TECHNISCHE
UNIVERSITÄT
WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS-
& REGELUNGSTECHNIK

Reinforcement Learning: Case Studies in Single Pendulum Control and Timber Crane Grasping

DIPLOMA THESIS

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Univ.-Prof. Dr. techn. Andreas Kugi
Dr. techn. Minh Nhat Vu
Dr. techn. Tobias Glück

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Florian Lechner
Matriculation number 11704942

Vienna, November 2023

Complex Dynamical Systems Group
A-1040 Wien, Gußhausstr. 27–29, Internet: https://www.acin.tuwien.ac.at

Preamble

This thesis was created at the Automation and Control Institute of the University of Technology Vienna in collaboration with the Austrian Institute of Technology.

The topic of forestry cranes was chosen because of my background with those machines in our family forests. While I was fond of forestry cranes from the beginning, I started the thesis with a natural scepticism towards machine learning, as it can be expected for algorithms learning correlations without a clear explanation why they learned what they learned. Only later I could see the brilliance of different reinforcement learning algorithms.

I would like to express my gratitude to Univ.-Prof. Dr. techn. Andreas Kugi for the supervision of my thesis. This endeavor would not have been possible without Dr. techn. Tobias Glück, responsible for the cooperation with the Austrian Institute of Technology, who had always a plan on how to progress in the topic of reinforcement learning, starting at the very basics. Additionally I am deeply indebted to Dr. techn. Minh Nhat Vu for his supervision and his advice on scientific writing. I am also thankful to Dipl.-Ing. Florian Grander und Dipl.-Ing. Lukas Flatz for the help with the physics engine.

Vienna, November 2023

Abstract

This thesis investigates the application of reinforcement learning (RL), a machine learning paradigm that learns from experience, to automate a specific task in timber harvesting - grasping a log with a grapple. RL's potential to learn optimal solutions without explicit models makes RL a promising method for forest machine automation. The thesis comprises three key chapters: an overview of the theoretical foundations of RL, an investigation of the application of RL to an inverted pendulum system on a cart, and a specific investigation of the automation of forestry cranes. In the first chapter, the general introduction to RL including popular approaches is presented. Then, the inverted pendulum on a cart system is used to test RL's capabilities and demonstrate RL's adaptability in different problem domains. The final chapter designs a learning-based control strategy for forestry cranes. It focuses on efficiently picking up a log of different diameters from a random initial configuration and aims to simplify the tasks for operators among the challenges posed by the forest environment.

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Anwendung von bestärkendem Lernen (Englisch: Reinforcement Learning), einer Disziplin des maschinellen Lernens, die von Ehrfahrung lernt, in der Automatisierung der Holzernte - das Greifen eines Stammes mit einer Zange. Das Potential des bestärkenden Lernens, eine optimale Lösung zu generieren ohne ein explizites Modell zu verwenden, machen das bestärkende Lernen zu einer vielversprechenden Methode für die Automatisierung von Holzerntemaschinen. Die Arbeit ist in drei Kapitel aufgeteilt: ein Überblick der theoretischen Grundlagen des bestärkenden Lernens, eine Untersuchung der Anwendung des bestärkenden Lernens auf ein inverses Pendel auf einem Wagen und der Untersuchung der Automatisierung eines Forstkrans. Im ersten Kapitel wird das Problem des bestärkenden Lernens formal beschrieben und Algorithmen zu dessen Lösung präsentiert. Das inverse Pendel-Wagen System wird verwendet, um die Fähigkeiten des bestärkenden Lernens auszutesten und die Anpassungsmöglichkeiten an unterschiedliche Arten von Problemen zu demonstrieren. Das finale Kapitel handelt vom Design einer Regelungsstrategie für Forstkräne. Der Fokus liegt dabei auf dem effizienten Aufsammeln von Baumstämmen mit unterschiedlichen Durchmessern und aus zufälligen Positionen und zielt darauf ab, die Aufgaben der Maschinisten im Wald zu vereinfachen.

Contents

List of Figures

List of Tables

1 Introduction

Wood, a renewable resource, is one of mankind's most valuable commodities. While the importance of this resource has not changed over time, the methods of timber harvesting have changed considerably in recent decades. Until the 1950s, trees were felled using pull saws, and pulling the trunks with horses out of the forests was common practice in Austria. Then tractors began to replace horses and the introduction of the chainsaw made the work of the loggers easier. Nowadays, two types of machines that became popular in the 1990s dominate the wood harvesting process [1]. The first machine is the harvester. This is a motorized vehicle with a chainsaw unit mounted on a hydraulic crane. This enables it to chop down trees, limb them, and cut the trunks into transportable sizes. The resulting logs can then be transported by the second machine, the forwarder. The forwarder is a type of truck that is specialized for the rough terrain in the forests and is equipped with a grapple mounted on the crane that picks up the logs and loads them onto its loading platform. Its job is to take the logs from the forest to the roads, where regular trucks equipped with a similar crane can load the logs and transport them to their destination.

The use of these sophisticated machines significantly reduces the amount of work involved in timber harvesting. However, operating these machines is physically demanding and requires highly skilled operators. Therefore, full or even partial automation of these machines could simplify the tasks for the operators. With the exception of the hydraulics, the cranes mounted on top of these machines are very similar to industrial robots, which are usually fully automated. The difference that makes the automation of forestry machines a challenge is that they do not operate in a well-defined environment but in the harsh environment of a forest. Several attempts to automatically control forestry cranes have been investigated, ranging from model-based control to model-free data-driven methods. This thesis explores how reinforcement learning (RL) can be used to solve a skilled task in the timber harvesting process. This task involves grasping a tree log with the grapple, which is relevant for both forwarders and log carriers.

RL is a domain of machine learning. Compared to other domains such as supervised learning, where labeled data is used, or unsupervised learning, where unlabeled data is used, reinforcement learning learns from experience. The RL agent learns from interactions with the actual environment for which it is being trained, or with a representation, such as a simulation of the environment. For control tasks, RL can find optimal solutions without providing exemplary solutions, as is necessary for supervised learning. This is achieved since optimality for RL is defined by a reward function and the learning process aims to maximize this reward. Conceptually, this is similar to optimal control, where a cost function is minimized. The difference is that RL can work completely model-free by collecting all the necessary information through interaction.

This thesis consists of three main parts:

- The first part is a brief introduction to the theoretical foundations of RL, including basic algorithms as well as state-of-the-art RL algorithms.
- In the second part, the capabilities of RL are investigated on a well-known system, the inverted pendulum on a cart, to gather information about different algorithms and possible reward function structures. It also investigated how to combine several problems to solve them at once with RL, showing that exploring the full state space can be advantageous for the resulting control strategy. The resulting control strategy is demonstrated on a real system to show the transfer from simulation to reality.
- The last chapter focuses on the automation of a forestry crane. The objective is to develop a solution that enables a crane to effectively pick up a tree log of different diameters from any initial configuration.

2 Reinforcement Learning

Lewis and Vrabie gave a description of reinforcement learning (RL):

> "Reinforcement learning refers to an actor or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions. This is based on evaluative information from the environment and could be called *action-based learning*. RL implies a cause and effect relationship between actions and reward or punishment. It implies goal directed behavior at least insofar as the agent has an understanding of reward versus lack of reward or punishment" [2, p. 33].

Since we are in a stochastic setting, it is important to understand the *basics of probability theory*. We use sans-serif letters such as $\mathsf{x}, \boldsymbol{\mathsf{x}}$, and $\boldsymbol{\mathsf{X}}$ to represent *random variables* (scalars, vectors and matrices) and serif letters such as $x, \mathbf{x}$, and $\mathbf{X}$ to represent the corresponding *deterministic variables* or events.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is used to formally describe the reinforcement learning (RL) problem. In a MDP, there are two elements that interact with each other, as shown in Figure 2.1. The agent makes a decision and performs an action u, based on the state x and reward signal r received from the environment. The environment responds to this action in accordance with its internal dynamics by returning to the state to which it has been changed and by giving the reward.

The response of the environment can then be used by the agent to improve future decisions and maximize the reward. A MDP is said to be finite [3, p. 48] if the set of states $\mathcal{X}$, the set of actions $\mathcal{U}$, and the set of rewards $\mathcal{R}$ are all finite.

This interaction results in a sequence of states

$$\boldsymbol{\mathsf{X}}_{[0:N]} = \{\boldsymbol{\mathsf{x}}_0, ..., \boldsymbol{\mathsf{x}}_N\} \tag{2.1}$$

and a sequence of actions

$$\boldsymbol{\mathsf{U}}_{[0:N-1]} = \{\boldsymbol{\mathsf{u}}_0, ..., \boldsymbol{\mathsf{u}}_{N-1}\}\ , \tag{2.2}$$

where N is the horizon length of an episode. Together they form the trajectory (also called an episode).

$$\boldsymbol{\tau} = \{\boldsymbol{\mathsf{X}}_{[0:N]}, \boldsymbol{\mathsf{U}}_{[0:N-1]}\}\ . \tag{2.3}$$

There are two settings for learning and optimization:

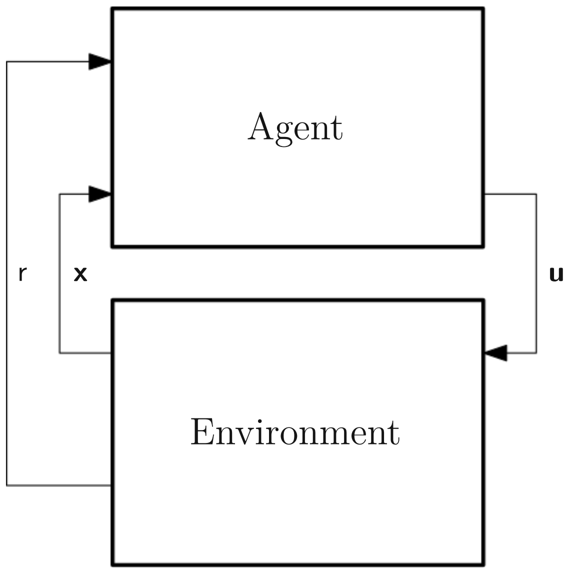


Figure 2.1: Interaction of agent and Environment in a Markov decision process.

- The *episodic/sequential setting*, where the experience is broken up into a series of episodes/sequences. It is our goal to maximize the cumulative reward within a single episode.
- The *continuing setting*, where the task doesn't have clear episode boundaries. It is our goal to maximize the cumulative reward over an infinite or very long time horizon, i.e. $N = \infty$.

To measure the quality of a trajectory, the environment creates a sequence of rewards for every time step

$$\mathsf{r}_{[0:N]} = \{\mathsf{r}_0, ..., \mathsf{r}_N\} \ . \tag{2.4}$$

2.1.1 The Environment

The environment can either be deterministic or stochastic. In the deterministic case, its dynamics can be described as

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, ..., N-1 \ . \tag{2.5}$$

While in control theory stochastic systems are handled by introducing a stochastic variable w_k to the right side of equation (2.5), in RL, stochastic systems are described with a conditional probability distribution, where the next state is sampled depending on the previous state and action according to

$$\mathbf{x}_{k+1} \sim p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, ..., N-1 \ . \tag{2.6}$$

For the latter case also the first state $\mathbf{x}_0$ of an episode is sampled from a distribution as well with

$$\mathbf{x}_0 \sim p_0(\mathbf{x}_0) \ , \tag{2.7}$$

and so are the rewards

$$\begin{aligned} \mathsf{r}_k &\sim p_r(r_k|\mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, ..., N-1 \\ \mathsf{r}_N &\sim p_r(r_N|\mathbf{x}_N) \ . \end{aligned} \tag{2.8}$$

With this construction, the system defined by (2.6) satisfies the Markov property, stating that $\mathbf{x}_{k+1}$ depends only on quantities of the previous time step k, i.e.

$$p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) = p(\mathbf{x}_{k+1}|\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, ..., N-1 \ . \tag{2.9}$$

Using those probability distributions, the probability of observing an N-step trajectory $\boldsymbol{\tau}$ under a policy π is

$$p(\boldsymbol{\tau}) = p_0(\mathbf{x}_0) \prod_{k=0}^{N-1} p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \ . \tag{2.10}$$

2.1.2 The Agent

The law that the agent follows to choose its actions is called the policy and can be either deterministic or stochastic. In the deterministic case, the action can be calculated from a function with

$$\mathbf{u}_k = \pi(\mathbf{x}_k) \ , \tag{2.11}$$

for the stochastic case again a probability distribution

$$\mathbf{u}_k \sim \pi(\mathbf{u}_k|\mathbf{x}_k) \tag{2.12}$$

is used, see [4, p. 13]. For a finite MDP, the policy can be organized as lookup tables.

With (2.10) and (2.12), the probability of observing an N-step trajectory $\boldsymbol{\tau}$ under a policy π reads as

$$p_\pi(\boldsymbol{\tau}) = p(\mathbf{x}_0) \prod_{k=0}^{N-1} p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k|\mathbf{x}_k) \ . \tag{2.13}$$

The agent employs an environmental state to determine actions through its policy, concurrently utilizing the reward signal to enhance the policy. For N-step trajectories, the return is the cumulative weighted reward from step k to N, represented by

$$\mathsf{R}_k(\boldsymbol{\tau}) = \sum_{k'=k}^{N} \gamma^{k'-k} \mathsf{r}_{k'} \ . \tag{2.14}$$

The discounting factor $\gamma \in (0, 1)$ is used. The smaller γ is chosen, the lower future rewards in r_k will be weighted, thereby giving more significance to immediate rewards. Conversely, rewards received for high occupancies in the future are weighted more heavily. Thus, this parameter essentially represents the distance of foresight into the future.

Since a stochastic environment and policy can be used, improving the policy is equivalent to finding the policy that maximizes the expected return by solving the optimization problem.

$$\pi^* = \arg\max_{\pi} E_\pi\{\mathsf{R}(\boldsymbol{\tau})\} \;, \tag{2.15}$$

where $E_\pi\{\mathsf{R}(\boldsymbol{\tau})\}$ describes the expected return, given that all the actions are sampled from policy π. The resulting policy π^* is an optimal policy.

Figure 2.2 shows in detail how the agent and environment interact with each other. In the environment, the conditional probabilities for the next state $\mathbf{x}_{k+1}$ and the reward r_k define how the environment reacts to the action $\mathbf{u}_k$ provided by the agent based on the policy π. The learner tries to solve the RL problem by maximizing (2.15).

Once an optimal policy is found, only the inner control loop (dotted square labeled in Figure 2.2) is needed for deployment. To put this in the context of control theory, the policy then acts as a state feedback controller to stabilize the dynamics of the environment.

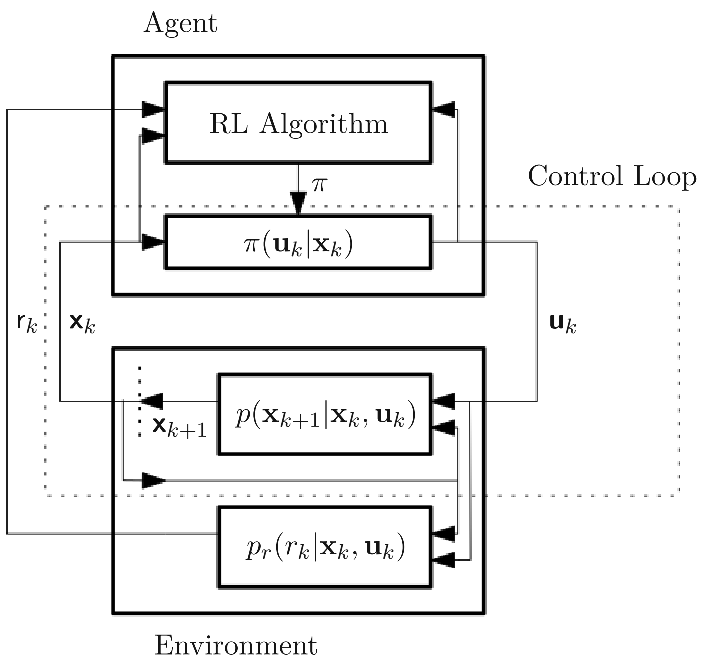

Agent
RL Algorithm
π
Control Loop
$\pi(\mathbf{u}_k|\mathbf{x}_k)$
r_k
$\mathbf{x}_k$
$\mathbf{u}_k$
$\mathbf{x}_{k+1}$
$p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)$
$p_r(r_k|\mathbf{x}_k, \mathbf{u}_k)$
Environment

Figure 2.2: Agent-environment interaction model.

2.1.3 Value Functions

Because of the Markov property, the future after visiting a particular state depends only on that particular state. Therefore, the probabilities of the trajectories are the same for each visit to a state. This causes the expected return after each visit to a state to be constant, given the same policy.

The mapping of expected returns is called the state value function of the policy π and is defined as

$$V^\pi(\mathbf{x}) = E_\pi\{\mathsf{R}_k(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}\}\ . \tag{2.16}$$

The same assumption holds true in addition to the starting state, the first action is fixed and afterwards policy π is followed. This gives the action value function

$$Q^\pi(\mathbf{x},\mathbf{u}) = E_\pi\{\mathsf{R}_k(\tau)|\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}\ , \tag{2.17}$$

see [3, p. 58].

We can infer the state value function from the action value function via

$$V^\pi(\mathbf{x}) = \underset{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}{E}\{Q^\pi(\mathbf{x},\mathbf{u})\}\ . \tag{2.18}$$

In contrast, the inverse relation requires knowledge of the environment, since

$$Q^\pi(\mathbf{x},\mathbf{u}) = \underset{\mathsf{r}\sim p_r(r|\mathbf{x},\mathbf{u})}{E}\{r\} + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{V^\pi(\mathbf{x}')\}\ . \tag{2.19}$$

The latter relations are known as Bellman expectation equations, which are derived next.

2.1.4 Bellman Expectation Equations

By splitting the return in the definitions of the value functions (2.16) and (2.17) into the immediate reward and the return of the following state, the Bellman expectation equations can be derived. For the state value function $V^\pi(\mathbf{x})$ this derivation reads as

$$\begin{aligned}
V^\pi(\mathbf{x}) &= E_\pi\{\mathsf{R}_k(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}\}\\
&= E_\pi\{\mathsf{r}_k + \gamma\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}\}\\
&= \underset{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}{E}\{\mathsf{r}_k|\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} + \gamma E_\pi\{\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}\}\\
&= \underset{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}{E}\{\underset{\mathsf{r}\sim p_r(r|\mathbf{x},\mathbf{u})}{E}\{r\} + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{E_\pi\{\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_{k+1} = \mathbf{x}'\}\}\}\\
&= \underset{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}{E}\{\underset{\mathsf{r}\sim p_r(r|\mathbf{x},\mathbf{u})}{E}\{r\} + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{V^\pi(\mathbf{x}')\}\}
\end{aligned} \tag{2.20}$$

and for $Q^\pi(\mathbf{x},\mathbf{u})$ it reads as

$$\begin{aligned}
Q^\pi(\mathbf{x},\mathbf{u}) &= E_\pi\{\mathsf{R}_k(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}\\
&= E_\pi\{\mathsf{r}_k + \gamma\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}\\
&= E\{\mathsf{r}_k, |\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} + E_\pi\{\gamma\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}\\
&= \underset{\mathsf{r}\sim p_r(r|\mathbf{x},\mathbf{u})}{E}\{r\} + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{\underset{\mathbf{u}'\sim\pi(\mathbf{u}'|\mathbf{x}')}{E}\{E_\pi\{\mathsf{R}_{k+1}(\boldsymbol{\tau})|\mathbf{x}_{k+1} = \mathbf{x}', \mathbf{u}_{k+1} = \mathbf{u}'\}\}\}\\
&= \underset{\mathsf{r}\sim p_r(r|\mathbf{x},\mathbf{u})}{E}\{r\} + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{\underset{\mathbf{u}'\sim\pi(\mathbf{u}'|\mathbf{x}')}{E}\{Q^\pi(\mathbf{x}',\mathbf{u}')\}\}\ ,
\end{aligned} \tag{2.21}$$

see [3, p. 59].

2.1.5 Bellman Optimality Equations

If the inequality

$$V^{\pi'}(\mathbf{x}) \geq V^{\pi}(\mathbf{x}) \quad \forall \mathbf{x} \; , \tag{2.22}$$

holds true, then π' is a better policy than π. This definition implies that an optimal policy has the highest values for all states and that only one unique value function can be optimal. For state values the optimal value function is defined as

$$V^*(\mathbf{x}) = \max_{\pi} V^{\pi}(\mathbf{x}) \tag{2.23}$$

and for action values, it is defined as

$$Q^*(\mathbf{x}, \mathbf{u}) = \max_{\pi} Q^{\pi}(\mathbf{x}, \mathbf{u}) \; . \tag{2.24}$$

Using (2.20) and (2.23) the Bellman equation for $V^*(\mathbf{x})$ can be derived with

$$\begin{aligned} V^*(\mathbf{x}) &= \max_{\pi} \underset{\mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x})}{E} \{ r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ V^{\pi}(\mathbf{x}') \} \} \\ &= \max_{\mathbf{u}} \{ r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ \max_{\pi} V^{\pi}(\mathbf{x}') \} \} \\ &= \max_{\mathbf{u}} \{ r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ V^*(\mathbf{x}') \} \} \end{aligned} \quad . \tag{2.25}$$

Similarly, this derivation can be done for the action value function with

$$\begin{aligned} Q^{\pi}(\mathbf{x}, \mathbf{u}) &= \max_{\pi} \{ r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ \underset{\mathbf{u}' \sim \pi(\mathbf{u}'|\mathbf{x}')}{E} \{ Q^{\pi}(\mathbf{x}', \mathbf{u}') \} \} \} \\ &= r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ \max_{\mathbf{u}'} \{ \max_{\pi} Q^{\pi}(\mathbf{x}', \mathbf{u}') \} \} \\ &= r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ \max_{\mathbf{u}'} Q^*(\mathbf{x}', \mathbf{u}') \} \; , \end{aligned} \tag{2.26}$$

using (2.21) and (2.24), see [3, p. 62f]. Those equations for the optimal value functions are called Bellman Optimality Equations.

2.2 Value Based Methods

Under the assumption that the optimal action-value function is known, an optimal policy can always be derived by

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u}) \; . \tag{2.27}$$

This shows that it is sufficient to calculate value functions and create policies for the concept of value-based methods.

2.2.1 Generalized Policy Iteration

The term generalized policy iteration is used for the repeated process of evaluating a policy to a certain extent and then improving the policy based on this evaluation of the old policy [3, p. 86].

Equation (2.27) shows how to get the optimal policy from an optimal value function, but the same method can be applied to the value function of the current policy. Applying

$$\pi'(\mathbf{x}) = \arg\max_{\mathbf{u}} Q^{\pi}(\mathbf{x}, \mathbf{u}) \;, \tag{2.28}$$

always chooses the action with the highest value according to the old policy. The policy is made "greedy" with respect to the value function. This method always improves the policy, with the exception that the policy is already optimal. Then the new policy is also optimal according to (2.27) and its value function satisfies the Bellman Optimality Equation. This method is called policy improvement and can similarly be done with the state value function, but with the downside that knowledge from the environment is necessary as (2.28) with the conversion (2.19) shows [3, p. 76-79].

The algorithm for the policy evaluation part can be freely chosen. Model-based dynamic programming or model-free Monte Carlo methods are just some examples.

2.2.2 Dynamic Programming

Are the dynamics of the environment fully known, dynamic programming can be used to evaluate policies, by turning Bellman equations into update rules. Combining this with the idea of generalized policy iteration results in methods to calculate an optimal policy [3, p. 73f].

Policy Iteration

Using the Bellman expectation equations as an update rule for the action value function reads as

$$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow r(\mathbf{x}, \mathbf{u}) + \gamma \underset{\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E} \{ \underset{\mathbf{u}' \sim \pi(\mathbf{u}'|\mathbf{x})}{E} \{ \hat{Q}^{\pi}(\mathbf{x}', \mathbf{u}') \} \} \;. \tag{2.29}$$

$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$ represents the current estimate of the action value function for following policy π and can be initialized arbitrarily. The calculation of the expectation values is only possible if the environment's dynamics are known. For a finite set of states and actions, the expectations can be calculated with

$$\underset{\mathsf{y} \sim p(y|z)}{E} \{ f(y) \} = \sum_{y} p(y|z) f(y) \;. \tag{2.30}$$

Using this equation to evaluate the policy until convergence and iterate alternately with policy improvement gives policy iteration, see [3, p. 80].

Value Iteration

Value iteration is when policy improvement follows after just one step of the iterative policy evaluation. The so-created update rule

$$\begin{aligned}\hat{Q}^\pi(\mathbf{x},\mathbf{u}) &\leftarrow r(\mathbf{x},\mathbf{u}) + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{\hat{Q}^\pi(\mathbf{x}', \arg\max_{\mathbf{u}'} \hat{Q}^\pi(\mathbf{x}',\mathbf{u}'))\} \\ &\leftarrow r(\mathbf{x},\mathbf{u}) + \gamma \underset{\mathbf{x}'\sim p(\mathbf{x}'|\mathbf{x},\mathbf{u})}{E}\{\max_{\mathbf{u}'} \hat{Q}^\pi(x',u')\}\end{aligned} \tag{2.31}$$

is equal to using the Bellman Optimality Equation as an update [3, p. 82f].

2.2.3 Monte Carlo Methods

If the dynamics of the environment are unknown, Monte Carlo methods can be used for policy evaluation. The principle of these methods is to sample trajectories using the current policy. The value of a state can then be estimated with the mean over the actual returns following visits to this particular state. By the law of large numbers this mean converges to the expectation of the return and therefore to the value [5].

A formal description of this method can be given using n different sampled trajectories $\boldsymbol{\tau}^{(i)}$. Then the action value function can be calculated with

$$Q^\pi(\mathbf{x},\mathbf{u}) = \frac{\sum_{i=1}^n \sum_{k=0}^{N^{(i)}} \mathbb{1}[\mathbf{x}_k^{(i)} = \mathbf{x}, \mathbf{u}_k^{(i)} = \mathbf{u}] \mathsf{R}_k(\boldsymbol{\tau}^{(i)})}{\sum_{i=1}^n \sum_{t=0}^{N^{(i)}} \mathbb{1}[\mathbf{x}_k^{(i)} = \mathbf{x}, \mathbf{u}_k^{(i)} = \mathbf{u}])}\ , \tag{2.32}$$

where the expression $\mathbb{1}[\cdot]$ becomes one if the equations in the brackets are true and is zero otherwise. Using this formula every visit of a state contributes to the mean. therefore this method is called every-visit Monte Carlo. Together with generalized policy iteration, (2.32) can be used to find optimal policies for episodic environments.

2.2.4 Temporal-Difference (TD) Learning

Temporal-difference methods combine sampling transitions like Monte Carlo methods and update value functions based on old estimates as in dynamic programming. The big difference to Monte Carlo methods is that observed transition data $(\mathbf{x}_k, \mathsf{r}_k, \mathbf{x}_{k+1})$ can be instantly used to update the current value function estimate. This is done using a value function approximation table $\hat{V}^\pi(\mathbf{x})$, where the current state value estimates are stored. For the simplest version of TD, the approximation table is then updated for just one state according to

$$\hat{V}^\pi(\mathbf{x}_k) \leftarrow \hat{V}^\pi(\mathbf{x}_k) + \alpha[\mathsf{r}_k + \gamma \hat{V}^\pi(\mathbf{x}_{k+1}) - \hat{V}^\pi(\mathbf{x}_k)]\ . \tag{2.33}$$

The update is towards the TD target $\mathsf{r}_k + \gamma \hat{V}^\pi(\mathbf{x}_{k+1})$, which is an estimation of the return [3, p. 119f].

The learning rate $\alpha \in [0,1]$ specifies the balance between using the old estimate and the new estimate consisting of the sampled reward and the discounted value of the sampled next state. For infinite visits of every state, the approximation table converges to the value function.

SARSA

TD learning can also be used similarly for estimating action-value functions. In this case, the current and following action have to be added to the used data $(\mathbf{x}_k, \mathbf{u}_k, \mathsf{r}_k, \mathbf{x}_{k+1}, \mathbf{u}_{k+1})$. The resulting algorithm updates its estimates with

$$\hat{Q}^\pi(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \hat{Q}^\pi(\mathbf{x}_k, \mathbf{u}_k) + \alpha[\mathsf{r}_k + \gamma\hat{Q}^\pi(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) - \hat{Q}^\pi(\mathbf{x}_k, \mathbf{u}_k)] \tag{2.34}$$

and is named SARSA because of this data sequence used for calculation (state, action, reward, state, action). Using action values allows model-free policy improvement. Therefore SARSA can not only be used for policy evaluation, but for the whole policy iteration, and therefore it can solve MDPs without any knowledge of the environment [3, p. 129].

Q-learning

Another algorithm also using action values is Q-learning [6]. The update rule

$$\hat{Q}^*(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \hat{Q}^*(\mathbf{x}_k, \mathbf{u}_k) + \alpha[\mathsf{r}_k + \gamma \max_{\mathbf{u}} \hat{Q}^*(\mathbf{x}_{k+1}, \mathbf{u}) - \hat{Q}^*(\mathbf{x}_k, \mathbf{u}_k)] \tag{2.35}$$

is similar to that used for SARSA, with the exception, that instead of the estimate of the action value of the occurring state-action pair, the maximum of all possible action value estimates is used. With this construction, Q-learning directly approximates the optimal value function, regardless of the policy used.

2.3 Policy Based Methods

Policy-based methods do not rely on value functions to create policies but calculate policies directly. Therefore the policy is parameterized with a parameter vector $\boldsymbol{\theta}$. The so created policy is denoted $\pi_{\boldsymbol{\theta}}$. To improve the policy gradient ascent,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \ , \tag{2.36}$$

can be used, updating the parameter vector in the direction of the gradient of some objective function $J(\pi_{\boldsymbol{\theta}})$, with the step size parameter α [3, p. 321].

2.3.1 Policy Gradient Theorem

To calculate the gradient, the objective function has to be defined. For this objective function, different definitions can be used. It can be an average reward, an average value, or the value of the starting state [5]. A very compact derivation is possible using the last of those objective functions and considering only episodic tasks.

The value of the starting state is by definition equal to the expected return after the starting state and therefore the expected return of the full trajectory, because the trajectory starts in the starting state. This can be denoted as

$$J(\pi_{\boldsymbol{\theta}}) = \underset{\mathbf{x}_0 \sim p_0(\cdot)}{E}\{V^{\pi_\theta}(\mathbf{x}_0)\} = E_\pi\{\mathsf{R}(\boldsymbol{\tau})\} \ , \tag{2.37}$$

using the probability a trajectory occurs under the parameterized policy

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = p(\mathbf{x}_0) \prod_{k=0}^{N-1} p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k) \ . \tag{2.38}$$

The gradient of the objective function can then be calculated with

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \nabla_{\boldsymbol{\theta}} E_{\pi}\{\mathsf{R}(\boldsymbol{\tau})\} \\ &= \int_{\boldsymbol{\tau}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\mathsf{R}(\boldsymbol{\tau}) \\ &= E_{\pi_{\boldsymbol{\theta}}}\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\mathsf{R}(\boldsymbol{\tau})\} \\ &= E_{\pi_{\boldsymbol{\theta}}}\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)\mathsf{R}(\boldsymbol{\tau})\}\end{aligned} \tag{2.39}$$

using the identity

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \tag{2.40}$$

and the logarithm rules to use the reformulation

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \log p(\mathbf{x}_0) + \sum_{k=0}^{N-1} \log p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) + \log \pi_{\theta}(\mathbf{u}_k|\mathbf{x}_k) \ . \tag{2.41}$$

The parts not dependent on $\boldsymbol{\theta}$ are eliminated by the gradient [7].

According to (2.39) the probability of every action is changed proportional to the return of the whole episode, although actions at the end of an episode cannot influence rewards at the beginning because of causality. Therefore it is reasonable to replace the full return with the partial return $\mathsf{R}_k(\boldsymbol{\tau})$ starting at step k. The policy gradient then changes to

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = E_{\pi_{\boldsymbol{\theta}}}\{\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)\mathsf{R}_k(\boldsymbol{\tau})\} \ , \tag{2.42}$$

and can be used in this form for the Reinforce algorithm [8].

2.3.2 Reinforce

The Reinforce algorithm [9] uses Monte Carlo sampling to create an estimate of the policy gradient. Using only the experience from the latest sampled trajectory $\boldsymbol{\tau}$ gives an unbiased sample of (2.42). Therefore

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \approx \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)\mathsf{R}_k(\boldsymbol{\tau}) \tag{2.43}$$

is an approximation of the policy gradient. This approximation can be used for one gradient ascent update of the parameter vector $\boldsymbol{\theta}$, then the new policy $\pi_{\boldsymbol{\theta}}$ is used to gather the next trajectory. Reinforce can only use data gathered by the same policy, that the improvement is done for.

Although (2.43) gives an unbiased estimate of the policy gradient, the estimate has high variance, because the return can change heavily from one episode to another. This variance can be reduced by introducing a baseline $b(\mathbf{x}_k)$ that can depend on the state but not on the action. Subtracting that baseline from the return gives the new policy gradient estimate

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \approx \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)[\mathsf{R}_k(\boldsymbol{\tau}) - b(\mathbf{x}_k)] \ . \tag{2.44}$$

The obvious choice for a state-dependent function is the state value function $V^{\pi}(\mathbf{x}_k)$, but a constant baseline like the mean return over the actual trajectory is also a viable option for a baseline [10, p. 32f].

2.4 Actor Critic Methods

If the principles of value and policy-based methods are combined the resulting algorithms are classified as Actor-Critic methods. Similarly to Reinforce, the policy gradient is used to update a parameterized policy, but instead of updating proportional to the return of the sampled episode, the critic is used for this update. One option for the critic is to learn the action value function. Plugging this into (2.44) and using the state value function as a baseline gives

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \approx \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k) A^{\pi}(\mathbf{x}_k, \mathbf{u}_k) \ . \tag{2.45}$$

as policy gradient to update the actor, with

$$A^{\pi}(\mathbf{x}_k, \mathbf{u}_k) = Q^{\pi}(\mathbf{x}_k, \mathbf{u}_k) - V^{\pi}(\mathbf{x}_k) \ . \tag{2.46}$$

as the advantage function [10, p. 135].

2.4.1 Proximal Policy Optimization (PPO)

Gradient ascent updates the parameter vector $\boldsymbol{\theta}$. Therefore the step size parameter α influences the amount $\boldsymbol{\theta}$ changes. However, the subject of interest is the resulting policy $\pi_{\boldsymbol{\theta}}$, and the amount the policy changes is not proportional to the change in $\boldsymbol{\theta}$. Therefore, it is possible that even a small step size parameter causes a big drop in the performance of the new policy because it has changed too much compared to the old policy.

Trust region policy optimization (TRPO) [11] is a method to ensure the new policy changes are not too much. Therefore a modified objective is introduced. Instead of generating the parameter vector from the maximization problem

$$\arg\max_{\boldsymbol{\theta}} E_{\pi_{\boldsymbol{\theta}}}\{\log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k) A^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)\} \ , \tag{2.47}$$

as the gradient ascent algorithm seeks to do with the gradient approximation (2.46), the new optimization problem reads as

$$\begin{aligned} &\arg\max_{\boldsymbol{\theta}} \, E_{\pi_{\boldsymbol{\theta}}}\left\{ A^{\pi_{\boldsymbol{\theta}_{old}}}(\mathbf{x}_k, \mathbf{u}_k) \frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{u}_k|\mathbf{x}_k)} \right\} \\ &\text{s.t.} E_{\pi_{\boldsymbol{\theta}}}\{D_{KL}(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)||\pi_{\boldsymbol{\theta}_{old}}(\mathbf{u}_k|\mathbf{x}_k))\} \leq \delta \ , \end{aligned} \tag{2.48}$$

where $\boldsymbol{\theta}_{old}$ is the parameter before the optimization step. The Kullback-Leibler divergence,

$$D_{KL}(p(y)||q(y)) = \underset{y \sim p(y)}{E}\left\{ \log \frac{p(y)}{q(y)} \right\} \ , \tag{2.49}$$

gives a measure of the difference between the new and the old policy, so the amount the new policy deviates from the old one can be bound with the parameter δ.

Proximal policy optimization (PPO) [12] introduces two methods to efficiently solve the TRPO problem. The first one is to reformulate the constraint into a penalty and the second one uses a clipped objective function. Because the second version is superior in terms of simplicity and performance the penalty version is neglected here. Maximizing the clipped objective function reads as

$$\arg\max_{\boldsymbol{\theta}} E_{\pi_{\boldsymbol{\theta}}}\{\min(A^{\pi_{\boldsymbol{\theta}_{old}}}(\mathbf{x}_k, \mathbf{u}_k)C, A^{\pi_{\boldsymbol{\theta}_{old}}}(\mathbf{x}_k, \mathbf{u}_k)C_{clip})\} \ . \tag{2.50}$$

Using the abbreviation

$$C = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{u}_k|\mathbf{x}_k)} \ , \tag{2.51}$$

the first part under the minimization is the TRPO objective. In the second part, the difference is that the multiplication is done with the clipped ratio

$$C_{clip} = \text{clip}\left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{u}_k|\mathbf{x}_k)}, 1-\epsilon, 1+\epsilon \right) \ , \tag{2.52}$$

that bounds the probability ratio C in the neighborhood around 1 specified by ϵ. Depending on the sign of $A^{\pi_{\boldsymbol{\theta}_{old}}}$, the minimization uses either $1-\epsilon$ as the lower bound or $1+\epsilon$ as an upper bound for the ratio C in the objective function. Keeping the ratio C around 1 is equal to a small change from the old to the new probability and therefore guarantees small changes in the policy.

In this form, the PPO algorithm is simple to use with just a few parameters to tune. It generates good results on different sorts of problems and the sensitivity to its parameters is low compared to other algorithms. Therefore, it is currently a state-of-the-art algorithm in RL and is one of the used algorithms in this thesis.

3 RL Application: Control Single Inverted Pendulum on Cart

For several years, the inverted pendulum has been a popular benchmark for research in nonlinear control and reinforcement learning, see, e.g., [13, 14]. Inverted pendulums with increasingly complex kinematics were published, such as the single pendulum [15], the triple pendulum on a cart [16], and the spherical pendulum [17]. Although it features a simple structure, the pendulum on a cart is an underactuated nonlinear system. In addition, it has a non stable equilibrium point and shows non-minimum phase behavior [17]. Thus, it represents a challenging task in nonlinear control.

Recently in the field of RL, scholars have been trying to close the gap between learning in simulation and application in the real world. Of course, the inverted pendulum remains a common benchmark. In [18], the swing-up and stabilizing of an inverted pendulum on a cart with Deep Deterministic Policy Gradient (DDPG) [19] are investigated. It is shown that the learned policy is robust and can be easily adapted to the changed environment by retraining. An integration between an RL approach and classical control is presented in [20]. Therein, the swing-up action is performed with Deep Q-Learning [21] and the stabilization is controlled by a PID controller. This chapter is primarily concerned with designing a policy that can

- swing-up or swing-down the inverted pendulum on a cart,
- do stabilization,
- and move the cart to a pre-defined position,

at the same time.

Simulation and experimental results show the efficiency of the trained RL policy.

3.1 Modelling

The schematic of the considered inverted pendulum on a cart is shown in Figure 3.1. This system is identical to the real-world system presented in the previous thesis [22]. The system consists of two degrees of freedom, $\mathbf{q}^{\mathrm{T}} = [x, \theta]$. The linear joint x moves the cart and is actuated by a motor. The rotary joint θ, where the pendulum is mounted onto the cart, is not actuated. To move the cart, a force is applied by the motor. Since a cascaded PI-controller [22] is designed for the considered system, the acceleration u of the actuated joint is considered to be the control input instead of the force action on the cart. Using the Euler-Lagrange formalism [23], the equations of motion are in the form

$$\begin{bmatrix} \dot{x} \\ \dot{v} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v \\ 0 \\ \omega \\ \frac{gma\sin(\theta)-d\omega}{a^2m+J} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ \frac{ma\cos(\theta)}{a^2m+J} \end{bmatrix} u \ , \tag{3.1}$$

x and v are the cart's position and velocity, respectively. θ and ω are the pendulum's angle and angular velocity, respectively. J, m, a are the moment of inertia, the mass of the pendulum, and the distance from the hinge joint to the pendulum's center of mass (CoM). More details on this setup can be obtained in [22]. Parameters of this system are presented in Table 3.1

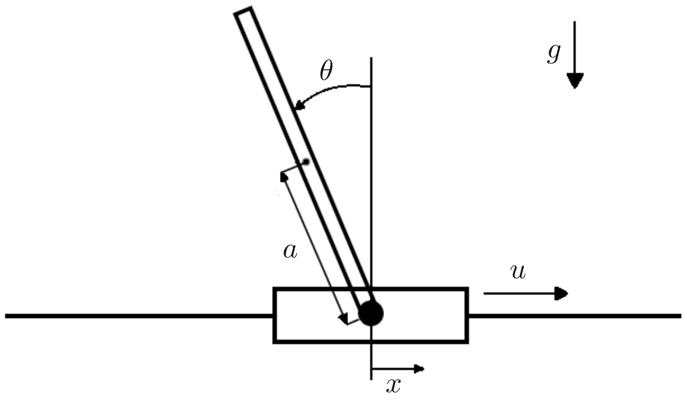


Figure 3.1: Schematic representation of the inverted pendulum on a cart.

Table 3.1: Parameters of the considered system.

Label	Description	Value	Unit
m	Mass of the pendulum	0.3553	kg
a	Distance from the hinge to the pendulum's CoM	0.42	m
J	Moment of inertia of the pendulum	0.03614	Nms^2
d	Friction coefficient of the hinge joint	0.005	Nms

3.2 RL Environment

Since balancing the inverted pendulum on a cart is a popular benchmarking problem in RL, there are already implementations of an RL environment for this problem. OpenAI provides the Gym [24] for setting up the environment. Different from other RL packages, OpenAI Gym provides an interface to communicate with several state-of-the-art RL algorithms. This helps to quickly build an RL environment for specific tasks and to train with different existing baselines. In general, for setting up an RL environment using the OpenAI Gym package, there are the following important functions. Firstly, *the init function* is used to randomize all the states of the system for the beginning of an episode. Secondly, by using the equations of motion of the system (3.1) together with an action input, the *step function* computes the next states of the system in a fixed time step of 20 ms. In this chapter, the Runge–Kutta 45 is utilized for the step function. Thirdly, *the observation function* is built to gather input data for computing the action of the agent. Note that in this chapter, the network architecture of the agent consists of two hidden layers. Each layer has 128 neurons. Different to the standard Open AI gym environment for the pendulum on a cart, the observations are extended, see Table 3.2. This extension helps to achieve the three goals mentioned earlier. Finally, *the termination function* defines stopping criteria for an episode. The episodes are terminated either after 500-time steps or when the cart reaches its left or right limit. During training, we randomly initialize the states of the pendulum in valid ranges of the system states. In the following, more details on the observation function and the design of the reward function are introduced.

3.2.1 Observations and Actions

For the balancing or swing-up problem, only the physical states of the inverted pendulum are used as observations. If the goal is to learn to position the cart at an arbitrary position along the runway, this is no longer sufficient and an additional observation is necessary. In this case, the desired position of the cart x_d is introduced. During training, this new state is initialized with a random constant value for each episode so that the agent can learn to minimize the difference between the actual position and this desired position. After training, this target state can be used to give the agent a target value for the position of the cart, which can be set arbitrarily independent of the inverted pendulum system. Similarly, since the agent wants to learn both swing-up and swing-down, a new observation o_d is introduced. This observation has only two values, i.e., 1 for the swing-up action and -1 for the swing-down action. Table 3.2 summarizes the used observations. Note that due to the interlocking representation of the pendulum angle between 0 and 2π, sine and cosine functions are utilized to ensure the continuity of this state.

3.2.2 Reward Function

In RL, the reward function determines the behavior of the agent which can maximize this function. In the presented pendulum environment, the episode is terminated prematurely when the cart moves beyond its limits. This is to encourage the agent to avoid these condi-

Table 3.2: Observations and actions of the system.

	Label	Description	Range	Unit
Observations	x	Position of the cart	$[-0.65, 0.65]$	m
	v	Velocity of the cart	$[-\infty, \infty]$	m/s
	$cos(\theta)$	Cosine of the pendulum angle	[-1,1]	
	$sin(\theta)$	Sine of the pendulum angle	[-1,1]	
	ω	Angular velocity of the pendulum	$[-\infty, \infty]$	rad/s
	x_d	Desired position of the cart	[-0.65,0.65]	m
	o_d	Desired pendulum configuration	$\{-1, 1\}$	
Actions	$u = \ddot{x}$	Acceleration of the cart	-10...10	m/s^2

tions, as no more rewards can be collected after termination. In the used reward function, only positive rewards are considered to avoid complications from early termination.

To learn a strategy for swinging up and positioning depending on the desired cart position x_d and the desired orientation state o_d, the reward function embeds multiple instructions for the agent. To realize this, the reward function r is constructed from two parts. First, r_x is used to reward the agent when the cart reaches the desired position x_d

$$r_x = \left(1 - \frac{|x - x_d|}{s_x}\right)^2 \tag{3.2}$$

Second, for the desired orientation, the reward part reads as

$$r_\theta = \left(1 - \frac{|\theta - \theta_d|}{s_\theta}\right)^2 , \tag{3.3}$$

with

$$\theta_d = \begin{cases} \pi, & \text{if } o_d = -1 \\ 0, & \text{if } o_d = 1 \, . \end{cases} \tag{3.4}$$

The restriction $\theta \in (-\pi, \pi]$ is automatically fulfilled by calculating θ from its sine and cosine representation. To keep the rewards positive for all valid state configurations, scaling parameters are used. User-defined parameters are listed in table 3.3. Each part

Table 3.3: User-defined parameters of the reward function.

Label	Value	Unit
s_x	1.3	m
s_θ	π	rad

can be used to solve the respective sub-tasks.

To solve both sub-tasks simultaneously, a combination of the two reward parts is employed in the form

$$r = r_x r_\theta \, . \tag{3.5}$$

3.3 Results

In this section, the simulation and experimental results for training the RL agent are presented. First, statistical results for training the RL agent using common state-of-the-art RL approaches, i.e., baselines, are presented. Then, the best trained RL agent is used in the real model of the system to validate the sim2real gap.

3.3.1 Training performances of the RL baselines

To this end, the open-source package Stable-Baselines3 [25] is utilized. It offers implementations of state-of-the-art RL approaches, i.e., Advantage Actor-Critic (A2C) (2.45), Deep Deterministic Policy Gradient (DDPG) [26], Proximal Policy Optimization (PPO) [12], Soft Actor-Critic (SAC) [27] and Twin Delayed DDPG (TD3) [28].

Training performance for the first reward term r_x

For this first statistical result, only the reward term r_x for driving the cart to the desired position x_d is employed in the training process. Instead of using (3.2), the exponential formulation of r_x is applied in the form

$$r_x = \mathrm{e}^{-|x-x_d|} . \tag{3.6}$$

Note that the initial value of the pendulum angle θ is set to zero, with the addition of the small initialization noise in the range of 0.05 rad. Additionally to the other termination criteria, an episode is terminated when θ falls out of the safe range of $[-45°, 45°]$. Thus, in this simulation, both placing the cart at x_d position and stabilizing are considered. Figure 3.2 shows the results of various RL baselines together with their default settings from the Stable-Baselines3 package. If the cart is at the desired location x_d from the beginning, the cumulative reward can reach a maximum of 500. This is because the reward value of 1 is given at each step for the total maximum of 500 steps. Thus, a final cumulative reward value of $\approx$ 400 is considered a good result in this case.

As presented in Table 3.2, all baselines achieve a cumulative reward over 400. However, the training time for each RL baseline is different. The A2C and PPO methods are much faster than others in the training times. While A2C, PPO, and SAC approaches achieved good results in every trial, the DDPG and TD3 approaches did not converge every time.

Training performance for the second reward term r_θ

For this second statistical result, only the reward term r_θ (3.3) is utilized in the training process. Note that, for only considering the swing-up action of the pendulum, the following modified r_θ is used

$$r_\theta = 1 - \frac{|\theta|}{s_\theta} . \tag{3.7}$$

The statistical results are collected three times, only the best trial is presented in Figure 3.3. Figure 3.3 shows that after 10^5 training steps, four of the algorithms, i.e., DDPG,

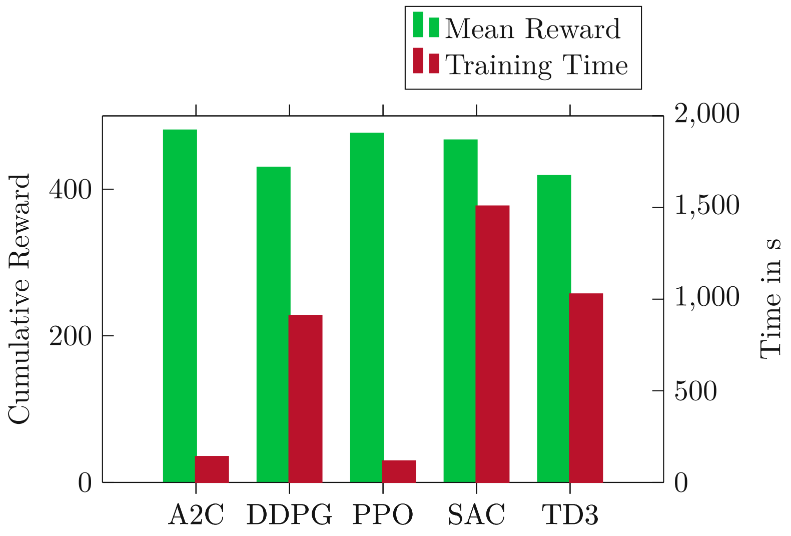


Figure 3.2: Training results of 10^5 learning steps of RL baselines for the first reward term r_x.

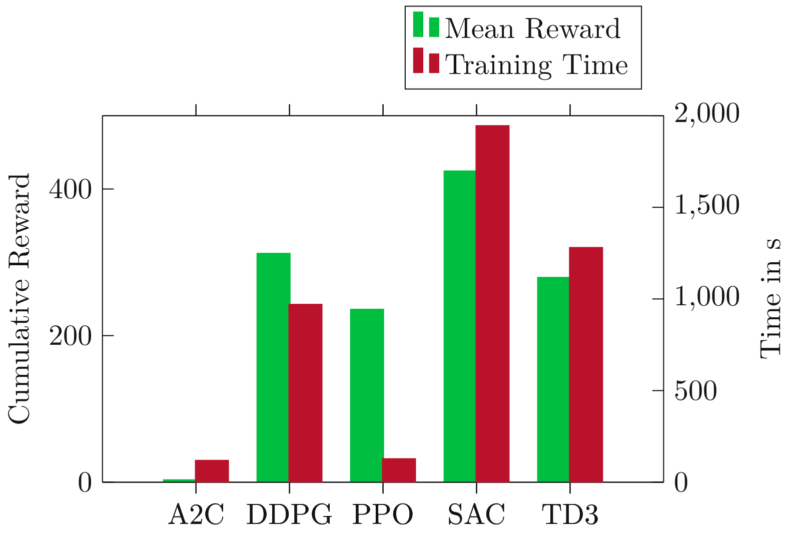


Figure 3.3: Training results of 10^5 learning steps of RL baselines for the second reward term r_θ.

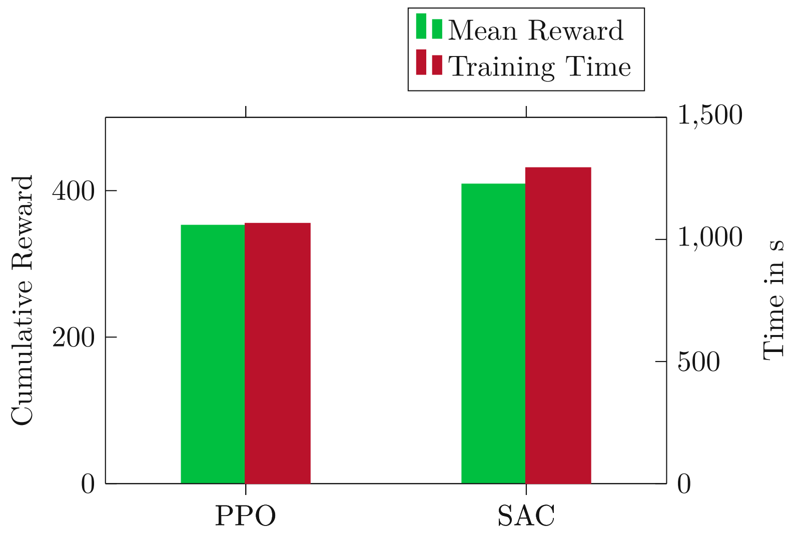


Figure 3.4: Results for learning to swing the inverted pendulum (r_θ) until a reward above 350 is achieved.

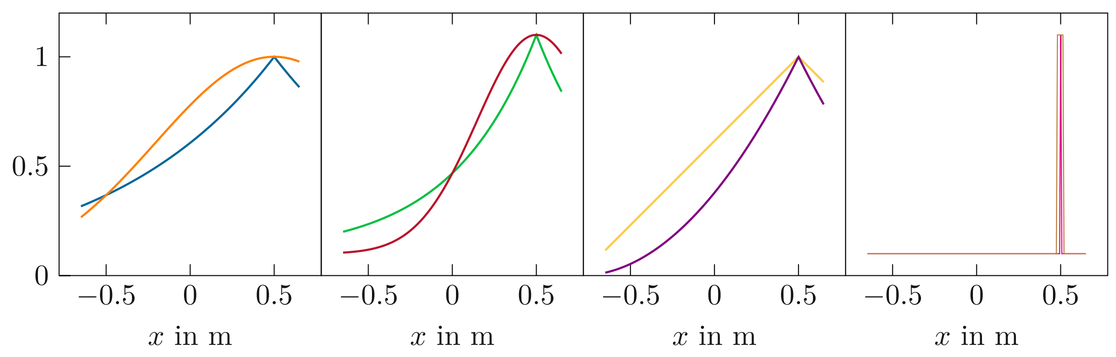
PPO, SAC, and TD3, can achieve a significant gain, while A2C could not improve the strategy in any trial. Note that TD3 could not create a good policy in every trial, while DDPG, PPO and SAC created good results consistently.

So far, only PPO and SAC perform robustly on these statistical tests without the need for fine-tuning. To further compare these two algorithms, the training was repeated with more training steps. Additionally, when a reward of 350 was reached, the training was aborted. Figure 3.4 shows the ratio between training time and collected reward which is similar for these two algorithms. Only the number of calculated time steps differs greatly. While PPO requires 880e3 training steps, SAC is finished after 80e3 training steps.

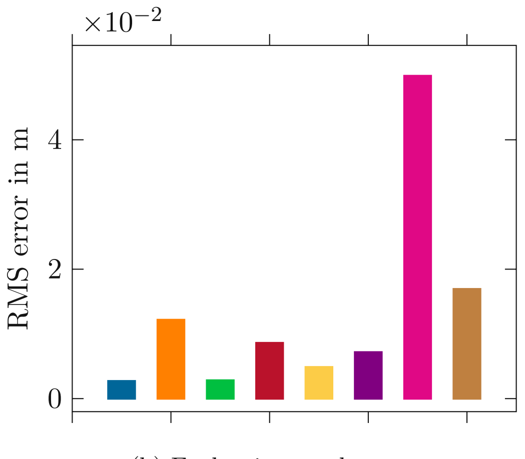
Statistical results of different reward formulations of r_x

The shape of a reward function is an important factor affecting the performance of an RL algorithm. To further investigate which form of reward functions is best suited for the balance and positioning problem, PPO is used to train the system with eight different reward functions of r_x for 10^5 training steps. The mean square error of the position of the cart at the end of an episode w.r.t. the target position x_d over 10^3 episodes is considered for the evaluation process. To mitigate the effects of random initial conditions during training, the RL agent is trained 3 times and the best policy is chosen.

All functions used are described in Figure 3.5c and are illustrated in Figure 3.5a for a target x_d of 0.5. The exponential function guarantees non-negativity. A linear and a quadratic argument of the exponential function, marked by the blue and orange color in Figure 3.5c, are considered. Scaled versions of these two functions are also employed, marked by the green and red colors in Figure 3.5c. Therein, an offset of 0.1 is added since this scaled exponential function returns a value very close zero when the cart is near



(a) Plotted reward functions for $x_d = 0.5$.



(b) Evaluation results.

	Reward function
🟦	$\mathrm{e}^{-\lvert x-x_d\rvert}$
🟧	$\mathrm{e}^{-(x-x_d)^2}$
🟩	$\mathrm{e}^{-\lvert 2x-x_d\rvert}+0.1$
🟥	$\mathrm{e}^{-(2x-2x_d)^2}+0.1$
🟨	$\frac{1}{1.3}(1-\lvert x-x_d\rvert)$
🟪	$\frac{1}{1.3^2}(1-\lvert x-x_d\rvert)^2$
🟪	$\mathbb{1}[\lvert x-x_d\rvert<0.005]+0.1$
🟫	$\mathbb{1}[\lvert x-x_d\rvert<0.02]+0.1$

(c) Reward functions.

Figure 3.5: Evaluation of reward shapes for positioning the cart while balancing the pendulum.

the limits. This can hinder the learning process. Additionally, a linear decay function and a squared version, marked by the yellow and purple color in Figure 3.5c, are also investigated in this statistical test.

The results are shown in Figure 3.5b. Five of the six continuous reward functions result in an average accuracy of less than 1 cm. The main difference is that the smooth exponential functions are outperformed by their non-smooth counterparts as well as by the linear decay functions. With sparse rewards, marked by the violet and brown color in Figure 3.5c, choosing the right threshold is crucial for obtaining good results.

3.3.2 Experimental results for the combined reward function

In the previous statistical results, PPO and SAC were the two algorithms with the best performance. For the combined reward function r in (3.5), SAC is chosen since this algorithm is more sample-efficient than PPO [29]. For the training process, the default parameters are used. The size of the network is reduced to two hidden layers with 128 neurons in each layer. This helps to realize the learned policy in a real hardware system, i.e.,DSPACE Microlabbox 1202, of the single pendulum on a cart [22].

While the training is performed with fully observable states without any noise, the velocity of the cart and the angular velocity of the pendulum cannot be measured directly in the real system. Thus, they are obtained from the derivative of the cart position x. This position data is determined using rotary encoders, which involve quantization noise. By using the derivative, this noise is amplified and it must be checked if the agent trained on the idealized system can handle this noise, before applying it to the real system.

Figure 3.6 shows the results of applying the policy in a MATLAB/SIMULINK simulation of the system without quantization noise. The system starts at $-0.5\,\mathrm{m}$ with the pendulum hanging down and is commanded to transit to $0.5\,\mathrm{m}$ with the pendulum in the up-right position. The results of the simulation, where the quantization noise is added as in the real system, are shown in Figure 3.7. The most obvious difference is that the acceleration calculated from the policy is sensitive to noise. Especially when the pendulum is in its unstable position. Although this sensitivity has a significant impact on the trajectory, the cart makes some additional movements and reaches its target point. Next, the policy is applied to the real system. For comparison, the same commands, previously used for the simulation, are applied to the real system, resulting in the trajectory shown in 3.8. The measured trajectory is very close to the one simulated with noise, the only difference is in the acceleration.

To test how the agent reacts to an external disturbance, the pendulum is disturbed with a stick near its end. A trajectory is recorded, as shown in Figure 3.9, with a strong and a small disturbance. After the first strong disturbance, the agent drops the pendulum and swings it up again on the other side with the momentum that the pendulum gained when it fell. With the second smaller disturbance, the deviation of the pendulum angle is kept small and the pendulum comes back into balance without falling. More experiments are recorded at https://www.acin.tuwien.ac.at/en/8b92/.

3.4 Conclusion

The statistical simulation results of the different RL algorithms in Section 3.3.1 have shown that PPO and SAC are the most reliable of the algorithms implemented in Stable-Baselines3 for this task. Given the fact that these algorithms were introduced to outperform the older algorithms such as A2C and DDPG, this is a plausible result. Only the performance of TD3 is below average, considering that it is the latest of these algorithms. This could be due to inappropriate parameters, however, no parameter tuning was performed for the other algorithms either.

Regarding the reward function, the results in Section 3.3.1 have shown the general feasibility of positive rewards in combination with early termination. The need for positive rewards everywhere is emphasized by the fact that an offset must be introduced when the reward function goes somewhere near zero to ensure that the agent learns to balance the pendulum. With this offset modification, all presented reward functions can be used to

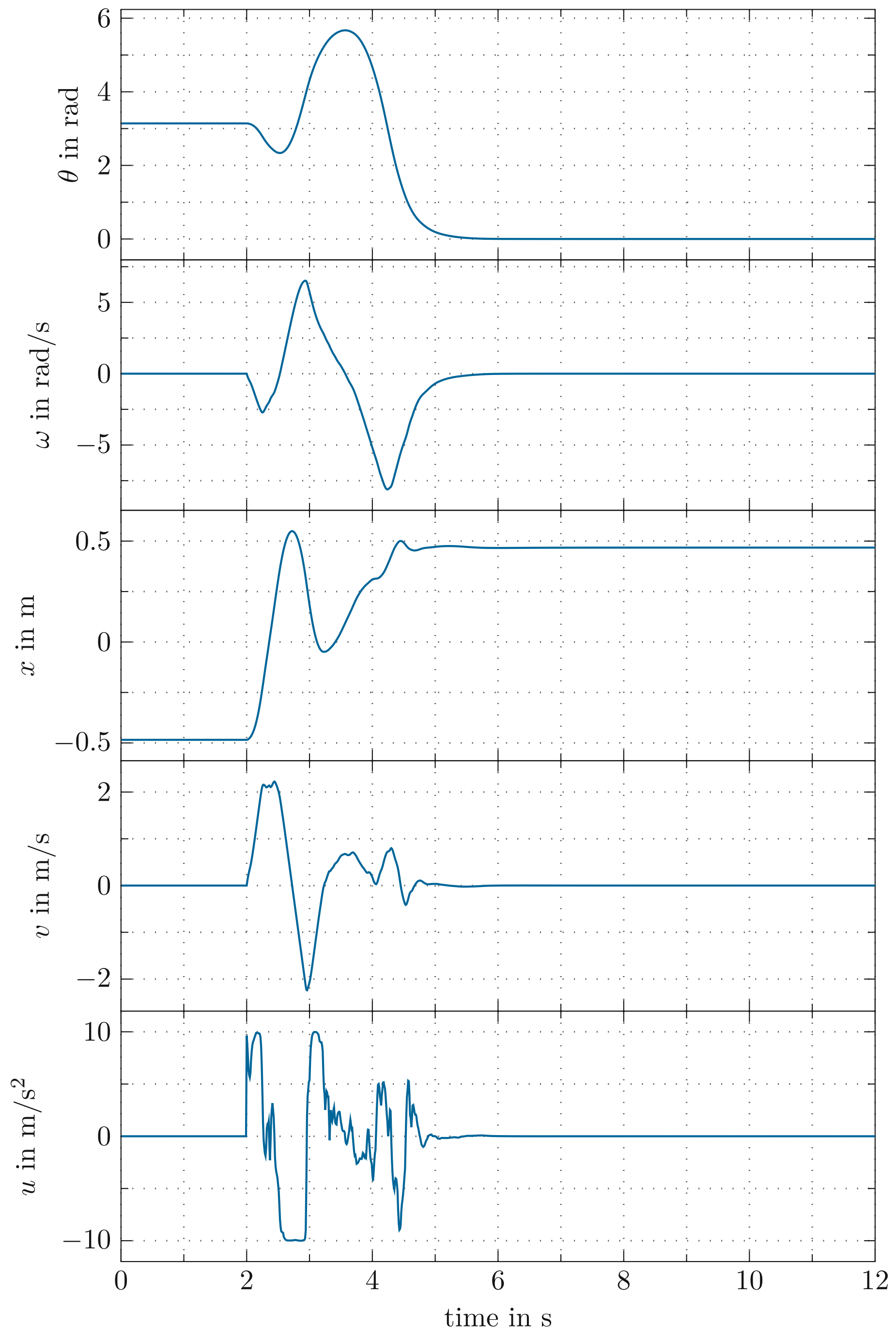


Figure 3.6: Simulation results without quantization noise for the swing-up when the cart is shifted by 1 m.

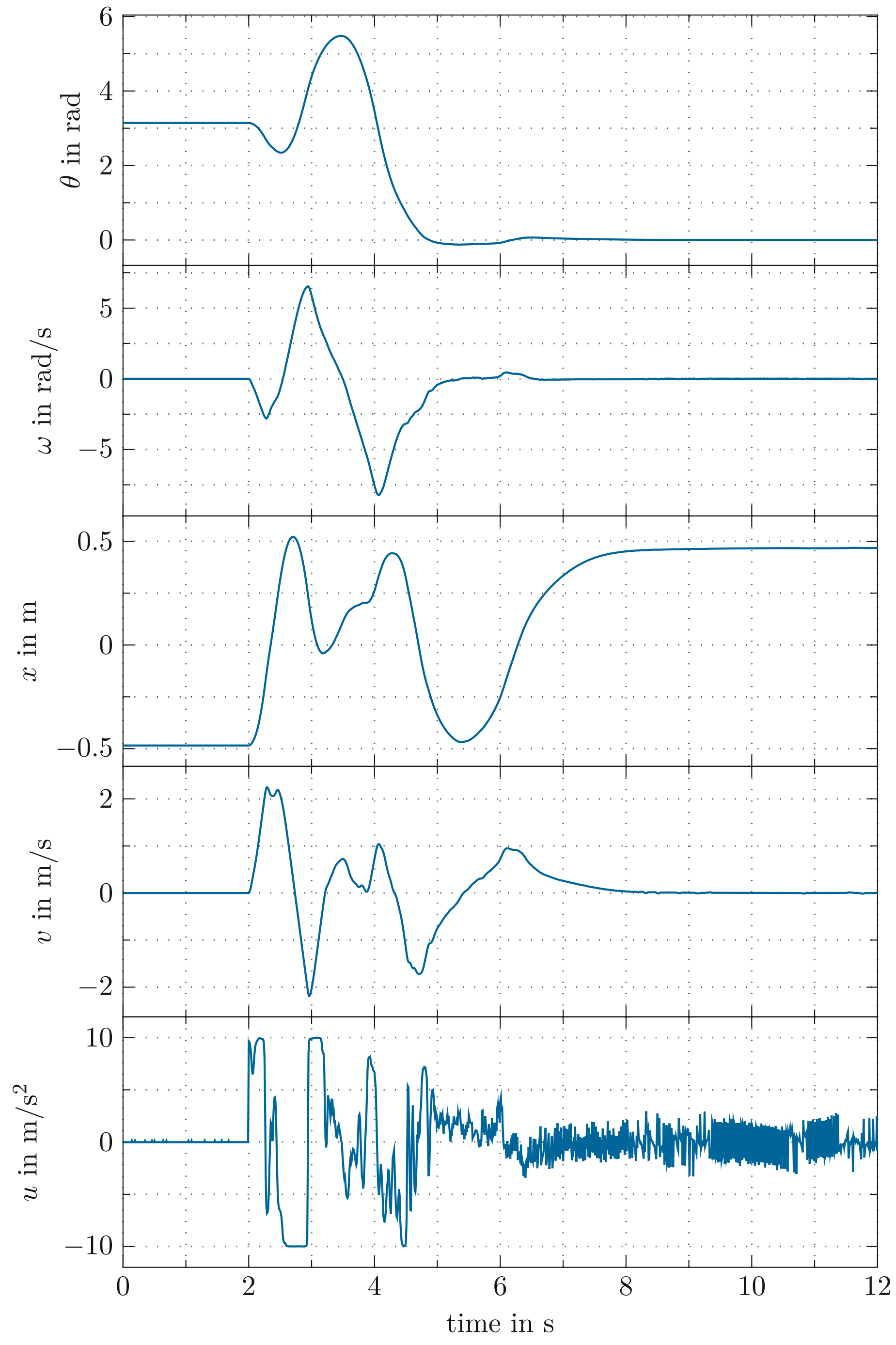


Figure 3.7: Simulation results with quantization noise for the swing up when the cart is shifted by 1 m.

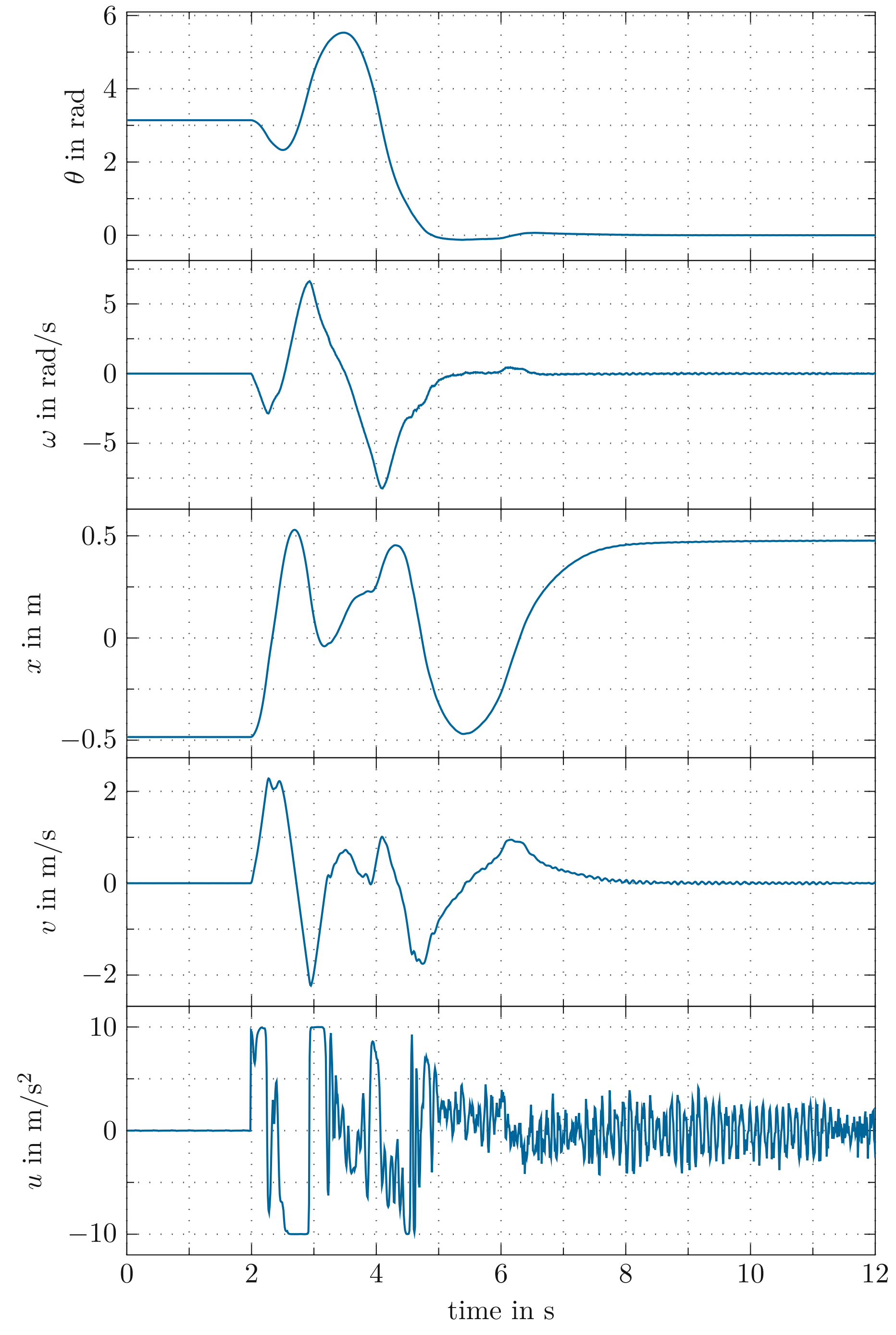


Figure 3.8: Measurement of the swing-up when the cart is displaced by 1 m on the real system.

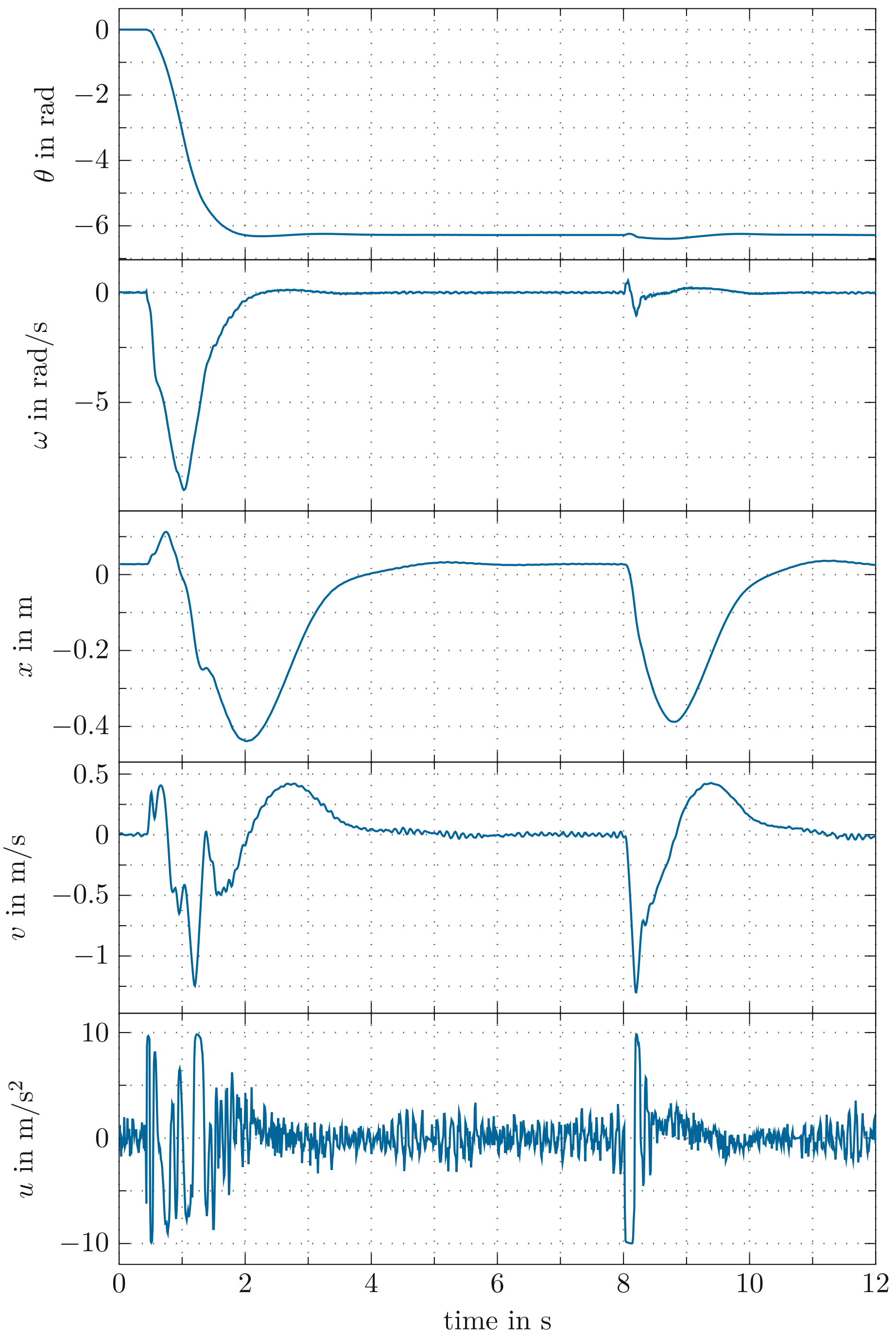


Figure 3.9: Measurement of the real system after a large and a following small disturbance.

solve the balance problem. In addition, the positioning accuracy is reasonable, especially for the continuous reward functions. For the non-continuous rewards, i.e., sparse reward functions, there seems to be an exploration problem when the high-reward region is too small, resulting in higher positioning errors. However, with the right size of the high reward area, the positioning accuracy comes close to that of continuous rewards.

Section 3.3.2 shows that a policy can learn to swing the inverted pendulum up or down on a cart and also learn to position the cart while balancing the pendulum. The design of reward helps to learn all tasks with one strategy. This avoids the problem where we have to switch strategies, but comes at the cost of increasing training time. The interesting aspect of this strategy is that it can collect data anywhere in the state space of the system. Therefore, the learned policy can counteract strong disturbances. In combination with the initialization of the episodes with values from the entire state space, the strategy has learned how to quickly bring the pendulum into the desired state. The experiments show that if the disturbance is weak, the pendulum can be balanced without falling. When the disturbances are strong, the policy can lean on the momentum of the system to achieve an immediate swing-up, after the pendulum fells. This behavior is observed in the real system, where the agent decides whether to let the pendulum fall or not depending on the disturbance.

4 RL Application: Log-grasping of a Forestry Crane

In this section, an RL application for grasping a log with a forestry crane is presented. In doing so, a physically based system dynamics of the forestry crane is computed using AGX Dynamics [30]. A brief introduction is given below to provide an overview of this chapter.

4.1 Introduction

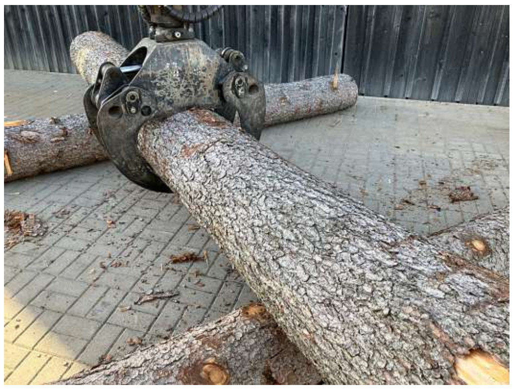
Forestry cranes are used to load wood logs from a pile or from the forest floor onto the bed of a truck. The loading cycle consists of moving the grapple over one or several wood logs, grasping them, moving the crane back to the truck bed, and placing the logs there.

In literature, a model-based [31], [32] and a learning-based approach [33] for controlling forestry cranes are investigated. In [34], [35], path planning approaches for the crane grapple are presented for moving from the initial configuration to the grasping position. Recently, Andersson et al. [36] presented an RL-based solution for the half-loading cycle, i.e. moving the crane to the log and grabbing the log. In [36], a small log next to a forwarder on the ground is considered. Additionally, the position and rotation of the log are random, however, the size of the log is constant.

Experienced operators use different strategies for grasping logs depending on the environment and log diameter. When grasping a log, the safest strategy is to wrap the gripper jaws around the log and then close the gripper. This helps to firmly lock the wood log in place, as shown in Figure 4.1a. Given the gripper design with the interlocking jaws, logs with diameters down to about 10 cm can be tightly grasped using this method. Additionally, smaller logs can be handled, as they can then only escape from the side of the gripper. The grapple in Figure 4.1a can also handle logs larger than the jaws can fully enclose. In this case, if the jaws can enclose more than half of the log, it can be held safely.

Another way to grasp small and light logs is to clamp them with the jaws from both sides instead of enclosing them completely, as shown in Figure 4.1b. This method works, however, it relies only on the clamping pressure of the jaws. Therefore, the log can be damaged and the grip is not very secure, especially if the gripper is not exactly over the center of gravity of the log.

Aditionally, the movements required to bring the log into the closed position in the grapple


(a) Secure grasp on a medium-sized log.

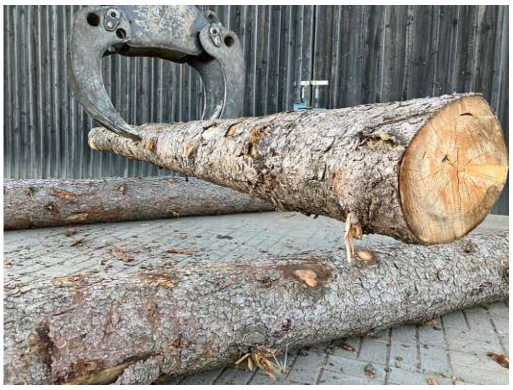

(b) Pinch grasp on a small log.

Figure 4.1: Different ways to grasp a log.

can vary depending on the environment. If the log is lying flat on the ground, an upward motion is required when closing the grapple after lowering the boom tip over the log to prevent the grapple from colliding with the ground. This movement can be seen in the sequence of images in Figure 4.2.

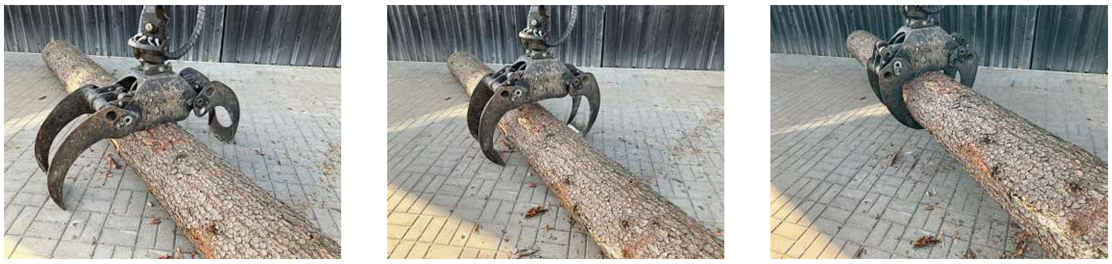


Figure 4.2: Graspig a log from the ground.

A simpler strategy can be applied if the log is supported on either side, as the bottom row in typical log piles is. In this scenario, shown in Figure 4.3, the upward motion is not necessary. The jaws can be closed after placing the grapple on top of the log without any extra movement.

While small and medium logs can be completely enclosed by the grapple, this is not possible for large logs. Therefore, the strategy for large logs is similar to that for supported logs. Place the open grapple on the log and close it to wrap the jaws around the log as much as possible and grip it tightly. In this scenario, the grapple cannot collide with the ground because the grapple jaws are too short compared to the log diameter. Given the above overview of the complexity of the grasping task, it is difficult to solve this task using only standard control strategies. One could combine task scheduling approaches with different control strategies. Another option is to use RL to train an agent in a synthetic

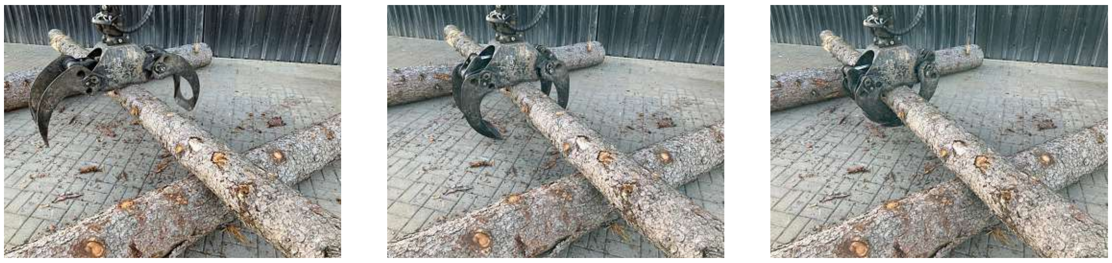


Figure 4.3: Grasping a log on supports.

environment with different log sizes and contact models.

In this chapter, an RL method is applied to solve the task in combination with a model simulated by a physics engine. The following scenario is considered. The jaw gripper is placed somewhere above the wood log, the RL-trained agent is utilized to perform the grasping task. Different diameters of a wood log are investigated to find a universal solution for this task.

4.2 Modeling

Figure 4.4 illustrates the schematic of the timber crane. In real-world applications, this crane is often attached to a truck to perform the loading of wood logs in the forest industry. The considered crane has a total of 10 joints. Since there are two synchronized joints at q_4 for extending the arm length and q_8 for grasping action, the total considered joints of the timber crane model are reduced to 8. Overall, the timber crane has a total of eight degrees of freedom (DoFs) $\mathbf{q}^{\mathrm{T}} = [\mathbf{q}_A^{\mathrm{T}}, \mathbf{q}_U^{\mathrm{T}}]$ consisting of six actuated DoFs $\mathbf{q}_A^{\mathrm{T}} = [q_1, q_2, q_3, q_4, q_7, q_8]$ and two non actuated joints $\mathbf{q}_U = [q_5, q_6]$. The first joint q_1 is the swivel joint for rotating the whole crane. The joints q_2 and q_3 can be used to raise and lower the first and second boom. The next joint q_4 is used for extending the telescopic boom. Additionally, q_5 and q_6 are unactuated joints that are used to ensure the center of mass of the log remains below the boom tip. The pivot joint q_7 allows the grapple to rotate endlessly to match the rotation of the log. The last joint q_8 is utilized for opening and closing the grapple. Note that only q_4 is a translational DoF while all others are rotational DoFs.

4.2.1 Kinematics

The kinematics of the forestry crane is described by transformations from a coordinate Frame $\mathcal{F}_i$ attached to joint i to a coordinate frame $\mathcal{F}_{i-1}$ attached to joint $i-1$

$$\mathbf{H}_{i-1}^{i} = \begin{bmatrix} \mathbf{R}_{i-1}^{i} & \mathbf{d}_{i-1}^{i} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathcal{SE}(3) \tag{4.1}$$

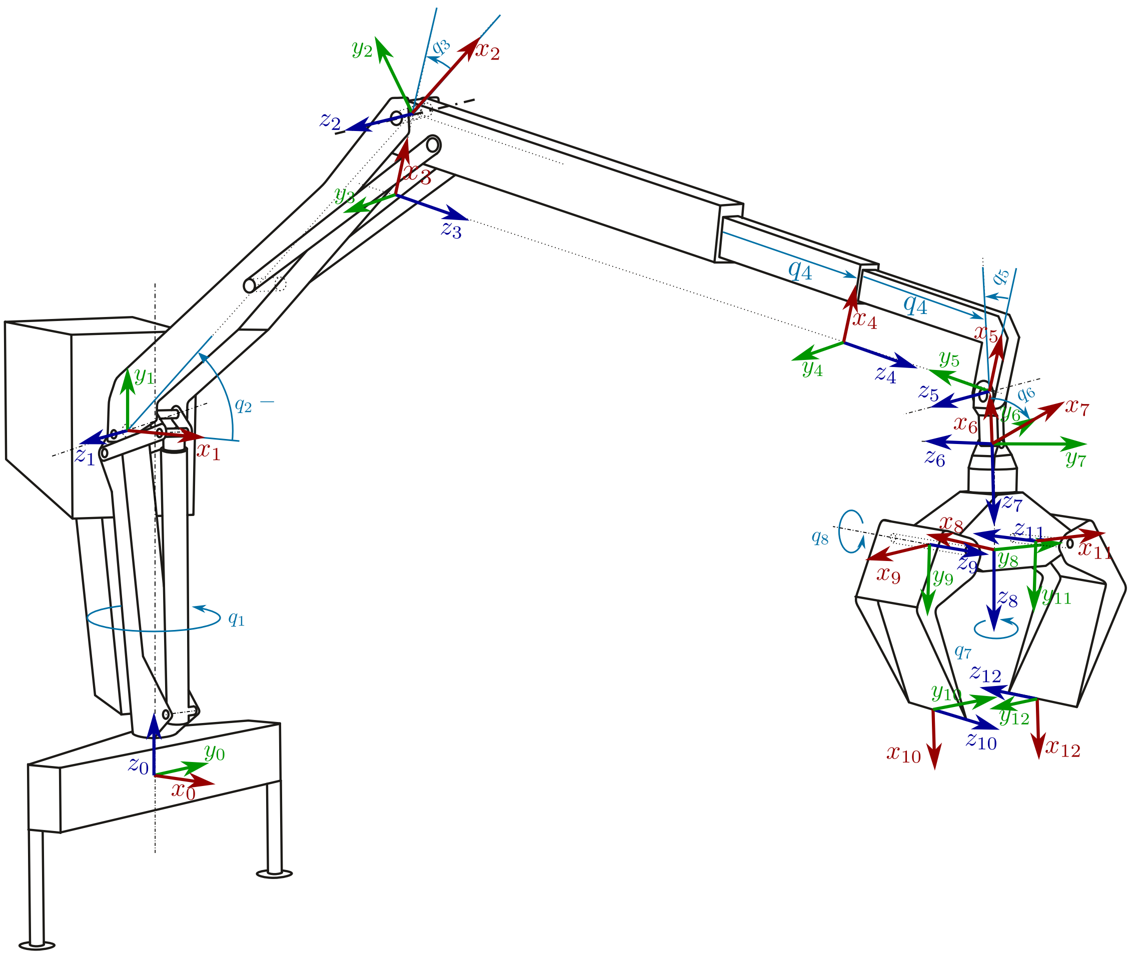


Figure 4.4: Schematic of the forestry crane.

where $\mathbf{R}_{i-1}^{i} \in \mathcal{SO}(3)$ is a three-dimensional rotation matrix and $\mathbf{d}_{i-1}^{i} \in \mathbb{R}^3$ is a three-dimensional translation vector. The coordinate frames are illustrated in Figure 4.4 selected according to the *Denavit-Hartenberg (DH) convention*, see, e.g., [23]. Note that frame $\mathcal{F}_{11}$ is defined by DH convention w.r.t. frame $\mathcal{F}_8$ instead of $\mathcal{F}_{10}$ due to the kinematic structure depicted in Figure 4.4. Hence, homogeneous transformations $\mathbf{H}_{i-1}^{i}$, $i = 1, \dots, 10, 12$ and $\mathbf{H}_8^{11}$ can be described using four DH parameters θ_i, d_i, a_i and α_i as

$$\mathbf{H}_{i-1}^{i} = \mathbf{H}_{Rz}(\theta_i)\mathbf{H}_{Tz}(d_i)\mathbf{H}_{Tx}(a_i)\mathbf{H}_{Rx}(\alpha_i) \ , \tag{4.2}$$

where $\mathbf{H}_{Ri}$ is a pure rotation around the i-axis and $\mathbf{H}_{Ti}$ is a pure translation in direction of the i-axis. The transformation from $\mathcal{F}_j$ to $\mathcal{F}_i$, $0 \le i < j$ can be computed using

$$\mathbf{H}_i^j = \begin{cases} \prod_{l=i+1}^{j} \mathbf{H}_{l-1}^{l} & , \text{ for } j \le 10 \\ \left(\prod_{l=i+1}^{8} \mathbf{H}_{l-1}^{l}\right)\mathbf{H}_8^{11}\mathbf{H}_{11}^{j} & , \text{ for } 11 \le j \le 12 \end{cases} , \tag{4.3}$$

where $\prod_{l=i+1}^{j} \mathbf{H}_{l-1}^{l}$ being the identity for $j \le i$. The DH parameters for the crane are

Table 4.1: Characteristic of the timber crane's joints.

Coordinate	Name	Actuated	Range	Unit
q_1	slewing joint	✓	[-3.71, 3.71]	rad
q_2	boom joint	✓	[-1.2, 1.56]	rad
q_3	arm joint	✓	[-0.91, 4.6]	rad
q_4	telescopic boom	✓	[0, 4.47]	m
q_5	tip joint		[-1.57, 1.57]	rad
q_6	tilt joint		[-0.79, 2.36]	rad
q_7	rotate joint	✓	$[-\infty, \infty]$	rad
q_8	grapple jaws	✓	[0, 3]	rad

Table 4.2: Denavit-Hartenberg parameters of the timber crane.

i	θ_i [rad]	d_i [m]	a_i [m]	α_i [rad]
1	q_1	2.425	0.1800	$\pi/2$
2	q_2	0	3.4931	0
3	q_3	0	-0.3925	$\pi/2$
4	0	$q_4 + 3.157$	0	0
5	0	q_4	0	$-\pi/2$
6	q_5	0	-0.2130	$-\pi/2$
7	q_6	0	0	$-\pi/2$
8	q_7	0.578	0	0
9	$-\pi/2$	0	0.3402	$\pi/2$
10	$\pi/2$	0	0.8566	0
11	$\pi/2$	0	0.3248	$\pi/2$
12	$\pi/2$	0	0.8566	0

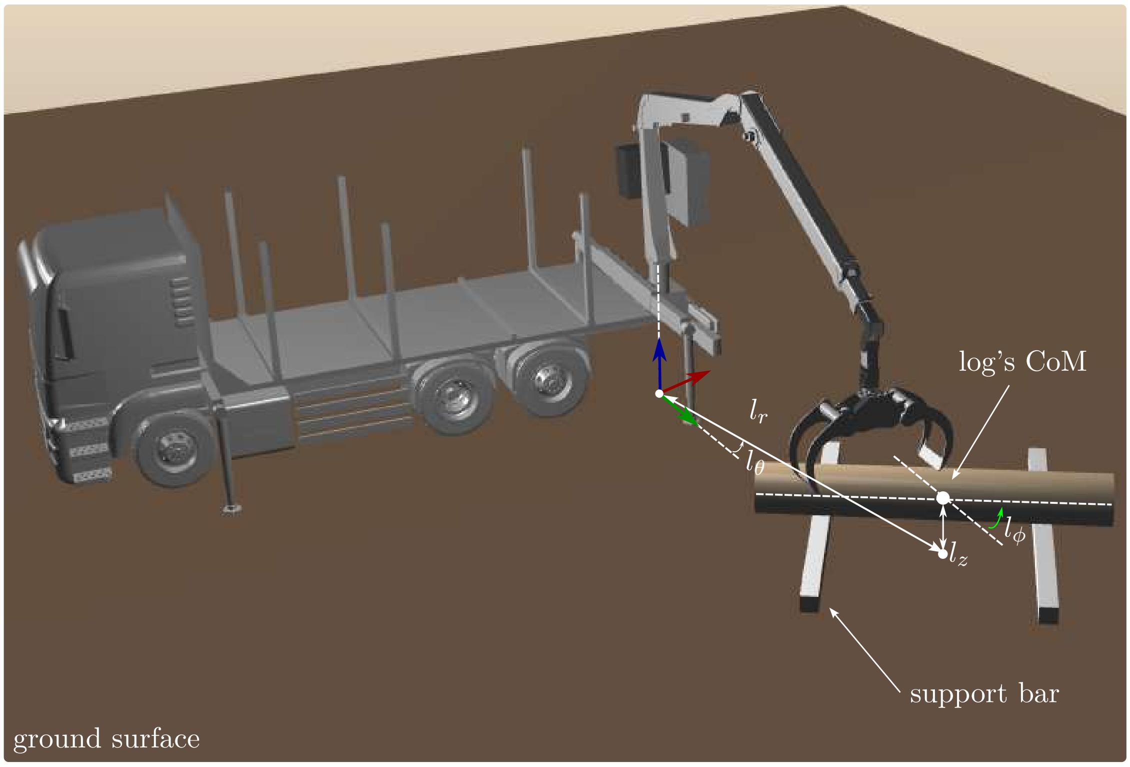


Figure 4.5: The forestry crane in AGX Dynamics simulator.

given in Table 4.2. Using the above kinematic relations, the wrist position of the grapple, $\mathbf{d}_g^{\mathrm{T}} = [g_x, g_y, g_z]$, is taken from

$$\mathbf{H}_0^8 = \begin{bmatrix} \mathbf{R}_0^8 & \mathbf{d}_g \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{4.4}$$

4.2.2 Simulator

A physics-based simulation is created using a commercial physic engine AGX Dynamics [30] which supports real-time simulation of multibody systems of nonsmooth contact dynamics and deformable terrain.

In AGX Dynamics, the dynamical system is comprised of multiple heterogeneous subsystems with stiff dynamics and unpredicted events where the connectivity and number of variables suddenly change. In this type of system, in addition to differential-algebraic equations of motion [23], the velocity of a physical body may change discontinuously with respect to a specific impact law, e.g., [37, 38], which can be expressed in terms of complementarity conditions and inequality. The system dynamics is solved by a non-smooth discrete element method [39]. Additionally, the contact forces are modeled using impact laws and kinematics constraints for unilateral contacts and friction.

A snapshot of the simulated environment is illustrated in Figure 4.5. The assembled model

Table 4.3: Properties of the log.

4D Pose	Label	Description	Initial Range	Unit
Displacement	l_r	distance (polar coordinate)	[5, 7]	m
	l_θ	angle (polar coordinate)	[-0.26, 0.26]	rad
	l_z	height	$0.2 + d/2$	m
Rotation	l_ϕ	rotation along the vertical axis	$[-\pi/2, \pi/2]$	rad

of the forestry crane (including the truck) consists of 38 rigid bodies and 10 active joints (two pairs of the joints are synchronized). The total operating weight of the forestry crane is ≈ 1981 kg. To create a realistic scenario, Young's modulus of the contact model between the six tires and the ground is set to 5.10^5 while the contact between two mounting supports (at the truck's cabin and at the back) with the ground is set to 10^8. Additionally, rotary joints are driven with velocity-controlled rotary motors while the telescopic boom joint q_4 is driven by a linear motor.

4.3 RL Environment

The training environment is created using OpenAI's Gym library [24]. In this environment, a 4 m long cylindrical log with a random diameter d ranging from 20 cm to 80 cm is considered. This log is placed on two supported bars, as illustrated in Figure 4.5. We assume that the location and orientation of the center of mass (CoM) of the log are given. Using the given pose of the log, the polar displacement l_r and the rotation angle of the log's CoM w.r.t. the origin of the frame $\mathcal{F}_0$ are computed. The use of polar coordinates is chosen because of the rotationally symmetrical workspace of the crane.

The vertical displacement w.r.t. the ground of the log's CoM l_z is considered. Additionally, the rotation angle of the log w.r.t. the world coordinate is taken into account. This augmented input, i.e., 4D pose of the log $[l_r, l_\theta, l_z, l_\phi]$, enriches the information which helps the agent to make a correct decision. For instance, if the grapple is not aligned correctly w.r.t. the log, no grabbing is possible. Thus, the rotation of the log along the vertical axis l_ϕ is an important property.

The 4D pose of the log is summarized in Table 4.3. Note that randomizing the log's pose and its diameter is very useful since we can compute a very general policy that can handle different log sizes in different situations. Note that, to focus on the grabbing action, the grapple is initially located above the log in close proximity.

The slewing joint q_1 is used to initially place the grapple close to the log by matching the angle l_θ. Additionally, a random offset in the range of [-10°, 10°] is added to the initial state of q_1. Without this random part in the initialization, there would be no incentive for the RL agent to move the slewing joint because it would be already correctly aligned. In Table 4.4, the initial configuration of the crane is summarized. An example of the initial configuration is depicted in 4.6.

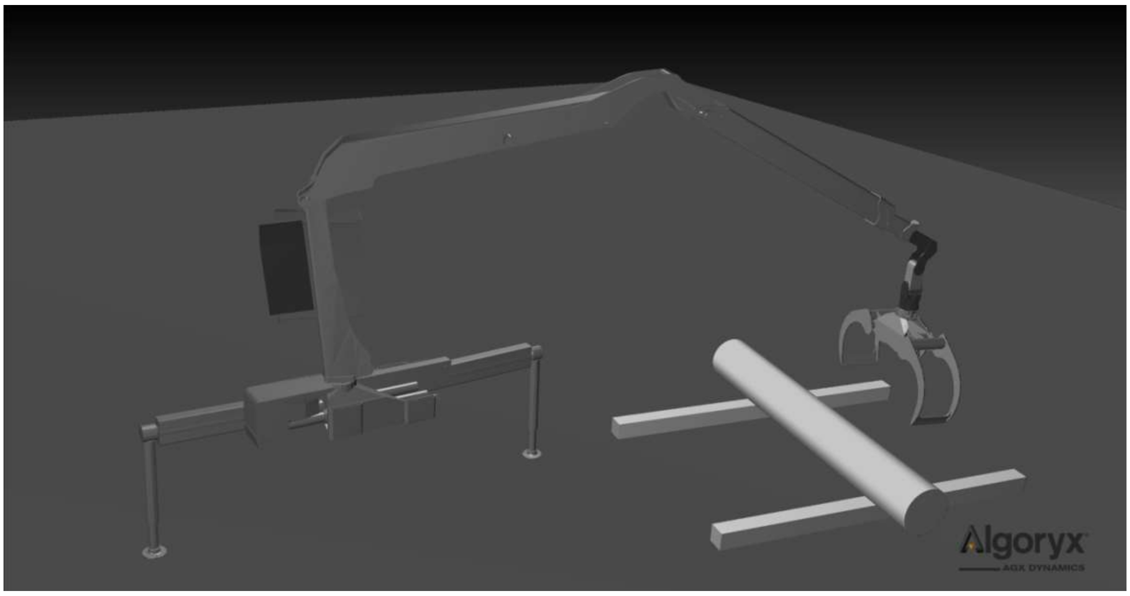


Figure 4.6: Example of an initial setup of the crane environment.

Table 4.4: Initial values of the crane joints.

Joint Coordinate	Initial Value	Unit
q_1	$\pi/2 - l_\theta + \text{rand}()$	rad
q_2	0.31	rad
q_3	0.79	rad
q_4	0.25	m
q_5	0.47	rad
q_6	$\pi/2$	rad
q_7	$\pi/2$	rad
q_8	1	rad

4.3.1 Actions and Observations

The used model of the crane is speed controlled, therefore, the actions are speed set points for every actuated joint. The speed set points for the actuated joints q_A are combined in the vector $\mathbf{u}^\mathrm{T} = [u_1, u_2, u_3, u_4, u_7, u_8]$. In the RL environment, observations are what the agent perceives after executing an action. First, all joint angles are included in the observation set, as well as the 4D pose of the log.

Additionally, we utilize the relative 4D pose in the observation set which helps to simplify the agent's task. This is based on the following reason: "The agent can only learn a useful grabbing strategy if the grapple is close enough to the log and is sufficiently aligned".

Table 4.5: Summarizing of the observations and actions.

	Observations	Actions
Joints	q_1	u_1
	q_2	u_2
	q_3	u_3
	q_4	u_4
	q_5	
	q_6	
	q_7	u_7
	q_8	u_8
4D Log Pose	l_r	
	l_θ	
	l_z	
	l_ϕ	
Relative 4D Pose	δ_x	
	δ_y	
	δ_z	
	δ_ϕ	

Thus, the relative 4D pose vector is computed

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_z \\ \delta_\phi \end{bmatrix} = \begin{bmatrix} g_x - l_x \\ g_y - l_y \\ g_z + g_{z_{off}} - l_z \\ g_\phi - l_\phi \end{bmatrix} , \tag{4.5}$$

the grapple rotation along the vertical axis g_ϕ is in the form

$$g_\phi = q_1 - q_7 , \tag{4.6}$$

$[g_x, g_y, g_z]$ are taken from (4.4). Because of the symmetric structure of the grapple and the wood log, two possible configurations are possible for grabbing a log. To indicate both configurations as valid options, the restriction of $\delta_\phi \in [-\pi/2, \pi/2]$ can be achieved with a modulo operation. The user-defined parameter g_{zoff}, see Table 4.6, is used to specify the target position of the wood log w.r.t the grapple in a way that a stable grasp can be achieved when this position is reached. Since the log's diameter d can be varied in the range of [0.2, 0.8] m, g_{zoff} can be used to shift the target position up or down w.r.t. the grapple position $\mathbf{d}_g$ to adapt to the log diameter. Table 4.5 summarizes observations and actions for this RL environment.

4.3.2 Episode Termination

Each training episode has a limitation of 10^3 time steps. In the following, termination criteria are listed in detail.

- To teach the agent to avoid joint limits, an episode is terminated early if one of the joint limits is violated.
- Another case that triggers termination is when the log is more than 8 meters away from the grapple, which can happen when the crane tries to push the log.

4.3.3 Reward Function

To gradually teach the forestry crane to grasp a wood log, five terms are combined to build the reward function

$$r = r_c + r_d + r_o + r_t + r_l \,. \tag{4.7}$$

These terms are explained in the following using the positive weights $w_\square$ and the scaling parameters $s_\square$

- The first term r_c is a constant value to encourage exploration and avoid early termination. This value $r_c = 0.1$ is chosen to be negligible once the forestry crane gains reward from other terms.
- The second term
$$r_d = w_d e^{-s_d|\boldsymbol{\delta}|} \,, \quad s_d > 0 \,, \tag{4.8}$$
with $\boldsymbol{\delta}$ defined in (4.5), is utilized to teach the forestry crane to align the grapple to the log's yaw angle and to minimize the distance between the grapple position $\mathbf{d}_g$ (4.4) and the log's CoM.
- To perform the grasping action, the grapple has to be opened while moving down to the target location. Then the forestry crane starts closing the grapple. Thus, the third reward term is given by
$$r_o = w_o \begin{cases} (1 + \frac{q_8}{s_o}) & \text{if } |\boldsymbol{\delta}| < \epsilon \\ (1 - \frac{q_8}{s_o}) & \text{if } |\boldsymbol{\delta}| \geq \epsilon \end{cases} , \quad s_o > 0 \,. \tag{4.9}$$
This reward term returns the highest value for an open gripper if $|\boldsymbol{\delta}|$ is larger than a threshold. When $|\boldsymbol{\delta}|$ becomes smaller, which means that the grapple is close and is well-aligned to the wood log, closing the grapple is most rewarded.
- The next reward term r_t is given if the grapple touches the wood log. The grapple consists of three parts. The first and second parts are the two jaws. The third part is the upper plate where two jaws are mounted. This reward function is defined as
$$r_t = w_t \begin{cases} (t_1 + t_2 + s_t t_3) & \text{if } |\boldsymbol{\delta}| < \epsilon \\ 0 & \text{otherwise} \end{cases} , \quad s_t \geq 0 \,, \tag{4.10}$$
with t_1, t_2, and t_3 are three binary variables that are set to 1 if the log is touched by the corresponding parts of the grapple.

- The final term is used to reward the goal of lifting the log and simultaneously penalize the action $\mathbf{u}$, i.e., the motor speed setpoints when the log is already lifted from the ground in the form

$$r_l = \begin{cases} w_l e^{-|\mathbf{u}|} & \text{if } l_z > l_{z_0} + s_l \\ 0 & \text{otherwise} \end{cases}, \quad s_l > 0\,. \tag{4.11}$$

  Note that using only this function for the total reward (4.7) can also lead to a successful grasp. This intuitively allows the robot to try out all actions that could lead to mastering the grasping task. However, other terms, e.g. r_d, r_o, r_t, can be considered as curricula [40] that could teach the robot to accomplish the task in a faster computation time and in a more robust way.

By selecting different user-defined parameters, two policies are designed for the grasping task. The parameters for these two policies, A and B, are presented in Table 4.6. In order to lift the log, contacts between the two jaws and the log are necessary. The contact between the log and the upper plate of the grapple is optional, although it is an indicator of a good grasp. Different values of s_t for two policies are selected to adjust the importance of this factor in the total reward function. Note that the final goal is to grasp the log and lift it from the ground. Thus, the parameter w_l is set to be dominant over other parameters in both policies.

The main difference between the two policies is the contact reward r_t (4.10). Policy B takes into account the contact point between the log and the upper part of the gripper, while policy A neglects this factor, i.e., $s_t = 0$. Additionally, the parameter g_{zoff} is differently defined for the two policies. For policy A, a constant offset $g_{zoff} = 0.5$ is used for all log sizes such that they all fit into the grapple. For policy B, the target position of the log in the grapple is adjusted so that it only has to touch the upper part of the gripper, taking into account the given log diameter, $g_{zoff} = 0.125 + d/2$.

Table 4.6: Parameters of the reward function.

	Policy A	Policy B
w_d	1	1
w_o	1	1
w_t	1	1
w_l	10	5
s_d	4	4
s_o	3	3
s_t	0	0.5
s_l	0.1	0.1
ϵ	0.5	0.5
g_{zoff}	0.5	$0.125 + d/2$

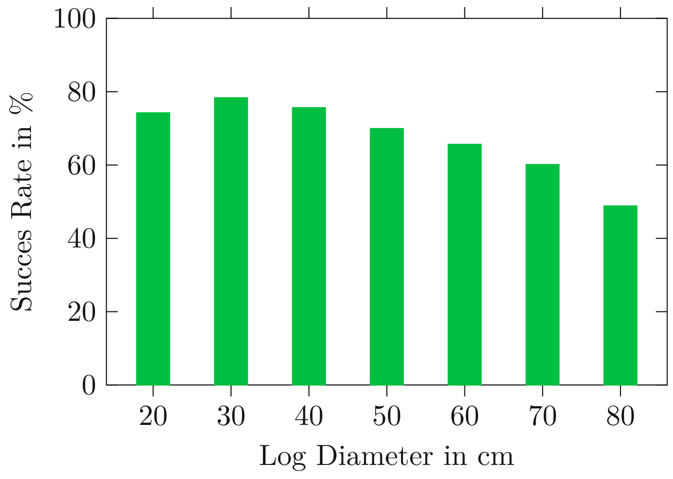


Figure 4.7: Success results of Policy A for different log diameters.

4.4 Simulation results

The AGX 2.26.1.5 package is used to create the environment of the forestry crane, see Figure 4.5. We utilize PPO [12], an on-policy algorithm, for training two policies A and B in an Intel i9 12000K CPU with 64 GB of RAM memory. The agent, i.e., the decision-making module, consists of two hidden fully connected layers. Each layer has 256 neurons. Each policy is trained over 25e6 steps by using an Adam [41] optimizer.

To evaluate two policies, the following criteria are considered. First, a successful grasp is counted if the log is lifted 10 cm above the ground. Another criterion is that not only the successful grasp is achieved, but also that a secure grip is executed. In this case, a successful episode is counted when the log is lifted and all three parts of the grapple touch the log. Evaluations are statistically reported over 10^3 episodes.

4.4.1 Policy A

The evaluation of policy A results in an overall success rate of 68.7%. For policy A, there is no direct information about the log's diameter in the observations or the reward function. Therefore, one can assume that the results are independent of the log's diameter. However, the success rates vary for different log diameters. To analyze this, the evaluation is performed for different fixed diameters. The result of this test is shown in Figure 4.7. Policy A shows the best performance for small to medium logs, i.e., $d \in [20, 40]$ cm while the performance decreases for larger logs.

A successful grasp on a large log can be seen in Figure 4.8. Therein, the jaws close around the log and create a secure grip. Another example of grasping a small log can be seen in Figure 4.9. In this example, the agent performs the pinch grasp, see Figure 4.1. Indeed, the use of the relative 4D pose (4.5) in the reward r_d (4.8) gives incentives for the agent to minimize the distance between the log's CoM and the fictive point under the

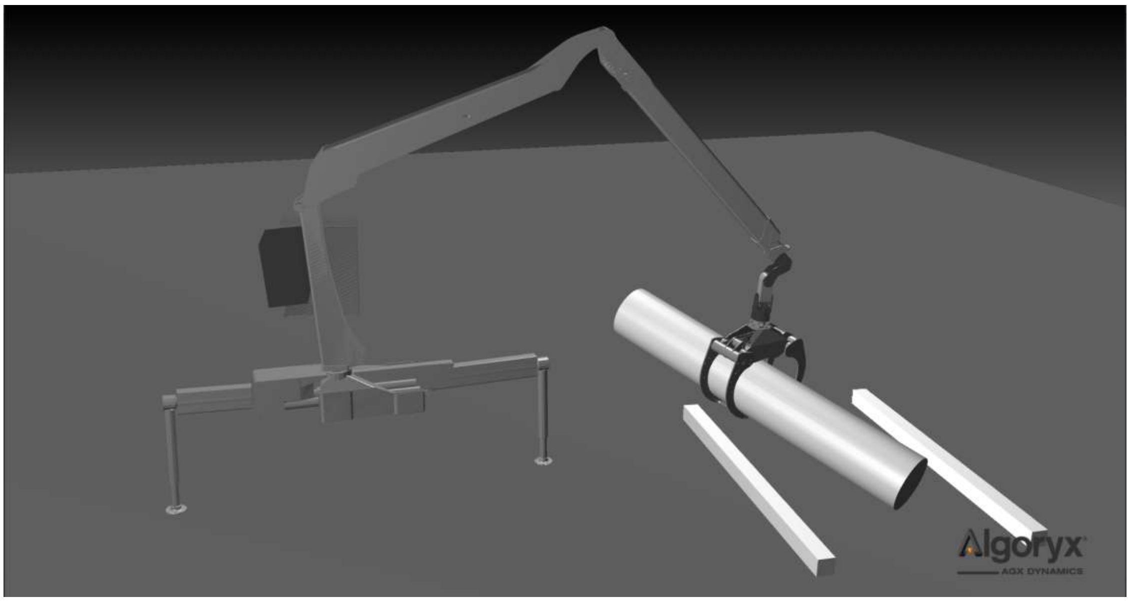


Figure 4.8: Snapshot of a successful grasp of a big log using Policy A.

grapple defined by the grapple position $\mathbf{d}_g$ (4.4) and $g_{zoff} = 0.5$. For large logs, the offset g_{zoff} matches roughly the radius of the log, whereas for small logs, the constant value of g_{zoff} encourages a gap between the log and the grapple. The agent responds to this incentives, by facilitating the pinch grasp on small logs. To obtain quantitative results on how often the problem is solved with a secure grip, policy A is tested with the second success criterion, resulting in a success rate of 16.7%.

4.4.2 Policy B

In policy B, the reward function is modified to increase the successful rate of the second criterion, i.e., secure grasp. Since the constant offset between the grapple and the center of the log makes a secure grasp difficult with small logs, a variable distance $g_{zoff} = 0.125 + d/2$ is introduced. This distance depends on the diameter of the log and defines the reference point at which the log must be located in order to be gripped securely. This addition means that the observations contain a certain amount of information about the trunk diameter, although this information is not given directly.

Training with this modified reward function leads to policy B, which achieves an overall success rate of 66.8%, close to the performance of policy A. With a total number of secure grasps of 36.4%, this strategy can significantly improve performance in terms of safe grasps using the modified reward function. The ratio of secure grasps over successful grasps is 54.5%.

While policy A shows a slight drop in performance for larger logs, the same correlation is even higher for policy B, as can be seen in Figure 4.10. For small to medium-sized logs, this policy performs much better than policy A, which could be a direct result of

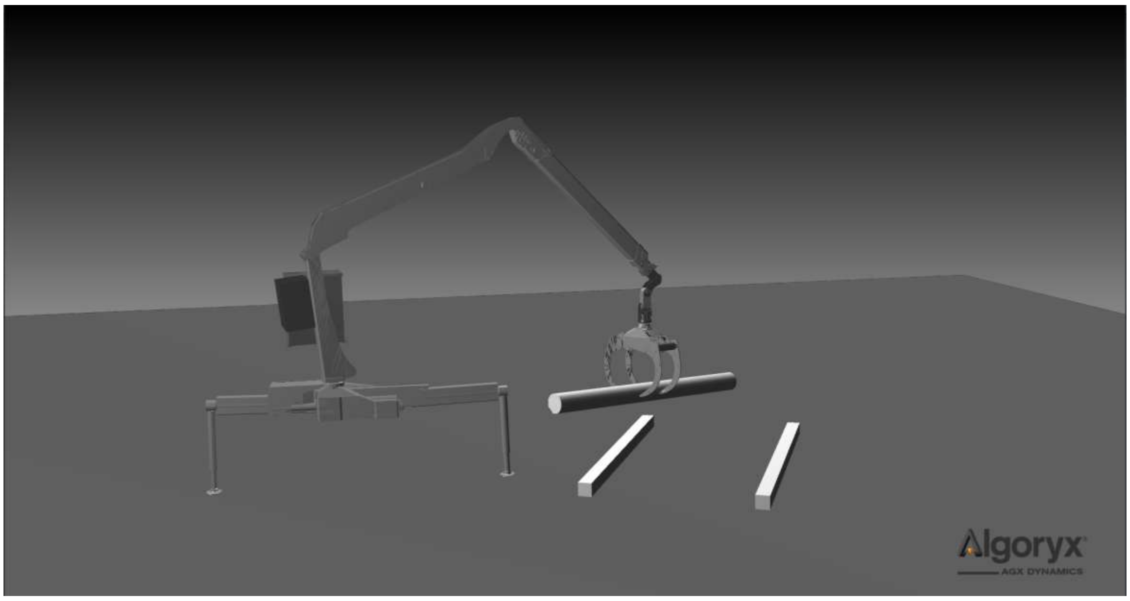


Figure 4.9: Successful grasp of a small log using policy A.

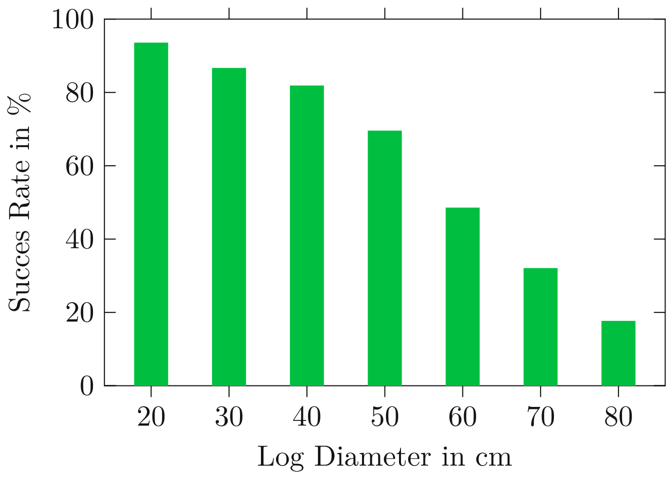


Figure 4.10: Success results of policy B for different log diameters.

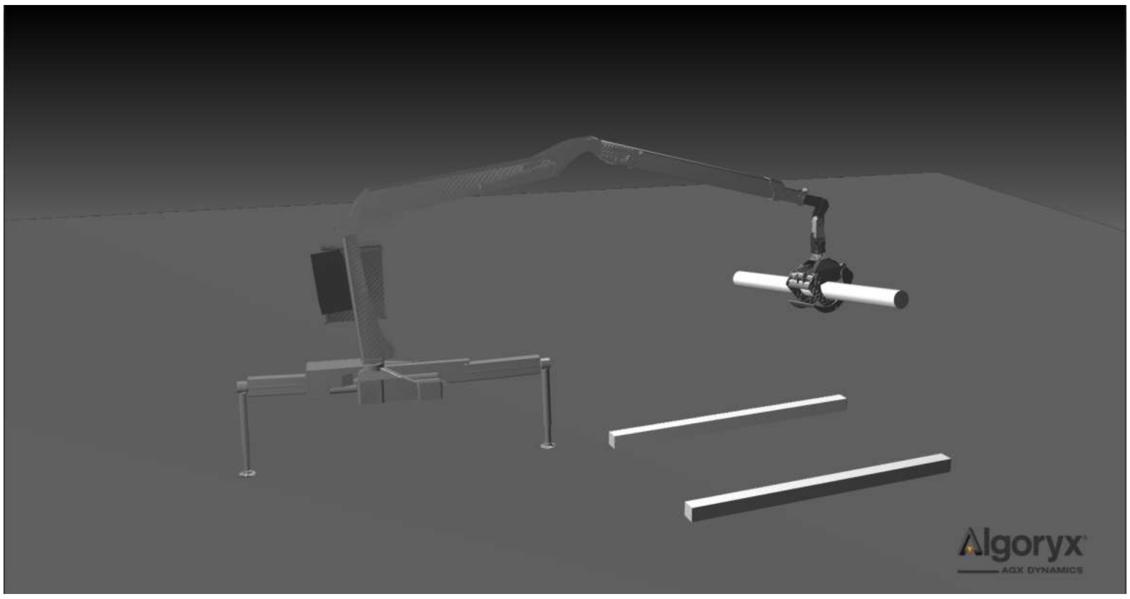


Figure 4.11: Successful grasp of a small log using policy B.

the grasping strategy for smaller logs. Policy B does not demonstrate the pinch grasp for small logs. Instead, secure grasps are mostly applied, see Figure 4.11. For larger diameters, the performance of policy B decreases rapidly. An example of a successful grasp on a large log is illustrated in Figure 4.12.

4.4.3 Discussion

Although the scenario with the log supported on the ground is common in reality, a scenario with the log lying flat on the ground is even more common. The latter is particularly interesting for applications in which the logs are collected directly from the forest floor. Since the two policies were trained for the supported log, the question arises as to how they can be transferred to a scenario with the log lying on the ground. The tests for this scenario result in an overall success rate of 58.9 % for policy A and 44.5 % for policy B. In this case, using the pinch grip on small logs could be advantageous. This is because it is easier to pinch the log than to place the jaws exactly between the ground and the log to ensure a secure hold. Figure 4.13 shows an example of how the forestry crane picks up the log in this scenario in the event of a successful grasp.

Policy B performs very poorly with the largest logs. A common behavior of the crane, when it fails, is that the grapple is not open wide enough to enclose the large circumference of large logs. In this case, the system tries to push the grapple down onto the log, which is not possible due to the collision between the jaws and the log and gets stuck in this position. An example of this is shown in Figure 4.14. This behavior could be due to the fact that the information about the diameter is only indirectly included in the observations and the gripper would be open wide enough for smaller logs.

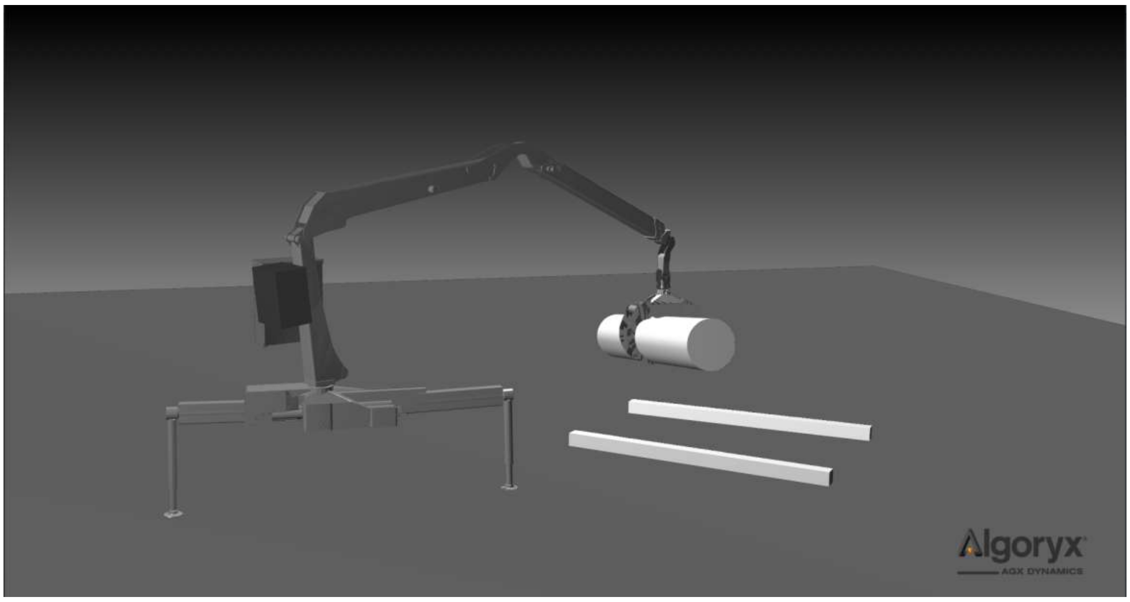


Figure 4.12: Successful grasp of a big log using policy B.

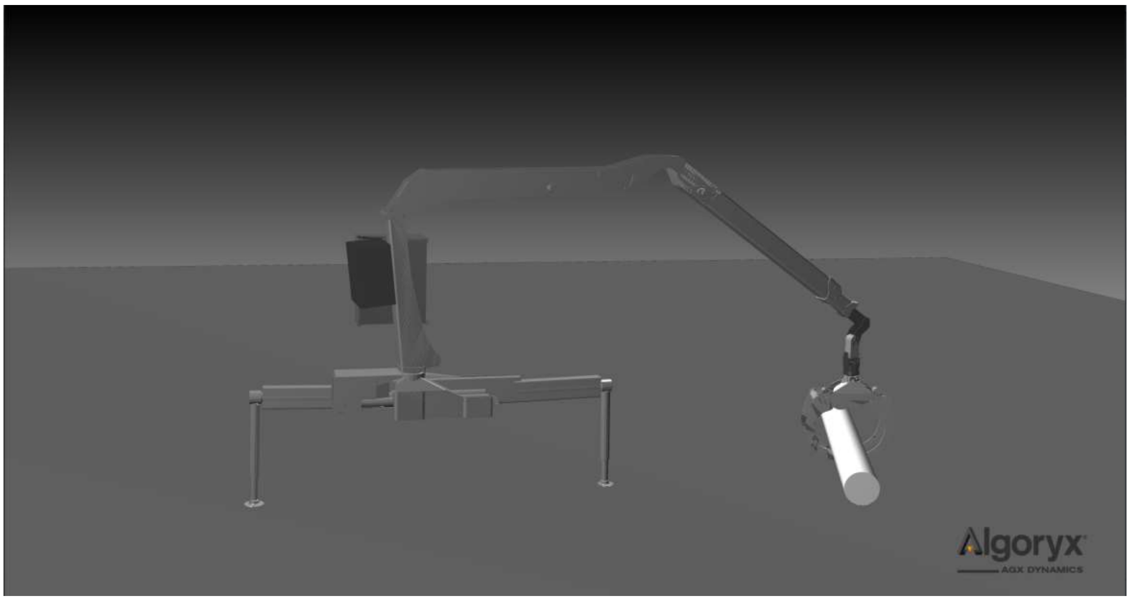


Figure 4.13: Successful grasp of a log from the ground using policy B.

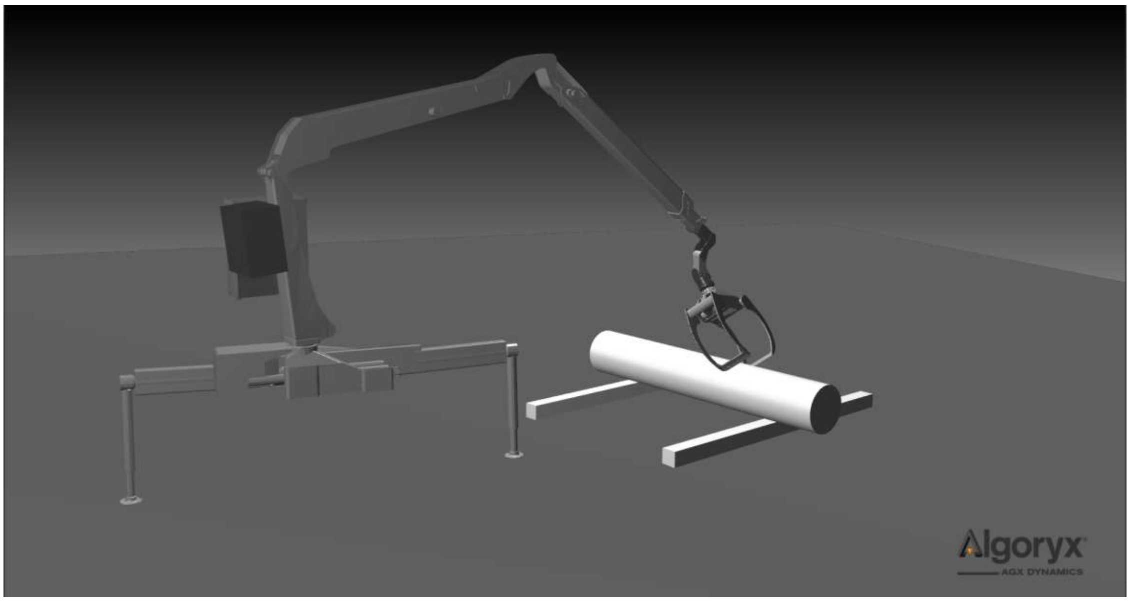


Figure 4.14: Failed grasp attempt: The grapple did not open wide enough.

Another type of failure is shown in Figure 4.15. The crane has successfully approached the log, opened the jaws wide enough, and even achieved a good grip by closing the jaws, but still remains stuck in this position instead of simply lifting the grapple together with the log. One reason for this behavior could be that the gripper is not exactly where it should be due to the relative 4D position. In this case, the grapple is already clamped to the log and the crane movement can no longer improve the relative position error, so the agent stops moving.

The third method of failure depends on the approach. If the grapple is not lowered onto the log from above, it could push the log away from the side as it approaches. Once the log is rolled off the supports, it is in a largely unexplored position from which the agent cannot recover, as shown in Figure 4.16. However, if the log remains on the supports, there is a chance that the agent can still successfully grab the log after a slight push from the side

4.5 Conclusion

The success of policy A shows the general feasibility of applying RL to the problem of grasping tree logs. The resulting policy is able to grasp logs with different diameters. Although the grasping strategy for large logs is similar to those used by expert operators, this is not the case for small logs.

With changes to the reward function, the grasping skills are altered, as shown by the results of policy B. By rewarding contact between the log and all three parts of the gripper, the agent is encouraged to use the closed grasp. Changing the calculation of the relative

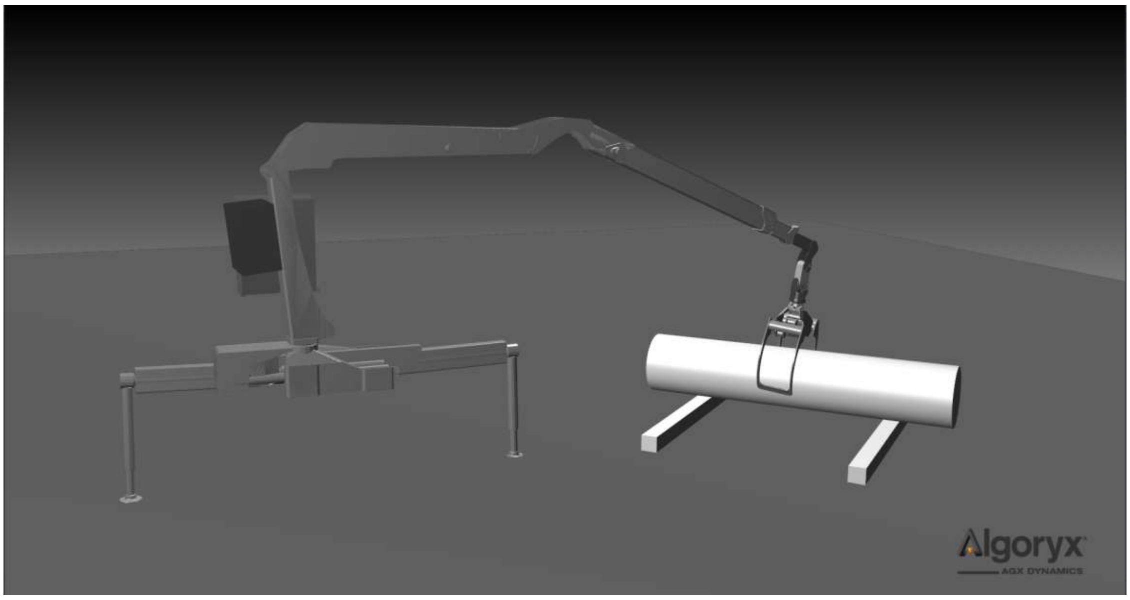


Figure 4.15: Failed grasp attempt: The crane did not lift the log, although it grasped it.

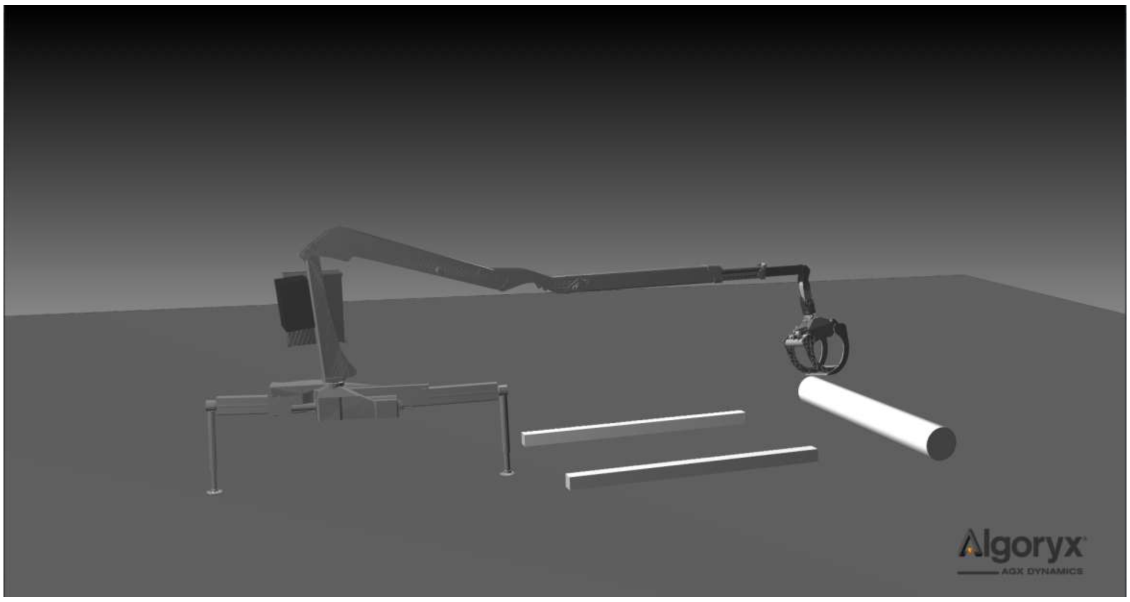


Figure 4.16: Failed grasp attempt: The grapple pushed the log away.

4D position to a diameter-dependent version removes the incentive to position the gripper in relation to the log in such a way that the pinch grasp is the best option for small logs. This changes the grasping strategy for small logs so that the pinch grasp is no longer used.

While strategy A is completely independent of log diameter, strategy B is not. In order to obtain a strategy that is independent of log diameter and behaves like strategy B in terms of the grasping strategy for small stems, a possible solution could be to use the touch reward for all grasping parts and keep the reward depending on the relative 4D appearance low compared to the other rewards.

Changing the parameters in the reward function for policy B could potentially improve the success rate for large wood logs. Setting these parameters could help the agent to overcome positions where the gripper cannot reach the log due to partially closed jaws or where the crane stops after grasping the log.

Bibliography

[1] *Holzernte früher und heute*, [Accessed on 23-October-2023], 2021. [Online]. Available: `https://biodiv-im-wald.online/holzernte-frueher-und-heute`.

[2] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* The MIT Press: Cambridge, Massachusetts, 2018.

[4] C. Szepesvári, *Algorithms for reinforcement learning.* Springer Nature: Cham, Switzerland, 2010.

[5] D. Silver, *Ucl course on rl*, [Accessed on 19-September-2023], 2015. [Online]. Available: `https://www.davidsilver.uk/teaching/`.

[6] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, England, 1989.

[7] J. Achiam, *Spinning up*, [Accessed on 19-September-2023], 2020. [Online]. Available: `https://spinningup.openai.com/en/latest/`.

[8] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[9] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[10] L. Graesser and W. L. Keng, *Foundations of deep reinforcement learning.* Addison-Wesley Professional: Boston, Massachusetts, 2019.

[11] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[14] K. H. Lundberg and T. W. Barton, "History of Inverted-pendulum Systems," *IFAC Proceedings Volumes*, vol. 42, no. 24, pp. 131–135, 2010.

[15] Y. Xu, M. Iwase, and K. Furuta, “Time Optimal Swing-up Control of Single Pendulum,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 123, no. 3, pp. 518–527, 2001.

[16] T. Glück, A. Eder, and A. Kugi, “Swing-up control of a triple pendulum on a cart with experimental validation,” *Automatica*, vol. 49, no. 3, pp. 801–808, 2013.

[17] M. N. Vu, C. Hartl-Nesic, and A. Kugi, “Fast swing-up trajectory optimization for a spherical pendulum on a 7-dof collaborative robot,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 114–10 120.

[18] C. A. Manrique Escobar, C. M. Pappalardo, and D. Guida, “A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole,” *Applied Sciences*, vol. 10, no. 24, p. 9013, 2020.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International Conference on Machine Learning (ICML)*, 2014, pp. 387–395.

[20] A. K. Pal and T. Nestorović, “Swing up and balance of an inverted pendulum using reinforced learning approach coupled with a proportional-integral-derivative controller,” in *IEEE International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2022, pp. 1–6.

[21] J. Fan, Z. Wang, Y. Xie, and Z. Yang, “A theoretical analysis of deep q-learning,” in *Learning for Dynamics and Control (L4DC)*, 2020, pp. 486–489.

[22] A. Eder, “Ausgangs- und eingangsbeschränkter steuerungsentwurf für arbeitspunktwechsel des doppel- und dreifachpendels,” M.S. thesis, Automation & Control Institute (ACIN), TU Wien, 2011.

[23] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Wiley: New York, New York, 2006.

[24] G. Brockman *et al.*, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[25] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 348–12 355, 2021.

[26] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[28] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International Conference on Machine Learning (ICML)*, 2018, pp. 1587–1596.

[29] C. Banerjee, Z. Chen, and N. Noman, “Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[30] Algoryx Simulation AB, *AGX Dynamics*, [Accessed on 17-October-2023]. [Online]. Available: `https://www.algoryx.se/agx-dynamics/`.

[31] J. Kalmari, J. Backman, and A. Visala, "Nonlinear model predictive control of hydraulic forestry crane with automatic sway damping," *Computers and Electronics in Agriculture*, vol. 109, pp. 36–45, 2014.

[32] P. L. Hera and D. O. Morales, "Model-based development of control systems for forestry cranes," *Journal of Control Science and Engineering*, vol. 2015, pp. 27–27, 2015.

[33] R. Dhakate, C. Brommer, C. Bohm, H. Gietler, S. Weiss, and J. Steinbrener, "Autonomous control of redundant hydraulic manipulator using reinforcement learning with action feedback," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 7036–7043.

[34] D. Ortiz Morales, S. Westerberg, P. X. La Hera, U. Mettin, L. Freidovich, and A. S. Shiriaev, "Increasing the level of automation in the forestry logging process with crane trajectory planning and control," *Journal of Field Robotics*, vol. 31, no. 3, pp. 343–363, 2014.

[35] P. La Hera, D. O. Morales, and O. Mendoza-Trejo, "A study case of dynamic motion primitives as a motion planning method to automate the work of forestry cranes," *Computers and Electronics in Agriculture*, vol. 183, p. 106 037, 2021.

[36] J. Andersson, K. Bodin, D. Lindmark, M. Servin, and E. Wallin, "Reinforcement learning control of a forestry crane manipulator," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 2121–2126.

[37] Y. Hurmuzlu and D. B. Marghitu, "Rigid body collisions of planar kinematic chains with multiple contact points," *The International Journal of Robotics Research*, vol. 13, no. 1, pp. 82–92, 1994.

[38] M. N. Vu, J. Lee, and Y. Oh, "Walking control algorithm of the 5-link robot based on operational space control," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, 2017, pp. 1532–1537.

[39] M. Servin and D. Wang, "Adaptive model reduction for nonsmooth discrete element simulation," *Computational Particle Mechanics*, vol. 3, pp. 107–121, 2016.

[40] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 7382–7431, 2020.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations ICLR*, 2015.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, im November 2023

Florian Lechner