Controlling an inverted pendulum with a robot arm

Department of Data Science and Knowledge Engineering
Maastricht University

Esther Kemper, Lukas Lindner Herrero, Pablo Angel Pérez Soto,
Nik Vaessen, José Luis Velasquez Sosa, Simon Wengeler

March 4, 2019

Abstract

In control theory, the inverted pendulum - being a nonlinear, unstable control problem with a relatively simple setup - has long been of interest. Typically, the pendulum is set up on a cart-pole. This research paper looks into controlling an inverted pendulum attached to a 2-jointed robot arm with the SARSA and DQN algorithms. The controllability of the physical set up is investigated. Experiments are performed on a simulation. Results show difficulty in converging to a good control policy and future research is advised.

Keywords: inverted pendulum, closed-loop control, reinforcement learning.

Introduction

Closed-loop control is an area of interest for a wide array of applications. Noisy real world processes require more than just feed-forward control to ensure reliable performance of the system. Closed-loop or feedback control can be used to control factory robots, as well as autonomous control of drones and also has applications in rocket science. The environment the systems are operating in can experience all kinds of noise and change. Thus, the systems have to be robust to such variations.

The inverted pendulum has extensively been studied since it exhibits challenging nonlinear unstable behaviour while being relatively simple to set up and describe. Opposed to this papers' robot arm approach, most research has been done with the cart-pole problem where a pendulum is attached to a rail-guided cart, which restricts the movements of the system on a two-dimensional horizontal line [8].

For controlling the pendulum in more traditional settings, a variety of different approaches have been studied, including the control of kinetic energy [3] [4], using nonlinear controllers [5] [9], PID controllers and Linear Quadratic Regulators (LQR) [7] [15], neural networks [1] [8], fuzzy logic [12] [18], reinforcement learning [11] [16] and often, combinations of these techniques.

Because of the large body of work on the topic it is interesting to study variations on the classical problem that have not been explored yet. As such, this work proposes an adapted version of the problem in which the pendulum is attached to the end of a 2-jointed robot arm. Furthermore, it should be noted that work on the inverted pendulum problem is often concerned with simply controlling the pendulum without the explicit goal of swinging the pendulum into an upward position or balancing it. Some of the referenced papers either discuss the general problem or only one of the mentioned sub-problems but the goal of this project is to perform both tasks.

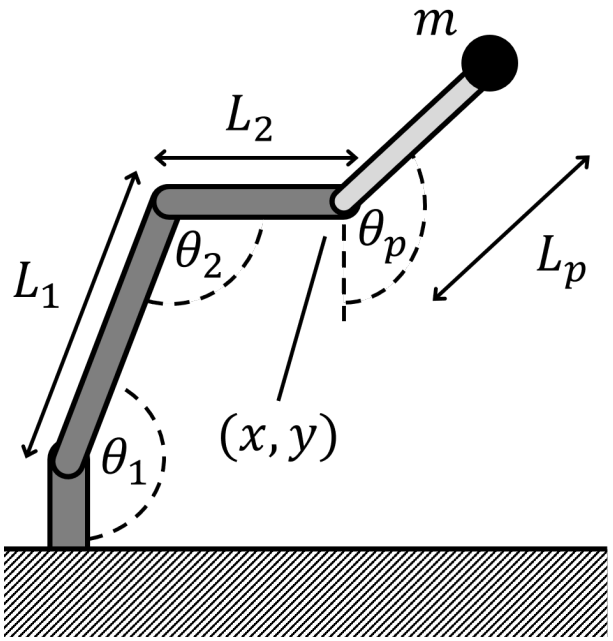

Figure 1: Schematic of pendulum setup

Parameter	Description
L_1, L_2, L_p	Length of lower arm segment, upper arm segment and pendulum respectively
θ_1, θ_2, θ_p	Angle of lower segment, upper segment and pendulum
m	Mass of the pendulum bob
x	Horizontal position of the pendulum support point
y	Vertical position of the pendulum support point
b	Damping constant on the pendulum axle
g	Gravity (9.81 ms^{-2})

Table 1: Description of system variables

In Figure 1 a schematic of the pendulum attached to a robot-arm is shown. The relevant variables of the system for modelling purposes are described in Table 1.

Problem definition

The project deals with closed-loop control of a robot arm balancing an inverted pendulum. The robot arm consist of two joints, each which is actuated by a servo motor. The pendulum is attached to the top module and is able to freely swing 360° along one axis in two dimensions. The bottom module of the arm is connected to a horizontal base, which allows the arm to stand upright.

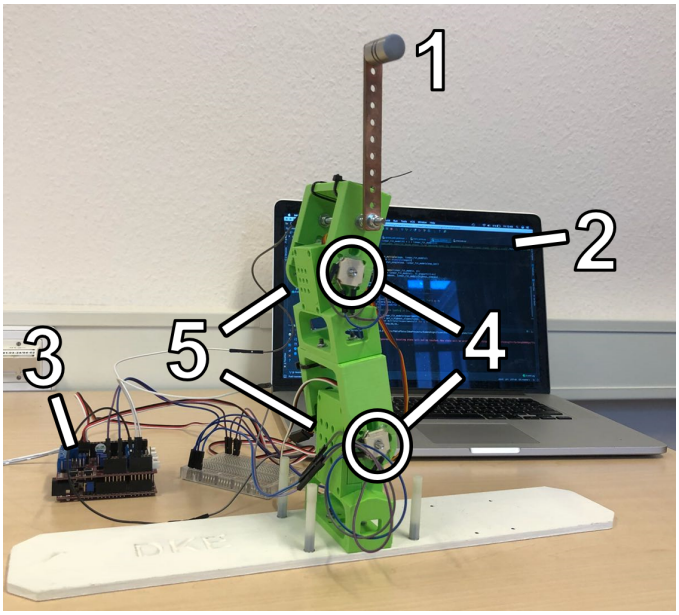

Figure 2: Picture of pendulum setup

In Figure 2 the physical setup of the system is shown. Three potentiometers (4) are used to determine the angular positions of the pendulum (1). The two joints contain servo-motors (5). The sensory data is collected by a micro-controller (3) and sent via serial communication to the PC (2) which is acting as main processor. The PC processes the sensory data, determines the actions and sends the required movements back to the micro-controller (3). This micro-controller in turn controls the servo motors.

There are three problems associated with controlling the pendulum. The first problem, swing-up, starts with the pendulum hanging down and the motor joints standing up-right. The pendulum must then be controlled to get it into a position pointing upwards with a low velocity as quickly as possible. The second problem, balancing, starts with the pendulum pointing up while it has a low velocity and the joints are pointing upwards as well. The goal with balancing is to control the pendulum in such a manner that it keeps pointing into the upper direction as long as possible. The third problem combines both tasks. The pendulum starts in the same configuration as the swing-up tasks, but the goal is to control it in such a way that the pendulum is in the upper region for as much time as possible.

The swing-up problem is considered over when the pendulum reaches the goal state for the first time or when it has failed to do so for a certain amount of time. The balancing task is considered done when the pendulum falls below a certain angle or when it has been able to balance it for a certain amount of time. The combined problem is never considered done unless a certain amount of time has passed. Throughout this paper this amount of time has been defined as 200 actions.

Research questions

The initial research plan was to compare two different control approaches, namely the classical control theory approach with the machine learning approach. Comparisons would have been made in terms of robustness against noise and stability of these two systems.

However, after encountering several issues with the control theory approach (as addressed in section 1.1), the decision was made to only experiment with different reinforcement learning approaches. Herein, the research questions addressed, are:

1. “Is the physical system controllable?”
2. “Is the chosen approach, reinforcement learning, appropriate to solve this problem?”
3. “Can one controller be used for both swing-up and balancing the pendulum?”

The first research question asks if the system is controllable, or if the same states can be reached performing

the same actions given the same initial state. This is necessary in order to be able to control the system and for a reinforcement learning algorithm to swing up and balance the pendulum. It is expected that the system is controllable since the hardware behaviour should be predictable. In the second question the chosen approach is evaluated. The cart-pole system has been solved with a reinforcement leaning approach. Since this problem is very similar to the cart-pole system it is expected that it is also appropriate for this setup. In order to answer the last question, research must be done into whether is it considerably more efficient to have two separate controllers switching according to the pendulums state. It is expected that learning the general combined task will be easier to manage because the controllers do not need to be switched, but that the separate task will most likely by easier to learn.

The structure of the report will be as follows. In Section 1 the mathematical model and simulation will be explained. Section 2 will go over the reinforcement learning approach. After that the experiments are described, followed by the results and discussion of these results. Lastly, the conclusion will be given.

1 Mathematical model and simulation

As the given list of relevant literature shows, the use of mathematical models of the physical system is widespread in research. Therefore, one attempt to perform control in the described variation of the classical cart-pole problem was to construct and use such a model.

It should be noted that while this model will be detailed in the following section, it was decided during the course of the project that the reinforcement learning approach should take priority. Because of this, no complete control method using said mathematical model was developed; however, it was used for simulating the system. This section should therefore serve to give a basis to and inspire future research.

The relevant variables of the system have already been described in the previous section (Figure 1). The movement of the pendulum (its angular position θ_p) is described by the following differential equation (the derivation of which can be found in [14]):

$$\ddot{\theta}_p = \frac{\cos\theta_p}{L_p}\ddot{x} - \frac{\sin\theta_p}{L_p}\ddot{y} - \frac{b}{mL_p^2}\dot{\theta}_p - \frac{g}{L_p}\sin\theta_p \tag{1}$$

Given that the controllable variables are $\ddot{x}$ and $\ddot{y}$, two general approaches to control the movement of the pendulum are possible. The first is to determine the necessary movement of the anchor point (x, y) and use inverse kinematics to calculate the necessary motor movement, since only the joint angles θ_1 and θ_2 are directly controllable in the described system. The second is to describe the two variables in terms of θ_1, θ_2 and their derivatives.

Because the movement of the point (x, y) is very restricted (especially given the physically small experimental setup), which would have to be accounted for when using the first approach, the second method was chosen. Since the equations for x and y are conceptually the same (only with swapped signs and trigonometric functions), only the equations for x will be developed in the following. The x position can be described as:

$$x = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2 - \pi) \tag{2}$$

Only θ_1 and θ_2 are time-dependent variables, thus the derivative is:

$$\dot{x} = L_1\dot{\theta}_1 \cos\theta_1 + L_2(\dot{\theta}_1 + \dot{\theta}_2)\cos(\theta_1 + \theta_2 - \pi) \tag{3}$$

And finally, the second derivative used in Equation 1 is:

$$\begin{aligned}\ddot{x} = L_1(&\ddot{\theta}_1 \cos\theta_1 - \dot{\theta}_1^2 \sin\theta) \\ &+ L_2((\ddot{\theta}_1 + \ddot{\theta}_2)\cos(\theta_1 + \theta_2 - \pi) \\ &- (\dot{\theta}_1 + \dot{\theta}_2)^2 \sin(\theta_1 + \theta_2 - \pi))\end{aligned} \tag{4}$$

The dynamics of the system can now be expressed as a state-space system using the relevant variables θ_p, θ_1, and θ_2 (the non-standard notation s for the state vector is used because x and y are both variables):

$$\boldsymbol{s} = \begin{bmatrix}\theta_p \\ \dot{\theta}_p \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2\end{bmatrix}, \dot{\boldsymbol{s}} = \begin{bmatrix}\dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2\end{bmatrix} \tag{5}$$

where $\ddot{\theta}_p$ is obtained by substituting $\ddot{x}$ in Equation 1 by Equation 4 (and performing the equivalent substitution for $\ddot{y}$). The two joints' angular acceleration $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are free variables and can be used as control inputs to the system.

A naive method of controlling a state space system is to express it in linear state space form and obtain a linear controller, for instance using an optimization method such as LQR [7] [15]. There are two main problems with this approach for the given system. Firstly, the dynamics are highly nonlinear, thus one would have to perform piecewise linearization, also known as gain scheduling [10], in order to use it. Moreover, the control inputs

and the state variables are multiplied in various summands, which makes it difficult to rewrite the system in the terms of $\dot{s} = As + bu$. In combination with non-mathematical issues such as imprecision of hardware, the control theory approach seemed infeasible to pursue further within the given time. For future research, it could help to look into methods of nonlinear control such as Lyapunov based methods, which have been applied to the inverted pendulum problem in [6] and [2] among others.

Despite the infeasibility of the control theory approach, the developed state-space system was used in a simulation built for testing and training the reinforcement learning model. The system was simulated by using a differential equation solver and integrating the state variables over time using fixed time steps. States for the reinforcement learning methods are observed after some time in order for the last action to take effect. This time can be varied by integrating the system for a variable number of steps.

Since servo motors are used in the real system, the control inputs represent the target positions for these motors. In order to emulate the servo motors' response to a set point input in software, an acceleration controller was developed. It controls the free variables of the simulation, the angular accelerations of the two joints. The acceleration of the motors was set according to the proportional distance of the motor to the set point and the current velocity. Essentially a proportional controller was used to compute the target velocity and then another proportional controller was used to set the acceleration accordingly.

2 Reinforcement learning

Reinforcement learning is a machine learning technique wherein an agent is tasked to find and perform optimal decisions in an environment. The environment is often a Markov decision process, or MDP for short [17]. In an MDP, an agent and environment interact with each other at discrete time steps, $t = 0, 1, 2, 3, ..., T$. At each time step t, the agent receives a representation of the state of the environment, $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states of the environment. Based on the given state s_t, the agent selects an action, $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all actions which the agent can perform when in s_t. The environment transitions to a new state S_{t+1} due to the selected action a_t, which is provided to the agent along with a reward r_t according to a reward function $R(s_t, s_{t+1})$. This process is displayed in Figure 3.

Reinforcement learning is suitable when supervised learning is not possible due to the lack of labeled exemplary data. Instead, the agent has to learn from the reward function $R(s_t, s_{t+1})$, which determines the quality of an action. The goal of the agent is to maximize the rewards it will get over its life-time, as seen in equation 6. The agent has control over maximizing G_t by modifying its policy π_t such that it will take actions which lead to states with high rewards. The policy π_t maps all the states to probabilities for selecting each possible action guiding the agent on choosing the actions with highest probability. This requires exploration, potentially until the best output for any given input is found. If an action leads to a high reward, its probability is strengthened and if it produces a low reward, its probability is weakened.

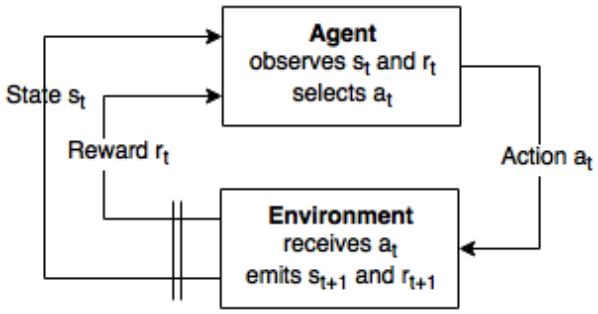

Figure 3: Schematic of a Markov decision process whereby an agent and environment interact

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{6}$$

The total reward G_t an agent will see over the rest of its lifetime at time step t is the sums over the rewards it will receive between t and the terminal step T. Future rewards get incrementally reduced by the discount rate γ unless γ equals 1. Therefore, the discount rate is used to determine how much the agent should think about future rewards versus rewards at the current moment, as a reward received after k time steps will only be worth γ^{k-1} times what it would be worth if it was received immediately. In this way the agent tries to select actions so that the sum of discounted rewards it receives in the future is maximized.

The problem of controlling the robot arm to balance the pendulum can be modeled as a finite MDP. All three different problems associated with controlling the pendulum, as explained in the problem definition, have the same theoretical state and action spaces. They do differ in the reward function. There are slight differences between the models of the simulation and the real-world

system. Both systems are modeled in the following manner:

1. The state space $\mathcal{S}$ is a vector in $\mathbb{N}^4$ wherein the values are respectively the positions of the pendulum, the position of the lower and upper joint, and velocity of the pendulum. For the real-world system, the positions are measured by the potentiometers and thus can have a value between 0 and 1023. For the pendulum, a value of 60 correspondents with hanging down and a value of 572 with balancing up. As the joints can only move 180 degrees, the values can only be between 256 and 768, with 512 being completely upright. The velocity is calculated by taking the difference between the current and previous potentiometer position, taking into account the direction to correctly deal with the pendulum going from 0 to 1023 or vice versa. The boundary is also 0 and 1023 for the velocity. For the simulation, the positions and velocities are in a vector in $\mathbb{R}^6$ and are calculated by equation 5. The bounds for the positions are $-\pi$ and π, with $\pm\ \pi$ hanging down and 0 being up. Due to the range of motions on the joints, their values are only between $\pm\frac{\pi}{2}$. The velocities are theoretically unbounded but practically in the range -6π and 6π.

2. The action space $\mathcal{A}$ is a vector in $\mathbb{N}^2$ wherein the values are respectively the position to move the lower and upper joint towards. The servo module in Arduino allows rotation of the shaft of the motor in a range between 0 and 180 degrees. However, to avoid putting to much strain on the servo motors and potentially damaging them, only actions between 45 and 135 are executed. Thus the lower and upper bounds of the values of $\mathcal{A}$ are 45 and 135, respectively. The simulation uses the same action space and internally converts an action to the correct radian value.

3. Each transition between states is a stochastic process with unknown transition probabilities. Whenever an action a_t is executed, the next state s_{t+1} will be observed by the agent after 100 milliseconds have passed. In the simulation, the new state is calculated by applying the state equations 6 times in a row.

4. The reward function can differ depending on the task. If only swing-up is desired, equation 7 can be used as a reward function. The agent receives a negative reward as long as it has not swung up the pendulum. When the agent does swing up the pendulum, it receives a high reward. The episode will end whenever the high reward is given or when t=200. This means the maximum reward the agent can get is 100 and the minimum reward is -200. The upper region is defined as the area between $\pm\frac{\pi}{6}$ for the simulation and 572 ± 100 for the real-world system. Note that if the velocity is higher than 2π for the simulation or 200 potentiometer units for the real-world it is not considered a swing-up.

$$R_{\text{swing}}(s_t, s_{t+1}) = \begin{cases} -1 & |s_{t+1_0}| < \frac{\pi}{6} \ \ \& \ |s_{t+1_1}| < 2\pi \\ 100 & \text{otherwise} \end{cases} \tag{7}$$

The task of balancing the pendulum has the reward function given in equation 8. The agent will receive a positive reward as long as the pendulum is in the upper region, defined as the same area where the pendulum needs to be for a swing-up. Whenever the pendulum leaves this region or the pendulum has been balanced for 200 steps the episode is stopped. This means that the minimum reward the agent can receive is 0 while the maximum reward the agent can receive is 200.

$$R_{\text{balance}}(s_t, s_{t+1}) = \begin{cases} 1 & |s_{t+1_0}| < \frac{\pi}{6} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The reward function for the combined task is shown in equation 9. The agent will receive a reward of 0 unless the pendulum is in the upper region with a relative low velocity. The upper region has been slightly increased to $\pm\frac{\pi}{4}$ for the simulation and 572 ± 150 for the real-world system. Note that the reward the agent receives tends to go to 1 with a low velocity and tends to go to 0 with a high velocity. The episode will terminate when 200 actions have taken place. Thus the minimum reward the agent can receive is 0 and the maximum reward the agent can receive is 200.

$$R_{both}(s_t, s_{t+1}) = \begin{cases} e^{-|s_{t+1_0}|} & |s_{t+1_0}| < \frac{\pi}{4} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

By transforming the problem of swing up and balancing the inverted pendulum to a MDP, known reinforcement learning techniques can be applied to be able to potentially learn a well-behaving controller. The following sections describe the two techniques which where applied to attempt to solve the described MDP's.

2.1 Learning the state-action function

Q-learning [17] is used to find a policy π_t that maximizes the reward received over the long run. In order to find

the optimal policy π_t^*, first a value function is defined for the state-action pair (s,a), that returns the expected reward when choosing an action a in a state s:

$$Q(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \tag{10}$$

Then, the optimal Q^* value function is defined to return the maximum expected reward when choosing a state-action pair (s,a).

$$Q^*(s,a) = max\, \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \tag{11}$$

If the optimal value $Q^*(s',a')$ of the state s' at the next time step was known for all the possible actions a', then equation 11 can be rewritten as the Bellman equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma max_{a'} Q^*(s',a')|s,a] \tag{12}$$

where the optimal strategy is to select the action a' which maximizes the expected value of $r + \gamma Q^*(s',a')$. The optimal policy the agent should use is then

$$\pi_t^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

.

2.2 SARSA

The SARSA algorithm tries to find π_t by iteratively updating a linear approximation of $Q^*(s,a)$. SARSA is an online adaptation to the Q-learning approach because it uses the learned Q-values to make decisions instead of doing the complete search with random actions. Once these values have been learned, the optimal action from any state is the one with the highest Q-value. After being initialized to zero, Q-values are estimated on the basis of experience as follows:

1. Select an action a from the current state s. This produces a reward r and leads to a new state s'.
2. Update the Q-table:

   $$Q(s,a) = Q(s,a) + \alpha[r + \gamma \operatorname*{argmax}_{a'} Q(s',a') - Q(s,a)]$$

   where α is the learning rate an γ is the discount factor.
3. Both steps are repeated until an optimal Q-table is obtained.

This approach is described in algorithm 1.

In order to achieve a high reward the agent selects actions that have performed well in the past. At every time point, the agent must make a decision to either exploit this knowledge or explore different actions. This decision is based on the exploration rate ϵ. This value dictates whether the next action is going to be random or an already explored option. The value of ϵ is decreased over time. This way the agent explores a lot of random actions in early episodes and exploits this knowledge in the later stages. The learning rate, γ, helps control how fast the Q value is modified. The implementation starts with a high learning rate, which allows fast changes, and lowers the learning rate as time progresses.

Algorithm 1 SARSA

```
Initialize Q-table Q randomly
Initialize environment E
Initialize exploration value ε
for episode = 1:number of episodes do
    s = E.reset()
    for step = 1:200 do
        if random number > ε then
            a = argmax_a Q(s,a)
        end
        else
            a = randomly selected
        end
        s' = E.transition(s, a)
        r = E.reward(s, s')
        Q(s,a) += α[ r + γ argmax_a' Q(s',a') - Q(s,a)]
        if s is terminal then
            break
        end
    end
    decrease ε until a minimum value is reached
end
```

For the continuous state and action spaces to be controlled by a tabular algorithm, like SARSA, both spaces have to be discretized. The discretization of the state space is derived by using boundaries for every dimension in a (min, max) form and an integer value that determines the number of uniformly distributed observations on the dimension, where observation 0 represents observations outside the bounds. The state space used for the SARSA is represented in a 4-tuple $(\theta_p, \dot{\theta}_p, \theta_1, \theta_2)$. Therefore a state space represented by (20,3,7,7) has 20 possible observations of θ_p, 3 possible observations of $\dot{\theta}_p$, 7 possible observations of θ_1, and 7 possible observations of θ_2.

The action space is discretized in a similar manner, a bound for motor movement is given in a pair $(min°, max°)$ where 90° is up right. An integer value c that determines the number of uniformly distributed actions per motor. Therefore, the size of the action

space is c^2. Actions spaces are represented in a 3-tuple $(min^\circ, max^\circ, c)$. An action space represented by (70,110,9) is an action space with a min° of 70°, a max° of 110° and with 9 possible actions per motor.

2.3 Deep Q-Network

The problem with a linear function approximation such as a table for $Q^*(s, a)$ is that the amount of parameters grow exponentially based on the dimensions of the state and action space. The deep Q-Network [13] algorithm uses a deep neural network to approximate $Q(s, a)$ with $Q(s, a, \theta)$, with θ being the neural network parameters. The dimensions of θ can be chosen arbitrarily, which makes it possible to only use a small amount. The general process is described in algorithm 2.

The problem with using a deep neural network for reinforcement learning is that there is no labeled data for supervised learning. To approximate $Q^*(s, a, \theta)$ iteratively with equation 12, the labelled data is generated using the neural network parameters itself. For a given experience of state s, action a, reward r and next state s', a training sample (x, y) is defined by

$$x = Q(s, a, \theta)$$

and

$$y = r + \gamma \; argmax_{a'} Q(s', a, \theta)$$

unless s' is a terminal state, which means no future reward can be received and $y = r$. In other words, the neural network predicts the expected future reward for the state s and action a, and based on the observed next state s' and reward r this expectation can be modified. This modification can be done in the usual way by defining a loss function

$$L(\theta) = (x - y)^2$$

and backpropegating that loss throughout the network with the gradient

$$\Delta L(\theta) = \alpha * 2(x - y)$$

To get a varied, independent amount of training samples experience replay is used. At every time step, the experience tuple (s, a, r, s', $s' == terminal$) is stored in a memory buffer with a capacity C. The buffer acts as a first-in-first-out queue when the capacity has been reached. At the end of every episode, stochastic gradient descent is used on a minibatch of size B to update θ.

Algorithm 2 Deep Q-network with experience replay

```
Initialize neural network Q with parameters θ randomly
Initialize environment E
Initialize memory M with capacity C
Initialize exploration value ε
for episode = 1:number of episodes do
    s = E.reset()
    for step = 1:200 do
        if random number > ε then
            a = argmax_a Q(s, a, θ)
        end
        else
            a = randomly selected
        end
        s' = E.transition(s, a)
        r = E.reward(s, s')
        if s is terminal then
            fill M with the tuple (s, a, r, s', true)
            break
        end
        else
            fill M with the tuple (s, a, r, s', false)
        end
    end
    samples = B randomly selected tuples out of M
    foreach (s, a, r, s', t) ∈ samples do
        a' = argmax_a' Q(s', a', θ)
        if t then
            target = r
        end
        else
            target = r + γ argmax_a' Q(s', a', θ)
        end
        backproperate αΔ(target − Q(s, a, θ))^2 on Q
    end
    decrease ε until a minimum value is reached
end
```

The state space which was given to the neural network was not discretized, but the action was discretized in the same manner for SARSA, based on the task.

3 Experiments

Different experiments were run on the real-world system and the simulation in order to be able to answer the research questions.

3.1 Hardware controllability

To test the ability to control the real-world system, the system was given the same 10 randomly generated input commands in 100 equal experiment runs. 10 moves

were chosen, because the reinforcement learner sends 10 commands per second. After each step, the state of the system expressed in the pendulum position, lower motor position, upper motor position and pendulum velocity, was recorded in order to compute the variance of each of these variables across the 100 experiments. Especially important for the reinforcement learner are the values "pendulum_position" and "pendulum_velocity", since they form the main part of the reward function. If these readings don't yield reliable values, the problem might become too volatile for the reinforcement learner.

3.2 Reinforcement strategies

To test whether reinforcement learning was a suitable control strategy for the tasks, the SARSA and DQN algorithms were ran on the simulation.

For the SARSA algorithm different state spaces and action spaces were tested (see appendix Table 4) in order to find out which combination of partitions of the states resulted in a faster convergence of the Q function for balancing the pendulum. Each episode started with an initial state of ($\frac{23\pi}{24}$,0,π,π) and moved 200 steps. The exploration parameter ϵ was decreased in a manner of $1 - \frac{episode}{total\ episodes}$. The learning rate α was decreased linearly in the bound (0.8, 0.1). The only conditions that were met in the set up of the state and action spaces were $\theta_1 = \sqrt{|actionspace|}$ and $\theta_2 = \sqrt{|actionspace|}$. This way, for every action taken by any motor would correspond to a corresponding state observation of the corresponding motor.

As there were a lot of parameters (see appendix Table 5 to tune in order for DQN to learn properly, a hyperparameter search was performed to try and find the values which worked well on the combined task, which means the experiments were only performed using the reward function described in equation 9. Due to the large number of parameters, as well as their possible values, it was not feasible to explore the entire parameter space. Therefore, instead of iterating over the entire space it was decided to choose combinations of parameters uniformly at random to gain insight into the performance of single parameter values and hopefully identify those values yielding the best performance. The learning rate was kept stable over the entire experiment. Epsilon was decreased linearly over time in a randomly chosen amount of steps at the start of the experiment. In total 341 experiments were performed.

3.3 Single-agent sufficiency

To be able to answer whether a single agent is sufficient to perform both swing-up and balancing at the same time, each of the three different tasks was run with the DQN algorithm as controller. The parameters were selected based on the results of the hyperparameter search and reasonable guesses. They can be found in appendix Table 6. The algorithm was trained for a total of 5000 episodes, where each episode was 200 steps. The state, reward, selected action and the estimated q-value of that action was recorded for each step. The action space was changed based on the task. For balancing, the action space was 9 steps between 70 and 100 for a total of 81 options. Swing-up had an action space with 5 steps between 45 and 135 for a total of 25 options. The general task had an action space with 9 steps between 45 and 135.

4 Results

The results for the individual experiments are presented in the following section.

4.1 Hardware controllability

The results for the controllability experiments for "pendulum position", "pendulum velocity", "servo1 position" and "servo2 position" are given in the histograms in Figure 4, 5, 6, 7 respectively. Results for both servo motors seem to follow a normal distribution. Servo2s' distribution is skewed to the right, while servo1 is not. However, both seem to follow a normal distribution with a low standard deviations. For the pendulums' readings, a large standard deviation from the mean was found (pendulum position and velocity show similar behaviours). For the same ten commands, the mean position of the pendulum is at 166.37. In 32% of the runs, the readings are between [180, 220], 20% between [220, 260] and 8% between [280, 370]. The distribution is even more skewed to the left, since 25% of the runs, the pendulum was positioned between [10, 80] and 15% between [80, 150].

4.2 Reinforcement strategies

The results for the different runs of the SARSA algorithm are shown below: Table 2 shows the amount of episodes that each state space simulated. Table 3 shows the average cummulative reward of the first and last 10% of the episodes in the run. Figures 8, 9, 10, 11, 12, 13 show the trend of the cumulative reward throught the episodes.

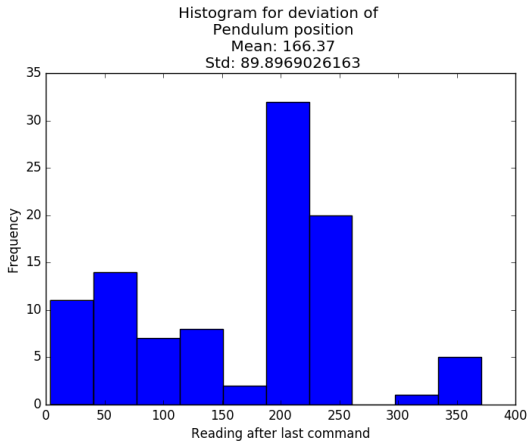

Figure 4: Histogram for "pendulum position" after 10 commands over 100 runs

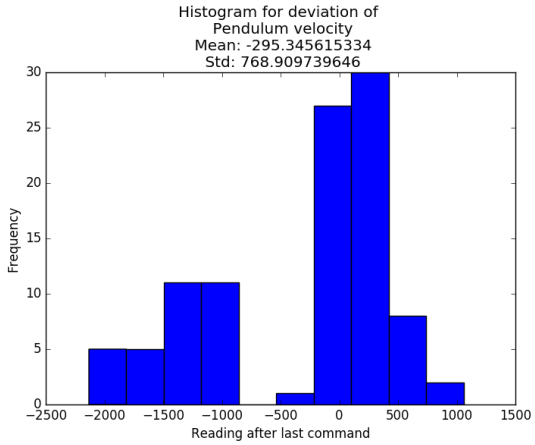

Figure 5: Histogram for "pendulum velocity" after 10 commands over 100 runs

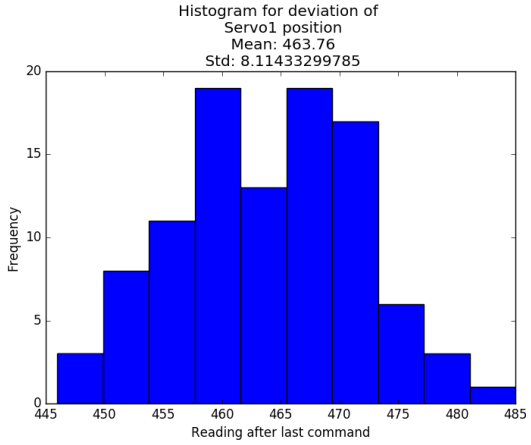

Figure 6: Histogram for "Servo1 position" after 10 commands over 100 runs

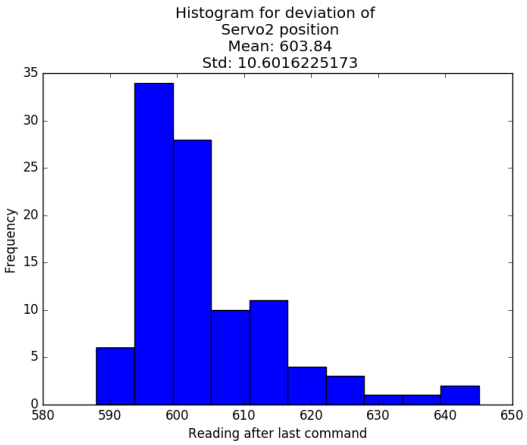

Figure 7: Histogram for "Servo2 position" after 10 commands over 100 runs

state	episodes
(16,3,7,7)	250000
(16,3,9,9)	250000
(16,3,11,11)	100000
(20,3,7,7)	500000
(20,3,9,9)	500000
(20,3,11,11)	500000

Table 2: Episodes ran in training

state	First 10% average	Last 10% average
(16,3,7,7)	0.0941	0.4265
(16,3,9,9)	0.1091	0.4203
(16,3,11,11)	0.3770	0.9774
(20,3,7,7)	0.2041	2.0818
(20,3,9,9)	0.2587	1.6893
(20,3,11,11)	0.2031	2.3146

Table 3: Average reward on first and last 10% of run per state space

4.3 Single-agent sufficiency

The results of the randomized hyperparameter search have been compiled in the form of several violin-plots. For each possible choice of parameter value, they show the distribution of the average cumulative reward per episode (see Equation 6) for each experiment. While the plots for all parameters can be found in Appendix A, only one is used here for illustrative purposes. Figure 14 shows the performance of the agents with the specified learning rates across all experiments performed for the hyperparameter search. While the maximum values differ slightly across different parameter values, the overall distribution is almost identical.

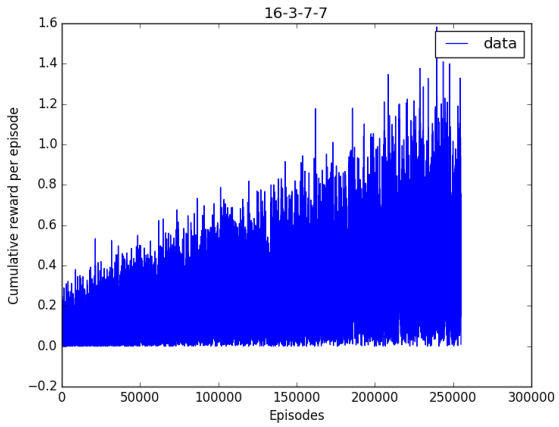

Figure 8: Cumulative rewards for each episode for state space (16,3,7,7) and an action space of(45, 135, 7)

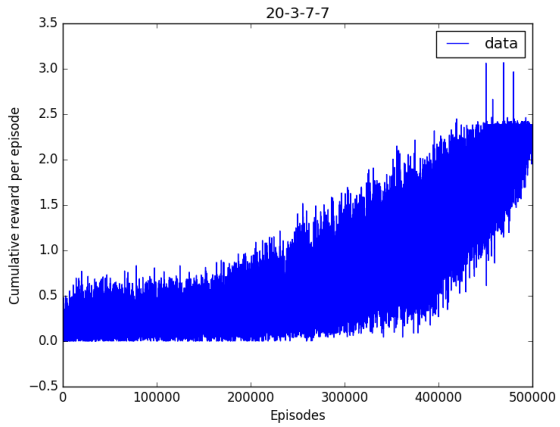

Figure 11: Cumulative rewards for each episode for state space (20,3,7,7) and an action space of (45, 135,7)

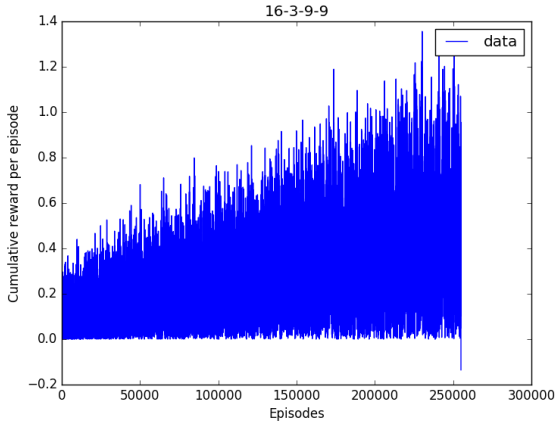

Figure 9: Cumulative rewards for each episode for state space (16,3,9,9) and an action space of (45, 135, 9)

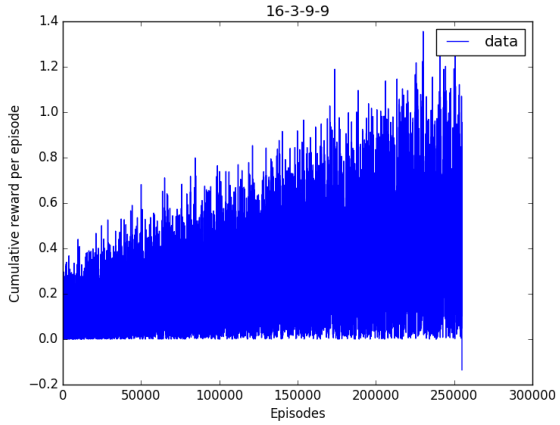

Figure 12: Cumulative rewards for each episode for state space (16,3,9,9) and an action space of (45, 135,9)

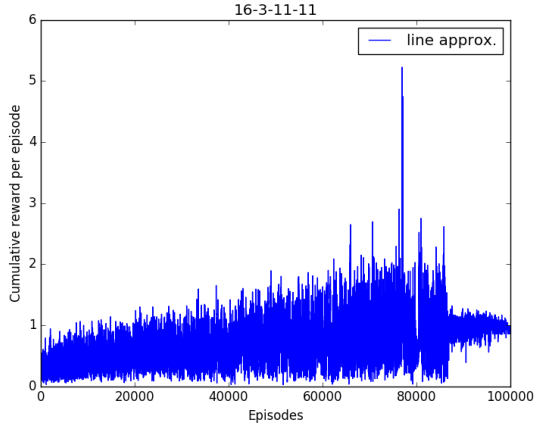

Figure 10: Cumulative rewards for each episode for state space (16,3,7,7) and an action space of (45, 135,11)

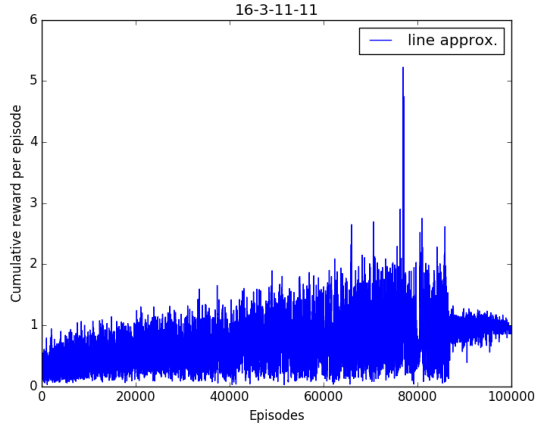

Figure 13: Cumulative rewards for each episode for state space (16,3,7,7) and an action space of (45, 135,11)

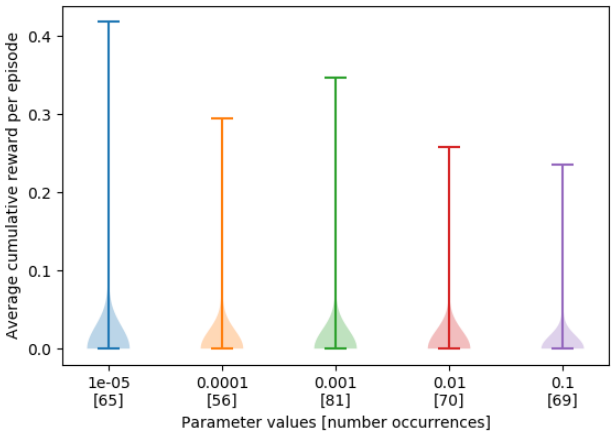

Figure 14: Violin plots of the performance of agents with the specified learning rates across all hyperparameter experiments

For the deep Q-network several metrics were recorded for experiments using all three described reward functions.

For the first, more general reward function (see Equation 9), Figure 15 shows the cumulative reward for each episode across all 5000 episodes as well as a trend line fit to the data. The latter shows that there is a downward trend, the cumulative reward per episode decreases over time on average. The same is shown for random action selection across all episodes in Figure 16.

In Figure 17 the maximum Q-value for three time steps per episode is plotted across all episodes. For about half of the episodes the value is relatively constant and then decreases steeply until the end.

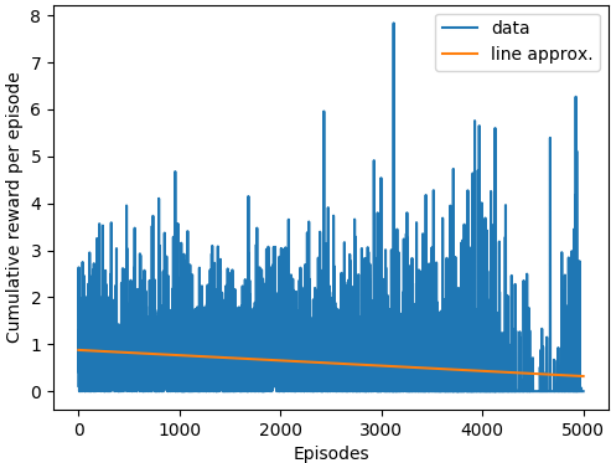

Figure 15: Cumulative rewards for each episode for the DQN

Figure 4.3 shows histograms of the angular position of the two motors. In the top left corner a histogram over

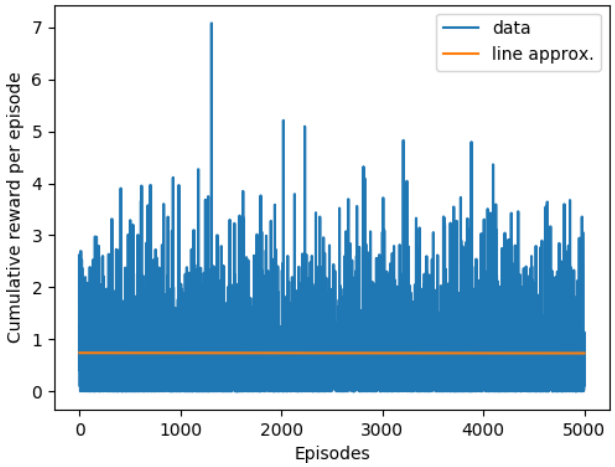

Figure 16: Cumulative rewards for each episode using random action selection

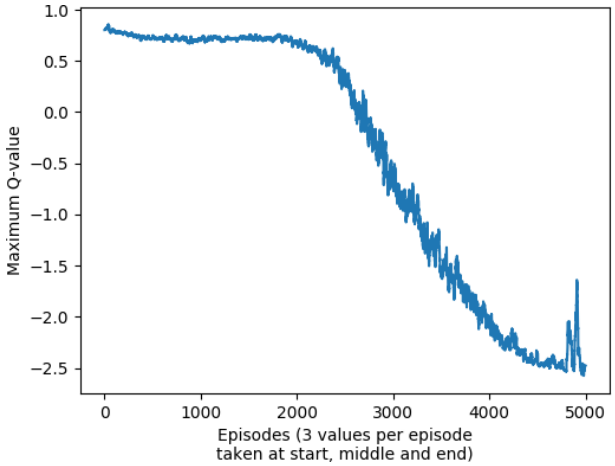

Figure 17: Maximum Q-values for states observed at the start, middle and end of each episode for the DQN (plot smoothed out using a Savitzky-Golay filter)

all episodes is shown. The top right corner shows a histogram over the first 500 episodes and the bottom figures show histograms over episodes 2500 to 3000 and the last 500 episodes respectively. From these plots it can be seen that the distribution of positions follows a normal distribution in the first 500 episodes. As time progresses, more positions are around -0.8π for the lower motor and -0.6π for the upper motor.

Histograms of the difference in consecutive motor positions are displayed in Figure 4.3. The left plot shows the distribution of the first 500 episodes, the right plot over the last 500 episodes. Over the first 500 episodes the values follow a normal distribution, whereas for the last 500 almost all of them are 0.

For the balancing reward function (see Equation 8) and the setting of the balancing task, the cumulative reward over all episodes is shown in Figure 20. The trend line shows an upward trend. The maximum Q-value at three time points per episode is displayed in Figure 21. It also increases over time.

Using the swing-up reward function (see Equation 7) for the swing-up task gives a cumulative reward over all episodes as shown in Figure 22. Again, an upward trend can be observed. Figure 17 shows the maximum Q-value, which decreases for the first 2000 episodes and then increases again.

5 Discussion

Since the lower servo (servo1) has a higher mechanical load to take than the upper servo (servo2), the servo-result of this experiment is more or less expected. The distribution of Servo1 is more skewed than the distribution of Servo2. However, both seem to follow a normal distribution and since the standard deviation is quite small, it suggests that the impact on the systems' behaviour is relatively low. The pendulum however, seems to move in an unanticipated, abnormal manner. The reason could lie in the potentiometers themselves, since they have to deal with dead-zones. But additionally, the reason could lie in small impreciseness of the worn out servos behaviour, that could actually have a high impact on the pendulums' swing. Sometimes the servos' movement is enough to swing the pendulum over the top, sometimes it is not enough and the pendulum falls back rather than overshooting the top. If after 10 moves, the outcome is significantly different, it means that after 1 second of running, the learner already has to deal with a significant uncertainty. Whatever the exact reason for the unanticipated behaviour of the pendulum is, it is hard for the reinforcement learner to adapt to such different behaviours of the hardware when sending the same actions. This experiment concludes that the potentiometers seem to give a reliable output in a normal distribution for the servos.

Considering the slow learning in all the SARSA runs with the increase of reward points per episode, which are in the magnitude of 10^{-6}, it can be said that some degree of learning is occurring. However, given that the maximum reward possible is 200 points and assuming that that the determined linear trend continuous, the number of episodes would have to be in the magnitude of 10^8. This may show that either the action space discretization or the observation space discretization were not done in a meaningful way to approach the Q function, or that the model just needs more time to train. However, the action spaces (45, 130, 7), (45, 130, 9), and (45, 130, 11) had movements of 12°, 9°, and 8° respectively in each motor. Taking into account the difficulty of the task, these step sizes may be too large to be able to control the system easily.

The hyperparameter search for the DQN algorithms did not yield many usable results. With the chosen metric for good performance over all experiments, few of the most important parameters have significant differences across the possible values. It is questionable whether averaging over the results of experiments performed using random combinations of parameters gives a good indication of a their performance in general. A more structured approach to choosing hyperparameters may be advisable (e.g. genetic algorithms). Additionally, considering the difference in performance of the DQN with different reward functions, determining hyperparameters after deciding on a reward function and action space would make more sense.

The DQN performs quite poorly on the general swing-up and balancing task. Figures 15 and 16 show that even with a random policy the average cumulative reward per episode is higher. Additionally, the network's performance actually decreases as training progresses.

This is mirrored in the development of the maximum Q-value over time. While the majority of actions is still performed randomly (during the first half of each run), i.e. the agent is still exploring, the maximum Q-value is roughly constant. However, as more actions are selected by the network itself, the maximum Q-value drastically decreases. This indicates that the agent does not have the expectation of receiving a lot of positive rewards. This may be due to the large continuous action space and relatively small exploration time.

The general reward function only gives significant rewards when the pendulum is in a small region around the upward position and has a low angular velocity. From

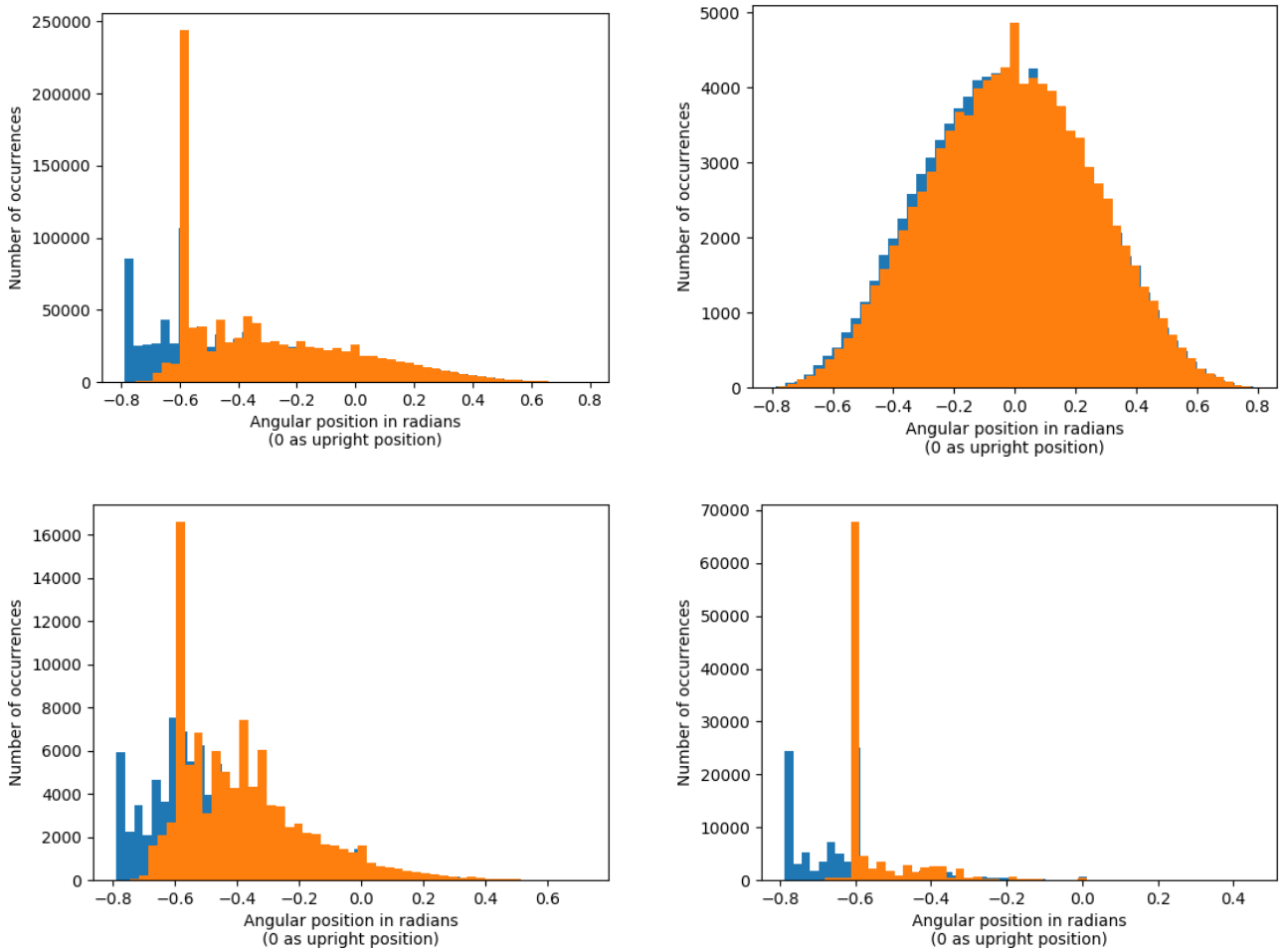

Figure 18: Histograms of the angular position of the two motors (lower in blue, upper in orange). The top left plot shows the distribution over all episodes, the top right over the first 500 episodes, the lower left over episodes 2500 to 3000 and the lower right over the last 500 episodes.

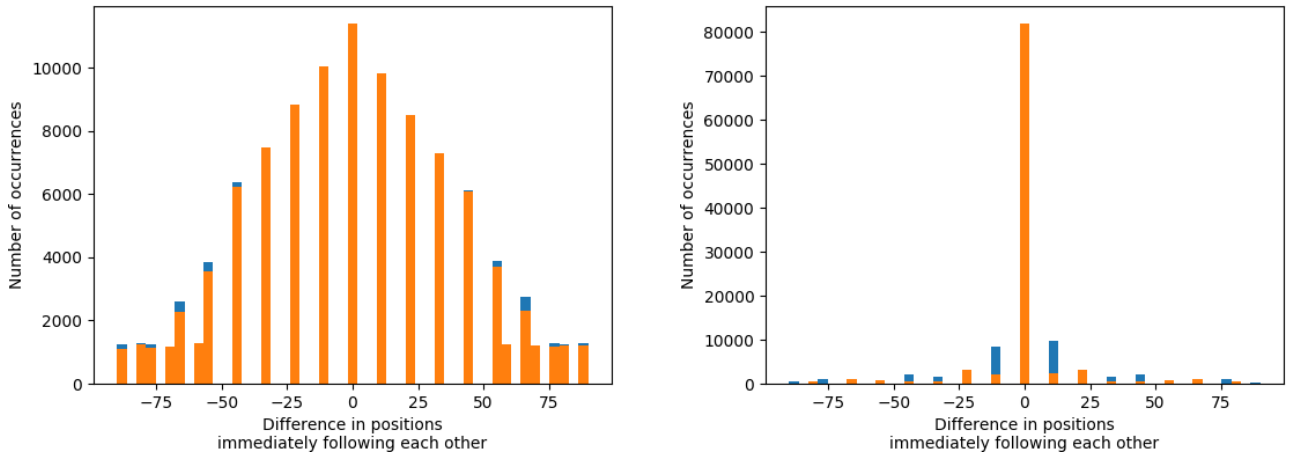

Figure 19: Histograms of the difference in consecutive motor positions (lower in blue, upper in orange). The left plot shows the distribution over the first 500 episodes, the right plot over the last 500 episodes.

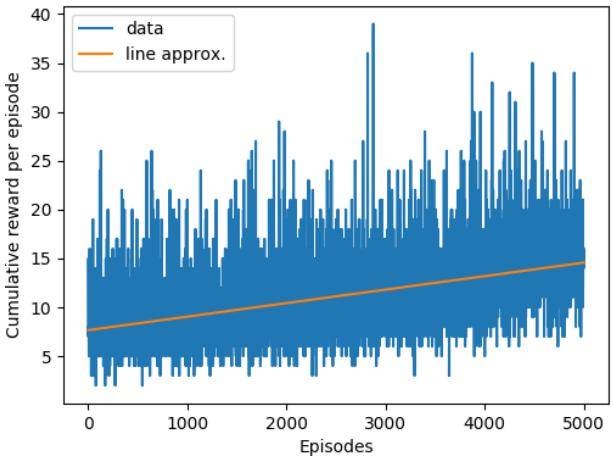

Figure 20: Cumulative rewards for each episode for the DQN using the balancing reward function

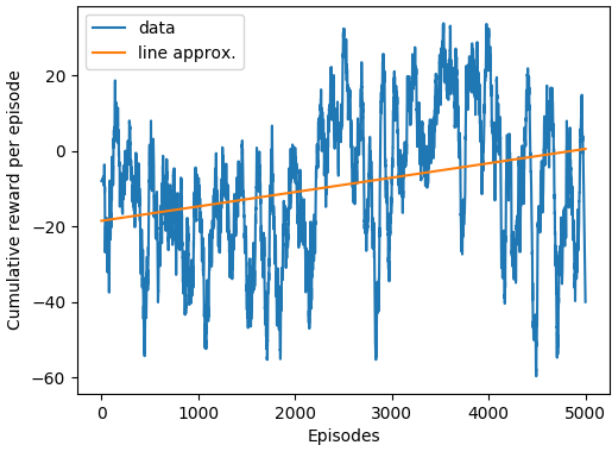

Figure 22: Cumulative rewards for each episode for the DQN using the swing-up reward function (plot smoothed out using a Savitzky-Golay filter)

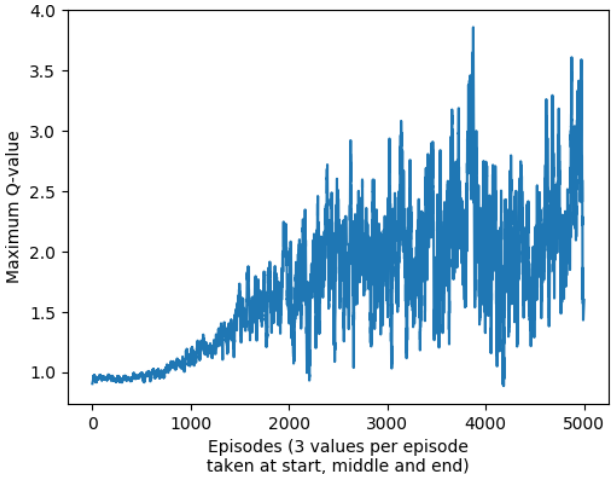

Figure 21: Maximum Q-values for states observed at the start, middle and end of each episode for the DQN using the balancing reward function (plot smoothed out using a Savitzky-Golay filter)

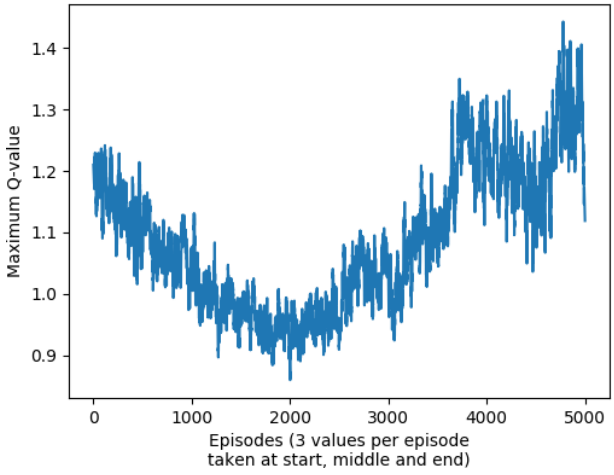

Figure 23: Maximum Q-values for states observed at the start, middle and end of each episode for the DQN using the swing-up reward function (plot smoothed out using a Savitzky-Golay filter)

the history of states over the experiment it was determined that the pendulum is in such a state about 10% of the time. This is a likely contributor to the observed difficulty of converging to an optimal Q function.

The network chooses certain actions and positions preferentially towards the end of the experiment, as evidenced by the data displayed in Figures 4.3 and 4.3. This shows that the network is capable of learning, however the policy that it is learning does not have the desired outcome of swing-up and balancing.

In contrast, the performance of both agents using more specific reward functions is promising. While the increase in cumulative reward per episode is not especially large, there is a clear upward trend in the data. The maximum Q-values also increase over time, as the network learns from positive reinforcement. Apart from the adjusted reward functions, both of these agents also operate on different action spaces. The balancing agent has an action space of the same size but smaller bounds and the swing-up agent uses a smaller action space with the same bounds. The improved performance on their respective tasks may possibly be attributed to more appropriate actions spaces.

6 Conclusion

The studied variation of the classic inverted pendulum problem proved to be more difficult than anticipated. The results show that the controllability of the hardware setup described in the introduction is limited. While the sensor readings from the joints when similar commands are applied only differ little and seem to follow a normal distribution, the pendulums position and velocity are abnormal. Since the simulation is fully deterministic it is controllable and suited to further study the problem. Furthermore, the controllability of real-world system can be improved by changing the hardware. For example, the potentiometer for observing the pendulum position could be replaced by a camera. While this would likely increase the time and processing power, noise from the potentiometer's dead zone would not cause any problems in obtaining the state. Furthermore, attaching better servos more strongly to the case would guarantee that the system doesn't wear out and the servos' moves always apply the same forces to the pendulum.

The results of the performed experiments are not sufficient to conclude that the chosen approach, reinforcement learning, works well on the given problem since performance is fairly poor. However, some of the approaches are promising and with further development may be better suited to solve the given problem. In particular, the training time used for most experiments may be much too short to learn a good policy in such a complex environment. Furthermore, the choice of action space is likely critical for good performance and should be explored before considering other parameters.

The results of experimenting with the swing-up and balancing task separately indicate that using two controllers for the combined task may be preferable to designing just one for both tasks. The performance of this more general controller was even worse than choosing actions randomly. Conversely, the two other controllers performed reasonably well on their respective tasks and improved with training. While the transition between two such controllers may be non-trivial, using a combined controller holds promise and would be an interesting direction for future research.

It is unfortunate that the selected reinforcement learning did not succeed in controlling the pendulum as desired. However, this variation on the classical inverted pendulum problem is interesting and also largely unexplored. The results presented here show that more research is required to control the described system, but there is promise in doing so.

References

[1] Charles W Anderson. "Learning to control an inverted pendulum using neural networks". In: *IEEE Control Systems Magazine* 9.3 (1989), pp. 31–37.

[2] MJ Anderson and WJ Grantham. "Lyapunov optimal feedback control of a nonlinear inverted pendulum". In: *Journal of dynamic systems, measurement, and control* 111.4 (1989), pp. 554–558.

[3] Karl Johan Åström and Katsuhisa Furuta. "Swinging up a pendulum by energy control". In: *Automatica* 36.2 (2000), pp. 287–295.

[4] Alan Bradshaw and Jindi Shao. "Swing-up control of inverted pendulum systems". In: *robotica* 14.4 (1996), pp. 397–405.

[5] Marvin Bugeja. "Non-linear swing-up and stabilizing control of an inverted pendulum system". In: *EUROCON 2003. Computer as a Tool. The IEEE Region 8*. Vol. 2. IEEE. 2003, pp. 437–441.

[6] Carlos Aguilar Ibanez, O Gutierrez Frias, and M Suarez Castanon. "Lyapunov-based controller for the inverted pendulum cart system". In: *Nonlinear Dynamics* 40.4 (2005), pp. 367–374.

[7] Akhil Jose et al. "Performance study of PID controller and LQR technique for inverted pendulum". In: *World Journal of Engineering and Technology* 3.02 (2015), p. 76.

[8] Sung S Kim, Geun H Lee, and Seul Jung. "Implementation of a Neural Network Controller on a DSP for Controlling an Inverted Pendulum System on an XY Plane". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 5439–5443.

[9] K Udhayakumar1 P Lakshmi. "Design of robust energy control for cart-inverted pendulum". In: *International Journal of Engineering and Technology* 4.1 (2007), pp. 66–76.

[10] Douglas J Leith and William E Leithead. "Survey of gain-scheduling analysis and design". In: *International journal of control* 73.11 (2000), pp. 1001–1025.

[11] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[12] Mario E Magana and Frank Holzapfel. "Fuzzy-logic control of an inverted pendulum with vision feedback". In: *IEEE transactions on education* 41.2 (1998), pp. 165–170.

[13] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[14] Erik Neumann. *Movable Pendulum.* URL: `https://www.myphysicslab.com/pendulum/moveable-pendulum-en.html` (visited on 01/01/2018).

[15] Lal Bahadur Prasad, Barjeev Tyagi, and Hari Om Gupta. "Optimal control of nonlinear inverted pendulum system using PID controller and LQR: performance analysis without and with disturbance input". In: *International Journal of Automation and Computing* 11.6 (2014), pp. 661–670.

[16] Lasse Scherffig. "Reinforcement learning in motor control". In: (2002).

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* Vol. 1. 1. MIT press Cambridge, 1998.

[18] Jianqiang Yi and Naoyoshi Yubazaki. "Stabilization fuzzy control of inverted pendulum systems". In: *Artificial Intelligence in Engineering* 14.2 (2000), pp. 153–163.

Appendices

Observations	**Values used**
$(\theta_p, \dot{\theta}_p, \theta_1, \theta_2)$	(20,3,7,7)
	(20,3,9,9)
	(20,3,11,11)
	(16,3,7,7)
	(16,3,9,9)
	(16,3,11,11)
learning rate, α	From 0.8 to 0.1
exploration rate, ϵ	From 1 to 0.05
discount rate, γ	0.95

Table 4: Q-table parameter values used in experiments

DQN parameters	**Possible values**	**Amount of possible values**
state space dimension	6	1
action space dimension	81	1
number of episodes	[1000, 2000, 4000]	3
number of steps per episode	[50, 100, 200]	3
memory size	[100, 200, 400, 600, 800, 1000]	6
batch size	$[5\%, 10\%, 25\%, 50\%, 100\%]$ of memory size	5
epsilon start	[1, 0.5]	2
epsilon finish	[0.05, 0.01]	2
epsilon decay steps	[10%, 50%, 90%]	3
	percentage of number of episodes	
discount rate, γ	[0.9999, 0,999, 0.99, 0.9, 0.5]	5
learning rate, α	[0.1, 0.01, 0.001, 0.0001, 0.00001]	5
amount layers	[1, 2, 3]	3
amount nodes per layer	[10, 20, 50, 100, 200]	5

Table 5: DQN parameter values which where experimented with during the hyperparameter search

DQN parameters	Values chosen
state space dimension	61
action space dimension	81 and 25, depending on task
number of episodes	5000
number of steps per episode	200
memory size	10000
batch size	64
epsilon start	1
epsilon finish	0.05
epsilon decay steps	4500
discount rate, γ	0,995
learning rate, α	0.0001
amount layers	2
amount nodes per layer	20

Table 6: The hyperparameters chosen for experimenting with the performance on the 3 separate tasks

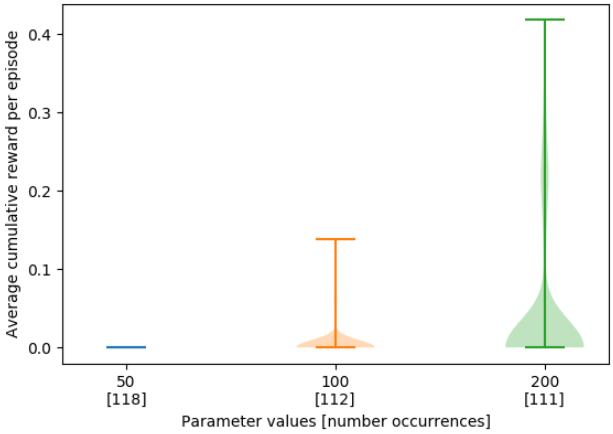

Figure 24: Comparing performances between different number of steps for the DQN

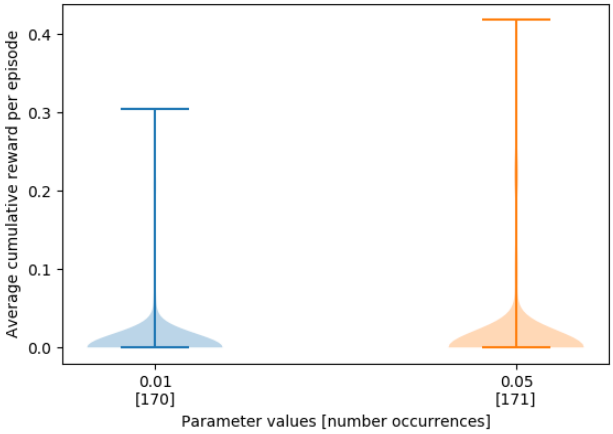

Figure 25: Comparing performance between different minimum epsilon values for the DQN

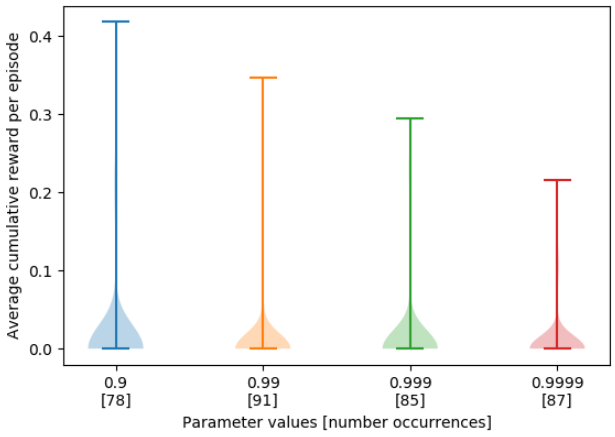

Figure 26: Comparing performance between different discount rates for the DQN

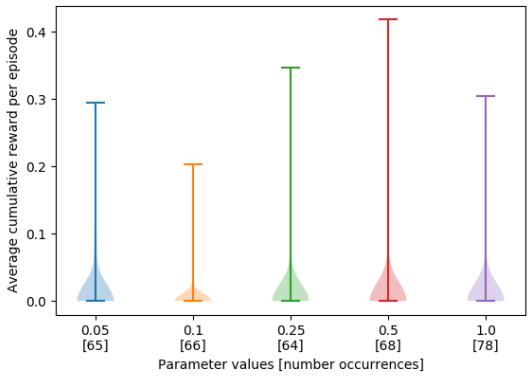

Figure 27: Comparing performance between different amounts of batch sizes for the DQN

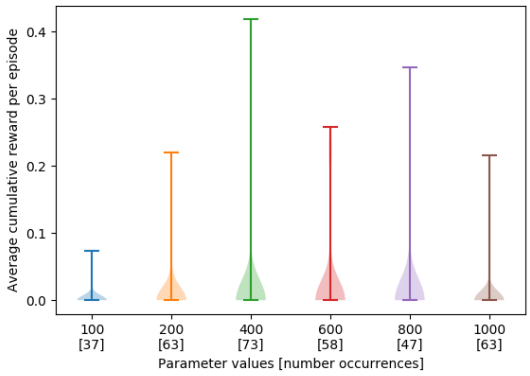

Figure 28: Comparing performance between different amounts of memory sizes for the DQN

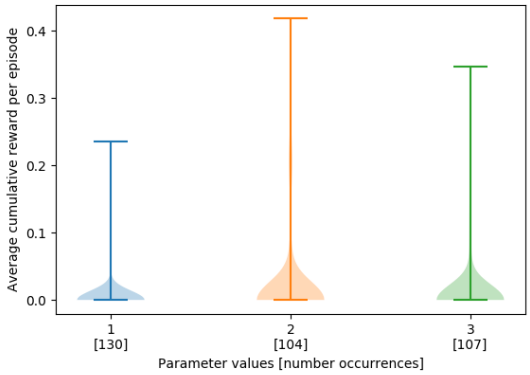

Figure 29: Comparing performance between different amount of layers for the DQN