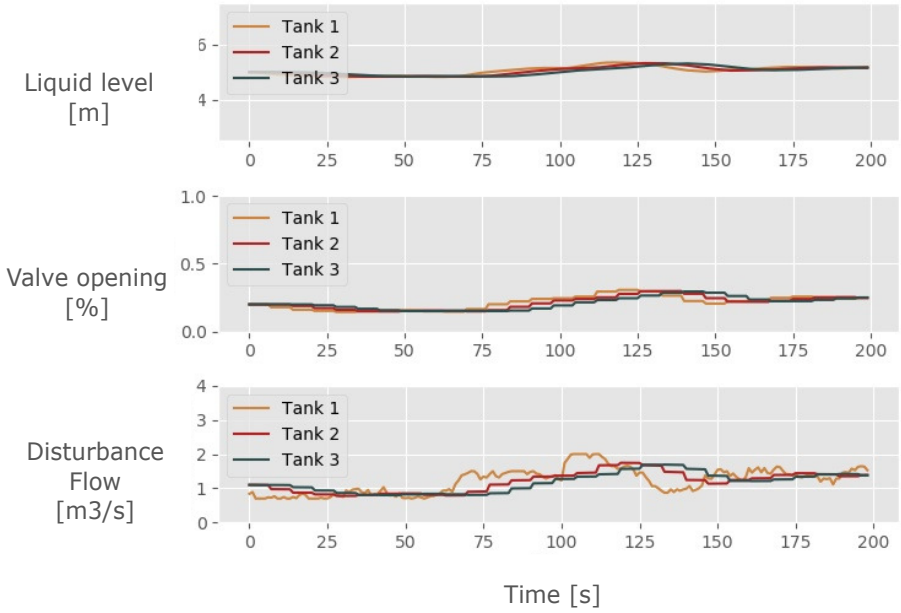

Figure 40: Evaluation of P-controllers of the first three tanks for the six tank case

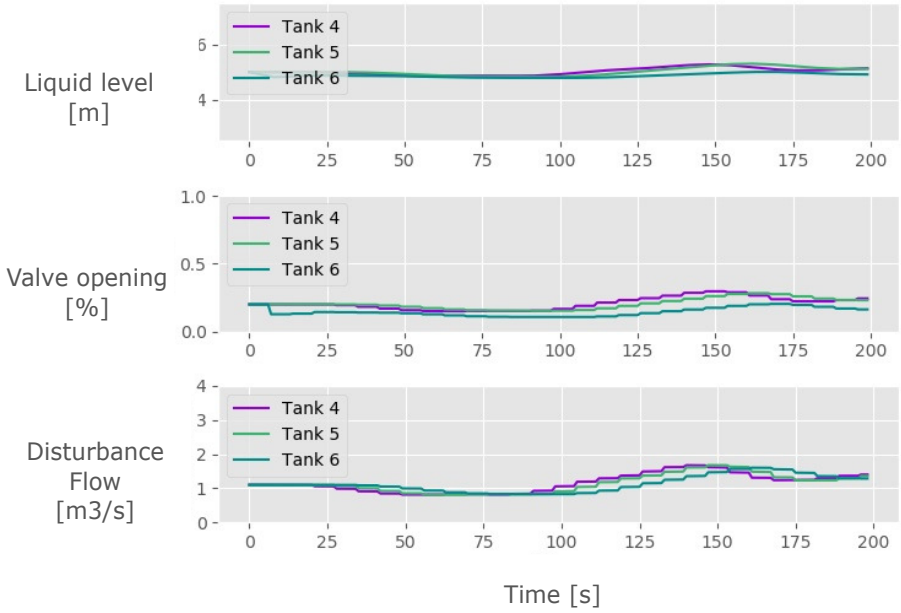

Figure 41: Evaluation of P-controllers of the last three tanks for the six tank case

From fig. 40 and fig. 41 the p-controller manages to keep the liquid level close to the set point within the 5 and 6 meter range. From the disturbance plot in the figures, the propagated disturbance is amplified for each tank in the series as the liquid levels after the disturbance step increases more for the succeeding tanks thank. However, the p-controllers manages to counteract the disturbance without breaking any constraint.

5.3.2 Evaluation of RL-algorithms

Following is the control usage of two RL algorithms, DQN and REINFORCE. Running the evaluation script in appendix F.2, the pre-trained neural network was used with its weights not altered throughout the episode, and the exploration rate was set to zero. In the case for REINFORCE was the z_{var} set to 0.01 to have some uncertainty into the input variable z.

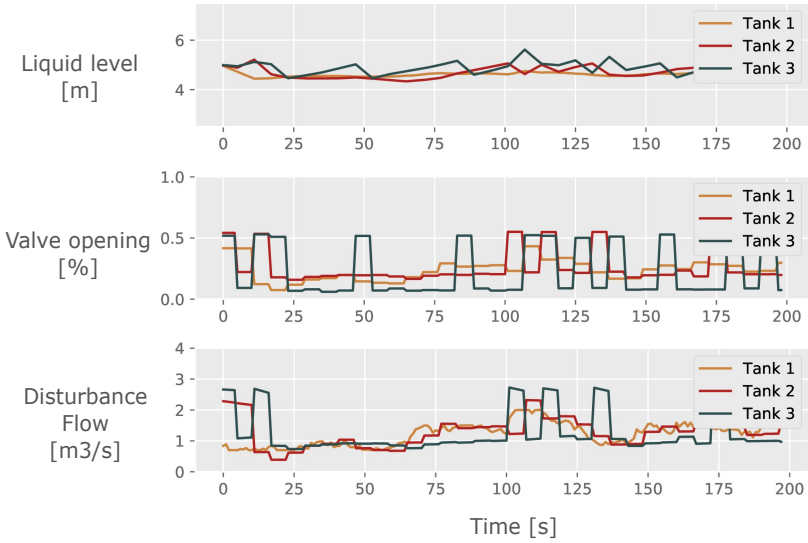

Figure 42: Evaluation of DQN-controllers of the first three tanks for the six tank case

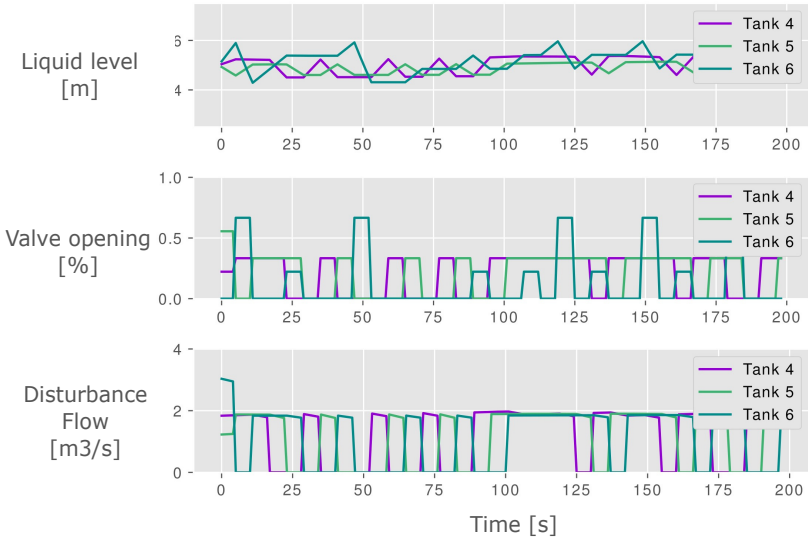

Figure 43: Evaluation of DQN-controllers of the last three tanks for the six tank case

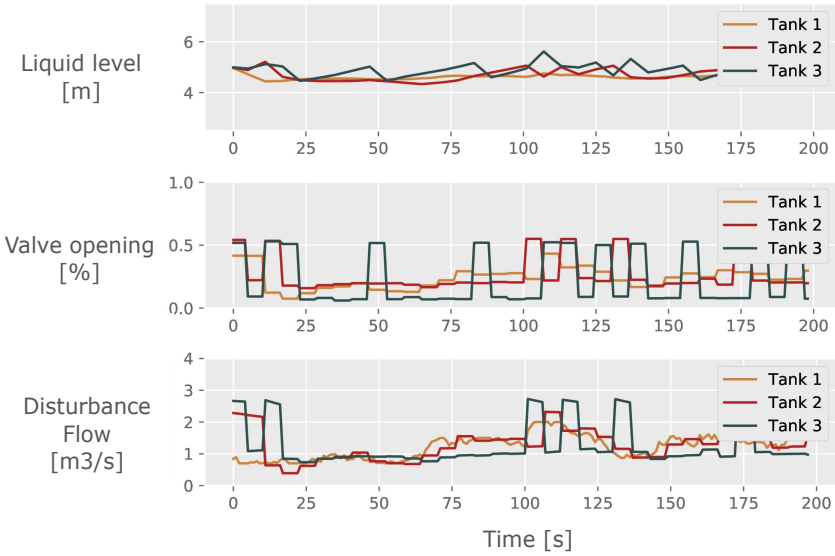

Figure 44: Evaluation of REINFORCE-controllers of the first three tanks for the six tank case

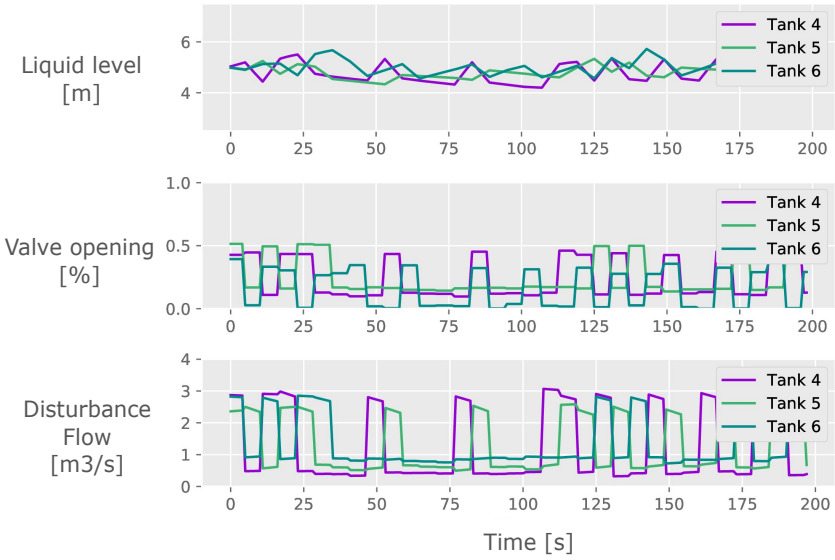

Figure 45: Evaluation of REINFORCE-controllers of the last three tanks for the six tank case

Both RL evaluation managed to keep the liquid level between 4 and 6 meters which correspond to the perfect score it achieved during training. However, compared to the P-controller the agents can be seen to favoring large instead of small incremental input changes. This is to be expected as this was the same behavior for the previous cases and the agents are only trained for maximizing the reward without any restrictions to the valve position trajectory.

The DQN agents in fig. 42 and fig. 43 follows the same pattern as for the two tank case where the optimal policy is found to shift between a small set of actions. The REINFORCE method shows smaller action changes compared to the DQN. However, this only applies to the earlier agents as observed in fig. 44. The agents for tank 4-6 in fig. 45 seems to fall into the same patterns with favoring a steady state action and shifting to a higher opening when the liquid level exceeds 5 meters.

6 Discussion

From the evaluations in section 5 the p-controllers showed superior performance for controlling the tank level for all three cases. This is due to the RL-controllers input trajectory leading to an oscillating behavior in the tank levels whereas the P-controller shows a more smooth and stable trajectory. As discussed earlier in section 5 is oscillatory behavior not preferable as it may lead to instability. The reason for this behavior is mainly a consequence of the design of the reward function. Additionally may the choice of RL-methods and choice of hyper-parameters may have contributed to the problem.

6.1 Design of reward function

The reward function in algorithm (5) states that the position between 4 and 6 is irrelevant for the agent for receiving rewards. One can argue that by keeping the level at 5 meters the agent would have the largest margin for errors and therefore would try to keep the level close to 5 meters. However, from the evaluations the DQN and REINFORCE does not find this policy but rather to oscillates around the setpoint. With respect to the Markov Reward Process would these two approaches have no differences as the same reward is returned regardless. This means that the agent has no incentive for favoring smaller input changes as long as the level is between 4 and 6 meters. This method of control shows a lot of similarities with zone control which in MPC have proved to result in large input changes. A stable MPC with zone control can be difficult to obtain, as the control systems keep switching between MVs when transitioning from controlled to non-controlled states [37]. The same behavior can be seen for the RL-controllers with the large input changes shown in the evaluation plots in section 5.

A way of making the agent favor smaller input changes could be to introduce the action as a variable in the reward function. For example, can the agent be penalized by making large action changes. This rework of the reward function would lose the Markov property for the current state observation as the state is dependent on previous actions. To preserve the Markov property, the agent's history of inputs could be included as a state observation. One way of implementing this is to use the action history as input nodes. However, for large action histories could a better approach be to introduce a Recurrent Neural Network in the agent's network. With the introduction of a RNN could

additionally a system delay be included in the simulation, a feature that was not implemented in this thesis. In process systems are delays always present in some way and with the introduction of RNN in the agent's network could the environment for the agent be more relatable to a real process system.

The evaluation of the auto-tuned p-controllers in appendix E and the RL-controllers shows similar tank level oscillations. The auto-tuned p-controllers finds the τ_c with the smallest SSE without any restrictions to the input change. This behavior is reflected in the evaluation of RL-controllers as DQN, REINFORCE and A2C only try to maximize the reward. Both methods use only the numerical minimization as a criterion for evaluating the best policy whereas the higher τ_c provides a less optimal but rather smoother control policy. The evaluation from the auto-tuned p-controllers emphasizes that only minimizing the deviation from the setpoint is not enough for creating a stable tank level controllers. This follows into the argument of including input constraints into the reward function for improved RL-control policy.

6.2 Performance between DQN-controller and REINFORCE-controller

From the evaluations, the REINFORCE-controllers can be observed to change the valve positions fewer times throughout the episode compared to the DQN-controller. Comparing the six tank case in fig. 44 and fig. 42 the DQN-controller deviates more from the setpoint than REINFORCE-controller. However, for the one tank case in section 5.1.2 are the deviations from setpoint larger for the REINFORCE-controller than the DQN-controller. The A2C-controller in the one tank case shows an even more deviation than the other two. The liquid level for all controllers can be observed to oscillate with approximately the same intensity. Based on the criteria of controlling the liquid level around 5 meters are none of the RL-controllers superior to the other ones, and all methods could be viable options for process RL-control. However, since the input changes in the REINFORCE- and A2C-controller are smaller compared to DQN -controller. The DQN-controller would be less favorable for controlling tanks in series. This is to be expected as Q-learning is more used for discrete deterministic MDPs, whereas the tank cases in this thesis are continuous and stochastic.

Even though the evaluation showed a more realistic application of using REINFORCE-controller for the presented cases, the training time which was needed was significantly higher for REINFORCE than DQN. During training was 50 episodes

roughly calculated in 1 second, meaning for the training time had no significant effect for the cases presented in this thesis. However, for systems that are more complex would this not scale well for the REINFORCE method. The implementation of A2C in fig. 32 shows that the usage of the actor-critic method could be a viable option for more computational demanding cases. Cases where the sample efficiency of data should be optimized so that the training time is of feasible length. The REINFORCE training time could potentially be decreased by using a more systematic exploration, for example by using methods used in adaptive controllers.

6.3 Performance between the RL-controllers and the conventional controllers

The high τ_c for the P-controller resulted in smaller changes in valve position. This was not the case for the RL-controller as the current action is independent of the previous actions. By comparing the performance of the auto-tuned P-controller in appendix E with the RL methods both methods shows an aggressive policy for controlling the liquid level. However, this is not the case for the fine-tuned p-controller. As discussed earlier could this dependency be implemented with an action dependent reward function and a Long Short-Term Memory architecture in a Recurrent Neural Network.

The training of the REINFORCE controller showed many resemblances to an adaptive controller. During training was the expected valve position shifted towards a setpoint which yielded a higher cumulative reward. This is also the case for the adaptive controller where the input is injected by a perturbation which evaluated how the input should be changed for a lower operation cost. However, where the adaptive controller would only evaluate the immediate cost, the REINFORCE methods would account for long term cost and weight these future rewards by a discounting factor. In the case of this thesis would an adaptive controller probably has the same policy as the REINFORCE-controller. This is most likely due to the approximate linear dynamics of the tank system. The Adaptive controllers would most likely shift the optimal input value around 20-30 % along with the policy of the REINFORCE-controller.

7 Conclusion

The well-tuned P-controllers in this thesis were proved to be better controllers, compared to the RL-controllers, for regulating tank levels. The reason for this was mainly the poor choice of reward function which did not penalize the RL-controller for making large input changes in the valve position. A different reward function dependent on valve position would probably lower the action changes and create a RL-controller which resemblance more of a P-controller. However, all RL-controllers managed to control the liquid level between the constraint which shows the possibility of using RL-controllers in process industries.

The RL-controllers during training did show a high sensitivity to changes in the hyper-parameters, as the agent would struggle to improve its policy with some sets of hyper-parameters. These include the learning rate, minimum exploration rate, action disturbance, and discount factor. This means that a strategy of choosing parameters is crucial for the RL-controllers performance. This is similar to the choice of choosing the correct tuning parameters for conventional controllers. However, for conventional controllers is not a training phase required before evaluation which means that creating a suitable RL-controller may be more time consuming than structural tuning of regular controllers.

The hyper-parameters which would be more relevant for more complex systems would be the structure of the neural network. For the cases in this thesis were only a shallow neural network with 2 hidden layers of 5 nodes used. The RL implementation would probably be sufficient with a linear function approximator in this thesis. However, the idea of RL-control is to substitute or compliment systems which the current control theory is not sufficient. This means that for process control would RL most likely be used as an alternative to or compliment Model Predictive Control and a non-linear function approximator would be required. Applying RL to complex systems means that the architecture of the network would need to be larger than in this thesis. This increase in network size would complicate the task of understanding the network's behaviour. Without more knowledge in deep learning would this implementation lead to a black box RL-controller which is not desirable as control engineers. The ability to understand how the network works is crucial for fine-tuning of parameters when applied. With more insight into the learning process of the neural networks could RL-controllers surge in popularity as they offer a general method for all types of controllers.

The RL controllers all managed to archive the goal of controlling the level be-

tween 4 and 6 meters for stochastic disturbance input which shows that using RL in process control could be an alternative for control. However, this thesis highlights the challenges related to the design of reward function, hyper-parameter sensitivity, and the tendency of black box modeling. With the current method of control will most likely RL-control not compete with current implementation but rather compliment areas where non-linear controllers like MPC are difficult to create. The nature of RL-controllers and general implementation also means that RL-control can be used as a method for automating the tuning process of controllers. The usage of continuous action space controllers like policy gradient and actor critics allows trials and errors of tuning parameters. Additionally could value based methods like DQN be used for systems where discrete decisions are required.

In summary are the potential of using RL-controller huge but difficult to implement. With increased research in deep learning and creation of RL-frameworks could RL-control be seen as a valid option in the future for process control. However, from the current iteration seen in publications and the lack of complete deep learning understanding is this option still a research field but could potentially compliment current non-linear controllers in the near future.

8 Further work

8.1 Improvements related to the thesis

From this thesis, the work of designing the reward function proved to be the most vital of the known challenges surrounding RL. The system could be improved by adjusting the reward function to also be a function of the valve position history to smoothen out the input trajectory. As discussed earlier could this be archived by changing the function approximator to be a time-dependent network. The reason for changing the network architecture is to maintain the critical Markov property. With a reward function that is dependent on the valve position history would the current state alone not be enough information for the agent to learn the optimal policy. Examples of a time-dependent network would be implementing an AR model or a recurrent neural network.

The RL-implementation for the DQN-controllers in the two and six tank cases consisted of a multi-agent system. This resulted in a similar multi-agent system for the REINFORCE-controllers. Whereas the scalability of one agent was poor for the multi-tank DQN implementation is this not necessarily the case for REINFORCE and AC. In this thesis was the REINFORCE controllers implemented with one agent controlling the outflow of one tank with the network having one output node correlating to the valve position opening. A different approach could be to collect all the agents into one network with each output node representing the valve position opening of one tank in the system. This could potentially decrease training time as only one network would be required for controlling all the tanks.

The RL controllers in this thesis all showed promising results for controlling a process system. These results could be further capitalized and the RL-controllers could be used for exploring more complex systems. However, the cases in this thesis showed that regular controllers were superiors compared to RL-controllers which indicates that RL-controllers for more complex systems would be more difficult to create. The RL-controllers property of long term reward should be the main attraction for using RL-controller in process systems. This means that RL-controllers should be used for systems that require immediate sub-optimal actions in favor of long term rewards. However, RL-controllers show high variance in performance and is mainly the reason why RL-controllers are still only a research area in the process industry. In applications would this high variance propagates into economical and health risks which are high prior-

ities in process control. A large process facility could not risk the health of its employees and millions of dollars on a black box controller.

8.2 Different approaches of RL in process control

With further exploration of RL in process control could RL-controllers like DQN be explored for discrete problems. For example could challenges related to deciding an optimal structure of a plant be tried out with RL. Often have these problems been solved with Mixed Integer Programming (MIP), and by branch and bounding. an RL-implementation could potentially find the optimal structure given a predefined reward function.

The general implementation of RL shown that any decision variable which prompts feedback signal from a state could be implemented with RL for finding the optimal policy. In this thesis was this decision variable the MV for the system but other decision variables could also be tried out. In particular could other hyper-parameters or variables which are often found by engineering intuition be tested in RL. For example, could non-static tuning parameters to different controllers be used as decision parameters. As discussed earlier is the usage of RL only relevant when dealing with long term evaluation along with a time-series of multiple decisions. This means that using RL for evaluating a constant tuning parameter like τ_c for the P-controllers in this thesis would not be suitable as only one decision was made for tuning the controllers.

From fig. 32 the actor-critic method A2C showed a significantly faster training time than the REINFORCE method with a continuous action space, that DQN could not achieve. This method could be further explored with an implementation for the two and six tank cases. This thesis proved that the implementation of A2C requires extensive deep learning knowledge and collaboration between the chemical- and computer engineers is required for developing realistic applications of RL in process control. However, with the development of RL-frameworks could this knowledge requirement be decreased.

A suggestion for RL methods to be further explored is the PPO and DDPG methods which OpenAI has shown great performance with large MDP [36]. Another extensive method of creating RL-controllers is combining SL with RL as DeepMind did with the AlphaStar [20]. This approach uses SL to replicate professional performance but then improved with RL. In process control could SL be used to replicate controllers behavior and then further be improved with

RL for more complicated systems.

References

[1] Brent R. Young William Y. Svrcek Donald P. Mahoney. *A Real Time Approach to Process Control*. 3 ed. 2014.

[2] *Norsk Petroleum Statens inntekter*. `https://www.norskpetroleum.no/okonomi/statens-inntekter/`, downloaded mars 2019. 2017.

[3] *Norsk Petroleum Eksport av olje og gas*. `https://www.norskpetroleum.no/produksjon-og-eksport/eksport-av-olje-og-gass/`, downloaded jan 2019. 2017.

[4] Duncan A. Mellichamp Francis J. Doyle Dale E. Sebord Thomas F. Edgar. *Process Dynamics and Control*. 3 ed. 2011.

[5] Sigurd Skogestad. *PID control. Practical issues*. Slides to course TKP4140 Process Control. Norwegian University of Science and Technology, Department of Chemical Engineering. 2017.

[6] Sigurd Skogestad. *PID Tuning using the SIMC rules*. Slides to course TKP4140 Process Control. Norwegian University of Science and Technology, Department of Chemical Engineering. 2017.

[7] Tor Aksel N. Heirung Bjarne Foss. *Merging Optimization and Control*. Norwegian University of Science and Technology, Department of Engineering Cybernetics. 2016.

[8] Johannes Jäschke and Sigurd Skogestad. *Plantwide process control Introduction*. Slides to course TKP14 Process Control, Advanced course. Norwegian University of Science and Technology, Department of Chemical Engineering. 2018.

[9] Bo Egardt. *Stability of adaptive controllers*. Vol. 20. Springer, 1979.

[10] Kartik B Ariyur and Miroslav Krstić. *Real time optimization by extremum seeking control*. Wiley Online Library, 2003.

[11] Khalid Tourkey Atta, Andreas Johansson, and Thomas Gustafsson. "Extremum seeking control based on phasor estimation." In: *Systems & Control Letters* 85 (2015), pp. 37–45.

[12] Keith L. Downing. *Supervised Learnng in Neural Networks*. Material to course IT3105 Artificial, Intelligence Programming. Norwegian University of Science and Technology, Department of Computer Science. 2010.

[13] Keith L. Downing. *Introduction to Artificial Neural Networks*. Slides to course IT3105 Artificial, Intelligence Programming. Norwegian University of Science and Technology, Department of Computer Science.

[14] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] Boston Dynamics. *Atlas*. `https://www.bostondynamics.com/atlas`. 2018.

[17] Arthur L Samuel. "Some studies in machine learning using the game of checkers." In: *IBM Journal of research and development* (1959).

[18] Z. Ge et al. "Data Mining and Analytics in the Process Industry: The Role of Machine Learning." In: *IEEE Access* 5 (2017).

[19] David Silver et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* (2016).

[20] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`. 2019.

[21] Jay H Lee and Weechin Wong. "Approximate dynamic programming approach for process control." In: *Journal of Process Control* 20.9 (2010), pp. 1038–1048.

[22] Thomas A. Badgwell, Jay H. Lee, and Kuang-Hung Liu. "Reinforcement Learning – Overview of Recent Progress and Implications for Process Control." In: *13th International Symposium on Process Systems Engineering (PSE 2018)*. Ed. by Mario R. Eden, Marianthi G. Ierapetritou, and Gavin P. Towler. Vol. 44. Computer Aided Chemical Engineering. Elsevier, 2018, pp. 71 –85. URL: `http://www.sciencedirect.com/science/article/pii/B9780444642417500082`.

[23] *Reinforcement Learning in TensorFlow with TF-Agents*. `https://www.youtube.com/watch?v=-TTziY7EmUA&t=697s`. Tensorflow Dev Summit. 2019.

[24] Eskild Ruud Mageli. *Reinforcement learning in process contrl*. `https://github.com/emedd33/Reinforcement-Learning-in-Process-Control`. Norwegian University of Science and Technology. 2019.

[25] Paul A Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[26] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[27] W Keith Hastings. "Monte Carlo sampling methods using Markov chains and their applications." In: (1970).

[28] Jorge Nocedal. and Stephen Wright. *Numerical Optimization*. second. Springer, 2006.

[29] Léon Bottou. "Stochastic Learning." In: *Advanced Lectures on Machine Learning*. Ed. by Olivier Bousquet and Ulrike von Luxburg. Lecture Notes in Artificial Intelligence, LNAI 3176. Berlin: Springer Verlag, 2004, pp. 146–168. URL: `http://leon.bottou.org/papers/bottou-mlss-2004`.

[30] `http://mathworld.wolfram.com/RosenbrockFunction.html`. Rosenbrock function.

[31] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5 (1943).

[32] Yoshua Bengio Xavier Glorot Antoine Bordes. *Deep Sparse Rectifier Neural Networks*. Université de Montréal. 2011.

[33] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3-4 (1992).

[34] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), p. 529.

[35] Gary M Burlingame, John C Kircher, and Charles R Honts. "Analysis of variance versus bootstrap procedures for analyzing dependent observations in small group research." In: *Small Group Research* 25.4 (1994), pp. 486–501.

[36] Prafulla Dhariwal et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[37] Alejandro H Gonzalez and Darci Odloak. "A stable MPC with zone control." In: *Journal of Process Control* 19.1 (2009), pp. 110–122.

A Tank equation derivation

The tank was modeled from a generic mass balance. The tank was assumed to no have only one inflow and no outflow.

$$\frac{d(\rho V_{tank})}{dt} = \rho q_{inn} - \rho q_{out} \tag{41}$$

Assuming the tank volume can be expressed as a perfect cylinder reformulates the mass balance to an equation of the height level.

$$\frac{d(h_{tank})}{dt} = \frac{1}{(\pi r_{tank}^2)}(q_{inn} - q_{out}) \tag{42}$$

The outflow goes through a valve which lets through a set amount defined by the valve equation $f(z)$. The liquid in the tank is assumed to be in-compressible with a low Mach number. The density is also assumed to be constant which means that the Bernoulli equation could be substituted for the outlet speed in $q_{out} = f(z)v_{out}A_{out}$.

$$\frac{1}{2}v_{tank}^2 + gz_1 + \frac{p_{tank}}{\rho} = \frac{1}{2}v_{out}^2 + gh_{tank} + \frac{p_{out}}{\rho} \tag{43}$$

The reference height is set to the outflow level, the liquid in the tank is assumed to have no kinetic energy before flowing out of the tank. The pressure in each tank is set to be atmospheric so the flow is only dependent on the height level. Adding the Bernoulli equation to the mass balance formulates the dynamic equation of the liquid level in the tank

$$\frac{d(h_{tank})}{dt} = \frac{1}{\pi r_{tank}^2}\Big[q_{inn} - f(z)A_{out}2gh_{tank}\Big] \tag{44}$$

B Parameters used in simulation

Below is the parameter used for the tank used in simulation.

Parameter name	Value
Height	10m
Radius	7-10m
Outflow pipe radius	0.5m
Initial level	5m
Soft constraint high	6m
Soft constraint low	4m
Hard constraint high	7.5m
Hard constraint low	2.5m

Table 6: Parameters used to the tank in the simulator. The width radius varies from tank 1 to tank 6 in the range between 7m to 10m

Below is the parameter used for the disturbance inflow to the first tank.

Parameter name	Value
Distribution model	Gaussian
Nominal flow	$1\frac{m^3}{s}$
Variance flow	$0.1\frac{m^3}{s}$
Max inflow	$2\frac{m^3}{s}$
Min inflow	$0.7\frac{m^3}{s}$

Table 7: Parameters used for the disturbance inflow to tank 1

C P-controller derivation

Since the tank model from eq. (35) is a differential equation of liquid height h was the equation linearized by Taylor expansion around the nominal values of the valve position and liquid height. The linearization can be seen in the following equation with the nominal values denoted by nom.

$$\frac{d\Delta h}{dt} = \frac{-f(z^{nom})A_{out}2g}{\pi r}\Delta h - \frac{A_{out}2gh^{nom}}{\pi r}\Delta f(z) + \epsilon \tag{45}$$

All the factors in front of Δh and $\Delta f(z)$ are now constant and are therefore substituted by the new constants called A and B. The error ϵ is removed in favor of an approximation of the original function.

$$\frac{d\Delta h(t)}{dt} \approx A\Delta h(t) + B\Delta f(z(t)) \tag{46}$$

Working with the approximation, the differential equation is then transformed by an Lapace transformation around $t = 0^+$.

$$h(s)s - \Delta h(t=0) = Ah(s) - Bf(z(s)) \tag{47}$$

The initial change in $\Delta h(t=0)$ is assumed to be zero. The notation of the state value h is substituted by the more commonly used notation x. The input variable $f(z)$ is also substituted by the notation u for the same reason. Reformulation eq. (47) results in the transferfunction G(s).

$$G(s) = \frac{\Delta x(s)}{\Delta u(s)} = \frac{K}{\tau s + 1} = \frac{\frac{-B}{A}}{\frac{-s}{A} + 1} \tag{48}$$

The gain K and τ is used with the SIMC tuning rule to evaluate the controller function. The system has no delay so the closed loop time constant K_c was evaluated as a function of only τ_c as seen in

$$K = \frac{A}{B} \tag{49}$$

$$\tau = \frac{1}{A} \tag{50}$$

$$K_c = \frac{\tau}{K\tau_c} \tag{51}$$

D Tuning of P-controllers

Below is the results from the running the tuning script in appendix D for all 6 tanks in the series as explained in section 4.2. 100 simulation was executed for each tau_c with sequential tuning starting from the first tank and backwards.

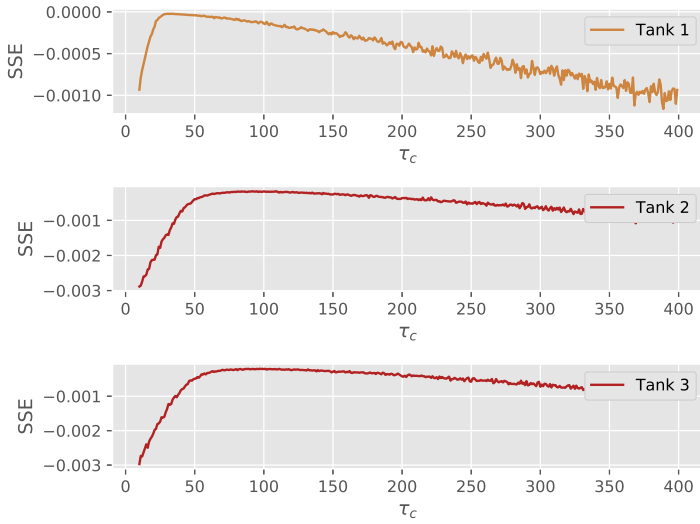

Figure 46: Mean squared errors of 400 τ_c evaluations for tank 1 to 3

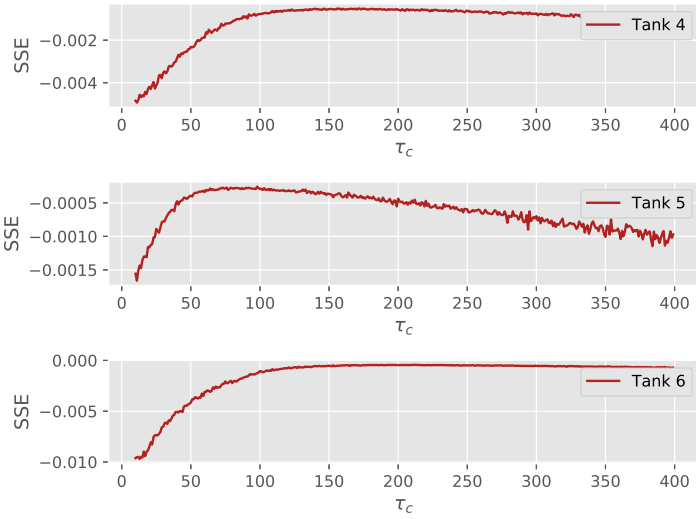

Figure 47: Mean squared errors of 400 τ_c evaluations for tank 4 to 6

E Auto tuned τ_c parameters for P-controllers

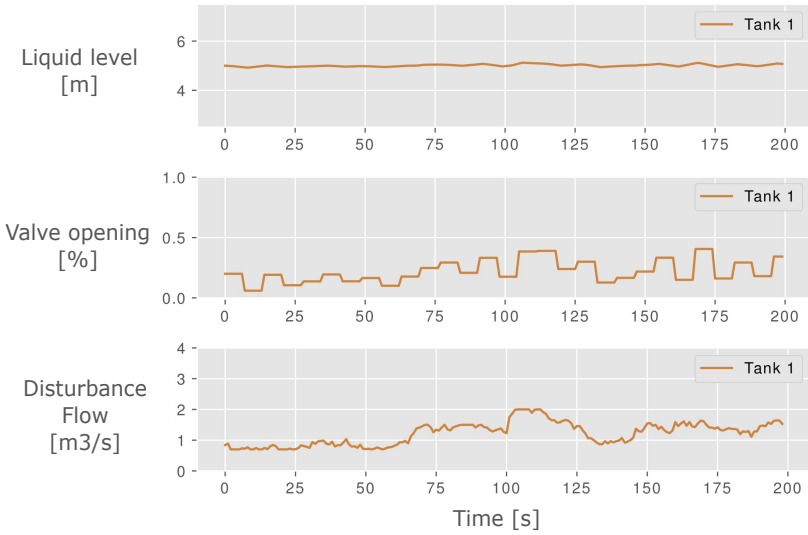

Figure 48: P-controller control for one tank evaluation plot with the τ_c parameters calculated using the tuning script

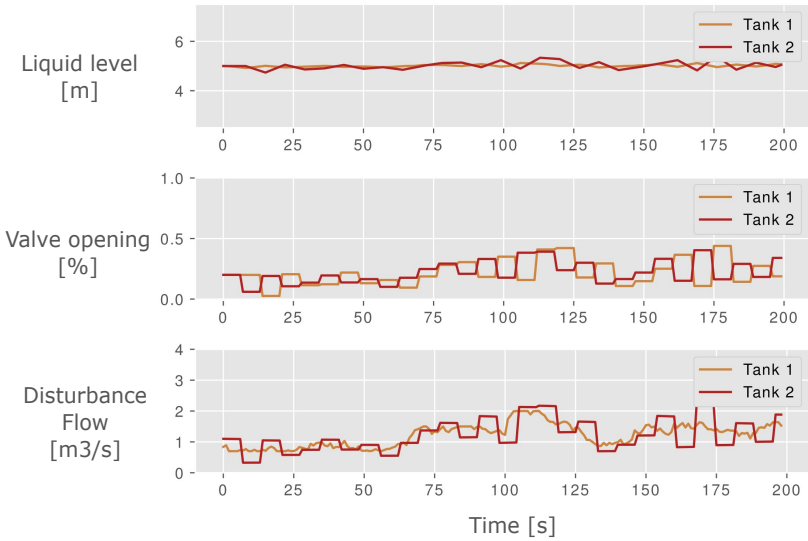

Figure 49: P-controller for two tank evaluation plot with the τ_c parameters calculated using the tuning script

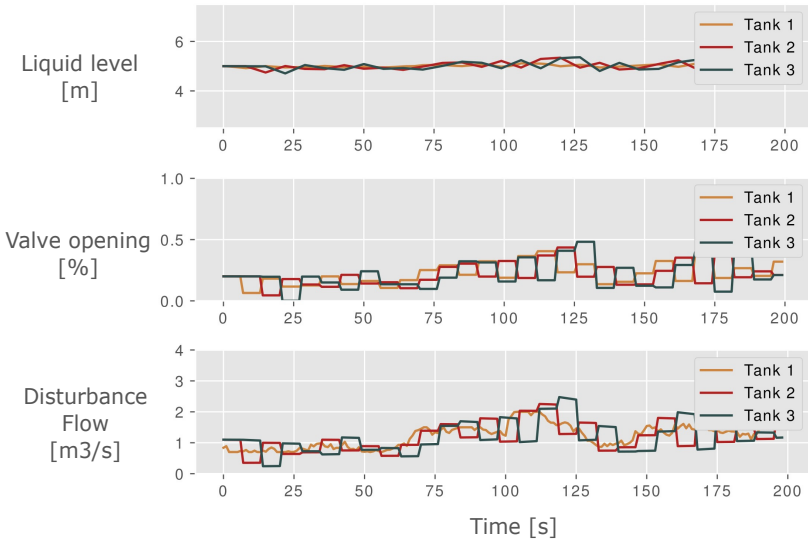

Figure 50: P-controller for six tank evaluation plot for the first three tanks with the τ_c parameters calculated using the tuning script

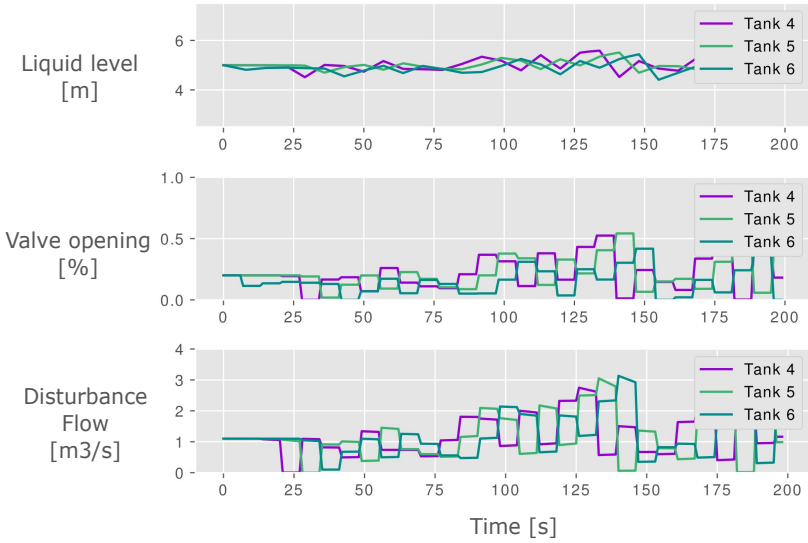

Figure 51: P-controller for six tank evaluation plot for the last three tanks with the τ_c parameters calculated using the tuning script

F Scripts

The scripts used in this thesis are available in the authors Github repository [24].

F.1 Tuning of p-controllers

The script appended below was used to evaluate different τ_c for the p-controllers used in one, two and six tank case. The method loops over a set of different τ_c and collects the SSE at all time steps for multiple episodes. The τ_c is then plotted with its corresponding mean error. The specific script below was used to tune the one tank case, and minor adjustments were made for tuning of the two and six tank case.

```
1  from main import main
2  import matplotlib.pyplot as plt
3  from params import TANK1_DIST, AGENT_PARAMS_LIST
4  import numpy as np
5  import sys
6
7  TANK1_DIST["pre_def_dist"] = False
8
9  tau_c_start = 10
10 tau_c_inc = 1
11 tau_c_end = 400
12 all_max_rewards = []
13 all_max_reward_values = []
14 number_of_tau_c_evaluations = 100
15 all_tau_c_app = []
16
17
18 def tune_controllers(tank_number=0):
19     rewards = []
20     max_reward = -99
21     max_reward_tau_c = 0
22     tau_c = []
23
24     tau_c = tau_c_start
25     tau_c_app = []
26     while tau_c < tau_c_end:
27         tau_c_app.append(tau_c)
28         reward = [
29             main(tau_c_tuning=tau_c, tuning_number=tank_number, plot=False)
```

```
            for i in range(number_of_tau_c_evaluations)
        ]
        rewards.append(np.mean(reward))
        if rewards[-1] > max_reward:
            max_reward = rewards[-1]
            max_reward_tau_c = tau_c_app[-1]
        sys.stdout.write(
            "\r"
            + "Tank "
            + str(tank_number + 1)
            + ": Current tau_c iteration: "
            + str(round(tau_c, 2))
        )
        tau_c += tau_c_inc
        sys.stdout.flush()
    print(f"\nSimulation Done for tank {tank_number+1}")
    print(max_reward, " with " + r"$\tau_c$" + ", max_reward_tau_c")
    all_max_reward_values.append(rewards)
    all_max_rewards.append([max_reward_tau_c, max_reward])
    all_tau_c_app.append(tau_c_app)
    return max_reward_tau_c


for i in range(1):
    max_reward_tau_c = tune_controllers(i)
    AGENT_PARAMS_LIST[i]["TAU_C"] = max_reward_tau_c

_, (ax1) = plt.subplots(1, sharex=False, sharey=False)
ax1.plot(
    all_tau_c_app[0], all_max_reward_values[0], color="peru", label="Tank 1"
)
ax1.set_ylabel("MSE")
ax1.legend(loc="upper right")
ax1.set_xlabel(r"$\tau_c$")

plt.tight_layout()
plt.show()
print(f"Best tau_c for Tank 1 was {round(all_max_rewards[0][0], 2)}")
```

F.2 Evaluation of controllers

The script appended below was used to evaluate the performance of the different controllers. The controller's exploration rate was set to minimum and the predetermined disturbance showed is fig. 15 was used for all evaluations. The specific script shown below was the evaluation of the DQN controller for the one tank case. Minor adjustments to the script were made for evaluation the REINFORCE, A2C, and P-controllers.

```
from models.Agent import Agent
from models.environment import Environment
from evalv_params import MAIN_PARAMS, AGENT_PARAMS, TANK_DIST
from params import TANK_PARAMS
import os
import matplotlib.pyplot as plt
import numpy as np
from rewards import get_reward_3 as get_reward
from rewards import sum_rewards

plt.style.use("ggplot")


os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"


def main():
    # ============= Initialize variables and objects ===========#
    environment = Environment(TANK_PARAMS, TANK_DIST, MAIN_PARAMS)
    agent = Agent(AGENT_PARAMS)
    z = []
    h = []
    d = []
    # ================= Running episodes =================#

    state, episode_reward = environment.reset()
    h_ = np.array([state[0][0][0]])
    h.append(h_)
    for t in range(MAIN_PARAMS["MAX_TIME"]):
        action = agent.act(state[-1])  # get action choice from state
        z_ = agent.action_choices[
            action
        ]  # convert action choice into valve position
        z.append(np.array(z_))
        terminated, next_state = environment.get_next_state(
            z[-1], state[-1], t
        )  # Calculate next state with action
```

```
        reward = sum_rewards(
            next_state, terminated, get_reward
        )  # get reward from transition to next state
        # Store data
        episode_reward.append(reward)

        state.append(next_state)
        h_ = []
        d_ = []
        for i in range(agent.n_tanks):
            d_.append(
                environment.tanks[i].dist.flow[t - 1] + environment.q_inn[i]
            )
            h_.append(np.array(next_state[i][0]))
        d.append(d_)
        h.append(h_)
        if environment.show_rendering:
            environment.render(z[-1])
        if True in terminated:
            break

        if not environment.running:
            break
    print(np.sum(episode_reward))
    _, (ax1, ax2, ax3) = plt.subplots(3, sharex=False, sharey=False)
    d = np.array(d)
    h = np.array(h[:-1])
    z = np.array(z)
    h *= 10

    ax1.plot(h[:-1, 0], color="peru", label="Tank 1")
    ax1.set_ylabel("Level")
    ax1.legend(loc="upper right")
    ax1.set_ylim(2.5, 7.5)

    ax2.plot(z[1:, 0], color="peru", label="Tank 1")
    ax2.legend(loc="upper right")
    ax2.set_ylabel("Valve")
    ax2.set_ylim(-0.01, 1.01)

    ax3.plot(d[:, 0], color="peru", label="Tank 1")
    ax3.set_ylim(0, 4)
    ax3.set_ylabel("Disturbance")
    ax3.legend(loc="upper right")

    # plt.legend([l1, l2, l3], ["Tank height", "Valve position", "Disturbance"])
    plt.tight_layout()
    plt.xlabel("Time")
    plt.show()
```

```
87
88
89  if __name__ == "__main__":
90      print("#### SIMULATION EVALUATION STARTED ####")
91      print("  Max time in each episode: {}".format(MAIN_PARAMS["MAX_TIME"]))
92      main()
```