Master's thesis

NTNU
Norwegian University of Science and Technology
Department of Engineering Cybernetics

Robin Honningsvåg Kleiven

Applying Reinforcement Learning for Controller Scheduling

Master's thesis in Department of Engineering Cybernetics
Supervisor: Ole Morten Aamo
June 2020

NTNU
Norwegian University of
Science and Technology

Robin Honningsvåg Kleiven

Applying Reinforcement Learning for Controller Scheduling

Master's thesis in Department of Engineering Cybernetics
Supervisor: Ole Morten Aamo
June 2020

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

In controller/gain scheduling the boundaries that need to be defined for the control switching/gain switching can be challenging. In this thesis, the possibilities of applying reinforcement learning (RL) for controller scheduling have been investigated. The algorithms used were the tabular methods; Q learning and state-action-reward-state-action (SARSA). The coupled tank system was used to investigate the research questions in this thesis. The state-action space was big and that caused the RL agent to oscillate between the controllers. The state-space was then reduced. This removed the oscillatory behavior and the agent was able to pick the optimal controllers. The controllers used were the linear quadratic regulator (LQR) together with a minimum and maximum controller. Additionally, it was studied what would happen if a proportional-integral (PI) controller and a proportional derivative (PD) controller were added. This did not give any new insight into the problem.

The coupled tank system was then made more complex by introducing more nonlinearities such that the solution required more than only one controller for satisfactory control. An emergency valve system was used that separated the system into 4 subsystems by adding an additional valve in each tank that opened when a condition was met. This condition varied from; water level too high and randomly switching on and off the valves. For each of these subsystems, an LQR was made. The agent was able to define a set of rules to switch between the correct LQR to control around a set point for each case with this emergency valve system, but it seems to not switch optimally to get to this equilibrium as fast as possible. The action choices of the agent have some oscillations and it was shown that lack of training was a big component that caused this and by increasing the training of the agent this was reduced. RL was successful in defining a set of rules for controller scheduling for this complex system.

Abstract - Norwegian

Ved regulering / parameter tilordning kan grensene som må defineres for kontrollbytte / parameterbytte være utfordrende. I denne oppgaven er mulighetene for å anvende forsterkningslæring (RL) for kontrollplanlegging blitt undersøkt. Algoritmene som ble brukt var tabellmetodene; Q-læring og tilstand-handling-belønning-tilstand-handling (SARSA). Det koblede tanksystemet ble brukt til å undersøke hypotesene i denne oppgaven. Handlingsrommet var stort, og det fikk RL-agenten til å svinge mellom kontrollerne. Tilstandsrommet ble deretter redusert. Dette fjernet den svingende atferden og agenten var i stand til å velge de optimale kontrollerne. Kontrollerne som ble brukt var den lineære kvadratiske regulatoren (LQR) sammen med en minimum -og maksimal -kontroller. I tillegg ble det studert hva som ville skjedd hvis en proporsjonal-integrert (PI) -kontroller og en proporsjonal-derivat (PD) -kontroller ble lagt til. Dette ga ingen ny innsikt i problemet.

Det koblede tanksystemet ble deretter gjort mer komplekst ved å innføre flere ikke-lineariteter slik at løsningen krevde mer enn bare en kontroller for tilfredsstillende kontroll. Et nødventilsystem ble brukt som skilte systemet inn i 4 delsystemer ved å legge til en tilleggsventil i hver tank som åpnet når et kriteria ble oppfylt. Dette kriteriet varierte fra; for høy vannstand og tilfeldig slå av og på ventilene. For hvert av disse delsystemene ble det laget en LQR. Agenten var i stand til å definere et sett med regler for å veksle mellom riktig LQR for å kontrollere rundt et settpunkt for hvert av tilfellene av dette nødventilsystemet, men det ser ut til å ikke bytte optimalt for å komme til likevekts punktet så raskt som mulig. Agentens handlingsvalg har noen svingninger, og det ble vist at mangel på trening var en stor komponent som forårsaket dette, og ved å trene agenten mere ble dette redusert. RL lyktes i å definere et sett av regler for kontroll planlegging for dette komplekse systemet.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning has been around for decades, but in recent years, more and more people are using it daily without even realizing it. It is a part of daily life for almost everyone. This is due to technological advancement, especially the increased use of mobile phones and social media. Mobile phones and other smart devices like smartwatches, smart fridges, tablets, etc have made it possible to collect an enormous amount of data. Lack of data has long been a problem with machine learning (3), but with the availability of smartphones, big companies such as Apple and Google have access to all the data they need to make search engines, spam filters, virtual assistants, etc with the help of machine learning algorithms.

Machine learning is as the word says; a machine that is learning. And for a machine to learn it needs something to learn from. It can learn how inputs affect output given examples (supervised learning), it can learn the underlying hidden structure of the data without prior knowledge (unsupervised learning) or by exploring the environment - reinforcement learning (RL). Both supervised and unsupervised learning need massive amounts of data, and the most used way is through an artificial neural network. An artificial neural network works by breaking down an example to simpler pieces and then build these pieces to form the full picture. Take a picture of a square. It has 4 corners and 4 edges. The neural network tries to find these and from that forms the entire image. To be able to do this with all kinds of squares the network needs a lot of examples to learn from. These examples are the data it will learn from, and by training the machine with the examples it will eventually learn to copy the examples. As in the example above it will learn how a cat is supposed to look like, what is spam and what is not spam. Unsupervised learning algorithms are used to pre-process the data during exploratory analysis or to pre-train supervised learning algorithms (4).

The increase of computational power and technological advancement has motivated more complex systems. More attention to the last learning method (RL) has increased in recent years due to that. Both (5) and (6) are examples of the increasing importance in a strategy

that can deal with such complex tasks. RL uses rewards/penalties to learn. The machine or the agent is the entity that interacts with an environment and based on what the agent does it receives rewards or penalties, depending on how good it performs. What is great about this is that one does not need to know the exact model of the system. This means that RL is flexible and can be used for many things such as; manufacturing (7), inventory management, optimize financial objectives (8), delivery management (9) and solving complex nonlinear control problems which has been done by; (10), (11) and (12). This research focuses on the latter, which is complex control tasks.

1.1 Problem description

The purpose of this research is to study a nonlinear dynamical system that uses more than one controller to operate around a set point. The system chosen for this is a coupled tank system. The idea is that there is an optimal switching strategy between the controllers to reach the set point as fast as possible, efficiently switch controllers if the dynamics change, and stay at this point for as long as needed. The main focus is the two latter points. To learn this optimal switching strategy RL is applied to this system with 3-6 controllers to see if the agent can figure out what the optimal path is. These controllers are; minimum controller, maximum controller, linear quadratic regulator (LQR), proportional-integral (PI), and proportional derivative (PD). The algorithm used is the tabular method; Q learning. This looks at each state and determines how good this state is given the action taken.

The coupled tank system is then made more complex by introducing more nonlinearities such that the solution requires more than only one controller for satisfactory control. Since this requires more than one controller for good control, 4 LQRs, each tuned around their linearization area, are used together with a maximum and minimum controller.

1.1.1 Research Questions

The research questions for this research are:

- Can an RL agent be efficiently used to define a set of rules for when to switch between controllers, that is controller scheduling?
- If so, how trivial could this be made?
- Can this be combined to control a complex system that needs more than one controller to achieve the control objective?

1.1.2 Motivation

With any given complicated problem it is normal to split the problem into smaller simpler pieces and then try to form the full picture by solving each individual piece. In nonlinear control it is common to linearize around an operating point, and use this linearized model when designing the controller. This is usually enough for most applications, but as the dynamics gets more and more complicated, a simple controller tuned to operate around

one set point is not enough. Control strategies such as gain scheduling and designing multiple controllers to operate around it's own set point have been used to solve these complicated problems (10), (6) and (5). But designing a switching strategy between the controllers and a rule for how the parameters should change depending on which state the system is in, is in it's own right a complex problem. If one could leave this problem to the RL agent it might end up less complicated.

1.2 Report structure

This report follows the guidelines of (13) to structure the report. The next chapter gives a brief review of similar work and discusses these. Chapter 3 gives a basic introduction to the theory needed to do the experiments. It consists mostly of the theory around RL and the different techniques used, as well as a brief segment about proportional integral derivative (PID) and LQR control. Chapter 4 introduces the system and its dynamics. It gives an overview of how the system works, what choices were made, why they were made, and lastly the linearized system is presented. Chapter 5 talks about how it started as a simple system to get everything to work and the extensions made at each new step. It briefly walks through the methods used in this work. First the design of the agent and controllers are presented, then an overview of how the communication between the system and agent works and lastly the extensions which were made to the system. Chapter 6 presents the results and discusses them in detail. The final section draws a conclusion with respect to the project as a whole.

Chapter 2

Literature Review

This chapter delves into the literature of nonlinear control, or more specifically, intelligent control strategies for nonlinear dynamical systems. We will look at different nonlinear systems and control of these systems but mostly focus on the coupled two-tank system and how RL can be used to control these systems with gain scheduling and switching of multiple controllers.

Nonlinear systems have been of great interest to engineers, biologists, mathematicians, and other scientists because most systems are nonlinear by nature. A common way to make controllers for these systems is to linearize it around a set point, or more than one set point, and operate it around these set points. For many industrial processes, a PID controller is sufficient. Sometimes more than one PID is used when the system in question has nonlinearities around certain areas, or when one controller simply can not fulfill the control objective. There are several strategies to control the liquid flow to a single or multiple coupled tank system as shown in (14), (15), (16), (17) and (18). These control-strategies seem to be good enough for this system, and (14) seems to be robust for parameter change as well. But would switching between multiple controllers/gain scheduling be just as efficient and robust for the general nonlinear system? The rest of this chapter looks at how one could switch between controllers or change the parameters in the controllers with the use of RL.

Gain scheduling is a method to change the gain parameters for a linear controller based on the state the system is in. The regular way to do this is defining areas for each gain parameter, and when the system transits to a new area the new parameters are chosen for the controller. The same approach could be used for switching between multiple controllers, this is called controller scheduling. This is done by designing multiple linear controllers for several operating points and interpolate to find a global controller (19). This paper will look at how this could be done intelligently, that is, with RL. In (20) they used a method called handicapped learning together with an RL scheme to solve a control problem. They used a nonlinear state encoding of the system, a new associative reinforcement

learning algorithm, and a novel reinforcement scheme to explore the control space to meet the scheduling constraints. They incorporated two learning heuristics; state recurrence and the ”handicapped learning” heuristics that this paper introduced. These techniques were used for basic set-point control in a continuously stirred tank reactor in which the temperature must be held at the desired set point. The scheme was able to learn satisfactorily, but this paper was written a long time ago, and since then, computational power has increased significantly. They tried to encode the states to not get an explosion of states in the traditional RL schemes. This could be done by a simple neuronal network today, and they do not need the handicapped learning scheme they developed to decrease the number of state encoding.

This next paper tried to control a biped robot (10). They did this by switching between multiple controllers. And the switching happens by the use of an RL agent. But this requires a complex switching mechanism. Hence something called a ”melting pot” is used. The melting pot is a central controller that uses the experiences of the other controllers to learn an average control policy. The central controller controls the robot in nominal conditions, and the other controllers, called peripheral controllers take over if the central controller's action deviates from each control policy. Both the peripheral controllers and the central controller use an adaptive cerebellar model articulation controller (CMAC) neural network. Instead of having one big network they use many with fewer inputs. This, and splitting between the central controller and peripheral controllers seemed to solve the problem with large inputs and they were able to accurately model the robot walking. But to be able to do this they used a support walking cart that the robot was pushing as well as adding extra weight to the biped robot to stabilize the movement. They also assumed a flat walking surface and that there exists a nominal behavior controlled by a minimal number of inputs. These assumptions limit the general usage of the whole system only to similar scenarios.

Control switching has also been applied to simpler systems such as (21) and (22). In (22) their goal is to make an RL scheme that combines traditional control theory to construct a global controller for unstable nonlinear systems. To see if this scheme works they try to control an acrobot. They construct several incomplete controllers that handle a subspace of the control problem and combine these to get a global controller. The incomplete controllers are LQR, sigma 1, and 2 controllers, a brake controller, and a zero torque controller. The sigma 1 and 2 controllers are to get the acrobot to a standing position, the LQR to keep it at the standing position and the brake and zero torque controller speaks for itself. State-action-reward-state-action (SARSA) is used as the RL scheme. SARSA is similar to Q-learning but uses the state and action for the current state and next state. This paper also discretized the state space instead of using a neural network which the previously mentioned papers did. They were able to show that this scheme achieved good stabilization and control for the acrobot, but one can not say the same for the general nonlinear system. This paper only simulated the movement of the acrobot and did not try it on a real system, which was the initial goal of the paper. They assumed that there was no noise, the state space was observable and that they knew the exact dynamics of the system.

Control switching by the use of RL was also done in (21). They used Q learning instead of the SARSA method together with 3 unknown black-box controllers, that were pre-tuned.

The goal of the paper was to compare the performance of the controllers one by one and the global controller defined by all three controllers. They used voltages to measure the flow and height of each tank. Each tank had a pressure sensor at the bottom that could read from 0-4 volts. This paper also used discrete states for the Q learning scheme but they discretized the error voltages into buckets of 0.2 volts. The results showed better responses with all three controllers combined. The overall strategy is explained in the paper, but there are a lot of details missing to be able to replicate the said experiment. The figures alone do not give enough details about the controllers for any reader to make the same controllers. They wrote that the agent chose a controller that took advantage of its best characteristics, but how should the reader know exactly what each controller's best characteristics are when they are not specified?

The use of local controllers combined with a global controller is a good strategy to control nonlinear and potentially unstable systems as shown in [(23), pages 12-14]. But the boundaries that need to be defined for the control switching can be challenging. The same applies to gain scheduling control strategies. With the use of RL one does not need to know everything about a system to be able to efficiently switch between controllers (22), and since general systems get more and more complex and more data is available than ever, it might be clever to rely more on a computer to do the job for us than to try and come with complex strategies to solve control problems. And using RL to define the rules for switching between the controllers is more flexible and scalable than defining each rule for each subspace individually.

Chapter 3

Background

In the last chapter, some of the previous work done on similar topics were discussed. In this chapter, some theoretical concepts about RL and general control theory will be introduced. The main focus will be about RL, where the basic concept will be explained. How an RL agent learns and how rewards are given. Then some of the most common RL techniques will be introduced like Q learning, deep Q-learning, SARSA, and double Q learning. Then neural network will briefly be presented. The theory behind the controllers used will be talked about in more detail; more specifically the PID and LQR equations and concepts will be introduced. All of the theory presented in this chapter is taken from the collection of these sources; (24), (25), (26), (27), (28), (29), (1), (30), (31), and (32). These sources are repeated throughout the chapter and specific citation of which chapter will also be specified if necessary.

3.1 Reinforcement learning

3.1.1 General reinforcement learning

The theory of RL is inspired by the psychology of behaviorism where we learn to behave based on our experiences. Every action has its consequences. Most of us learn that being close to the fireplace is good because it warms us. But to touch the fire is bad because the fire hurt us. This is the basic concept of RL. Through actions, we either learn that something is good; positive reinforcement, or bad; negative reinforcement.

Other than the agent and the environment, one can identify four main subelements of an RL system: a policy, a reward signal, a value function, and sometimes a model of the environment.

A policy is a description of how the agent should behave at any given time. It is a mapping from the perceived states of the environment to action to be taken in those states. The

whole learning process behind RL is to learn the optimal policy. The policy could either be a simple function, a lookup table, and in other cases, it might be a search process. In the mentioned cases, learning the policy would consist of learning the function, update the table to optimal values, and learn the best weights in the search process.

A reward signal defines the goal of the RL problem. On each timestep, a reward is calculated based on how well the agent did. The goal of the agent is to maximize the total reward over many timesteps. Like explained earlier, this can be tied to human and animal behavior. Pain is negative while pleasure is positive. We make decisions based on this feedback, and the same is used for RL. One can either use positive reinforcement, negative reinforcement, or both together as a reward metric. An example of this could be a robot walking through a terrain. Moving without colliding gives -1 reward. Hitting something gives -100 reward, and getting to the finish line gives +100 reward. This way the robot wants to get to the goal as fast as possible while avoiding collisions.

The value function serves a different purpose than the reward. The reward is an immediate indication of how good an action is in this state, whereas the value function is a measure of how good something is in the long run. In other words, the value of a state is the total amount of reward the agent can expect to get in the future, starting from that state. Both are important to find the optimal policy. One way to think of the value is exercise, you will not feel the benefits of exercising immediately, but the benefits will show in the long run [(24), pages 5-6].

3.1.2 Learning

MDP

An agent wanders around in an unknown environment to learn the best possible course of action that gives the best rewards to reach the desired goal. The challenge is to understand how these actions will affect future rewards. A good way to model these problems is with an MDP. This has become the best approach to solving RL problems.

"*MDPs are mathematically idealized form of the reinforcement learning problem for which precise theoretical statement can be made.*" [(24), page 53].

In all artificial intelligence, there is a tension between the wide range of application that can be made with RL and the mathematical tractability. An MDP is defined by a process of a 4-tuple (S, A, P_a, R_a)[(24), page 53]:

- S - finite states
- A - finite set of actions the agent makes in a state
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability action a in state s at t will lead the agent to the state s' in t+1
- $R_a(s, s')$ the immediate reward given to the agent when moving from s to s'

Figure 3.1 shows how the agent interacts with the environment

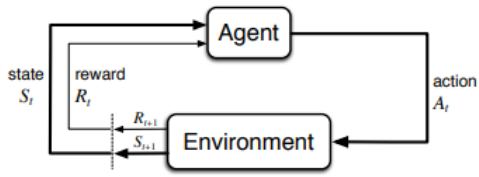


Figure 3.1: Agent interacting with the environment in MDP.

The agent and the environment continually interact with each other. The agent selects an action and the environment responds to these actions and new situations are presented to the agent. The RL algorithm must find an optimal policy by interacting with the MDP directly, such that the behavior of the agent in the environment is optimal. And for this to happen the algorithm needs to visit every action-state pair infinitely many times as stated in [(24), pages 113-142] and (25).

Monte Carlo and TD learning

There are two types of learning in RL. One is Monte Carlo where the rewards are collected at the end of each episode and then the maximum future reward is calculated. This means that at the end of each episode the agent looks at the total cumulative reward to see how well it did but the rewards are only given at the end of each episode.

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Where V on the left side is the expected future reward starting at that state. And the V on the right side is the former estimation of maximum future reward starting at that state. Alpha is the learning rate and G is the discounted cumulative reward [(24), chapter 5].
The other learning method is the TD method. This is also called a one-step method since it updates the estimate of the expected future reward at every timestep. It will update its value estimation V for every nonterminal state happening at that experience. The value at the next time step is calculated using the formula:

$$V(S_t) \leftarrow (1 - \alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1})) \tag{3.1}$$

Here the parameters mean the same as in the previous equation. γ is the discount value and $\gamma \mathbf{V}(S_{t+1})$ is the discounted value in the next timestep. The γ parameter indicates how far ahead the agent looks. If it is close to 1 it will prioritize rewards in the distant future. If the value is close to 0 only rewards in the immediate future will be considered. R is the reward for timestep t. And these together are called the TD target, which is the estimated value for the next timestep. The formula for TD learning is almost the same as that for Q-learning, which will be talked about later [(24), chapter 6].

Exploration vs Exploitation

When talking about learning one always talks about something called exploration and exploitation. Should the agent explore new states in the environment to possibly find better rewards or should the agent exploit known knowledge about the environment? This is the exploration vs exploitation trade-off. The way this is done is by choosing a number, which is usually called ϵ - epsilon, and let it decrease with time. At the start, this parameter is set to 1 to fully explore the environment because the agent does not know anything about it yet. When the epsilon parameter decreases the agent exploits current knowledge about the environment. In figure 3.2 it shows how the parameter should be changed based on the knowledge of the environment. The less the agent knows the more it should explore and the more the agent learns about the environment the more it should exploit known knowledge to achieve the best possible value.

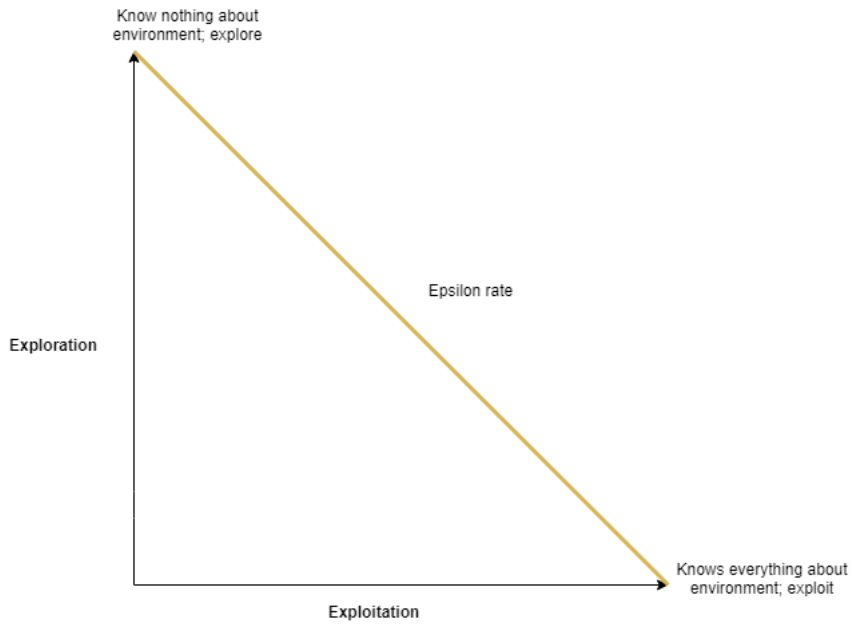


Figure 3.2: Exploration vs exploitation.

Reward function

When choosing the reward function for an RL problem it should be noted that this is not where you impart to the agent your knowledge about how to achieve what we want it to do. The reward function is a way of communicating to the agent what the goal is, not how we want it achieved [(24), pages 42-43]. The whole point of RL is gone if we hold the agent's hand and through the reward function, try to teach the agent how it should behave. If the goal, for a robot walking through a maze is to get out as fast as possible, it is logical to punish the robot for each step it is in the maze, and give a huge bonus reward once it is

out. The robot does not care about what we want it to do or how we want it to behave, it simply tries to maximize the reward function. We should not tell the robot what to do, all we should do is propose rewards for what is bad, and what is less bad or good. We can't directly tell the agent how to behave. If the goal is to clean the maze, one can reward the robot every time it picks up trash, and if you want that done fast punish the robot for each step it takes.

Designing a reward function for real-life applications is rarely a trivial task. The reward function can vary from what was described above. Simple numbers that reflect if the agent did something bad or something good. In this case, the agent needs to figure out how it got the reward/punishment, this is known as the credit assignment problem. The reward function can be an actual function as well. The distance from the goal can be used as a negative reward to encourage the agent to decrease that distance as fast as possible. One can use the states of a system as well, like velocity, position, and control effort. It is important to consider the relative sizes of these values such that their contribution is appropriately weighed. In general, it is important to define a reward function that is rich in information for change in state and action (28).

Convergence of RL

For many easier problems such as the general grid world problem (33) it is enough to put constant values for the parameters. When the problems become more complex, that is many states and actions, one needs some mathematical ground to prove that the RL algorithm converges to the optimal solution. Proofs of this will not be done in this paper. The discount factor and epsilon have already been discussed above. If exploration is not present in the algorithm, the agent might find the optimal path, but more often than not it will exploit current knowledge and the algorithm converges to a local minimum. Each state-action pair needs to be visited infinite times for the algorithm to converge [(24), pages 113-142] and (25).

The learning rate α needs to be something different than a constant. According to the sources (25) to guarantee convergence, the learning rate needs to fulfill two requirements:

$$\begin{aligned} \sum_{n=1}^{\infty} \alpha(t) = \infty \qquad (1) \\ \sum_{n=1}^{\infty} \alpha(t)^2 < \infty \qquad (2) \end{aligned} \tag{3.2}$$

This means that alpha needs to decrease for each timestep. Requirement 1 tells us to not decrease alpha too fast and 2 tells us that alpha decreases fast enough for it to converge. This will guarantee convergence for the RL algorithm with a probability 1 (25).

Value based and Policy based learning

There are three approaches to RL and those are value-based, policy-based, and model-based. As already discussed, the value-based approach tries to optimize the value function

$V(s)$. This function tells how much the maximum expected future reward the agent will receive at each state:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...|S_t = s] \tag{3.3}$$

In policy-based RL the policy function is directly optimized without the use of a value function. As said before, the policy describes the agent behavior at a given time. A policy function **a** $= \pi$**(s)** is learned that lets us map each state to the best action $action = policy(state)$. Policy-based RL can be both deterministic, where the agent will take the same action for a given state, or stochastic, where the action can be random. Lastly, there is a model-based RL, which will not be further explored in this paper (34).

3.1.3 Q-learning

Q-learning is a value based RL algorithm that uses something called an action-value function together with a look up table to solve a MDP. This function is almost the same as the value function in 3.3 [(24), page 70], but the difference is that it gives the action-value for the given state **and** action:

$$Q^\pi(s_t, a_t) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...|s_t, a_t] \tag{3.4}$$

The action-value function gives the quality of the action in that state given the state and the action. What this means is that it is a measure of how good the expected future reward of that action in that state is. The look-up table is just a table filled with state-action values; q values. A simple example is a world with 2 states (a 2-D grid) where an agent could take 4 possible actions; up, down, left, and right. For each state, there are 4 q values, where the best value gives the best action in that state. At the start, the Q table has initial values which are 0 for the most, and as the agent explores, the Q table will give better and better approximations by updating the table. The table is updated by using the Bellman equation [(24), page 90] or (35):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma maxQ(s_{t+1}, a_t) - Q(s_t, a_t)] \tag{3.5}$$

The general algorithm for Q learning is given in figure 3.3 below as well as a pseudo code:

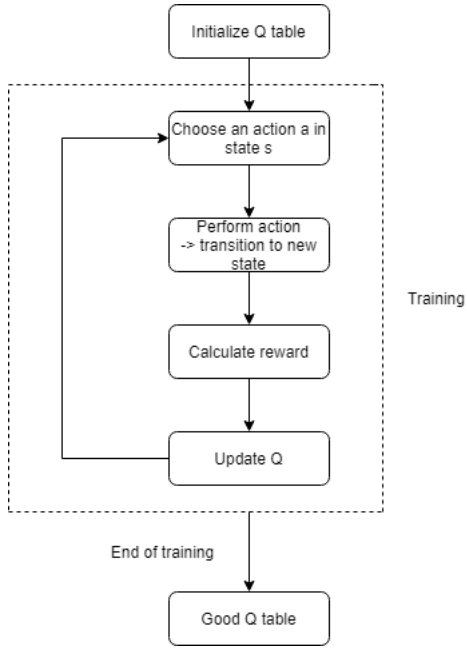


Figure 3.3: Flow chart of the Q learning algorithm.

1. Initialize the Q table's values Q(s,a).
2. For each episode in total episodes
3. For each step in each episode
4. Choose an action a in the current state s based on the current best estimate of Q(s,·)
5. Take the action and observe how the environment evolves (new state s') and the reward
6. Use the bellman equation 3.5 to update the Q value

The figure, as well as the pseudo-code, explains the main important parts of the algorithm, but some things need further explanation.
The initial Q values are arbitrarily chosen, according to the literature these values are set to 0, but there are other ways to initialize these (36). In step 4 what is meant by the best estimate of $Q(s, \cdot)$ is the action with the highest value for that state. For the 2-D grid example that would mean choosing the best of the 4 possible actions that have the biggest action value. But this alone means that the agent is greedy and will exploit current knowledge all the time and might miss possible better action is each state. Randomness is added to make the agent explore more. This is the ϵ parameter mentioned earlier. Whenever a random number between 0 and 1 is bigger than epsilon, the best-estimated action for that state is chosen, otherwise, a random action of the possible actions is chosen. At the start, the parameter is chosen to be big because initially the Q table only has 0's. And epsilon is

decreased when the agent has explored and updated the Q table to other values as depicted in figure 3.2.

In the next step, the action is taken and the environment changes. Based on this the reward is calculated and the maximum expected future reward, given the new state and all possible actions are calculated. These values are used to update the Q table with the equation 3.4. At the end of the training, the Q table can be seen as a complete, optimal description of how the agent should move to get to the desired goal (35).

SARSA

Next, several other similar RL techniques will be briefly introduced and explained: SARSA is an on policy algorithm that uses the current action performed by the current policy to learn the Q value. It uses the current state, current action, reward obtained, next state, and next action, hence the name SARSA [(24), pages 154-157].

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.6}$$

As seen the equation is almost the same as the Q update function, but instead of taking the maximum future expected value of all possible action given the new state, the Q value for the next state and action is taken.

Double Q learning

Both the Q learning algorithm and SARSA is greedy in the sense that they choose the optimal action in that state. In stochastic MDPs Q learning's performance can be poor because of the large overestimations of the action values. This leads to high positive bias values. In other words, Q learning struggles with maximization bias. A way to see this is to imagine an agent in a state. In this state, the agent has a number of actions to choose from, and all these actions have a true q value of 0 but the estimated values are uncertain and have distribution around 0. This means the average of the estimated values is above 0, thus a positive bias (29).

Algorithm 1 Double Q-learning

Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s', a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s', a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s', a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s', b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

Figure 3.4: Double Q learning algorithm.

One way to look at the problem is that the same samples are used to both determine the maximizing action and to estimate its value. To avoid this two Q functions are used and only one is updated at each step. This is chosen randomly. Then the action is chosen based on the sum of both the Q functions. Details of the algorithm is shown in figure 3.4.

3.1.4 Deep Q learning

Q learning and the algorithms discussed are perfectly fine to use even with big state spaces. But what about state spaces that are gigantic and seem to have infinitely many states? Classical Q learning is not scalable when the state-action space becomes enormous. The number of visits for each state-action pair in the Q learning algorithm has to approach infinity for it to converge to the optimal policy. This means for big state spaces the classical Q learning algorithm may never converge.

This issue is solved by deep Q learning. The deep refers to the use of a neural network to approximate the Q values for each action in that state as shown in figure 3.5

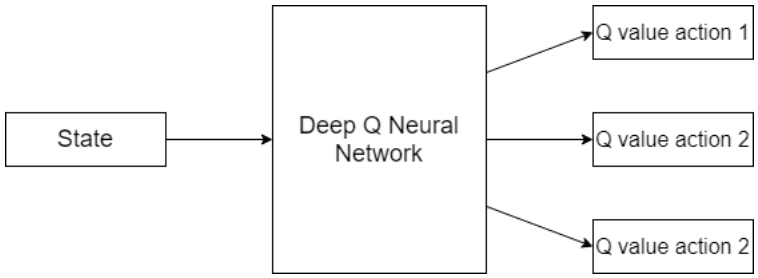


Figure 3.5: Deep Q learning.

Instead of updating a lookup table with 3.4 and using this table to transition from state to state, the weights in the neural network are updated. These weights function as paths between nodes in the neural network, and by tweaking them they will form a decision that picks the best action. The weights are updated with the following equation:

$$\Delta w = \alpha[(R + \gamma max_a \hat{Q}(s', a, w)) - \hat{Q}(s, a, w)]\nabla_w \hat{Q}(s, a, w) \tag{3.7}$$

The expression inside the square brackets is the TD error and functions the same way as the TD error as in previous learning techniques. Inside the parentheses is the target, that is the maximum possible value for the next state. The last expression is the gradient of the current predicted Q value. Most of the heavy lifting is done by the neural network. It is common to use a convolutional neural network (CNN) in deep RL and deep Q learning. How a basic neural network works will be explained in the next section.

The training in deep Q learning is more extensive than the traditional approach. It is smart to preprocess the input state whenever possible to reduce the complexity of the states to reduce training time. As said earlier the state space can be huge and the difference between

training time could be several hours. Another way to reduce the total amount of training a deep agent needs is through experience replay. At each timestep, the network receives a tuple (state, action, reward, next state) and then throw away the experience. The problem is that sequential samples from environment interaction are given to the neural network. This means that it can overwrite previous experiences with new ones. The solution to this is to have a replay buffer that stores the experience tuple while interacting with the environment and then use some of this buffer to train the neural network. And by sampling from the replay buffer at random the problem of correlation between tuples is avoided as well (37).

Neural network

A neural network consists of layers upon layers of nodes 3.6 that outputs a number depending on the node's attribute. This output number depends on all the inputs to the node and their corresponding weights. If the output is high, that is close to 1, the percepton is said to be lit up. In other words, the output number will determine how the output numbers of other nodes will be and how much they "fire".

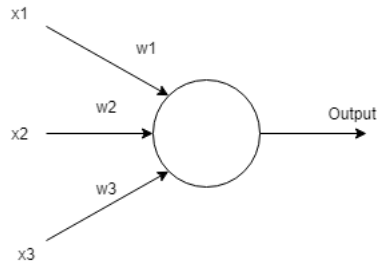


Figure 3.6: A single percepton.

In the figure below 3.7 a network with 4 layers is shown. The first layer is always called the input layer and the last is called the output layer. All the layers in between are called hidden layers. The inputs cause some pattern in the next layer, which causes some pattern in the next layer and this causes some pattern in the output layer. The one with the highest value is the network's best guess. The hidden layers work as collecting simple pieces from the full picture and puzzling them together to give an output. What is meant by that is that if one imagines that the network is trained to recognize shapes. The first hidden layer might pick up on edges, and the second might pick up on corners. This is not what the hidden layers do in actuality. But they do pick up on simple pieces and puzzle it together to form the full picture.

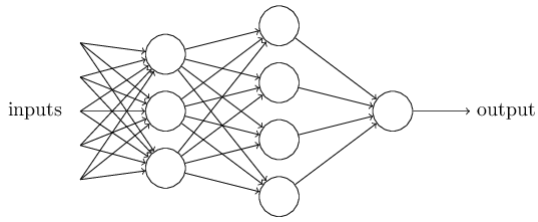


Figure 3.7: A simple neural network with 4 layers (1).

The network learns through something called gradient descent with backpropagation. What that means is that one starts at the output layer and calculates the gradient of the weights to adjust them to better values. How mathematics is and how the algorithm is in details will not be further explained in this paper.

This is the most basic of neural network. The one used in deep Q learning is a deep convolutional network. The basic idea of a convolutional network is to reduce the complexity of a regular network and to keep spatial features. If the parameter space becomes too big, the amount of data needed for training will increase significantly. In regular network the data is flattened to a 1-D vector, spatial information is lost. This is avoided by using convolutional layers that use filters to convolve with for example an image. This results in a new reduced grid of numbers. Using many of these filters will reduce the parameters into the fully connected layers. The fully connected layers is a neural network that was just introduced above. The whole purpose of the convolutional part of CNN is to reduce the parameters that go into the fully connected layers while keeping the most valuable information of the inputs.

3.2 Control theory

The basic problem with any closed-loop control tasks is to steer a system to the desired set point. This is done with feedback. The output of the system is compared to the desired set point and the error is used as an input to the controller. The output of the controller will depend on the control structure and this will determine the new state of the system (31). This is shown in figure 3.8

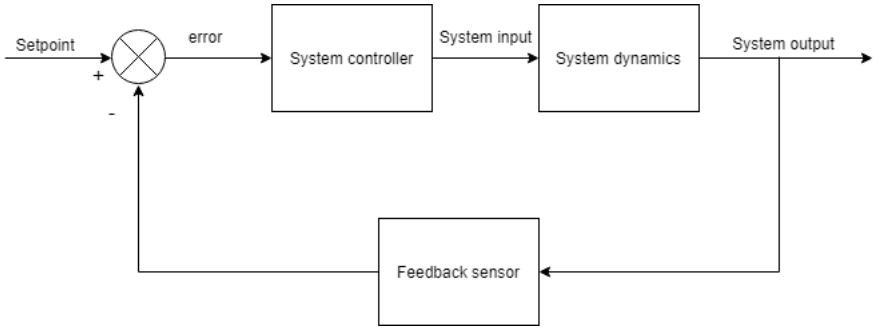


Figure 3.8: A block diagram of a simple control loop.

In this small section two approaches to this problem will be introduced; LQR and PID control. The intuition behind them will be explained as well as their mathematical formulation. This paper will however not focus on stabilizing proofs.

3.2.1 LQR

As explained above, the overall goal of a closed-loop control system is to make the process go towards the desired set point through feedback. This error is often used in something called a cost function, or loss function. As the name suggests, a cost function is how much it "costs" to be away from the set point. This could be a sum of deviation of different states like altitude and speed. This cost function needs to be minimized to achieve the control objective. The LQR algorithm reduces the amount of tuning work that needs to be done. An LQR is full state feedback controller that seeks to minimize the following cost function (30):

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{3.8}$$

For the system

$$\dot{x} = Ax + Bu \quad x \in \mathbb{R}^n,\ u \in \mathbb{R}^n,\ x_0\ given$$

This is the infinite horizon LQR. The solution that minimizes this cost function is the feedback control law: $u = -Kx$, where K is:

$$K = R^{-1} B^T P x$$

And to find P the algebraic Riccati equation needs to be solved:

$$0 = PA + A^T P - PBR^{-1} B^T P + Q$$

The Q and R matrices are the weight matrices. These are usually set to the identity matrix and after testing one fixes the parameters to meet the control objective. By looking at Q in

3.8 (30) it can be seen that it has something to do with the states. How much does each state determine how the controller should work, in other words, how much should the controller be punished for deviating from each state? Higher Q values mean the corresponding state matters more for the control objective. R in the equation is how much you want to penalize the control signal. High value for R means trying to stabilize the system with less weighted energy. This is called an expensive control strategy.

Integral action

State feedback controllers achieve desired steady response by tuning the parameters. But by using feedback one expects good performance, even with the presence of noise. This requires that the exact model is known, which can be demanding and is not wanted. By using the integral of the error as feedback one removes steady-state error. This is done by augmenting the description of the system with the state z:

$$\frac{d}{dt}\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ y - r \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ Cx - r \end{bmatrix}$$

The new state z is the integral of the difference between the output state and the set point. The control law with integral feedback is now:

$$u = -Kx - k_i z$$

3.2.2 PID control

There is a lot of literature on PID and it is probably the most known control strategy in modern times. PID uses three branches of correction of the error signal; proportional, integral, and derivative. In the proportional branch, the error signal is simply multiplied by a gain. In the integral branch, the error is first integrated and then multiplied by a gain. And lastly, in the derivative branch, the error is differentiated and then multiplied by a gain. The control law for a general PID is:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{3.9}$$

Where the K's are the gains for each respective branch. PI performance is good enough in most practical applications and the addition of the D term will increase the cost of the controller. This is why the derivative term is often dropped in the industry.

Chapter 4

Experimental Design

The coupled tank system is a typical system used in school to teach about flow, and mass balance. Quanser even has a development kit to do live experiments on (38). This is even supported by Matlab. This availability gives rise to a lot of literature and experiments. This system is used to answer the research questions posed in chapter 1.1.1. This chapter will describe the system in detail. The overall structure, the parameters, and why they were chosen, as well as the addition of complexity. The differential equations of the dynamics of the system are given; for the original system, the more advanced system, and the linearized system.

4.1 Base system

The system chosen for this thesis is a coupled tank system. The dynamics of the system are easy to understand and calculate and there is a lot of literature on the control and the dynamics of similar systems. A schematic of the system is given in figure 4.1. Both tanks have an outlet at the bottom. The first tank receives water from a pump, which can both pump water into the tank and suck water out of the first tank. The second tank receives water from the outlet of tank 1. At the bottom, there is a water reservoir where the water ends up in. The pump takes water from this reservoir and pumps it back to tank 1. The outflow of each tank depends on the height of each tank. The more water that is in tank 1, the more will flow out of tank 1.

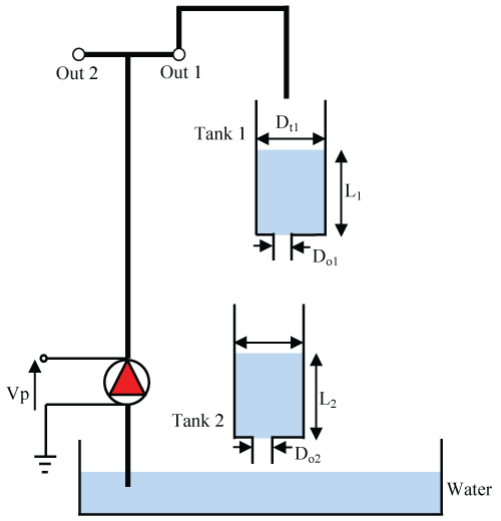


Figure 4.1: The coupled tank system used for the experiment (2).

This coupled tank system is the dynamical part of the system. There is another part of the total system. The goal of this thesis, as explained in 1.1.1 is to use RL to find an optimal switching strategy between multiple controllers. The starting system, or the base system, is the dynamical part together with this RL part. This is shown in figure 4.2. This system is easy to extend and make more complex by adding additional restrictions, adding another tank, extending to more than one set point, dynamical valves, having more than three controllers, etc. This is why it is presented as the base system. This is the simplest form of the system with the possibility to extend it further. The extending part of the system will be discussed later in the thesis.

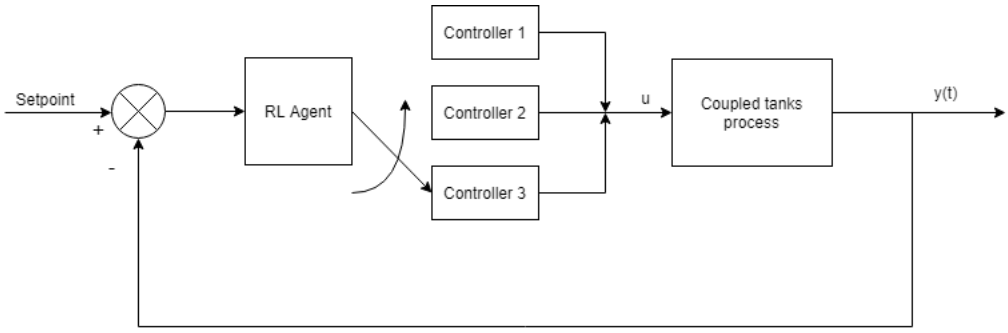


Figure 4.2: The base system.

The controllers chosen for the base system were max, min, and LQR. Max and min are as the name suggests, maximum, and minimum actuation voltage for the pump. The parameters and their values are listed below in table 6.1:

Table 4.1: Parameters of the coupled tank system

The parameters corresponds to the ones on figure 4.1.

Parameters	Value	Description
L1	0.5m	Height of tank1
L2	0.5m	Height of tank2
D_{t1}	0.1m	Diameter of tank1
D_{t2}	0.1m	Diameter of tank2
D_{o1}	0.015m	Diameter of outlet1
D_{o2}	0.016m	Diameter of outlet2
K_p	0.0002	Pump constant
MaxV	16V	Maximum voltage supplied to the pump
MinV	-10V	Maximum voltage supplied to the pump

The aim of the dynamics of the coupled system was that it should represent a real system, or close to a real system. How should the system act and what should the flow be in a logical sense. If maximum actuation is applied to the pump, the flow into the tank should be much bigger than the flow out at the bottom, even at maximum height. This was the motivation for trial and error testing to get reasonable values for the tank system. Additionally, there was an overflow condition added. Whenever the water level of each tank in the next time step is over 0.5m the new height is set to 0.5m. This way the water level in each tank will never exceed 0.5m.

4.1.1 System dynamics

As described earlier, the flow out of each tank depends on the water level. The way the flow rate is calculated is through the use of mass balance or mass flow. What is meant with that is: Change in mass $= Mass_{in} - Mass_{out}$ or that the change in total water in a tank is the difference between the water flow in and out. The volumetric change is given by:

$$\dot{V} = q_{in} - q_{out}$$

Where q is the flow rate. The goal is to keep the water level at a certain height. The change in height is:

$$\dot{h} = \frac{1}{A}(q_{in} - q_{out})$$

The flow out of each tank is:

$$q_{out} = c * \sqrt{2Gh}$$

Where c is the cross-sectional area of the valve where the water flows out. G is the gravitational constant and h is the height of that tank. The outflow of tank 1 is the inflow of tank 2. The inflow of tank 1 is what is pumped into the first tank from the pump. This pump is a constant multiplied with a voltage to generate a flow rate.

The differential equations for the coupled tank system is given by the equations below:

$$\begin{aligned} \dot{h}_1 &= \frac{1}{A_1}(Ku - C_1\sqrt{h_1}) \\ \dot{h}_2 &= \frac{1}{A_2}(C_1\sqrt{h_1} - C_2\sqrt{h_2}) \end{aligned} \tag{4.1}$$

Where C_1 and C_2 are constants given in the equations below. The small c's are the outlet cross sectional area.

$$\begin{aligned} C_1 &= c_1 * \sqrt{2G} \\ C_2 &= c_2 * \sqrt{2G} \end{aligned}$$

This system is a nonlinear system and it needs to be linearized to use an LQR. The linear system is given by:

$$\begin{aligned} \Delta\dot{h} &= \mathbf{A}\Delta h + \mathbf{B}\Delta u \\ \mathbf{A} &= \begin{bmatrix} -\frac{C_1}{2A_1\sqrt{h_1^*}} & 0 \\ \frac{C_1}{2A_2\sqrt{h_1^*}} & -\frac{C_2}{2A_2\sqrt{h_2^*}} \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} \frac{K}{A_1} \\ 0 \end{bmatrix} \end{aligned} \tag{4.2}$$

The desired set point is inserted for h_1^* and h_2^* to get the LQR to stabilize around the set point.

4.1.2 Additional complexity

This small section is here to present the extended differential equations and the parameters used for the additional valves. Two additional valves open only when the water level reaches a certain height. The parameters for this extended system are shown in table 4.2 below:

Table 4.2: Parameters of the coupled tank system with added emergency valves

The parameters corresponds to the ones on figure 4.1 but with two extra valves.

Parameters	Value	Description
L1	0.5m	Height of tank1
L2	0.5m	Height of tank2
D_{t1}	0.1m	Diameter of tank1
D_{t2}	0.1m	Diameter of tank2
D_{o1}	0.012m	Diameter of outlet1 in tank1
D_{o3}	0.008m	Diameter of outlet2 in tank1
D_{o2}	0.01m	Diameter of outlet1 in tank2
D_{o4}	0.006m	Diameter of outlet1 in tank2
K_p	0.0002	Pump constant
MaxV	16V	Maximum voltage supplied to the pump
MinV	-10V	Maximum voltage supplied to the pump

The dynamics of the system changes to:

$$\begin{aligned} \dot{h}_1 &= \frac{1}{A_1}(Ku - C_1\sqrt{h_1} - C_3\sqrt{h_1}) \\ \dot{h}_2 &= \frac{1}{A_2}(C_1\sqrt{h_1} + C_3\sqrt{h_1} - C_2\sqrt{h_2} - C4\sqrt{h_2}) \\ C_3\sqrt{h1} &= 0 \quad if \quad h1 < 0.375 \\ C_4\sqrt{h2} &= 0 \quad if \quad h2 < 0.375 \end{aligned} \tag{4.3}$$

Where C_3 and C_4 are calculated the same way as C_1 and C_2. This gives rise to 4 different linear systems each having their own LQR. These linear systems are numbered from 1-4:

- 1 - the original system
- 2 - where the extra valve in tank 1 is open
- 3 - where the extra valve in tank 2 is open
- 4 - where both extra valves are open

System 1: System 1 is the same as the basic system 4.2.

System 2:

$$\begin{aligned}\Delta\dot{h} &= \mathbf{A}\Delta h + \mathbf{B}\Delta u \\ \mathbf{A} &= \begin{bmatrix} -\frac{C_1+C_3}{2A_1\sqrt{h_1^*}} & 0 \\ \frac{C_1+C_3}{2A_2\sqrt{h_1^*}} & -\frac{C_2}{2A_2\sqrt{h_2^*}} \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} \frac{K}{A_1} \\ 0 \end{bmatrix}\end{aligned} \tag{4.4}$$

System3

$$\begin{aligned}\Delta\dot{h} &= \mathbf{A}\Delta h + \mathbf{B}\Delta u \\ \mathbf{A} &= \begin{bmatrix} -\frac{C_1}{2A_1\sqrt{h_1^*}} & 0 \\ \frac{C_1}{2A_2\sqrt{h_1^*}} & -\frac{C_2+C_4}{2A_2\sqrt{h_2^*}} \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} \frac{K}{A_1} \\ 0 \end{bmatrix}\end{aligned} \tag{4.5}$$

System 4:

$$\begin{aligned}\Delta\dot{h} &= \mathbf{A}\Delta h + \mathbf{B}\Delta u \\ \mathbf{A} &= \begin{bmatrix} -\frac{C_1+C_3}{2A_1\sqrt{h_1^*}} & 0 \\ \frac{C_1+C3}{2A_2\sqrt{h_1^*}} & -\frac{C_2+C_4}{2A_2\sqrt{h_2^*}} \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} \frac{K}{A_1} \\ 0 \end{bmatrix}\end{aligned} \tag{4.6}$$

Chapter 5

Methods

As discussed in chapter 3 a full RL system consists of more components. One needs an environment, which was discussed in the last chapter. As well as how this environment will evolve from timestep to timestep. The last thing needed is the RL itself. This chapter will describe the process of how the RL was designed to work with the system, how the state-action space was chosen, how the parameters were chosen, how the reward function was designed, and how the training + testing procedure was done. Then the values for the controllers are presented. Towards the end of the chapter, the additional complexity that was added to the system is shown. The last part of the chapter says what kind of libraries that were used in python for the design of the whole system.

The overview of the process is shown in figure 5.1. This is the general process for testing and experimenting with what works and does not work. Mostly the base system was used for this. If the results were not satisfactory something was changed. Most often the parameters were changed, but also big changes in the physical parameters of the tank system could be made as well.

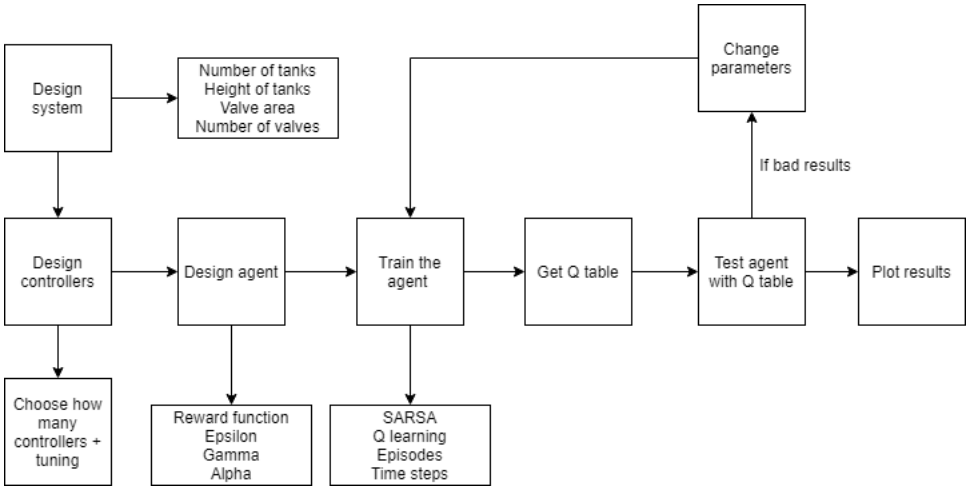


Figure 5.1: Overview of the process of designing and training.

5.1 RL agent

5.1.1 Agent design

The overall thought process, in the beginning, was to make a simple nonlinear system work with the RL agent, and then add complexity as it progressed. The dynamics of this system is simple and easy to make more complex. The real task is to design the RL agent. In chapter 3 the definition of an RL problem was defined. For this first base system, both Q learning and SARSA were chosen as the learning algorithms. To be able to use Q learning one needs to define the state space, action space, immediate reward, how each action is chosen, hyperparameters such as; ϵ, α, and γ and how the training/testing process was executed.

State-action space

Q learning is a tabular method, which means it uses a lookup table as a policy. This table could be viewed as a road map for the agent. The agent looks at its current situation, or this state, and tries to see what the best possible course of action is. This lookup table is updated by using the Bellman equation 3.1 and 3.6 in chapter 3. To define states in a system with continuous dynamics one needs to discretize the continuous states into buckets. What this means is that states could be defined as; far away, close, very close, and goal, as an example. The paper (21) used voltages to measure the height of the tanks and discretized each volt into 5 buckets. In this paper, however, the height of both tanks is directly used as states. To avoid training the agent longer than necessary, the state space was kept as small as possible while preserving good performance. The total number of buckets was experimented with, ranging from 3-160. All of these buckets represented 0.5m in each tank. This means that each state was ranging from 0.166m to 0.003125m. At the start, the

state space was chosen as 100, because 100 seemed like an easy number to start with. This, together with the 3 actions added up to a total of 30000 state-action space. According to the theory, each state-action pair should be visited infinite times before the policy converges to the optimal policy as stated in [(24), pages 113-142] and (25). Thus the state space was reduced to make the agent converge and to avoid unnecessary big state-action spaces. The chosen states for each tank ended up at 40 for a big portion of the base system, but as the project progressed the state space was reduced further to see how simple it could be done with good performance. This means that the size of the state-action space at the start was:

$$Total = 40 * 40 * 3 = 4800$$

And it could go as low as:

$$Total = 3 * 3 * 3 = 27$$

The discretization was mostly kept uniform in both tanks to keep in scalable. However other techniques were utilized. One of them was to have 3 areas in both tanks. Above setpoint, below set point, and an area around the set point. This area was set to 0.0125m. Figure 5.2 shows how this looks like.

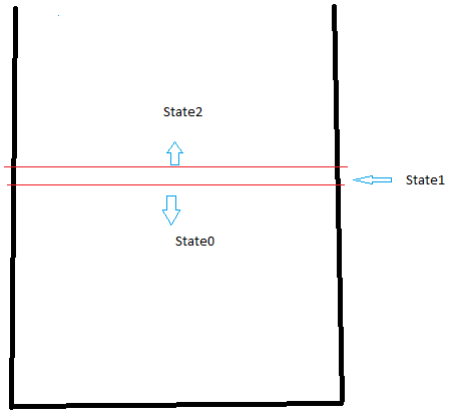


Figure 5.2: One of the discretization chosen for the base system.

This does not work when extending the system. The equilibrium point in tank 1 will move around and one can not use the same methodology. However, the tank that will be regulated around a setpoint will always have the same setpoint. Keeping this discretization in tank 2 and having a uniform one in tank 1 was done instead to keep it scalable. This was used as the go-to discretization and will be referred to as the 5-3 discretization for the remainder of this thesis.

Action selection

The way the actions were chosen was through a greedy strategy. It is a very common strategy in RL to decrease the epsilon from episode to episode. This will let us take advantage of both exploration and exploitation, and as stated in 3.1.2 it will help the agent converge to the optimal policy. The action is chosen based on how high the epsilon parameter is. If a random number between 0 and 1 is less than epsilon, a random action is taken. When a random action is not taken the action with the highest Q value in the state the agent currently resides is taken. This is simply done by looking at the current discrete state and what action gives the highest value:

$$action = np.argmax([qstate_1, qstate_2, :]) = np.argmax([0.2, 0.22, 0.21]) = 1$$

This action is passed to the dynamics of the system. Note that python is 0 indexed and index 1 is in position 2. The argmax function is from NumPy (39), a python library used for scientific computing, and returns the index with the highest value in an array. This is done for each step in an episode. The agent can switch between controllers freely at each timestep.

Reward

As discussed in chapter 3 the reward function, or simply the immediate reward, should not be a map for the agent on how you want it to reach the goal. Rather, it should be a way of communicating with the agent what the goal is. Keeping in mind what was written in chapter 3.1.2 there was still a lot of testing to get an appropriate reward function. The reward function does exactly what it is defined to do. Sometimes it finds a ”better” strategy, but all it has done is to find a loophole in the reward function. This is called reward hacking. Another problem that was discussed in 3.1.2 was the tuning of all the different terms in the reward function.

The thought process behind choosing a reward function for this system was that the agent should make the height of tank 2 be on set point, and get there as fast as possible. Trying many different rewards such as;

- Distance from the set point in the current step - this is simply set point minus the current height.
- Distance from the set point in the next step - the action taken in the current step affects the next step, this was more logical than taking the distance in current step.
- Distance for set point 2 steps ahead - it was observed that the second tank was not affected by the action taken in the previous step, a prediction was made to see how an action affected the state two steps ahead.
- An error gradient was formed based on the direction where the error moved. Moving away from set point was negative while moving towards the set point was treated better.

- Distance from the set point in the next step + the distance from the equilibrium point in tank 1. Given a set point in tank 2, there is an equilibrium point in tank1. This is used as a sort of set point in tank 1. The logic was that this is a coupled system, and thus one needs to take both tanks into account when making a reward. The outflow of tank1 can be seen as an input into tank 2.
- A combination of the above rewards. Especially the error gradient together with the distance from set point in tank 2.
- Using the set point in tank 2 as a goal state and reward when inside the goal or punish when outside the goal state.
- Adding the control input to the reward function and/or penalizing when the agent switches controller.

The above rewards were treated more as penalties. As in the examples given in chapter 3.1.2 being in an undesired state was penalized and being in a good state was less penalized to encourage the system to reach the goal faster. Moving away from the goal was penalized more than moving towards the goal. In general, everything was treated as negative rewards. Good actions and good states was less penalized, and whenever the agent did something bad it was punished by a lot.

The overall way of reasoning lead to the reward function:

$$R = -((error_{t+2})^2 + (error_{t+1})^2) \tag{5.1}$$

Here the error is referred to as the distance from the setpoint to the height of the water in tank 2. This reward function is one of the simplest ways to punish the system and is the one with the best performance. The whole point of making an RL agent is for it to understand what the goal is with only the relevant information. The error terms are squared such that the reward function is continuous.

Hyper parameters

Epsilon has already been discussed and why it has been set to that value. The discount factor γ is set to 0.9-0.99, this indicates that future rewards are valued more than the immediate rewards. This leaves the learning rate. As explained in section 3.1.2 in chapter 3 alpha needs to decrease. This is a simple system and absolute convergence is not that important in this thesis so a constant alpha works as well. But for the purpose of following state of the art methods the alpha was set to decrease, but not as fast as the literature suggests (25).

$$\alpha(n) = \frac{1}{1 + (0.005n)} \tag{5.2}$$

The table below shows what each parameter was set to:

Table 5.1: Hyper parameters for the RL agent

Parameters	Value	Description
α	See 5.2	Learning rate, or size step in learning
γ	0.9	Discount factor
ϵ	$1- > 0$	Exploration or exploitation
$Epsilon_{decay}$	0.995	Reduction in epsilon

Training and testing

In fig 3.3 the general procedure on how to train the agent is shown. First, the system that is going to be trained is chosen; base system, with added controllers or the emergency system. Then the algorithm is chosen; either SARSA or the regular Q learning algorithm. The training works by running episodes with a number of timesteps in each episode. At the start of the training, the Q table is initialized to 0 and the agent will fully explore in the first episodes since it currently knows nothing about the environment. The initial height of the tank is randomly initialized between 0 - 0.5m in each episode such that the agent will learn general scenarios. The epsilon parameter decreases each episode with the $Epsilon_{decay}$ parameter until it is 0.1 where it stays constant until all the episodes are done. With an epsilon of 0.1 means that the agent selects a random action 10% of the time, this is to prevent the agent from converging to a nonoptimal policy. In each time step, an action is selected by the agent, and this action is given to the dynamics of the system. Based on what action was given to the system, the state changes, and a reward is given. The optimal policy will be when the total accumulated reward is maximized. The final policy used to test the 40 state system was trained for 20000 episodes with 5000 timesteps each.

The Q table is saved and analyzed to see if the agent has converged if not, more episodes are run. The number of episodes needed varied, and could be as much as over 20000 episodes if the state space was big, or could be as little as 1000 when the state space was smaller. To test how good the policy is the epsilon parameter is set to 0 such that the agent only chooses the action with the highest q value, and in testing the Q table is not updated. This means that the agent uses the Q table as a map and will only do what the map tells it to do. The dynamics and the control actions chosen are plotted to study how good the agent did.

Disturbance

A simulated system without disturbances is unrealistic and too perfect. Thus a disturbance was added to simulate a more realistic system. The disturbance was a tiny leakage from both the valves. This leakage was simulated by using a sinusoidal that depended on the height of each tank. The higher the water level in each tank, the greater the pressure on the valves, and thus more water will leak.

5.1.2 Controller design

As discussed in the previous chapter, there were only 3 controllers to begin with; max, min, and LQR. The LQR controller was designed according to the theory in chapter 3 and the linear system 4.2. The Q and R were chosen based on how much each state was valued. They ended up at:

$$Q = \begin{bmatrix} 6000000 & 0 \\ 0 & 24000000 \end{bmatrix} \qquad R = 1$$

To remove the steady-state error an integrator was added to the LQR controller as well. The gain was chosen to be:

$$Klqr_i = \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

Additional controllers

After getting the base system to work, some additional controllers were added; PI and PD. This was done to see how the agent operated when the action space increased. The gains for the PI was:

$$Kp_{PI} = \begin{bmatrix} 500 \\ 500 \end{bmatrix} \qquad Ki_{PI} = \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

And for the PD:

$$Kp_{PD} = \begin{bmatrix} 500 \\ 500 \end{bmatrix} \qquad Kd_{PD} = \begin{bmatrix} 0.05 \\ 0.05 \end{bmatrix}$$

These controllers were added together with the 3 controllers mentioned in 4. This was done to study if the best characteristic of each controller would be used, as they investigated in (21). And lastly, only the PI, PD, and LQR controller were used together to see what the RL thought was the best switching strategy.

5.2 Additional complexity

There were many ideas about how to make the system more complex:

- Adding a third tank
- Introducing more valves in each tank and let some valves flow to a tank below or the reservoir
- Valves placed at the top of the tank to let water through to prevent overflow
- Valves that varied with the height
- A sudden drop in water level that simulated a crack in the tank

- Valves that varied with time.

All of these made the system more complex, but it didn't help to answer the questions in chapter 1.1.1. The last question demand that the system can not easily be controlled by one controller. Three tanks make the system more complex, but one controller is enough for set-point regulation. The dynamics need to change in such a way that more than one controller is needed to keep the desired tank at the set point. Making more water flow out of both tanks at some point will split the dynamics into more parts. That is, when there is no extra flow, there is extra flow in one of the tanks, there is extra flow in the other, and in both at the same time. To do that 2 extra valves that functioned as emergency valves were added. These are state-dependent and open once, and never close once they are open. When the water level in a tank reaches a certain height, the emergency valve opens to let more water through to prevent overflow. The threshold height for this was set to 0.375m. This could be tied to if the fluid is dangerous, and leakage can hurt a part of a bigger system. Having these 2 extra valves introduce a change in the dynamics and makes the system hard to control accurately with only one controller.

5.2.1 Design of emergency valve system

Adding to the RL agent

The agent in Q learning has states that describe where it is in the environment and actions that describe what the best course of action is in this state. The agent needs to know about the valves as well. This is done by adding 2 new states that can take the values 0 and 1; 1 for when the valve in either tank is open, and 0 for when they are closed. This means that the agent splits the overall system into 4 subsystems. One for when the valve in tank 1 is open, one for when it is open in tank 2, one when both are open and one when both are closed. The agent learns how to switch between the controllers in each subsystem and what is the optimal control switching in each subsystem

Adding to the controllers

As discussed in chapter 4 and above, the system is now 4 subsystems that together make the full system. For each of these subsystems, an LQR is made. Each of these has its linearization shown in chapter 4.1.2 and with Q and R as:

$$Q = \begin{bmatrix} 1000000 & 0 \\ 0 & 2000000 \end{bmatrix} \qquad R = 1$$

And the gain for the integrator is:

$$Klqr_i = \begin{bmatrix} 15 \\ 15 \end{bmatrix}$$

The system as a whole

Each of these LQRs had the same Q, R, and gain, but their dynamics are different and their setpoint as well. With these extra controllers and the extra valve states the total state-valve-action space would be very large and training would be slow. But as discussed above, the state space was reduced to be both scalable and still have good performance. The 5-3 discretization was used, which means the full state-valve-action space was:

$$Total = 5 * 3 * 2 * 2 * 6 = 360$$

The whole procedure with training and testing, parameters, and the reward was the same as it was in the base system. However, the training for the emergency system is a little different. Since the whole system has the valve as states each of these subsystems was trained by itself for 1000 episodes. And since this is a whole system, the full system was trained for just as many episodes as all the subsystems combined (4000).

To test the system some variations of the dynamics were used. This emergency system with the extra valves could have different dynamics. First, the system was run in the normal way where each valve would open if the water level would reach 0.375m. Several simulations were run to see if the agent chose the correct LQR. Another way of testing was to allow the valves to close after they opened to see if the agent would select the correct controller. And lastly, each valve was allowed to switch on and off randomly each 2500 timestep.

5.2.2 General discussion

The goal of designing the RL system talked about in this section was to make it scalable to bigger/more complex systems. There was a lot of testing with the number of states used, the number of controllers used, and nonlinearities that could be dependent on time. Most of the testing was done on the base system. Once a general approach was found for the base system, the only thing to do then was to change the dynamics, add controllers or add a tank, without changing the structure of the system. That means the number of states, the parameters used for training, or the set point for the desired tank to control. Making the system scalable was a high priority while designing the system.

It is worth to mention that 2 other methods were also used to train the agent. Both the double Q learning and deep Q learning were used as well. But it was noted that both of these methods were not suited for this problem. Double Q learning is suited for stochastic systems. This system has no randomness in it, thus the performance was not good. Deep Q learning is used when the state-action space is enormous and it is not feasible to train the agent with a tabular method. In this system the state-action space is not big enough for the benefits of the deep Q learning method to show.

5.3 Software and Hardware

This work did not use any hardware outside of a computer. The dynamics and RL interaction with the dynamics are simulated using python. The libraries used is NumPy (39) for matrix calculation and in general for the computing part. Pickle is used to save the Q table in a pickle file (40) so that it could be read and used by the agent. The standard matplotlib library from python is used to plot the results (41). And finally to solve the Riccati equation for the LQR the linear algebra library from SciPy was used (42). The system dynamics were implemented using the NumPy library, and the RL agent was implemented with the help of (43), (44), and (35).

Chapter 6

Results and discussion

This chapter will present all the results from the simulation made in python with the RL agent and the two tank system dynamics. The result from each sub step will be presented and then discussed. There will be plots from how the dynamics evolve when the agent is in charge of the controller scheduling, the switching of each controller and general plots that show performance and to illustrate certain points, if being discussed. First the base system will be presented. Then the additional controllers will be shown. At last the emergency system with different dynamics will be presented followed by a discussion to sum everything up.

6.1 Base system

The first part of this chapter will present the results from the base system. How the results evolved and what changes where made to get to the final results. The control actions are;

- 0 - Minimum
- 1 - Maximum
- 2 - LQR

6.1.1 Results

These first figures 6.1, 6.2, 6.3 and 6.4, show the performance with 40 states in both tanks and 3 actions; min, max and LQR. This adds up to 4800 total states that the agent needs to learn. The first figure shows the simulation of the dynamics for 5000 timesteps (50 seconds), and one can see that it reaches the setpoint of 0.25 m fairly quickly. The second figure shows the switching of the controllers done for each timestep. In general, the most optimal thing to do is to keep the water level in tank 1 high when tank 2 is below set point, and when tank 2 is over setpoint it is best to keep the water level in tank 1 low. In this

way, the water level in tank 2 would reach set point in the shortest amount of time. Once tank 2 is close to set point, tank 1 needs to get to its equilibrium point and then switch over to LQR. This is how one intuitively dwells on what is optimal or not. But this might not be what the agent thinks is optimal. Depending on what parameters are used, especially gamma, it is hard to say for sure that the agent's behavior is optimal or not.

In this example, the water in tank 2 is under set point, and the water level in tank 1 is low. The most optimal action would be 1 to get the water level in tank 2 up to the set point as fast as possible, and in figure 6.4 there are some oscillations at the beginning between all the controllers. At the very start it chooses 1 for some time, but mainly it switches between all the controllers. Once the water level in tank 2 is getting close to the set point the most optimal thing would be to pump water out of tank 1 until it is around the equilibrium point. That means the optimal action would be 0, but the figure shows that the agent is switching between 0, 1, and 2.

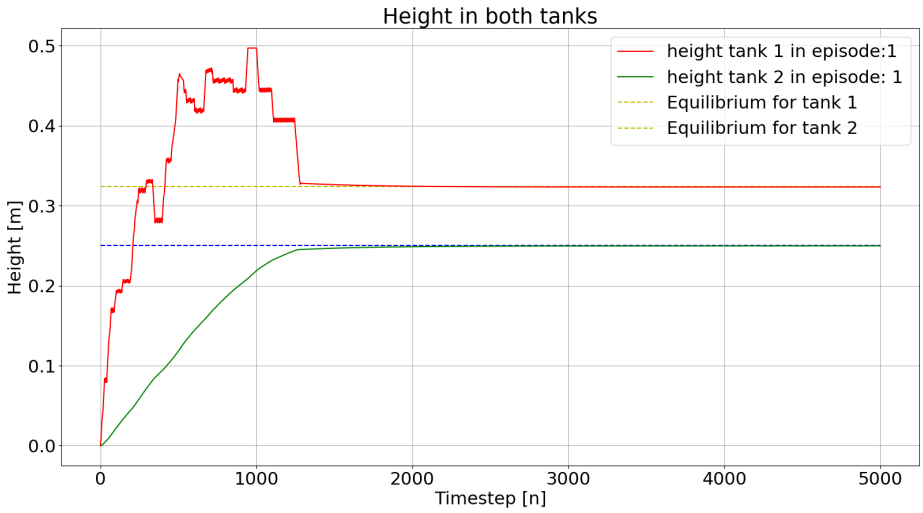


Figure 6.1: Simulation of the dynamics when the agent chooses what control action is used for the 40 state system.

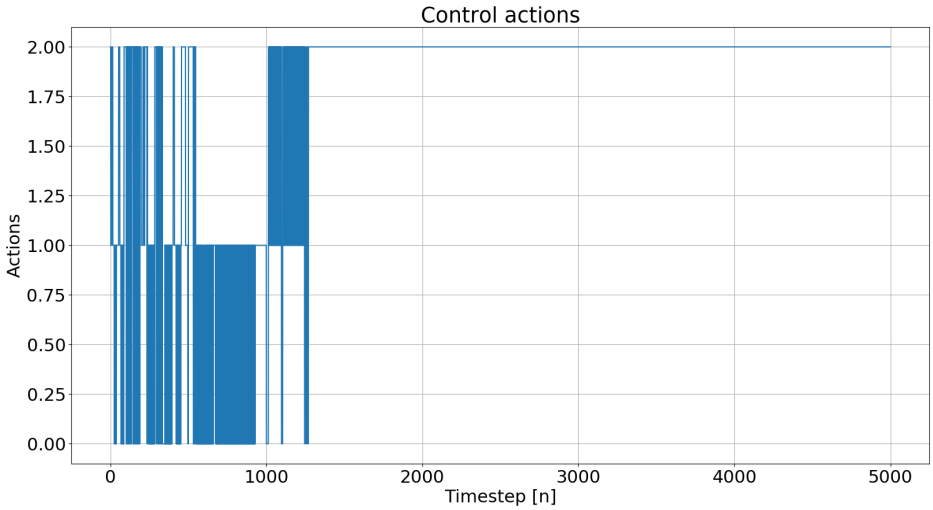


Figure 6.2: The control actions taken in the simulation. 0 is minimum, 1 is maximum and 2 is LQR.

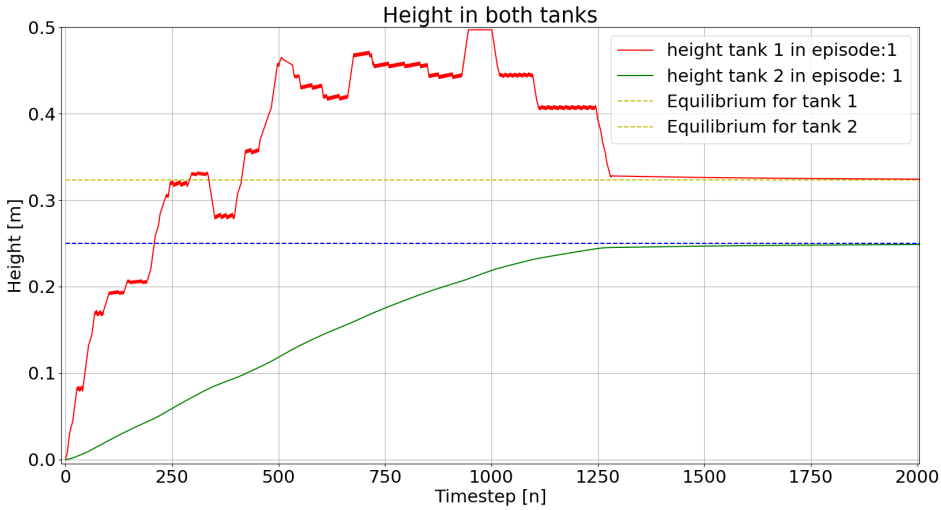


Figure 6.3: Zoomed in on fig 6.1.

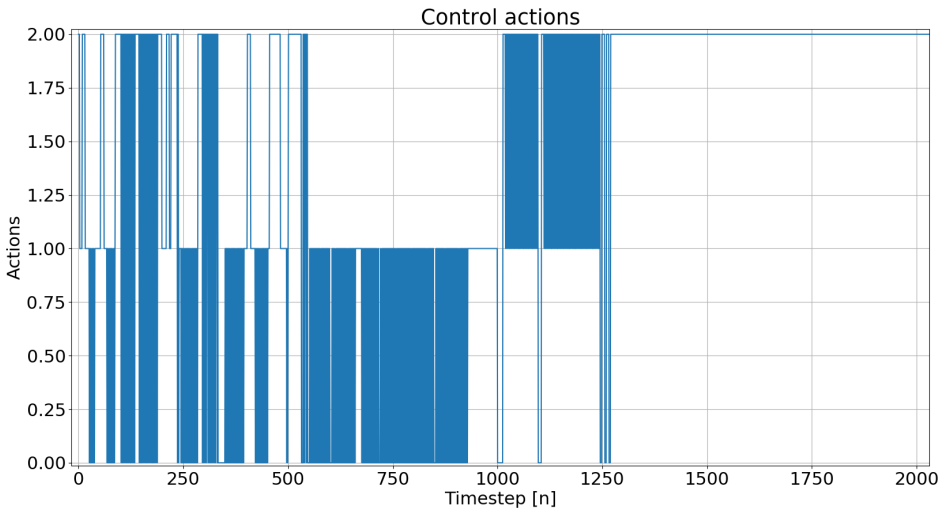


Figure 6.4: Zoomed in on fig 6.2.

Taking a closer look at the zoomed-in versions 6.3 and 6.4 one can see that the water level in tank 1 is kept high until tank 2 closes in on 0.25 m. It should keep a high level until tank 2 is close to set point, but it switches between 0 and 1, 1 and 2, and 0 and 2. This might be due to a local minimum or that the agent looks far into the future (big gamma) and wants to ”prepare” to get to equilibrium as fast as possible to avoid overshooting. On its way down it switches between 0, 2 and sometimes 1. This is not optimal, but both the LQR and the minimum controller have negative values, meaning both will contribute to pump water out of the tank. Again, this switching is most likely due to a big gamma. In these figures, a very big gamma was used (0.999) which means that future rewards matter more than immediate rewards. This switching between 0 and 1 and 1 and 2 might be because the agent tries to maximize the total accumulated reward and looks far ahead into the future. In general though, the agent seems to understand that the water level in tank 1 should be kept high when the water level in tank 2 is low, but it does not seem to behave optimally.

Reduction in states

The state-space was reduced to see if it was possible to further improve the performance of the system and to see how simple the discretization could be done while still maintaining good performance. The first discretization is the one discussed at the end of chapter 5.1.1 with 3 states, where there is a small bucket around setpoint in each tank and a state over and under that area.

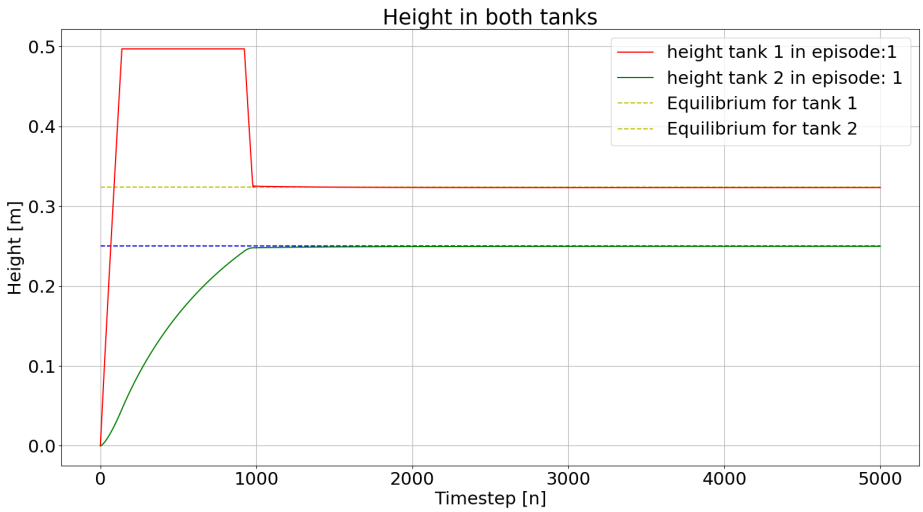


Figure 6.5: Simulation of the dynamics when the agent chooses what control action that is used with only 3 states for both tanks.

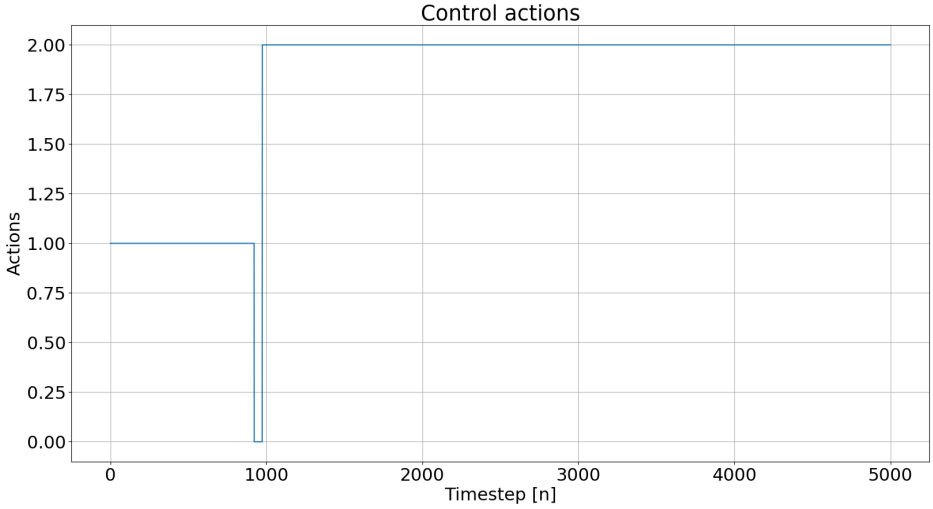


Figure 6.6: The control actions taken in the simulation. 0 is minimum, 1 is maximum and 2 is LQR.

As seen in figure 6.5 the water level in tank 1 is rising to the maximum and staying there, even though tank 2 seems to be very close to set point. This behavior is very different

from the one with 40 states in both tanks. Taking a look at the control actions in figure 6.6 one can see what was expected to be optimal. The water level in tank 1 is kept as high as possible for as long as possible such that the water level in tank 2 reaches its set point in the shortest amount of time. Once it closes into that point, it switches and pumps water out of tank 1 until tank 1 is close to equilibrium, where it switches to LQR.

Comparing figure 6.3 and 6.5 it is easy to see that the switching is gone and with these 3 states the agent was able to get the water level to set point within 1000 timesteps, but the one with 40 states did it in almost over 2000 timesteps. With only 3 states fixed the way they are, there are a limited amount of actions that can be done in each state. The separation between a good action and a bad action in each of these fixed states is more apparent for the agent than with a finer discretization. Even taking future rewards into account it would not make much of a difference with this setup. Future events will most likely be the same as recent ones, and 10-20 timesteps ahead the agent will most likely be in the same state as it was in at that point; either above or below the equilibrium. If the level in tank 2 is below set point, the only thing to do then is to pump max water into tank 1 until tank 2 is inside the bucket that is around the set point. Below, the Q table for this state-action space is shown.

Table 6.1: Q table

This is the q table for the simulation above.

States	Action 0	Action 1	Action 2
0 0	-9.4	-8.58	-9.35
0 1	-4.74	-5.03	-4.60
0 2	-9.53	-9.32	-9.51
1 0	-7.07	-6.49	-6.52
1 1	-4.58	-4.57	-4.56
1 2	-8.42	-8.16	-8.47
2 0	-6.51	-6.28	-6.52
2 1	-4.58	-4.67	-4.62
2 2	-8.13	-8.14	-8.14

This Q table shows what the agent thinks is the optimal action in each state. The value with the biggest value (smallest negative value) is the best action in that state. The states are as in chapter 5.1.1, 0 is under, 1 is inside the bucket, and 2 above. Having 40 states as figure 6.1 the agent can look at how an action in this state affects the states and rewards in the future. An action in one timestep will directly affect how the next state will be, but in this fixed 3 state discrete system, the next state following an action will most likely be the same state. This works well for a simple system with fixed non-changing equilibrium points such as this one, but not for a general system as the one later in this chapter.

Making it general

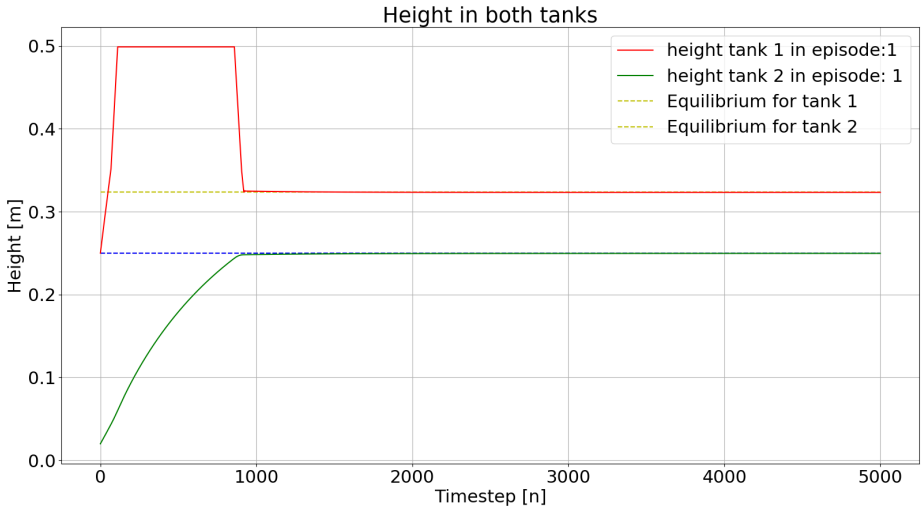


Figure 6.7: The dynamics of the 5-3 discretized system.

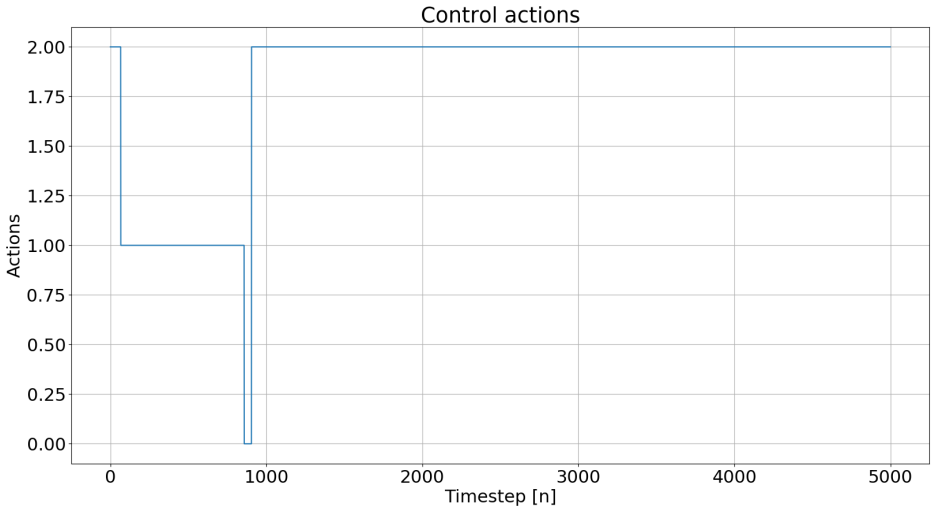


Figure 6.8: Actions taken for the 5-3 discretized system.

The above two figures show the same dynamics and controllers. This time tank 1 is discretized with 5 uniform states and tank 2 has the same discretization. As seen in figure 6.7 and 6.8 the agent seem to be utilizing the correct controller to achieve best performance. The water is pumped into tank 1 and is kept high until tank 2 is close to setpoint, then it pumps water out of tank 1 until tank 1 is close to equilibrium and then switches to the LQR.

6.1.2 Additional discussion

It is seen here that with fewer states the agent behaves more optimally. Having more states in the tanks leads to more training, and in this case the performance is better than with 40 states in both tanks. The plots from the 40 states were trained for 20 thousand episodes with 5000 timesteps, and the one with 5 uniform states in tank 1 and fixed states in tank 2 was only trained for 1000 episodes with the same amount of timesteps. While the latter only has 45 values to learn, the first has 4800 values to learn from, and according to the literature, each state-action pair needs to be visited an infinite amount of times before the learning converges [(24), pages 113-142] and (25). Even though the agent in the 40 states scenario trained more, the relative ratio between states and training was less than the 5-3 discretization.

The main focus is not to make the learning converge to the optimal, but rather study how RL could be used in real life to learn a switching strategy. Showing this is the most important, and since the performance was much better in the smaller state-space case, the state-space was kept small for the rest of the experiments.

The main algorithm used for learning was SARSA. Q learning was also used but the results gave no new insights or understanding and the results from those simulations have been omitted from this paper.

6.2 Base system + additional controllers

Here, the results with additional controllers are presented. First with all 5 controllers; max, min, LQR, PI, and PD. Then only the LQR, PI, and PD will be shown. The control actions are the same as in the previous section where 0 is min, 1 is max and 2 is LQR. Actions 3 and 4 are PI and PD respectively. But when only LQR, PI, and PD are used, 0 is LQR, 1 PI, and 2 PD. Only plots from the 5-3 discretization are shown.

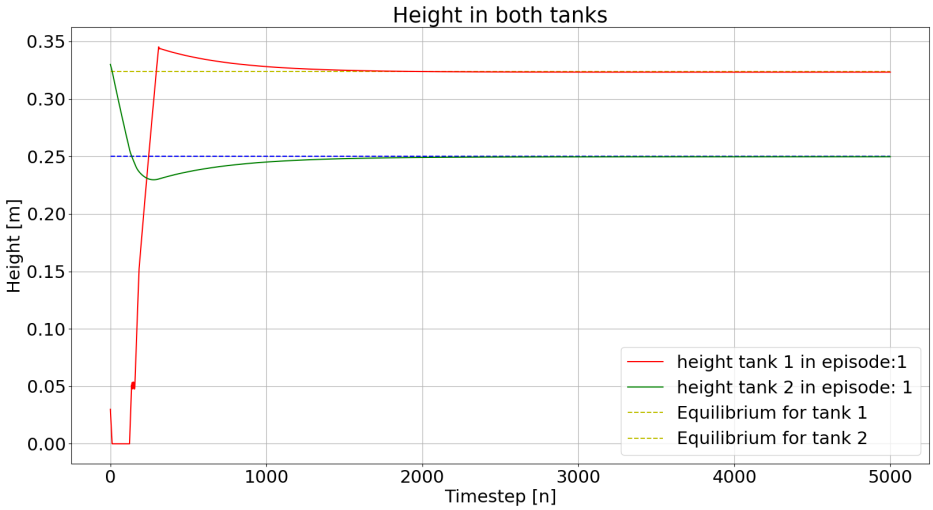


Figure 6.9: Simulation with the two extra controllers; PD and PI.

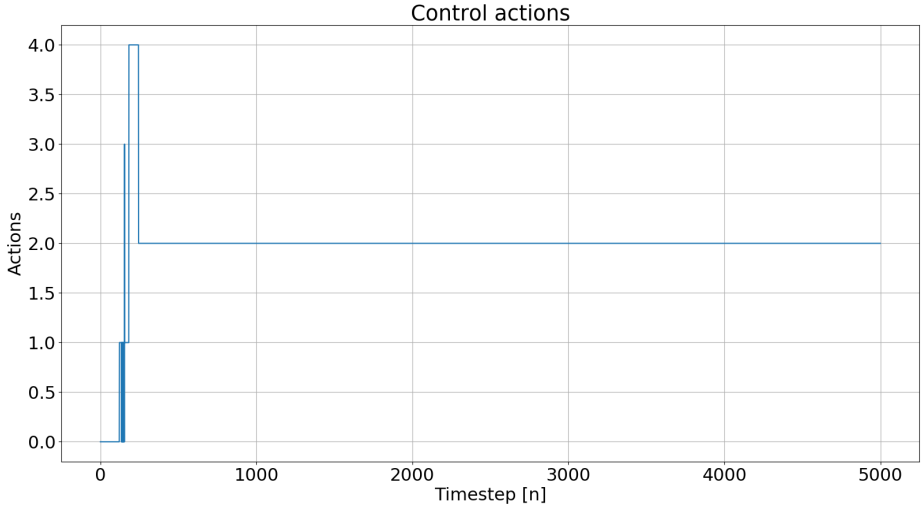


Figure 6.10: Actions taken for the dynamics in figure 6.9.

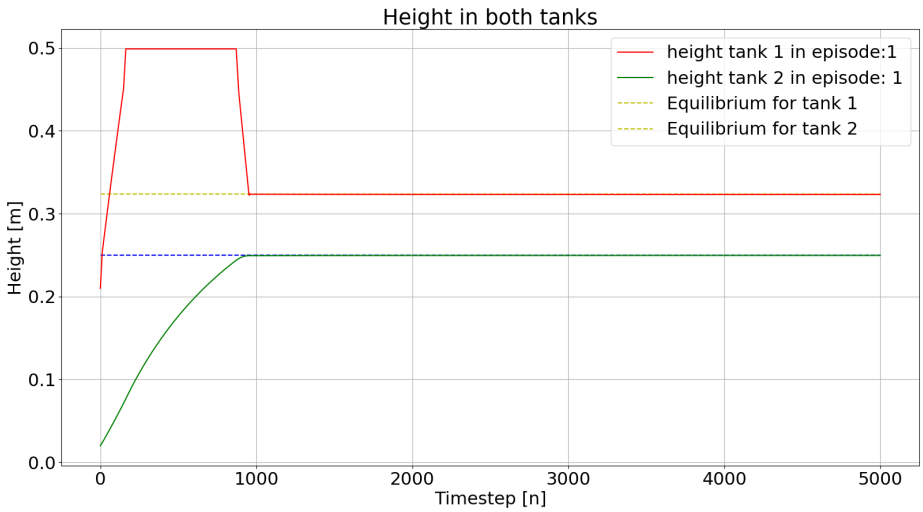


Figure 6.11: Simulation of the dynamics with only the LQR, PI and PD controllers

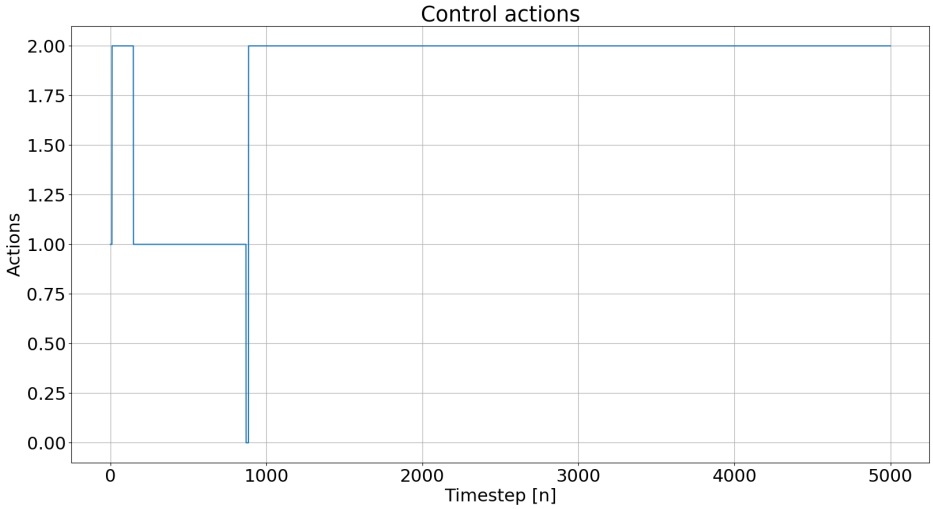


Figure 6.12: Actions taken for the dynamics in figure 6.11.

These figures don't add much to the understanding behind this thesis, neither do they help to answer the research questions. They are here, in addition, to show how the process was from the base system to the final system. As stated in chapter 5 the base system was

there to test and see if it would work with RL. These simulations helped to see if the agent could handle more control actions. The plots clearly show that this is no problem. It is interesting however that the agent chooses the LQR in figure 6.10 to regulate around set point, but in figure 6.12 the agent chooses the PD controller. This could be explained by the integrator and the derivative term in the controllers. Depending on where the water level starts, these values will be different. And by using a min-max controller that pumps in more/pumps out more water than each of these controllers, these terms will also vary from each respective scenario. With the availability with the min-max controllers, the 5 control system will choose the LQR and not the PD as in the 3 control case.

6.3 Emergency valve system

Below, the results for the emergency valve system are presented. The first 8 figures are plots of each subsystem and the actions taken. The chosen names for each of these systems are:

- Subsystem 1 - when no valves are open.
- Subsystem 2 - when the valve in tank 1 is open and the valve in tank 2 closed.
- Subsystem 3 - when the valve in tank 2 is open and the valve in tank 1 closed.
- Subsystem 4 - when both the valves are open.

The controllers for this system are:

- 0 - min controller
- 1 - max controller
- 2 - LQR for subsystem 1
- 3 - LQR for subsystem 2
- 4 - LQR for subsystem 3
- 5 - LQR for subsystem 4

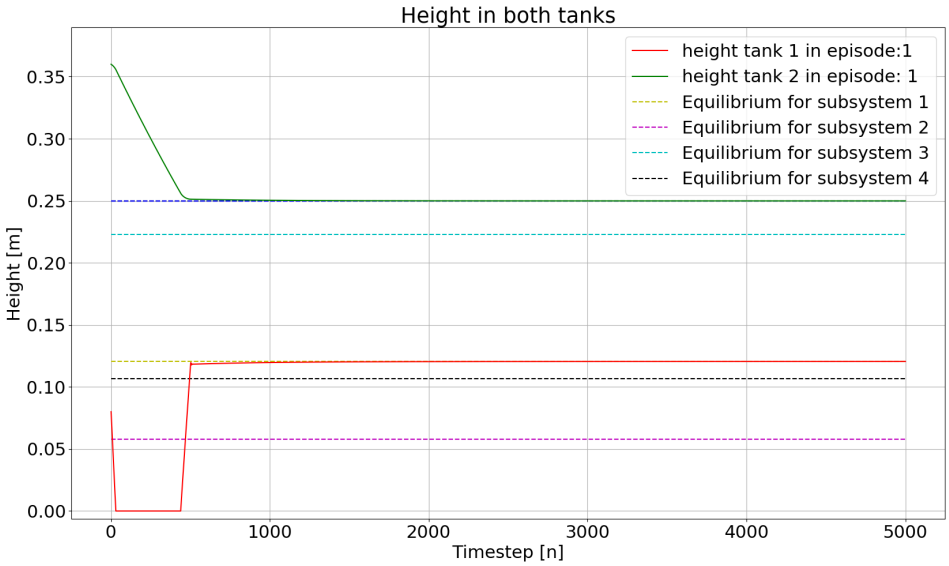


Figure 6.13: The dynamics of subsystem 1.

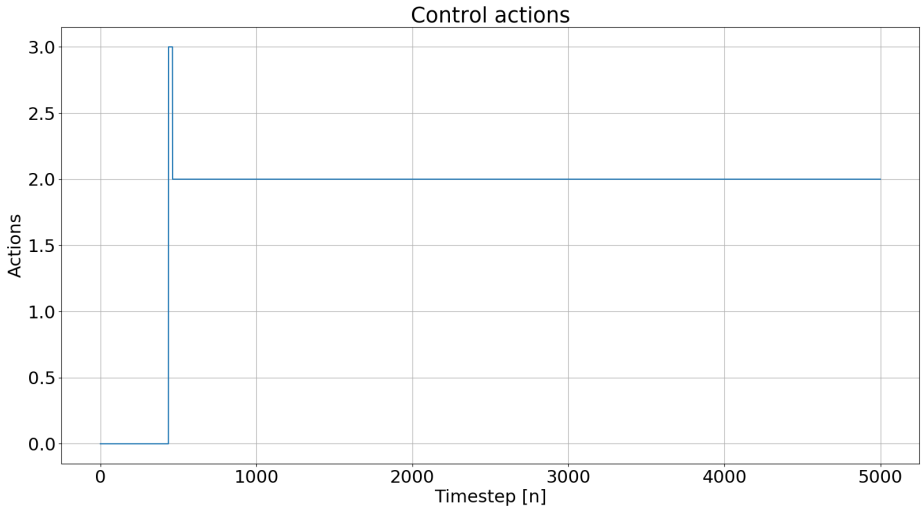


Figure 6.14: The actions taken for subsystem 1.

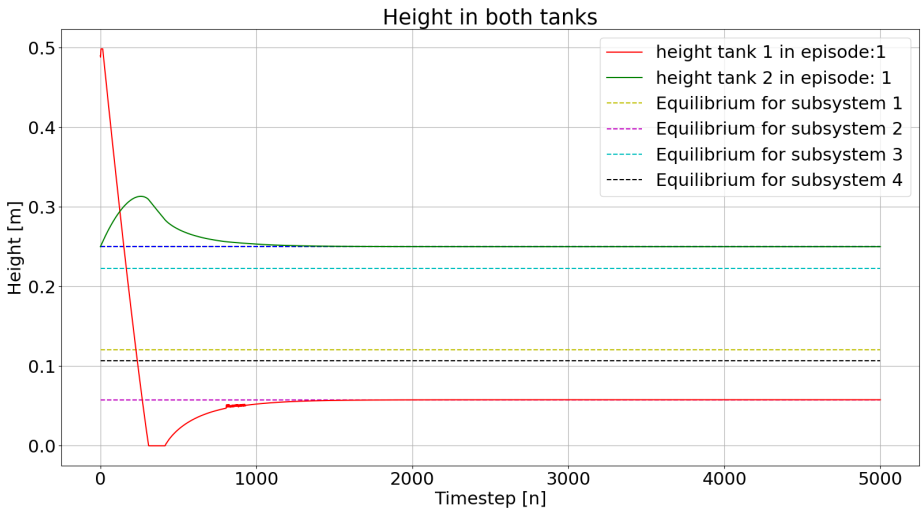


Figure 6.15: The dynamics of subsystem 2.

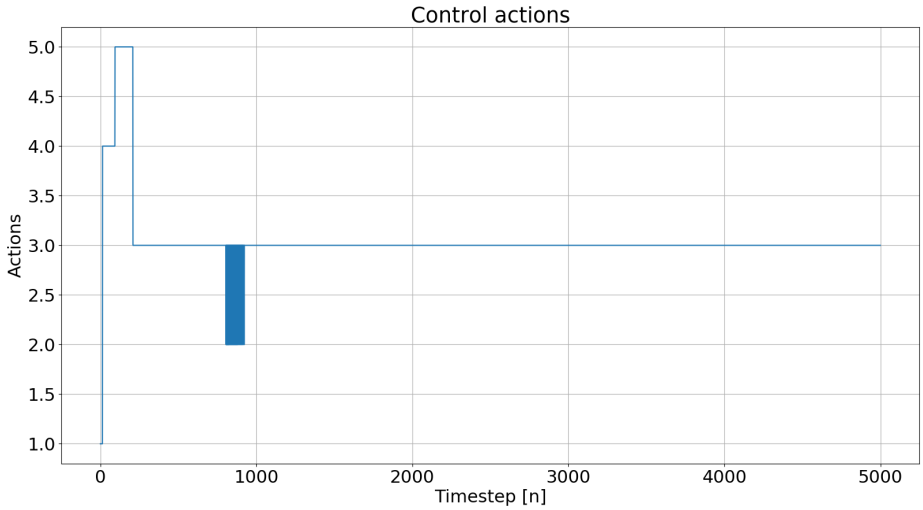


Figure 6.16: The actions taken for subsystem 2.

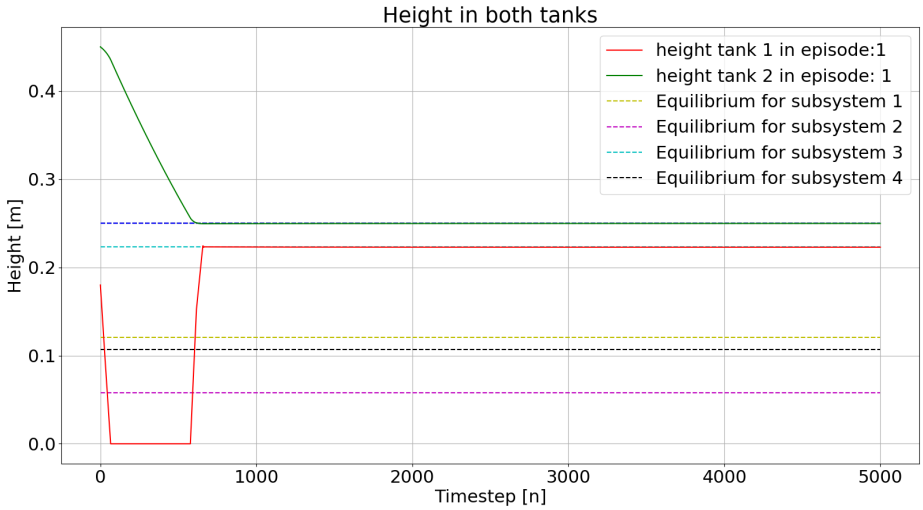


Figure 6.17: The dynamics of subsystem 3.

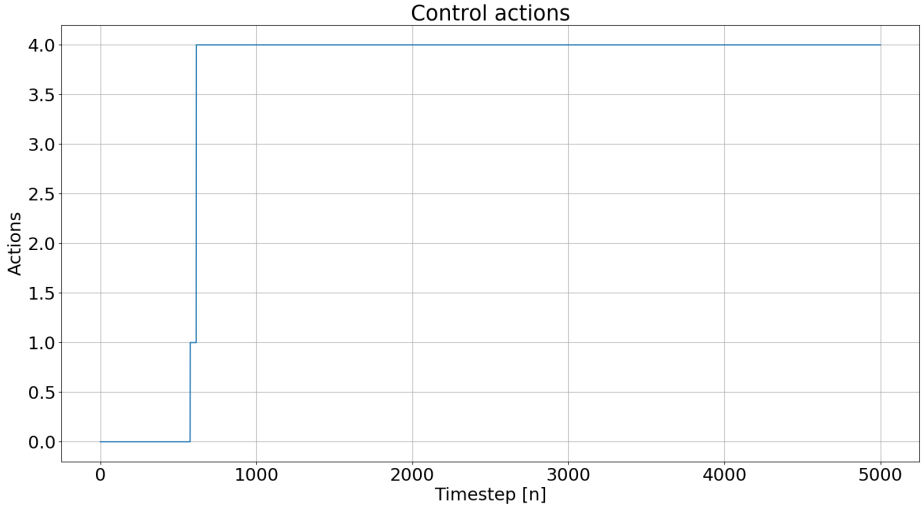


Figure 6.18: The actions taken for subsystem 3.

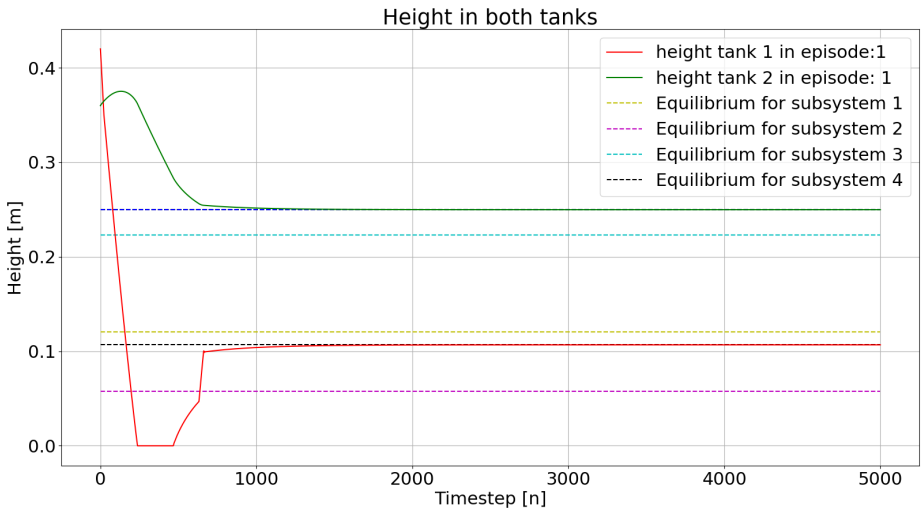


Figure 6.19: The dynamics of subsystem 4.

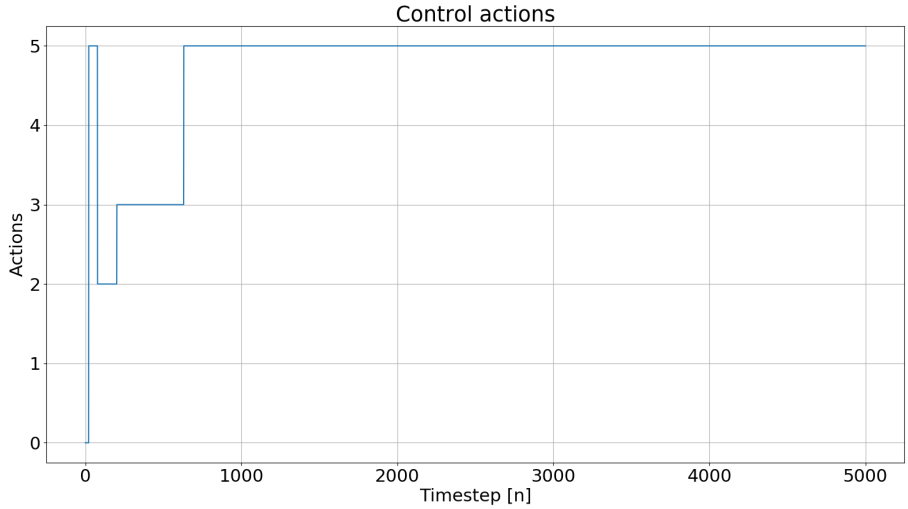


Figure 6.20: The actions taken for subsystem 4.

All of the plots above show how the agent chooses the actions based on the water level in each tank. In figure 6.13 and 6.14 the water level is never above 0.375m, and thus none of the valves are open. Figure 6.15 and 6.16 show when tank 1 starts above 0.375m and

the valve in that tank is open. Continuing, the same applies to figure 6.17, 6.18, 6.19 and 6.20. In each of these scenarios the agent chooses the correct controller to keep tank 2 at set point.

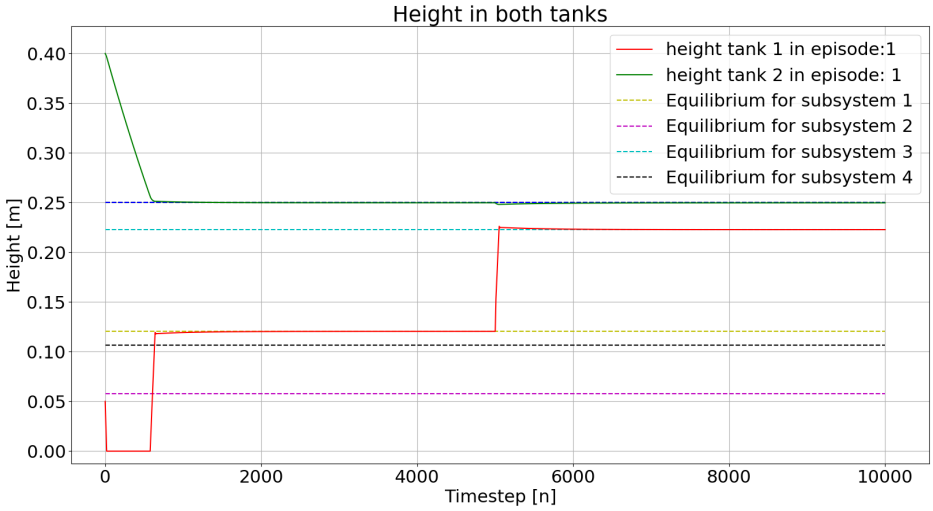


Figure 6.21: The dynamics with forced valve opening. The valve in tank 2 opens after 5000 timesteps.

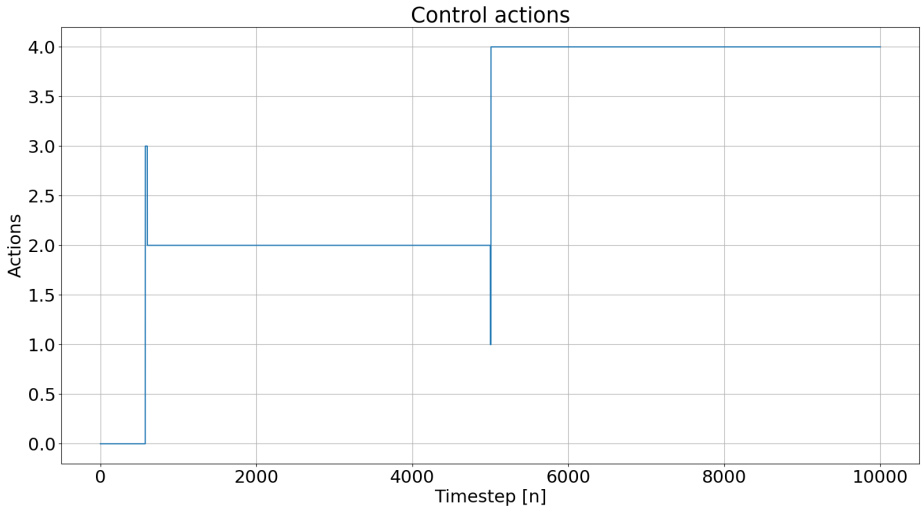


Figure 6.22: The actions the agent takes from figure 6.21.

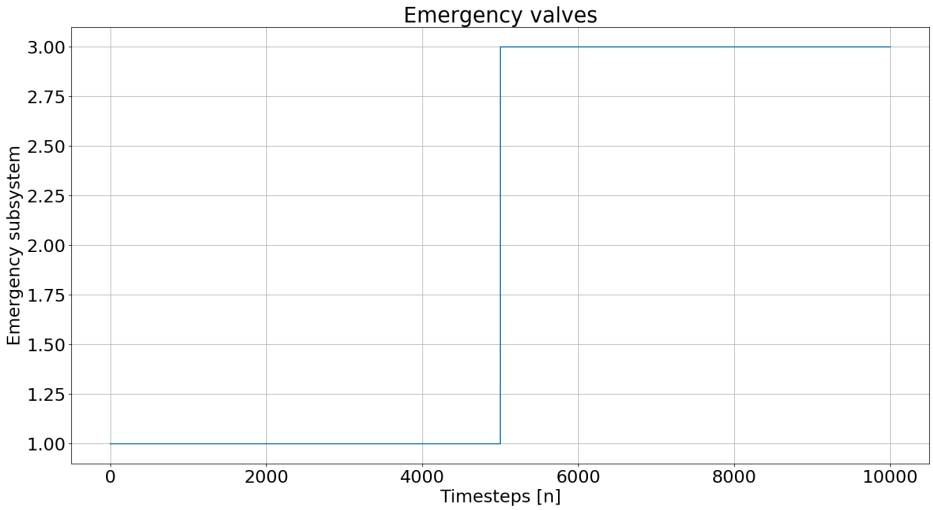


Figure 6.23: Which of the subsystem that the system is in.

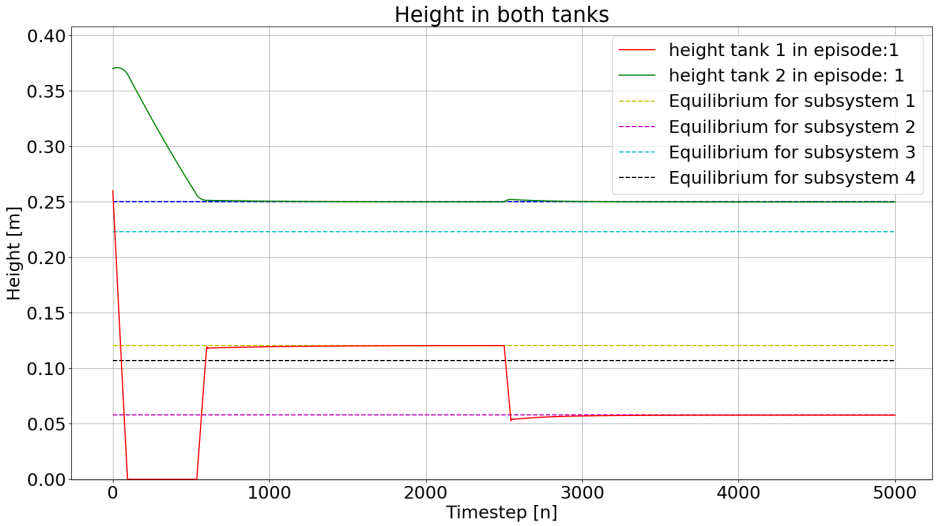


Figure 6.24: The dynamics with forced valve opening. The valve in tank 1 opens after 2500 timesteps.

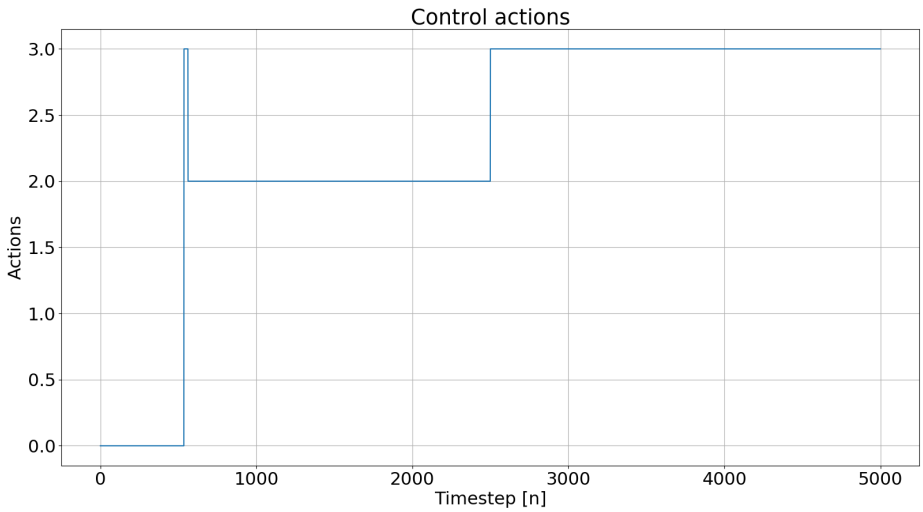


Figure 6.25: The actions the agent takes from figure 6.24.

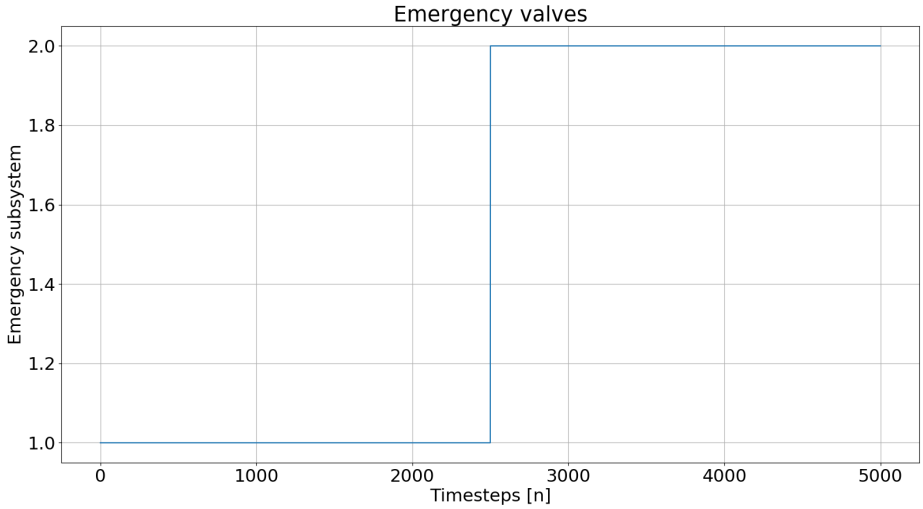


Figure 6.26: Which of the subsystem that the system is in.

Above in figure 6.21 and 6.22 a simulation where the valves are forcefully opened is shown. First, no valves are open, but from 5000 timesteps and beyond the valve in tank 2 opens. The agent chooses the correct controller and when the switching happens the agent

chooses controller 1, which is the maximum controller, to get the level in tank 1 to the new equilibrium point. One thing to note is that the agent chooses controller 3 at a small interval about 700 timesteps into the episode. This might be the agent that tries to avoid overshooting by looking into the future.

In figure 6.24 and 6.25 the valve in tank 1 opens from 2500 timesteps and beyond. Here the same thing is seen, the agent switches to the min controller immediately, but switches to the third controller before it settles for the LQR for subsystem 1. The agent chooses the correct controller to get to the first set point. When the valve opens the agent jumps directly to controller 3 which is the LQR controller for subsystem 2. One would think that the optimal thing to do is to pump water out as fast as possible until it is at the new equilibrium and then switch over to 3. But as stated earlier, when there is a lot of weight on future rewards it is hard to tell by the eye what is the most optimal thing to do in that exact instance. This behavior could be explained by what was said above, the agent tries to avoid overshooting.

Below in figure 6.27 and 6.28 the system starts in subsystem 4 and switches to subsystem 3, 1 and then 2 again. The agent seems to be handling this kind of switching easily. At the start, it chooses the LQR controller for subsystem 2 but quickly switches to the max controller to get the water level in tank 2 to 0.25 m as fast as possible. When it is close to the set point it overshoots a little and then pumps water out to get the level down to set point. Then it switches to LQR 4 before it settles for the correct LQR. This unusual switching can as stated earlier be due to high weights on future rewards. Another thing could be that the agent needs more training to converge. Figure 6.23, 6.26 and 6.29 show which subsystem the agent is in.

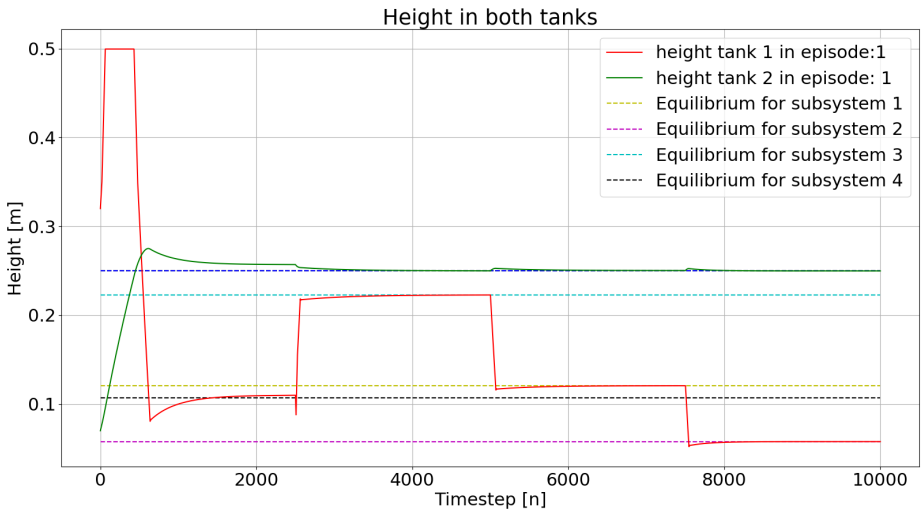


Figure 6.27: Both valves start opened and one and one closes. First tank 1 closes, then tank 2, and at last tank 1 opens again.

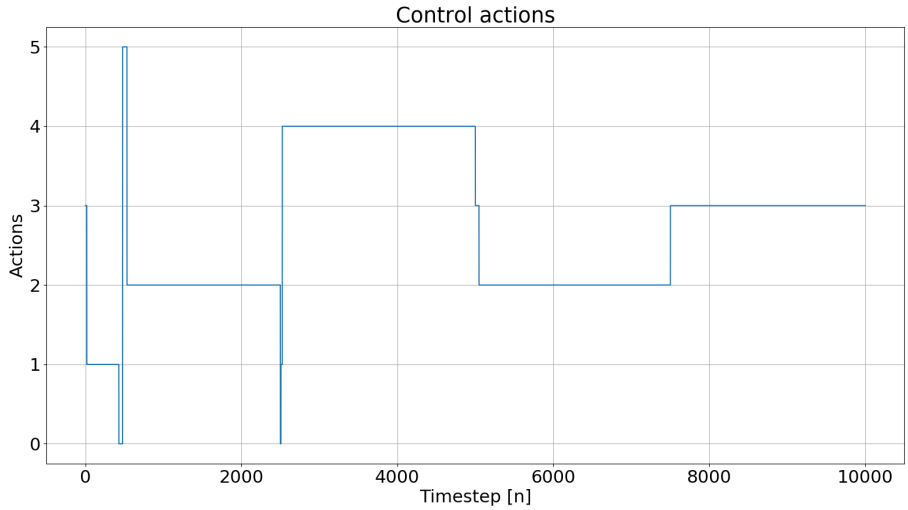


Figure 6.28: Actions taken from the simulation in figure 6.27

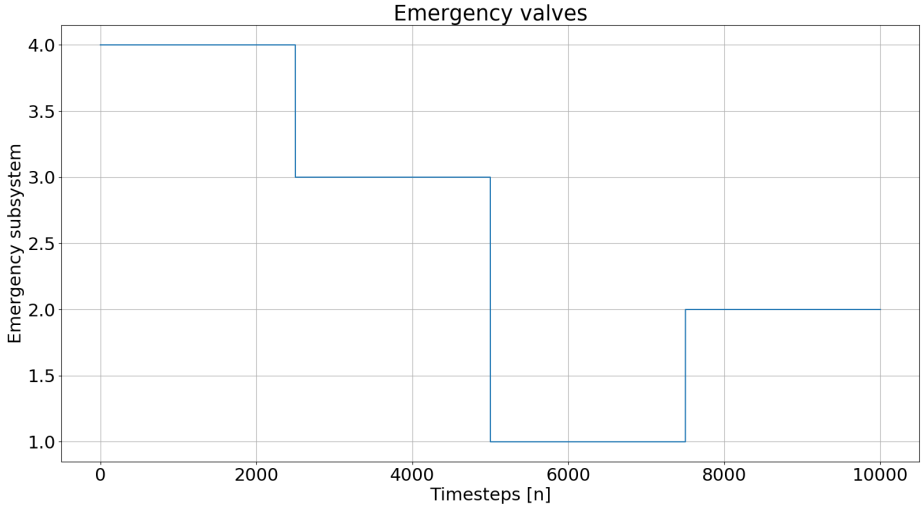


Figure 6.29: Actions taken from the simulation in figure 6.27

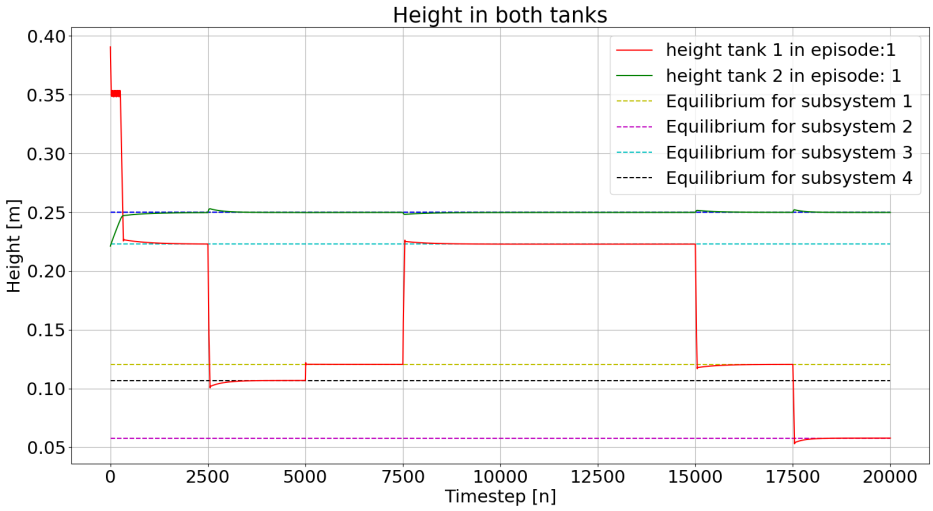


Figure 6.30: The dynamics of the system that randomly switches on and off the valves every 2500 timesteps.

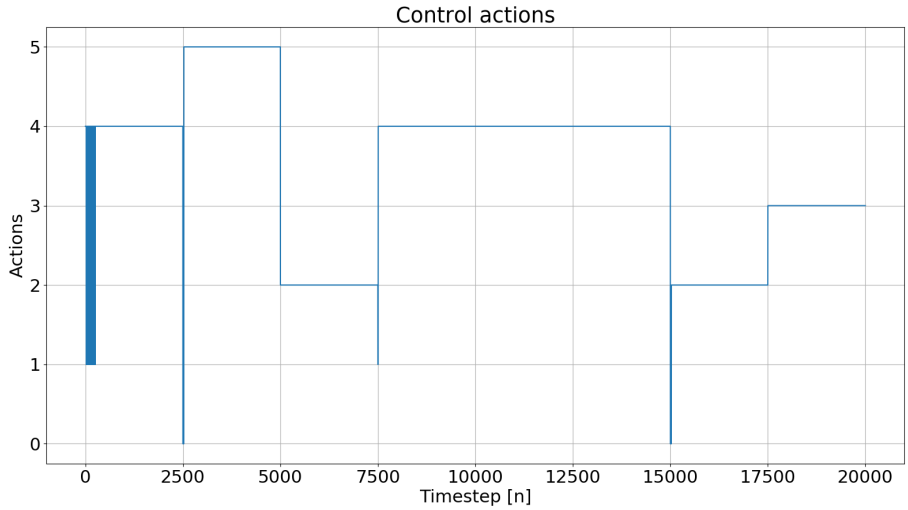


Figure 6.31: Actions the agent does for the dynamics in figure 6.30

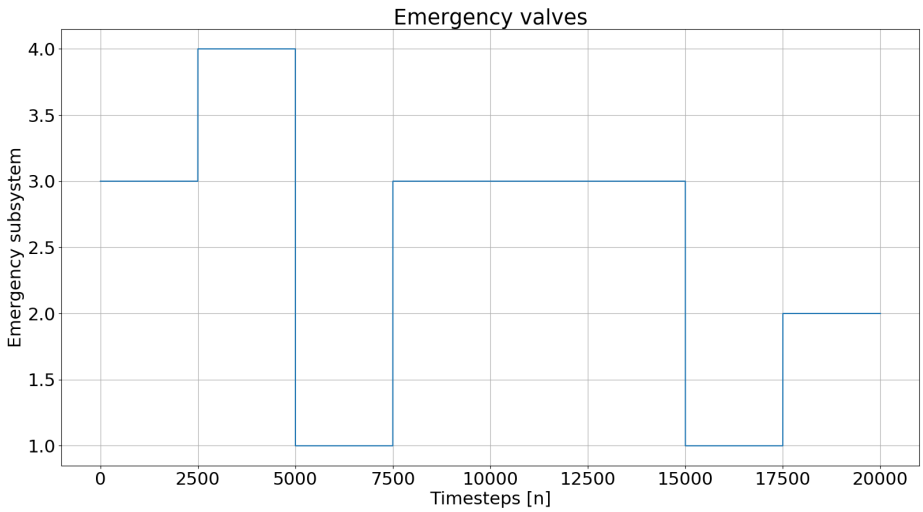


Figure 6.32: This shows which valve that is open from the dynamics in 6.30.

In figure 6.30 the dynamics of the system that randomly opens and closes the valves every 2500 timesteps are plotted. One can see on the red line that it moves to the equilibrium point for the current flow that goes through the tanks. In figure 6.31 the controller switching is shown. And the agent chooses the correct LQR based on which valve is open. Whenever both valves are open the water level in tank 1 is on the black dotted line and the controller chosen is 5, which is the correct controller for subsystem 4. This can be said for each segment.

6.3.1 Additional discussion

In this section one could see that the actions the agent chose were more optimal in the sense discussed earlier. The oscillatory behavior is almost gone and highly reduced in comparison to the 40 states case in figure 6.1. It was argued that the reason for this behavior was either weighting of future reward, local minimum or that the agent still has to converge to the optimal policy. What happens when one reduces the state space is equivalent to training the 40 state system more. This is in line with what is seen with the reduced state space versions. The agent seems to be acting more optimally when the state space is reduced. There is still some oscillatory behavior that can be seen in figure 6.31 and 6.16.

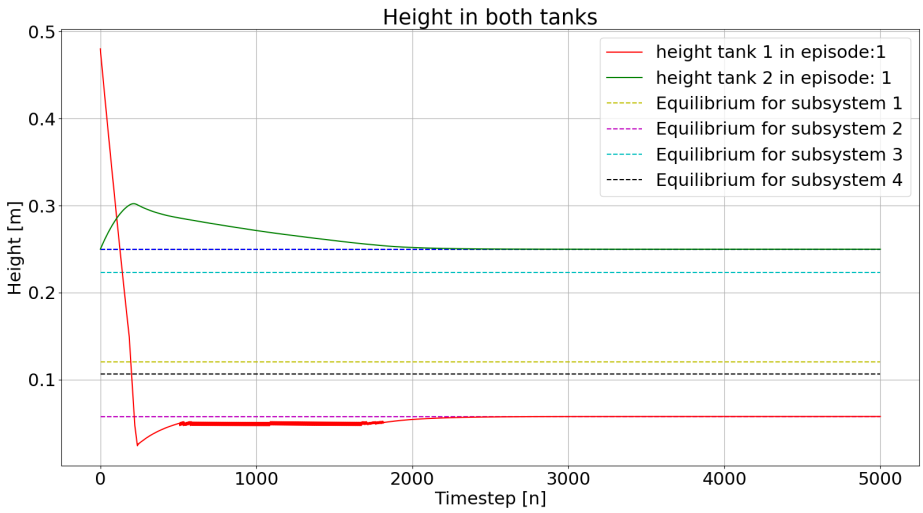


Figure 6.33: Same as in figure 6.15 with less training.

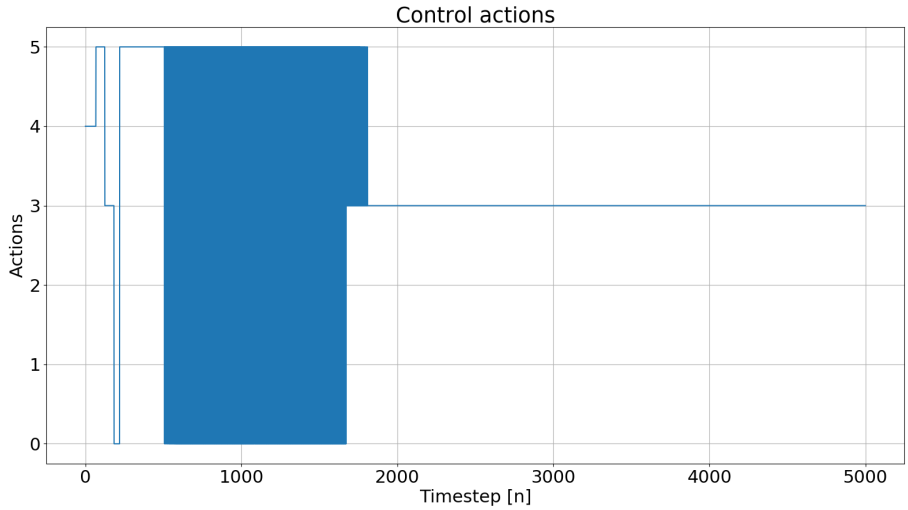


Figure 6.34: Same as in figure 6.16 with less training.

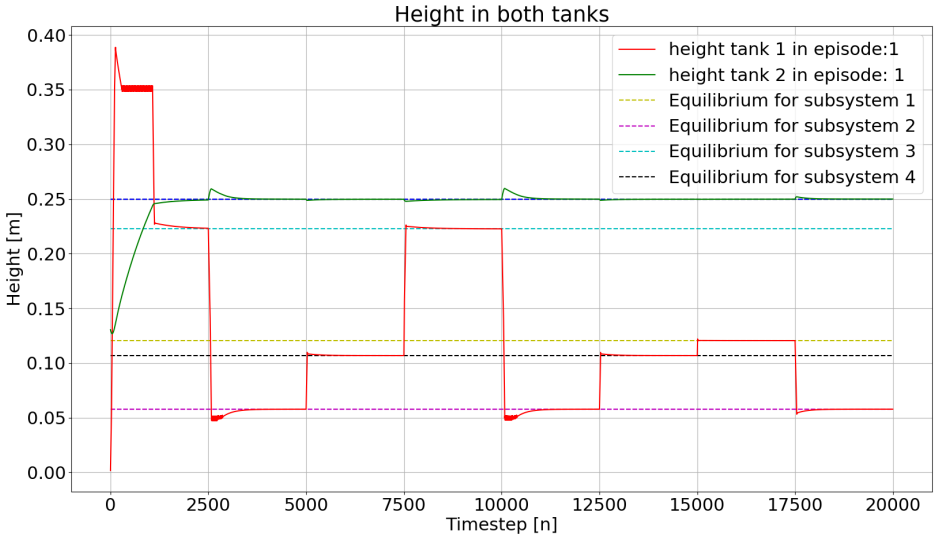


Figure 6.35: Valves open and closes randomly as in figure 6.30 but with less training.

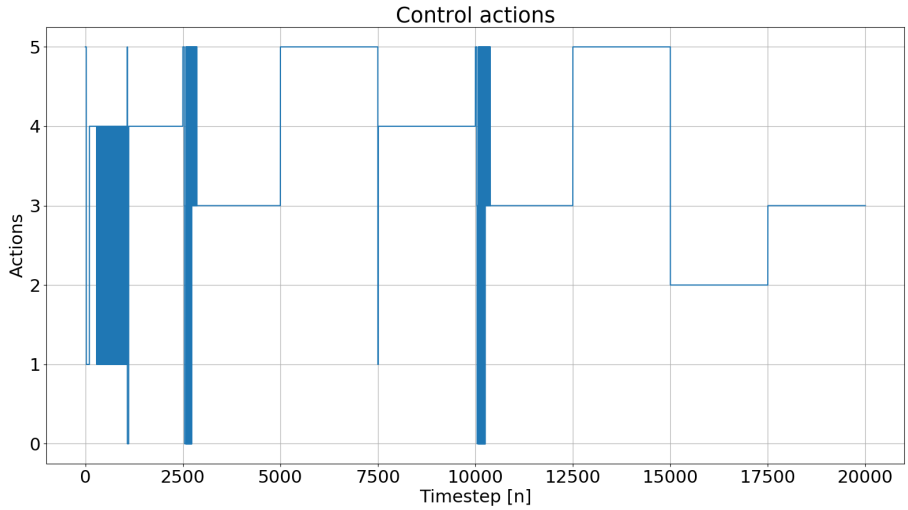


Figure 6.36: The actions taken for figure 6.35.

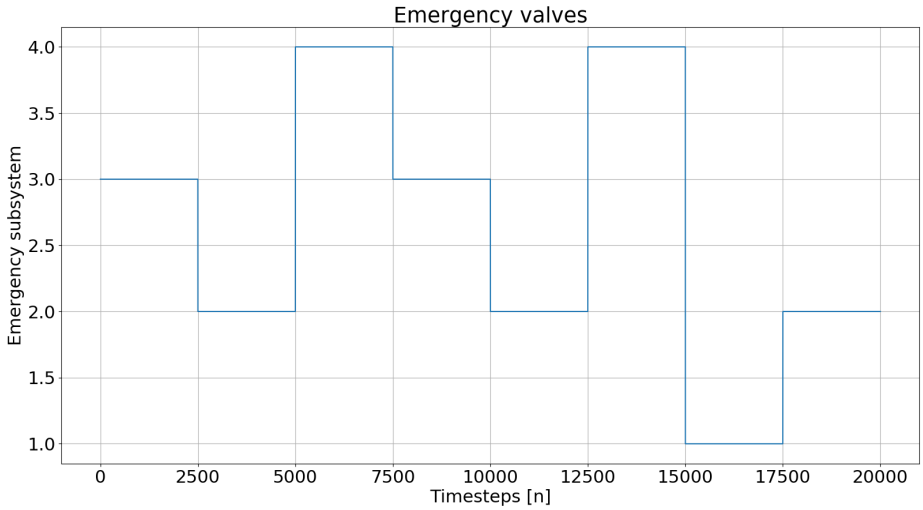


Figure 6.37: This shows which valves are open for 6.35.

Taking it further figure 6.33 and 6.34 show the same dynamics for subsystem 2 as in figure 6.15 and 6.16. It is clear that the amount of training is important to get optimal behavior. There is a lot more oscillation when training is reduced. And in figure 6.35 and figure 6.36 the dynamics + actions for the emergency system with random switching are shown. Compared to figure 6.30 and figure 6.31 there is much more oscillation. But as stated earlier, the goal is not to obtain optimal behavior, but to research the possibility for an RL agent to define the rules to switch between the correct controllers for this system.

Another way to remove this unwanted switching is by penalizing the control input. Whenever the agent switches controller it is punished, or add the controller value to the reward function. Both these ways were experimented with, but if one looks back at chapter 3.1.2 it stated that the reward function should not be a complete description of how the agent should behave. By trying to introduce the controller value into the reward function one would ”help” the agent behave the proper way. The main purpose of this thesis was to avoid an unnecessary amount of knowledge and build the system as simple as possible. Adding the controller to the reward function the proper way would probably decrease these oscillations, but would not follow the methods stated in the literature of how one should design a reward function [(24), pages 42-43].

Chapter 7

Conclusion and future work

The rules for switching between multiple controllers in a nonlinear system were made possible with the use of reinforcement learning. The agent was able to define areas for when to use different controllers. This was first shown with a simple coupled tank system. The agent was able to learn an optimal switching strategy to reach a set point and regulate around this area. It was made easier with the reduction of the state space. This improved the result for this system because the relative training to state ratio was much higher.

It was investigated further what the addition of 2 more controllers would do to the system. The agent was able to choose the appropriate controller. However, in one of the cases there was some oscillatory behavior most likely due to a local minimum, a policy that still has to converge to the optimum, or the future rewards have big weights.

Making the system more complex in such a way that it could not meet the control objective with only one controller was done by adding two more valves. These where emergency valves. These were either opened when the water level rose above 0.375m to avoid overflow, forcefully opened or closed in the middle of the episode, or randomly opened in fixed intervals. These additional valves separated the system into 4 subsystems, each with their own LQR. The agent was able to switch to the correct LQR depending on which valve was open. When switching between controllers there was some oscillatory behavior that could again be explained with what was stated above.

When the valves were forcefully opened and closed the agent was able to switch to the correct LQR when the system transitioned to the next subsystem. Switching from subsystem 1 to 3 the agent switched to the max controller and then over to the correct LQR. Other times it did not do this and switched directly to the next correct LQR. The agent looks far ahead into the future and it might avoid overshooting the water level in tank 2.

The agent was able to define a set of rules to switch between the correct LQR to control around a set point, but it seems to not switch optimally to get to this equilibrium as fast as possible. It was shown that by training the system less the oscillation was worse, hence

by training the system more it would eventually converge to the optimal policy and the oscillation would be gone, or at the least decrease.

The agent was able to efficiently switch between the correct controllers and define a set of rules to do this. The state-action-space was reduced from 30000 to 45 where the performance was even better in the latter case. This was combined to make the system more complex such that more than one controller was needed for satisfactory control. And the agent was still able to define rules for controller scheduling. The agent needed to be trained more for perfect switching, but it was enough to answer all the research question in chapter 1.1.1 properly.

This is a very simple system and there have been done controller scheduling for different systems (22) and gain scheduling + controller scheduling in (6). These two systems are more complex than the system in this thesis. For future work, one should try to implement this with a more complex system, maybe where gain scheduling is already utilized. Before the topic of the thesis was chosen, it was considered to cut RL out of the control scheme, which is to use artificial intelligence (AI) directly to control a system. This would be of great interest because one of the main reasons for this thesis was to study if one could drop a lot of the work and knowledge needed about the system to control it. Reflecting on this thesis, one had to know a lot about the system, about the controls and the theory of RL needed to be well known and well studied to make this system. Cutting out RL doesn't necessarily take less work, but it means that we would not need to know as much about the system and the controllers to meet the control objective, and we would let the AI do a lot of the work. This would make such tasks more scalable and adaptive.

Bibliography

[1] M. A. Nielsen, "Neural networks and deep learning," 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[2] A. A. e. a. Tijjani AS, Shehu MA, "Performance analysis for coupled - tank system liquid level control using mpc, pi and pi-plus-feedforward control scheme." *J Robotics Autom*, vol. 1, no. 1, pp. 42–53, 2017.

[3] M. Stewart, "The limitations of machine learning," 2019, [Online; accessed ¡18.04.2020¿]. [Online]. Available: https://towardsdatascience.com/the-limitations-of-machine-learning-a00e0c3040c6

[4] EDUCBA, "Supervised learning vs unsupervised learning," 2020, [Online; accessed ¡19.04.2020¿]. [Online]. Available: https://www.educba.com/supervised-learning-vs-unsupervised-learning/

[5] M. Nagayoshi, H. Murao, and H. Tamaki, "A reinforcement learning with switching controllers for a continuous action space," *Artificial Life and Robotics*, vol. 15, pp. 97–100, 08 2010.

[6] P. Pierpaoli, T. Doan, J. Romberg, and M. Egerstedt, "A reinforcement learning framework for sequencing multi-robot behaviors," 09 2019.

[7] W. Knight, "This factory robot learns a new job overnight," 2016, [Online; accessed ¡19.04.2020¿]. [Online]. Available: https://www.technologyreview.com/2016/03/18/161519/this-factory-robot-learns-a-new-job-overnight/

[8] P. TECHNOLOGIES, "Solving intelligence for investment management," 2018, [Online; accessed ¡19.04.2020¿]. [Online]. Available: https://pit.ai/

[9] M. Nazari, A. Oroojlooy jadid, L. Snyder, and M. Takáč, "Deep reinforcement learning for solving the vehicle routing problem," 02 2018.

[10] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283 –

302, 1997, robot Learning: The New Wave. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889097000432

[11] L. Yang, Z. Nagy, P. Goffin, and A. Schlueter, "Reinforcement learning for optimal control of low exergy buildings," *Applied Energy*, vol. 156, pp. 577 – 586, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030626191500879X

[12] W. Ilg and K. Berns, "A learning architecture based on reinforcement learning for adaptive control of the walking machine lauron," *Robotics and Autonomous Systems*, vol. 15, no. 4, pp. 321 – 334, 1995, reinforcement Learning and Robotics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0921889095000095

[13] NTNU, "Structuring an assignment," 2020, [Online; accessed ¡19.04.2020¿]. [Online]. Available: https://www.ntnu.edu/sekom/structuring-an-assignment

[14] N. B. Almutairi and M. Zribi, "Sliding mode control of coupled tanks," *Mechatronics*, vol. 16, no. 7, pp. 427 – 441, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741580600033X

[15] H. Pan, H. Wong, V. Kapila, and M. S. de Queiroz, "Experimental validation of a nonlinear backstepping liquid level controller for a state coupled two tank system," *Control Engineering Practice*, vol. 13, no. 1, pp. 27 – 40, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066103002946

[16] K. Tan, S. Huang, and R. Ferdous, "Robust self-tuning pid controller for nonlinear systems," *Journal of Process Control*, vol. 12, no. 7, pp. 753 – 761, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959152402000057

[17] K.-L. Wu, C.-C. Yu, and Y.-C. Cheng, "A two degree of freedom level control," *Journal of Process Control*, vol. 11, no. 3, pp. 311 – 319, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959152400000056

[18] A. Visioli, "A new design for a pid plus feedforward controller," *Journal of Process Control*, vol. 14, no. 4, pp. 457 – 463, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959152403000982

[19] F. Bianchi, R. Mantz, and C. Christiansen, "Gain scheduling control of variable-speed wind energy conversion systems using quasi-lpv models," *Control Engineering Practice*, vol. 13, no. 2, pp. 247 – 255, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066104000541

[20] "A reinforcement learning variant for control scheduling," *Guha*, pp. 479–485, 1990. [Online]. Available: https://papers.nips.cc/paper/337-a-reinforcement-learning-variant-for-control-scheduling.pdf

[21] J. D. d. M. A. D. D. N. A. J. J. L. F. A. A. R. Diniz, P. R. M. Pires and S. M. Kanazava, "Reinforcement learning for controlling a coupled tank system based on the scheduling of different controllers," in *2010 Eleventh Brazilian Symposium on Neural Networks*, Sao Paulo, Brazil, 2010, pp. 212–216.

[22] M. Nishimura, J. Yoshimoto, and S. Ishii, "Acrobot control by learning the switching of multiple controllers," pp. 67–71, 2004. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2Fs10015-004-0340-6.pdf

[23] A. R. T. Rafal Goebel, Ricardo G. Sanfelice, *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2014-2017.

[25] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *Journal of Machine Learning Research*, vol. 5, pp. 1–25, 2003.

[26] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: an overview," in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, vol. 1, Dec 1995, pp. 560–564 vol.1.

[27] S. McLeod, "Behaviorist approach," 2017, [Online; accessed ¡18.05.2020¿]. [Online]. Available: https://www.simplypsychology.org/behaviorism.html

[28] I. The MathWorks, "Define reward signals," 1994-2020, [Online; accessed ¡18.05.2020¿]. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/define-reward-signals.html

[29] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

[30] R. M. Murray, "Lqr control," 2016, accessed: 19.03.2020. [Online]. Available: http://www.cds.caltech.edu/~murray/courses/cds110/wi06/lqr.pdf

[31] K. J. Åström Richard M. Murray, *Feedback Systems*. Princeton University Press, 2008.

[32] K. Murphy, "A brief introduction to reinforcement learning," 1998, [Online; accessed ¡02.04.2020¿]. [Online]. Available: https://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html

[33] J. Zhang, "Solving the gridworld problem with reinforcement learning," 2019, [Online; accessed ¡03.04.2020¿]. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-implement-grid-world-from-scratch-c5963765ebff

[34] T. S. freeCodeCamp, "An introduction to reinforcement learning," 2018, [Online; accessed ¡20.05.2020¿]. [Online]. Available: https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/

[35] T. Simonini, "Q learning theory and explantion," 2018, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/

[36] L. Matignon, G. Laurent, and N. Fort-Piat, “Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning,” 09 2006.

[37] T. S. freeCodeCamp, “An introduction to deep q-learning: let’s play doom,” 2018, [Online; accessed ¡20.05.2020¿]. [Online]. Available: https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/

[38] Quanser, “Coupled tank educational kit,” 2020, accessed: 24.03.2020. [Online]. Available: https://www.quanser.com/products/coupled-tanks/

[39] T. Oliphant, “NumPy: A guide to NumPy,” USA: Trelgol Publishing, 2006–, [Online; accessed ¡02.04.2020¿]. [Online]. Available: http://www.numpy.org/

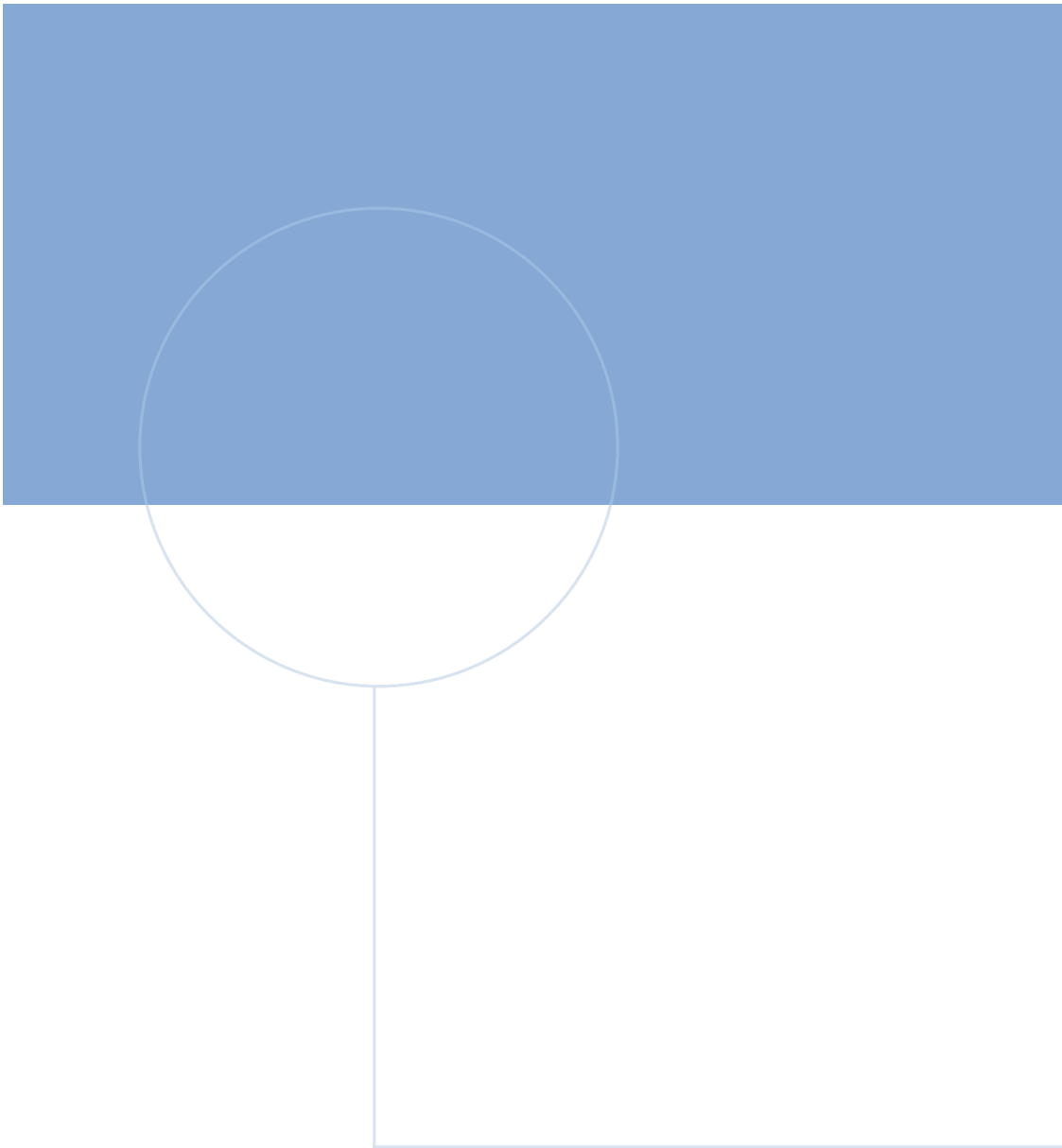
[40] python.org, “Pickle library documentation,” 2020, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://docs.python.org/3/library/pickle.html

[41] E. F. M. D. John Hunter, Darren Dale and the Matplotlib development team, “Matplotlib documentation,” 2012-2018, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://matplotlib.org/3.1.1/contents.html

[42] T. S. community, “Linear algebra toolbox for python,” 2019, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/linalg.html

[43] P. Harrison, “A q learning code example,” 2020, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://pythonprogramming.net/own-environment-q-learning-reinforcement-learning-python-tutorial/

[44] T. Simonini, “A q learning code example,” 2018, [Online; accessed ¡28.04.2020¿]. [Online]. Available: https://github.com/simoninithomas/Deep_reinforcement_learning_Course/blob/master/Q%20learning/FrozenLake/Q%20Learning%20with%20FrozenLake.ipynb



NTNU

Norwegian University of
Science and Technology