UPPSALA
UNIVERSITET

IT 20 083
Examensarbete 30 hp
November 2020

Simulation-Driven Machine Learning Control of a Forestry Crane Manipulator

Jennifer Andersson

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Simulation-Driven Machine Learning Control of a Forestry Crane Manipulator

Jennifer Andersson

A forwarder is a forestry vehicle carrying felled logs from the forest harvesting site, thereby constituting an essential part of the modern forest harvesting cycle. Successful automation efforts can increase productivity and improve operator working conditions, but despite increasing levels of automation in industry today, forwarders have remained manually operated. In our work, the grasping motion of a hydraulic-actuated forestry crane manipulator is automated in a simulated environment using state-of-the-art deep reinforcement learning methods. Two approaches for single-log grasping are investigated; a multi-agent approach and a single-agent approach based on curriculum learning. We show that both approaches can yield a high grasping success rate. Given the position and orientation of the target log, the best control policy is able to successfully grasp 97.4% of target logs.
Including incentive for energy optimization, we are able to reduce the average energy consumption by 58.4% compared to the non-energy optimized model, while maintaining 82.9% of the success rate. The energy optimized control policy results in an overall smoother crane motion and acceleration profile during grasping. The results are promising and provide a natural starting point for end-to-end automation of forestry crane manipulators in the real world.

Handledare: Daniel Lindmark
Ämnesgranskare: Thomas Schön
Examinator: Stefan Engblom
IT 20 083
Tryckt av: Reprocentralen ITC

To my grandmother Dagmar; the best person I will ever know, and to whom I owe much of my inner strength and warmth of heart.

Acknowledgements

First and foremost, I would like to thank my supervisor, Daniel Lindmark, for his encouragement and continuous support throughout the work on this thesis. He consistently encouraged me to rely on my own creativity and taught me to trust my instincts. I am sincerely grateful for your enthusiasm, constructive feedback, and for all the valuable discussions we have shared along the way.

My endless gratitude is extended to everyone at Algoryx Simulation for offering me the opportunity to work on a topic close to my heart, and for providing me with a great working environment despite an ongoing pandemic. In particular, I have highly appreciated the passionate participation of Martin Servin and Kenneth Bodin. Thank you for valuable feedback, and for showing never-ending enthusiasm for the research questions I have investigated in this thesis. I would also like to acknowledge Thomas Schön as the subject reader of this thesis. His feedback has been highly valued and appreciated. Thank you.

Finally, I am indebted to my adorable nephew, Bernard, for filling the occasional weekend breaks with so much joy, despite constantly choosing to watch Frozen over videos of my automated forestry crane. I promise to teach you everything I know about reinforcement learning in the future. I also want to thank my wonderful sisters, Josefine, Johanna and Jessica, for always being there for me, believing in me, and encouraging me to follow my dreams during my years of education.

Contents

Chapter 1

Introduction

1.1 Context

The forest ecosystem is one of the world's largest, with forests covering 31% of the global surface area (FAO and UNEP, 2020). In Sweden, productive forest land constitutes 57% of the total land area, and despite globally corresponding to less than one percent of commercial forest land, the rich forest environment has enabled the national forest industry to become a world-leading exporter of timber, pulp and paper, and a main driver of the Swedish economy (Royal Swedish Academy of Agriculture and Forestry, 2015). The success is directly contingent upon the efficiency of the forest harvesting and regeneration procedures, of which the former has been highly mechanized in recent history and continues to facilitate efficient solutions with the development of more advanced forest harvesting machines and technology.

Forwarding is an essential part of the forest harvesting cycle. A forwarder is a mechanical off-road vehicle tasked with transporting timber out of the harvesting site. The key equipment is a hydraulic manipulator repeatedly undergoing monotonous pick-and-place motion to collect and redistribute logs prepared by the harvester. Despite widespread automation in industry contexts today, forwarders mainly remain manually operated. While the ambition to deploy automatic and semi-automatic solutions has been present at least since the beginning of the century (e.g. Hera et al. (2008)), the comparatively slow automation progress in the forest industry can in part be traced to the very complex and dynamic environments in which forestry cranes are utilized, which inevitably complicates the automation process. For a human operator, manual control of a forestry crane can be a both mentally and physically exhausting task, requiring counterintuitive coordination of several hydraulic cylinders for many hours straight (Hera and Morales, 2019) and exposing the operator to extensive cabin vibrations following the motion of the crane (Fodor, 2017). In 2019, motion patterns of a forwarder under operation were analyzed using motion sensors (Hera and Morales, 2019). The authors conclude that the motion patterns of the crane joints are, as expected, highly repetitive. They argue that although automation of the entire forwarding operation is complex − indeed, in addition to the repetitive motion of the forestry crane, the task involves log recognition, strategic log selection and forest navigation − automating the repetitive expanding and retracting motion of the crane can be done using analytical methods. Such semi-automation of forestry crane control has been investigated by for example Hansson and Servin (2010), who presented a solution for shared control between the operator and a computer control system in unstructured environments. The findings suggest that reduced workload and/or increased performance can be achieved using semi-automation. Thus, if the forwarding task can be automated, either fully or in part, this can relieve operators both mentally and physically, while increasing overall efficiency and productivity in the forest harvesting industry.

This thesis looks to investigate possible solutions to the forwarding automation problem. With increasing automation in industry today, there is a growing demand for physics simulation tools, as these can reduce costs, increase performance and speed up automation processes across various domains. Progress in machine learning has excelled in recent years, revolutionizing fields from computer vision to robotics, and the potential for simulation-driven machine learning control for autonomous systems development in the automotive and robotics industry is therefore evident. By extension, this includes robotic machines and vehicles in the forest industry, where safe simulation training can provide a platform for mastering complex behavior in simulated unstructured environments without the risk of damaging the physical machine. Currently, many integral questions remain in order for machine learning automation in simulated environments to excel, such as what simulation precision and robustness is possible to obtain and required for reliable transfer between simulation and reality, optimal method selection, and more.

In this project, the grasping motion of a forestry crane manipulator is fully automated in a simulated environment using a branch of machine learning known as deep reinforcement learning. This helps answer some of these questions and provides an initial step towards end-to-end automation of forestry crane manipulators in the real world. The remainder of this chapter gives an overview of related research, followed by a concise problem statement and a discussion on the limitations constraining the work presented in this thesis.

1.2 Background

The grasping motion of a robotic arm can be defined as the end-effector's motion to securely grab an object in its gripper, lift it from the ground and move it to another location. In the forest industry context, the forestry crane manipulator can be regarded as a robotic arm performing repeated grasping motion to collect and transport timber from the harvesting site. This section aims to review previous machine learning automation efforts in the context of robotic manipulation, as well as previous research related to the automation of forestry crane manipulators.

1.2.1 Forestry Crane Automation

There are several major challenges to overcome in order to automate the entire forwarding process, including autonomous navigation in dynamic forest environments, strategic log selection, object recognition, obstacle detection, path planning, grasp detection, and, of course, the grasping motion itself. A completely autonomous system also requires advanced safety systems to be in place during operation. A particular challenge that separates this grasping task from factory-floor robotic grasping even under non-moving vehicles is the very dynamic and unstructured environment in which a forestry crane is required to operate. A perfected end-to-end autonomous forestry crane requires advanced log perception systems and intelligent log selection systems. Moreover, it needs to be robust to environmental disturbances in order to compensate for vibrations, master difficult weather conditions and navigate in unfamiliar, uneven terrains.

Due to the uneven terrain in forest environments, a crane operator must learn to collect logs from multiple different vehicle configurations, as the limitations of the manipulator are dependent on the vehicle position and inclination. Optimal forwarding also includes time- and energy efficiency as well as load optimization, in which the crane can adjust the position of logs relative to other logs in order to strategically grasp configurations of multiple logs. Thus, optimized forwarding depends on external optimization criteria that must be defined a priori.

We refer to Westerberg (2014) for a more elaborate analysis of the current logging process. An important result of this analysis regards human-operated forwarding. It is shown that the majority of the time is spent on crane manipulation. Thus, the heart of the forwarding task lies in the repetitive grasping motion of the forestry crane, essentially reducing the forestry crane automation problem to a complex robotic grasping problem in a highly unstructured environment. Indeed, the unstructured environment, in combination with the redundant kinematics of the crane configuration, is what makes automation of the forwarding task much more difficult than similar manipulation tasks in controlled environments. Previous research has analysed the motion patterns of forestry cranes under operation (Hera and Morales, 2019) as well as proposed solutions for 3D log recognition and pose estimation (Park et al., 2011), both important building blocks for future automation of forestry crane manipulators using visual sensory information. Mettin et al. (2009) showed that automation efforts indeed can increase performance compared to manual operation, suggesting that the full potential in the forwarding task is not met without some degree of automation.

Several successful semi-automation approaches, focused on trajectory planning and motion control assisting the crane operator, have been investigated in order to increase productivity and learning speed of the operator and reduce unnecessary workload, see for example Hansson and Servin (2010), Westerberg (2014) and Fodor (2017).

So far semi-autonomous solutions have been restricted to guiding and complementing the operator in routine tasks such as controlling the crane to the grasping position along a trajectory, while trajectory tuning, the grasping and releasing of logs as well as the intelligent analysis of the surroundings required for log selection and forest navigation are left to a manual operator. Due to the complexity of the forwarding task in the inevitably unstructured forest environment, end-to-end automation completely eliminating the involvement of a human operator has been viewed as a far-off utopia. However, recent advances in machine learning, including reinforcement learning agents that learn from experience and can be trained in a simulated environment, may rekindle these ambitions, or at the very least accelerate semi-automation efforts.

1.2.2 Machine Learning for Robotic Grasping

Robotic factory floor pick-and-place motion has been largely mastered through analytical, purpose-specific control algorithms. However, grasping in unstructured environments remains an open problem in robotics today. Deriving analytical algorithms is tedious and may prove impossible in many of the target contexts, for example due to object occlusion and varying object properties, backgrounds and illumination. This makes object identification from visual input data difficult, and a general algorithm must be adaptive to constantly changing environments. Alleviating this challenge, recent advances in the application of machine learning to areas such as computer vision have inspired progress within the area of robotics. This has proved important to the evolution of robotic grasping and grasp detection (e.g. Caldera et al. (2018)).

Robotic grasping can be divided in two primary steps; grasp detection and grasp planning. The previous determines the grasping pose and the latter refers to the process of determining the robotic path enabling a successful grasp, i.e. mapping the coordinates of the grasping region in the image plane to the coordinate system of the robot (Bicchi and Kumar, 2000). Finally, the planned trajectory is executed using a control algorithm. Deep convolutional neural networks (DCNN) are the most common deep learning architectures that have been used for grasp detection with input data from visual sensors, as argued in a review of deep learning methods in robotic grasping by Caldera et al. (2018). They conclude that the one-shot method, where the grasping region representation is found through DCNN regression, is the most promising in terms of real-time grasp detection, based on the research available at the time of their review.

There are multiple examples of successful applications of CNN's in grasp detection (e.g. Kumra and Kanan (2017)). In this case, visual input is often complemented with depth information using RGB-D images. Such deep learning methods assume that there is enough annotated data, including domain specific data, for the model to generalize well. For general grasp detection, researchers commonly use datasets such as the Cornell Grasp Dataset (Lenz et al., 2015), a labelled dataset of RGB-D images for single-object grasp detection. An analytical way to solve this problem is to use 3D model reasoning to detect the grasping regions in the training dataset. Compared to identifying the grasping regions from visual perception data, however, this is complicated and assumes that important physical properties of the object, such as the mass distribution and force profile, are known (Pinto and Gupta, 2016). Moreover, annotated datasets run the risk of not being general enough, and manual prediction of optimal *robotic* grasping poses may not be straightforward.

Empirical methods that rely on experience-based learning, through trial-and-error or demonstration, escape this challenge altogether. Such systems either require separate models for grasp detection and grasp planning, respectively, or merge the steps using a visuomotor control policy. Pinto and Gupta (2016) used a self-supervised approach inspired by the core of reinforcement learning, i.e. learning by trial-and-error, to train a CNN for grasp detection. Levine et al. (2016) developed a model combining learning of the perception and control systems using a guided policy search method to learn policies mapping the visual perception data to the robot motor torques in a single step. Their results showed significant performance improvement compared to non-end-to-end methods.

The main obstacle in applying supervised deep learning to grasp detection tasks is the lack of domain specific annotated training data, necessary to enable sufficient model generalization. Simulated data can partly solve this problem. For example, Viereck et al. (2017) designed a closed-loop controller for robotic grasping using sensor training data gathered entirely through simulation. They trained a CNN to learn a distance function to true grasps from the image data. The closed-loop control approach enables dynamically guiding the gripper to the target object, thus allowing for adaptation to environmental disturbances. This is an essential challenge to overcome in order to master robot manipulation in unstructured environments.

Supervised learning methods still require large labelled datasets which are difficult and time-consuming to produce. In light of this, interest in applying reinforcement learning for robotic grasp detection has increased. In the reinforcement learning framework, an agent learns from experience by receiving rewards or penalties for desired or undesired behavior while navigating through its environment. Thus, training data is assembled in real-time by the learning system itself. See Chapter 2 for a thorough introduction to the field of reinforcement learning.

Of course, this approach requires trial-and-error and involves high risk of damaging a physical agent. In a simulated environment, however, the agent can learn from repeated experience in a secure, virtual setting and the final knowledge can be transferred to the physical agent post training, depending on how well the simulated and physical environments correlate. Methods to ease model transfer between the simulated and real world have been explored by for example Tobin et al. (2017), in the particular case in terms of domain randomization for a deep neural network model used for robotic grasping.

In review literature covering deep learning methods in robotic grasp detection (Caldera et al., 2018), the authors conclude that applying reinforcement learning in the prediction of visuomotor control policies, or directly in the grasp detection problem, is a largely unexplored territory with promising potential when simulated environments can be utilized to speed up and mitigate damage in the training process. One of the main advantages of the reinforcement learning framework is its potential for end-to-end learning, where stable grasping regions can be learnt through trial and error without labelled datasets. Moreover, the sequential properties of the problem are naturally taken into consideration, enabling correction for dynamics in the environment through continuous strategy tuning, for example aiding the grasping process with pre-grasp object manipulation.

While reinforcement learning in complex environments often suffers from low sample efficiency and other dimensionality issues, combining the framework with deep learning, through function approximation and representation learning (see e.g. Lesort et al. (2018)), has accelerated progress in various areas, including that of robotic grasping. Recent *deep reinforcement learning* achievements in the field of robotic grasping include vision-based robotic grasping using two-fingered grippers (e.g. Quillen et al. (2018), Kalashnikov et al. (2018) and Joshi et al. (2020)) and dexterous multi-fingered grippers (Rajeswaran et al., 2018). Exemplifying the potential in the field, Kalashnikov et al. (2018) shows that vision-based reinforcement learning can yield models exhibiting promising generalization in both simulated and real-world grasping, also managing regrasping of dynamic objects and other non-trivial behavior that is required for success in unstructured environments.

1.3 Objective

In this thesis, the grasping motion of a hydraulic-actuated forestry crane manipulator with redundant kinematical structure is fully automated in a simulated environment. The main purpose is to increase knowledge of how machine learning methods can be applied in the development of physics-based simulation tools for industry automation in general, and forest crane manipulation in particular. Specifically, the potential of using *deep reinforcement learning* methods in simulation-driven end-to-end automation of a forestry crane manipulator is explored.

As discussed, full automation of the forwarding process is an overwhelmingly complex task. To this end, we limit our initial work to automation of the single-log grasping motion of a forestry crane manipulator mounted on a static vehicle on a fixed, horizontal surface. A perfected reinforcement learning agent could ideally perform the grasping task using solely visual sensory signals capturing the scene and sensory signals providing information on the actuator states. For simplicity, our work is limited to a smaller observation space, including only the state of the actuators and the position and orientation of the target log. Thus, the existence of an external perception system is assumed. This provides a natural starting point for future research.

The final outcome is a prototype of a simulated forestry crane manipulator, automated to perform single-log grasping under the preceding conditions using state-of-the-art deep reinforcement learning techniques, in particular the empirically stable Proximal Policy Optimization algorithm (Schulman et al., 2017). Two automation strategies are investigated; a multi-agent approach separating the task into the two subtasks of navigating to and grasping the log, and a single-agent approach using curriculum learning to achieve full automation of the grasping task. This is done in a simulated environment using the Unity 3D simulation platform (Juliani et al., 2020) together with the high-accuracy simulation and modeling SDK, AGX Dynamics[1].

1.4 Contribution

Though semi-automation has been explored before in the context of forestry crane manipulation, to the best of our knowledge, simulation-driven machine learning control of forestry crane manipulators is a topic previously not touched upon in machine learning or robotics research. Thus, our contribution is the first implementation of deep reinforcement learning control of a forestry crane manipulator.

1.5 Collaboration

This research is carried out in collaboration with Algoryx Simulation, a company based in Umeå, Sweden, focusing on the development of advanced physics simulation software. Founded in 2007, Algoryx Simulation has quickly become a leading provider of visual and interactive multiphysics simulation software and services. Their simulation engine, AGX Dynamics, lies at the core of this project. It is a physics-based simulation SDK enabling high-accuracy, real-time simulations of complex mechanical systems, thereby contributing to narrowing the gap between dynamical multibody system simulation and reality. Today, this simulation technology is used in a broad variety of applications ranging from product development and virtual deployment to system optimization, simulation training and engineering analysis in the automotive and robotics industry.

[1]https://www.algoryx.se/agx-dynamics/

Chapter 2

Theory

This section introduces the reinforcement learning framework, providing background and context to the method used in this thesis. We begin by introducing the theory behind Markov decision processes, and move forward discussing common exact solution methods. Next, we introduce the field of reinforcement learning, arriving at Proximal Policy Optimization; the state-of-the-art algorithm carrying the results produced in this thesis. The chapter is concluded with a discussion on reinforcement learning techniques that can be used to tackle particularly complex reinforcement learning problems.

2.1 Markov Decision Processes

The art of mastering automated sequential decision making in unstructured environments spans a variety of domains; from decision-theoretic planning and reinforcement learning to operations research, control theory and economics. Though domain-specific issues naturally persist, many such problems can, at least at a conceptual level, be formally described as *Markov decision processes* (Boutilier et al., 1999).

A Markov decision process is a mathematical framework formalizing sequential decision making in stochastic state-transition systems. The control of such stochastic, dynamical systems involves a decision maker, commonly referred to as the agent, interacting with its environment through actions and rewards. Through its actions, which influence the environment but not exclusively in ways fully predictable, the agent carries the system through a random sequence of states. The underlying objective of the agent is to maximize its utility for a specified purpose, such as bringing the system to a desired state. Markov decision processes can be either time-discrete or time-continuous, allowing the decision maker to make decisions at discrete or continuous time intervals. Which time-discretization better models the behaviour of a system depends on the system properties.

This section begins with a mathematical definition of the Markov decision process, which will be referred to as *MDP* in the remainder of this thesis, and continues with a discussion on the Markov decision problem and its solution. We restrict ourselves to the discrete-time and discrete and finite state- and action space Markov decision process, but the formalization can be extended to include continuous-time Markov decision processes with infinite state- or action spaces. For the interested reader, extensive literature has been written on the topic and a more in-depth overview of the elegant theory of Markov decision processes is provided by for example Puterman (1994).

2.1.1 Definition

A *Markov decision process* is a 4-tuple $\mathcal{M} := \big\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \big\rangle$, where $\mathcal{S}$ denotes the *state space*, $\mathcal{A}$ denotes the *action space*, $\mathcal{T}$ denotes the *transition probability function*, and $\mathcal{R}$ denotes the *reinforcement* or *reward function* An MDP is therefore defined by a set of states $s \in S$ and a set of actions $a \in \mathcal{A}$, as well as the transition probability function $\mathcal{T}$ and the reinforcement function $\mathcal{R}$, such that $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$.

Based on this definition, we note that spaces $\mathcal{S}$ and $\mathcal{A}$ are system properties, whereas $\mathcal{R}$ and $\mathcal{T}$ are model properties. Each element of $\mathcal{M}$ is described below:

i State space

The state space $\mathcal{S}$ is a set of all possible states s in the system. Typically, each state is a collection of important environment features needed to model the system in that particular state. Possible board configurations is a basic example of a state space in the board game context, where each state is constituted by its board configuration.

ii Action space

The action space $\mathcal{A}$ is a set of all possible actions a in the system. In each state, the decision maker, or agent, can choose an action from the entire set of actions, or, depending on the system, a subset of actions specific to the current state. Extending the board game example, the action space consists of all possible actions the player can take, which may vary depending on the state, i.e. current board configuration.

iii Transition probability function

Given a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, the system moves into a subsequent state $s' \in \mathcal{S}$. Aptly denominated, the transition probability function $\mathcal{T}(s, a, s')$ controls these state transitions by providing the proper probability distribution over all possible subsequent states s'. Thus, given any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the subsequent state $s' \in \mathcal{S}$ is determined by the transition probability function $\mathcal{T}(s_t, a_t, s_{t+1}) = P\big\{s_{t+1} = s' \mid s_t = s, a_t = a\big\} = P\big\{s_{t+1} = s' \mid s_t = s, a_t = a, s_{t-1}, a_{t-1}, ..., s_{t-N}, a_{t-N}\big\}$, where N is the number of preceding time steps and s_t denotes the state s at time t. Evidently, the state transition in only dependent on the currently visited state and currently applied action, i.e. it satisfies the *Markov property.* In this, each state is assumed to be *fully observable*, an often optimistic assumption (Arulkumaran et al., 2017). The theory of partially observable MDP's (POMDP) is omitted in our discussion on MDP's, but has been discussed for example by (Kaelbling et al., 1998).

iv Reward function

The reward function $\mathcal{R}$ specifies a scalar feedback signal that depends on the current state, action or state transition. This feedback signal is referred to as the reward. Here, we limit the discussion to deterministic reward functions based exclusively on actions and state transitions; $\mathcal{R}(s, a, s')$. Depending on its sign, the reward aims to encourage or discourage certain state transitions, thus controlling the target system evolution. Returning to our board game example, a simple reward function grants the decision maker a positive reward for state transitions to winning states, negative reward for the corresponding transitions to losing states, and zero reward for transitions to remaining states. In this way, the reward function is designed to specify the goal of the decision maker and guide the learning process.

To summarize the discrete-time MDP, we let s_t be the system state at time t. Given s_t at any given time step, the agent takes an action a_t, causing the system to move to the subsequent state s_{t+1}, sampled from the probability distribution $\mathcal{T}(s, a, s')$, and the agent to receive a reward $r_{t+1} = \mathcal{R}(s_t, a_t, s_{t+1})$. This is repeated until a terminal state is reached, or until the system has been modelled for a finite or infinite number of time steps.

In finite-time MDP's, an *episode* is defined as the time between the initial and terminal states. In the episodic task, the initial state is sampled from an initial state distribution, and the terminal state is commonly characterized by $\mathcal{T}(s, a, s') = 1$ and $\mathcal{R}(s, a, s') = 0$ for all $s' \in \mathcal{S}$ and $a \in \mathcal{A}$. This allows for treating episodic tasks similarly to continuing tasks mathematically, implying that our mathematical discussion in the following section holds for both types of MDP's.

2.1.2 Solution

Solving an MDP is a question of finding the *optimal policy* π^* in order to maximize the *cumulative reward*, or *return.* A deterministic *policy* π controls the agent's decision making process by mapping each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$; $\pi : \mathcal{S} \to \mathcal{A}$. A fixed, optimal policy π^* therefore yields a stochastic transition system where the distribution over states is stationary.

Note that the policy is not necessarily deterministic. In fact, the deterministic policy can be viewed as a special case of the stochastic policy, where only one action is performed with non-zero probability in each state. In general, the stochastic policy is defined by $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(a|s) \geq 0$ and

$\sum_{a \in \mathcal{A}} \pi(a|s) = 1$ for each state $s \in \mathcal{S}$. Thus, $\pi(a|s)$ is the probability that the agent takes action a in state s.

To determine the optimal policy, an optimality criteria needs to be defined. An MDP coupled with such a criteria is known as the *Markov decision problem* (Littman et al., 1995), to which the optimal solution is the optimal policy π^*. We will focus on a common optimality criteria in which the agent seeks to maximize the expectation of the discounted return defined according to (2.1). This is known as the *discounted, infinite-horizon* optimality criteria.

$$R_t = \sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau+1} \tag{2.1}$$

Here, an agent currently in state s_t aims to maximize the expectation of the discounted cumulative reward R_t, where $r_{t+\tau+1} = \mathcal{R}(s_{t+\tau}, a_{t+\tau}, s_{t+\tau+1})$ is the reward at each subsequent time step $t + \tau$ and $\gamma \in [0, 1)$ is the exponential discount factor.

The discount factor enforces larger weight to earlier rewards, and is often 1 in finite-horizon systems, i.e. episodic systems. If $\gamma^\tau = 0$, the optimality criteria is reduced to maximizing the expected immediate reward at each time step.

To find the optimal policy, each state is given a *value* or *state-action value*, through the value function $V : \mathcal{S} \to \Re$ or action-value function $Q : \mathcal{S} \times \mathcal{A} \to \Re$. Using the definition of the return specified in (2.1), $V^\pi(s)(2.2)$ denotes the expected return from being in state $s \in \mathcal{S}$, following the policy π. Similarly, $Q^\pi(s, a)(2.3)$ denotes the expected return from taking action $a \in \mathcal{A}$ while in state $s \in \mathcal{S}$, following the policy π. The state value function allows for policy evaluation, whereas the state-action value function carries information on which action maximises the expected return at a particular state.

$$V^\pi(s) = E_\pi\Big[\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau+1} \mid s_t = s\Big] \tag{2.2}$$

$$Q^\pi(s, a) = E_\pi\Big[\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau+1} \mid s_t = s, a_t = a\Big] \tag{2.3}$$

Using the recursive properties of the formulation, (2.2) can be reduced to depend only on immediate rewards and values of possible subsequent states s' under the policy π. Expanding (2.2) and applying the law of total expectation, we arrive at (2.4) and, the *Bellman equation* for the state-value function.

$$\begin{aligned} V^\pi(s) &= E_\pi\Big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s\Big] \\ &= E_\pi\Big[r_{t+1} + \gamma R_{t+1} \mid s_t = s\Big] \\ &= E_\pi\Big[r_{t+1} \mid s_t = s\Big] + \gamma E_\pi\Big[R_{t+1} \mid s_t = s\Big] \\ &= E_\pi\Big[r_{t+1} \mid s_t = s\Big] + \gamma E_\pi\Big[E_\pi\Big\{R_{t+1} \mid s_{t+1} = s'\Big\} \mid s_t = s\Big] \\ &= E_\pi\Big[r_{t+1} + \gamma V^\pi(s') \mid s_t = s\Big] \end{aligned} \tag{2.4}$$

Similarly, the Bellman state-action value equation can be derived from (2.3) according to (2.7). Here, a' denotes actions taken at the next state s'.

$$\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\Big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s,\ a_t = a\Big] \\
&= E_{\pi}\Big[r_{t+1} + \gamma R_{t+1} \mid s_t = s,\ a_t = a\Big] \\
&= E_{\pi}\Big[r_{t+1} + \gamma \sum_{a'} \pi(a'|s') E_{\pi}\Big\{R_{t+1}|s_{t+1} = s', a_{t+1} = a'\Big\}|s_t = s, a_t = a\Big] \\
&= E_{\pi}\Big[r_{t+1} + \sum_{a'} \pi(a'|s') Q^{\pi}(s',a') \mid s_t = s', a_t = a'\Big]
\end{aligned} \tag{2.5}$$

For a stochastic policy, the state value function and the state-action value function can be expanded into (2.6) and (2.7)

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')\Big) \tag{2.6}$$

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma Q^{\pi}(s',a')\Big) \tag{2.7}$$

We realize that the state value is the expectation value of the state-action value, averaged over possible actions and weighted by their probabilities, i.e. $V^{\pi}(s) = \sum_a \pi(a \mid s) Q^{\pi}(s,a)$ for each $s \in S$.

The Bellman state-value equation has remarkable implications, as its simplicity promises that the calculation of one state-value only depends on possible next state-values, as opposed to all subsequent state-values.

Now that the value function and state value functions are defined, we can define the optimal policy π^*. Given two stationary policies π_1 and π_2, π_1 is considered superior to π_2, i.e. $\pi_1 \geq \pi_2$ where $\pi_1 > \pi_2$ holds for at least one state, if and only if $V^{\pi_1}(s) \geq V^{\pi_2}(s)\ \forall\ s \in S$, and $V^{\pi_1}(s) > V^{\pi_2}(s)$ holds for at least one state. Thus, finding the optimal value function $V^*(s) = \max_\pi V^{\pi}(s)$ yields the optimal policy π^*, which dominates or equals all other policies π. It can be shown that at least one such optimal policy exists (Bellman, 1957). To find the optimal value function from our definition of the value function (2.6), we simply choose the action yielding the maximum value. The resulting expression is presented in (2.8), which is known as the *Bellman optimality equation.*

$$V^*(s) = \max_a \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^*(s')\Big) \tag{2.8}$$

Based on the previous definition, this gives the value for each state $s \in S$, following the optimal policy π^*. The optimal policy π^* can then be defined according to (2.9).

$$\pi^* = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^*(s')\Big) \tag{2.9}$$

To complete our discussion, we define the optimal action-state value function in a similar way (2.10).

$$Q^*(s) = \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma \max_{a'} Q^*(s',a')\Big) \tag{2.10}$$

Comparing (2.10) to (2.8), it is evident that $V^*(s) = \max_a Q^*(s,a)$. This yields a more elegant expression for the optimal policy, given in (2.11). Using this expression, solving an MDP by finding the optimal policy π^* is reduced to finding the optimal state-action value function $Q^*(s,a)\ \forall(s,a) \in \mathcal{S} \times \mathcal{A}$.

$$\pi^* = \operatorname*{argmax}_a Q^*(s,a) \tag{2.11}$$

2.2 Dynamic Programming

Solving a Markov decision problem amounts to finding the optimal policy π^* (2.9, 2.11) for the MDP given an optimality criteria, as discussed in section 2.1. *Dynamic programming* is a name given to a collection of model-based MDP solution algorithms requiring full knowledge of the environment. Of course, if such a comprehensive model of the environment exists, the linear system of $|S|$ equations, constituted by (2.6) $\forall s \in \mathcal{S}$, can be used to solve directly for the optimal value function $V^*(s)$. In fact, linear programming is an exact method that achieves this by solving the optimization problem of minimizing $\sum_s V(s)$ subject to $V(s) \geq \max_a \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V(s')\Big)\ \forall s \in \mathcal{S}$ and $a \in \mathcal{A}$ (Sanner and Boutilier, 2009). However, most algorithms, including dynamic programming algorithms, lend themselves to iterative methods to find the optimal value function and policy.

Dynamic programming algorithms integrate *policy evaluation*, finding the value function given a policy, with *policy improvement*, improving said policy, in different ways to find the optimal policy π^*. In this section, two principle dynamic programming algorithms are discussed; *policy iteration* and *value iteration.* The former was originally proposed by Howard (1960) and the latter by Bellman (1957), and both algorithms have since given rise to a range of modified and approximate versions (see e.g. Puterman and Shin (1978), Bertsekas and Tsitsiklis (1996) and Scherrer et al. (2012)). As before, we assume discrete-time MDP's with discrete and finite state- and action spaces in our discussion of these solution methods.

2.2.1 Policy Iteration

The *policy iteration* algorithm (Howard, 1960) iterates between policy evaluation and policy improvement. Policy evaluation involves finding the value function V^π for the policy π. We repeat the Bellman equation for the value function (2.6) in (2.12), adding the iteration index n and assuming a deterministic policy

π_m. Updating $V_n^{\pi_m}(s)$ iteratively ∀s $\in S$ in this way, it converges to $V^{\pi_m}(s)$ ∀s $\in S$ as $n \to \infty$.

$$V_{n+1}^{\pi_m}(s) = \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V_n^{\pi_m}(s')\Big) \tag{2.12}$$

The next step is to improve the policy. To do this, $Q^{\pi_m}(s,a)$ is obtained by evaluating $V^{\pi_m}(s)$ $\forall a \in A$ in each state $s \in S$ (2.13). If an action $a \in A$ exists such that $Q^{\pi_m}(s,a) \geq Q^{\pi_m}(s, \pi_m(s))$, the current policy π_m is updated, and the process is repeated with the improved policy π_{m+1} (2.14)

$$Q^{\pi_m}(s,a) = \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^{\pi_m}(s')\Big) \tag{2.13}$$

$$\pi_{m+1}(s) = \operatorname*{argmax}_a Q^{\pi_m}(s,a) \tag{2.14}$$

This repeated until the value function approximation converges to the optimal value function $V^*(s)$, and the optimal policy π^* has been obtained.

The value function can be shown to increase monotonically in each policy improvement iteration. Thus, given finite state and action spaces $\mathcal{S}$ and $\mathcal{A}$, there is an upper bound $|\mathcal{A}|^{|\mathcal{S}|}$ to the possible number of policies and thus the maximum number of iterations required until convergence, suggesting that policy iteration converges in a finite number of steps. In practice, the algorithm often converges a lot faster, balanced by its comparatively high complexity per iteration (Santos and Rust, 2004).

The computational algorithm for policy iteration is summarized in Algorithm (1), where σ denotes some specified tolerance for convergence.

Algorithm 1: Policy Iteration

```
Result: The optimal policy $\pi^*$
1. Initialization
   Initialize $V(s) \in \Re$ and $\pi(s) \in A$ arbitrarily $\forall s \in \mathcal{S}$.
2. Policy Evaluation
   while $\Delta \geq \sigma$ do
       $\Delta := 0.$
       for each $s \in \mathcal{S}$ do
           $v := V^\pi(s)$
           $V(s) := \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V(s')\Big)$
           $\Delta := \max(\Delta, |v - V(s)|)$
3. Policy Improvement
   for each $s \in \mathcal{S}$ do
       $\widetilde{\pi}(s) := \pi(s)$
       $\pi(s) := \text{argmax}_a \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V(s')\Big)$
       if $\widetilde{\pi}(s) \neq \pi(s)$ then return to 2; else $\pi^*(s) := \widetilde{\pi}(s)$
```

2.2.2 Value Iteration

Full convergence of the value function in the policy evaluation step at each iteration is not required for convergence to the optimal policy. To this end, Bellman (1957) introduced the *value iteration* algorithm, for which a single policy evaluation iteration suffices. Hence, the policy evaluation and policy improvement step are completely merged.

As per policy iteration, $V_n(s)$ is updated iteratively $\forall s \in \mathcal{S}$, though incorporating the policy improvement immediately according to (2.15). Like the policy iteration algorithm, it can be shown to converge to $V^*(s) \forall s \in \mathcal{S}$ as $n \to \infty$.

$$V_{n+1}(s) = \max_a \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V_n(s')\Big) \tag{2.15}$$

The value iteration algorithm can be shown to converge linearly to the optimal value function (e.g. Puterman (1994)). Puterman (1994) also proves that if the iterations are terminated under the tolerance $\sigma = \max_s |V_n(s) - V_{n-1}(s)| = \frac{\epsilon(1-\gamma)}{2\gamma}$, then the obtained value function $V_n(s)$ fulfills $\max_s |V_n(s) - V^*(s)| < \epsilon$, i.e. the value iteration algorithm converges to the ϵ-optimal value function.

Once the optimal value function is found, the optimal, deterministic policy π^* can be obtained from (2.16) as before. The resulting value iteration algorithm is summarized in Algorithm (2).

$$\pi^* = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^*(s')\Big) \tag{2.16}$$

Algorithm 2: Value Iteration

```
Result: The optimal policy $\pi^*$
1. Initialization
    Initialize $V(s) \in \Re$ arbitrarily $\forall s \in \mathcal{S}$.
2. Value Iteration
    while $\Delta \geq \sigma$ do
        $\Delta := 0$.
        for each $s \in \mathcal{S}$ do
            $v := V(s)$
            for each $a \in \mathcal{A}$ do
                $Q(s,a) := \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V(s')\Big)$
            $V(s) := \max_a Q(s,a)$
            $\Delta := \max(\Delta, |v - V(s)|)$
    for each $s \in \mathcal{S}$ do
        $V^*(s) := V(s)$
3. Policy Determination
    for each $s \in \mathcal{S}$ do
        $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} \mathcal{T}(s,a,s')\Big(\mathcal{R}(s,a,s') + \gamma V^*(s')\Big)$
```

2.2.3 Computational Complexity

Many Markov decision problems require large state spaces, in which the efficiency of the algorithms discussed in this section is questionable in practice. Consider a game like chess, where there are approximately 10^{43} possible states (Shannon, 1950). Finding the optimal value function and/or policy in this case would be considered computationally expensive even with a computational complexity linearly dependent on the state space, and not considering the number of iterations required for convergence.

Denoting the state and action spaces as before, the computational complexity of each iteration of the algorithms discussed so far is $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$ for value iteration and $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2 + |\mathcal{S}|^3)$ for policy iteration (Littman et al., 1995). Littman et al. (1995) shows that at worst, the run time of value iteration can grow faster than $\frac{1}{1-\gamma}$, where γ per usual denotes the discount factor. Policy iteration typically converges faster, but the number of iterations can still grow to be very large depending on the problem (Santos and Rust, 2004). Previous work has aimed to mitigate these issues by improving the algorithms in different ways. Such methods include the adoption of search algorithm elements in which only a relevant fraction of the entire state space is visited, and the adoption of asynchronous updating schemes through different versions of modified policy iteration (Wiering and van Otterlo, 2012). Reinforcement learning, to which the following section is dedicated, is another collection of MDP solution methods that have proven successful in many large-scale applications.

2.3 Reinforcement Learning

Though exact methods like the linear and dynamic programming algorithms discussed in Section 2.2 provide simple and beautiful model-based solutions to Markov decision processes, they rely on the assumption that full knowledge of the environment is accessible, which is often not the case. Moreover, even when a complete environmental model does exist, many large-scale problems in the field of sequential decision making require state spaces too large for these algorithms to be computationally feasible, as briefly touched upon at the end of the last section. This becomes evident when the exponential growth of the number of states with the number of state variables is considered. Consequently, the state spaces of complex problems can quickly become very large.

The aforementioned problems, sometimes described as results of the *curse of modeling* and the *curse of dimensionality* (Gosavi, 2004), are often tackled using an assembly of methods collectively known as *reinforcement learning.* The field of reinforcement learning has seen major advances in recent decades, providing successful adaptive control algorithms through a combination of concepts from fields such as dynamic programming, stochastic approximation and function approximation. More recently, the adoption of function approximation paradigms like deep learning has begun revolutionizing the scale of problems that can be mastered by reinforcement learning techniques.

In this section, the concept of reinforcement learning is introduced, followed by a description of a number of key algorithms and their contributions. We assume that the reader is familiar with machine learning in general, and deep learning in particular. For the unacquainted reader, rich literature has been provided on the subject, e.g. Goodfellow et al. (2016).

2.3.1 Perception-Action-Learning Framework

At its core, reinforcement learning revolves around an agent interacting with its environment and adapting its behaviour based on a feedback system, using previous experience to learn how to solve novel problems through a *trial-and-error* approach. It stands on a foundation rooted in behaviourist psychology (Sutton and Barto, 1998) and optimal control (Arulkumaran et al., 2017).

As we have seen, a reinforcement learning problem can be mathematically formulated as a Markov decision process, but in many real-world problems related to sequential decision making, a model of the environment is not fully accessible. In the language of Markov decision processes introduced in Section 2.1, this means that the MDP cannot be perfectly modelled, i.e. the transition function $\mathcal{T}(s, a, s')$ and the reward function $\mathcal{R}(s, a, s')$ are at least partially unknown. Algorithms adamant about solving Markov decision problems where this applies, which lie at the heart of reinforcement learning, are known as *model-free* solution methods.

Model-free methods naturally rely on exploration of the environment to compensate for the lack of global model information. If this is done to obtain a sufficiently accurate approximation of the transition function and the reward function, classical methods for solving MDP's remain valid. Most methods, however, attempt to directly estimate the state-action value function $Q(a, s)$ (Wiering and van Otterlo, 2012). This is where the formulation of a reinforcement learning problem deviates from that of optimal control problems in general, and what generates the characteristic *trial-and-error* description of reinforcement learning. We refer to this as perception-action-learning (Arulkumaran et al., 2017), where each iteration allows the agent to update its knowledge of the environment based on its experience.

The perception-action-learning concept is summarized in Figure 2.1. The success of algorithms resting on this notion relies on a proper *exploration-exploitation trade-off.* In essence, the agent needs to *explore* in order to learn, and *exploit* what it already knows in order to achieve its goal of maximising the return. A simple approach commonly used to accomplish this is to apply an *ϵ-greedy policy*, in which the agent simply follows the best policy with a probability of $1 - \epsilon$ and explores with a probability of ϵ (Wiering and van Otterlo, 2012), but several other well-proven methods exist as well (e.g. Kaelbling et al. (1996)).

Since the transition probabilities and the reward function are unknown, reinforcement learning algorithms cannot build upon our previous definitions of the state-value function (2.6) and the state-action value function (2.7). The following sections outline common reinforcement learning algorithms and how

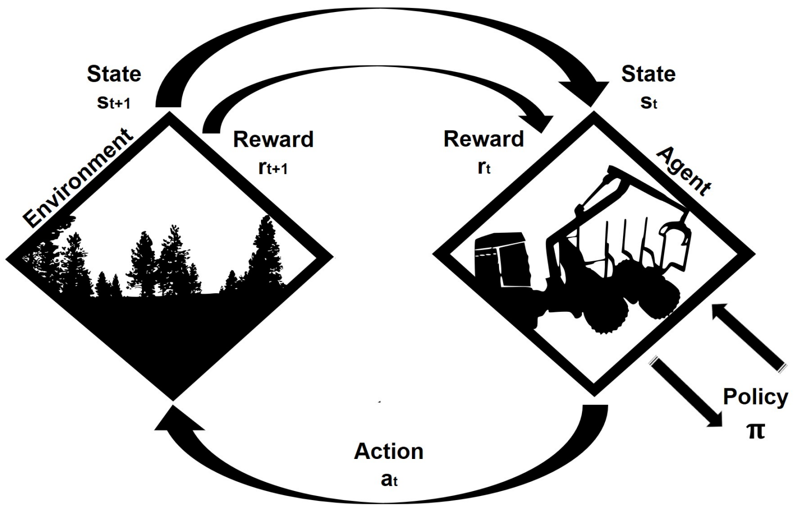


Figure 2.1: *The agent performs an action $a_t \in \mathcal{A}$ from the current state $s_t \in \mathcal{S}$, information about which it has received from the environment. This causes the system to transition to state s_{t+1}, and the agent receives information about the new state s_{t+1} and the current reward r_{t+1}. The agent continues exploring and exploiting its environment to improve its policy π throughout the learning process, with the goal of finding the optimal policy π^* generating the maximum return.*

they get around this problem, starting with a simple algorithm estimating these values based on previous estimates. This lays the groundwork for Proximal Policy Optimization (Schulman et al., 2017), the reinforcement learning algorithm used in the work presented in this thesis.

2.3.2 Temporal Difference Learning

Temporal difference learning is a fundamental solution method aimed at the *temporal credit assignment* problem in reinforcement learning (e.g. Sutton and Barto (1998)). In its simplest form, the state-value or state-action value of each state is stored in a lookup-table, which is updated continuously throughout the training process by means of *bootstrapping.* Of course, this does not alleviate us from the problem of requiring an enumerated state space, but in contrast to the dynamic programming algorithms, the need for a full model of the MDP is removed, and values are only updated for states visited throughout the learning process.

The most basic temporal difference learning algorithm is the *TD(0)*-algorithm (Sutton, 1988). To find the value function $V^{\pi}(s)$, an estimate of the return (discounted, infinite-horizon accumulated reward (2.1)) is calculated each iteration, such that the estimated return $\widetilde{R}_{t+1} = r_{t+1} + \gamma V(s_{t+1})$. In this way, the updated value estimate of state s, $V_{n+1}(s)$, is based solely on the immediate reward and estimated value of the actual subsequent state s_{t+1}, $V_n(s)$. For any state

s, observed reward r and immediately subsequent state s', a TD(0)-update is formulated according to (2.17), where $r + \gamma V(s') - V(s)$ is known as the *TD error* and $\alpha \in [0, 1]$ is the *learning rate*. The latter specifies the trade-off between prior and new information.

$$V_{n+1}(s) = V_n(s) + \alpha\Big(r + \gamma V_n(s') - V_n(s)\Big) \tag{2.17}$$

An extension of the TD(0)-algorithm is the *Q-learning* algorithm (Watkins and Dayan, 1992), which, as the name suggests, aims to estimate the state-action value function $Q(s, a)$ directly. It is highly reminiscent of TD(0), but, given a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, updates its estimate of the state-action value $Q_{n+1}(s, a)$ based on the immediate reward and the *maximum* state-action value of the immediately subsequent state s', i.e. r and $\max_a Q_n(s', a)$. Thus, this is an example of a so-called *off-policy* method, where each update is not necessarily based on the action taken according to the policy. Each Q-learning update is formulated according to (2.18).

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha\Big(r + \gamma \max_a Q_n(s', a) - Q_n(s, a)\Big) \tag{2.18}$$

SARSA (e.g. Singh et al. (2000)) is a corresponding *on-policy* algorithm, for which the single-step update is presented in (2.19). Here, a' denotes the action taken in the subsequent step according to the current policy.

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha\Big(r + \gamma Q_n(s', a') - Q_n(s, a)\Big) \tag{2.19}$$

With the proper learning rate α, Watkins and Dayan (1992) showed that the Q-learning algorithm is guaranteed to converge to the optimal state-action value function $Q^*(s, a)$ for discrete action-value functions, provided each state-action value is sampled enough times. If, in addition, the given policy converges to the greedy policy in the limit, the same is true for SARSA (Singh et al., 2000). Given the optimal state-action value function $Q^*(s, a)$, the optimal policy π^* is easily derived using (2.20).

$$\pi^* = \operatorname*{argmax}_a Q^*(s, a) \tag{2.20}$$

Our discussion on temporal difference learning algorithms is concluded with a summary of the Q-learning algorithm, presented in Algorithm (3). Minor changes can be applied for this outline to apply to SARSA and TD(0). The ϵ-greedy policy π is used to demonstrate the exploration strategy.

```
Algorithm 3: Q-Learning
  Result: The optimal state-action value function $Q^*(s,a)$
  1. Initialization
    Initialize $Q(s,a) \in \Re$ arbitrarily $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$.
    Let $\gamma \in [0,1)$ and $\alpha, \epsilon \in [0,1]$.
  2. Q-Learning
    for each episode do
      Choose an arbitrary starting state $s \in \mathcal{S}$.
      while $s \neq$ terminal state do
        Choose $x \sim U(0,1)$
        if $x < \epsilon$
          Choose a random action $a \in \mathcal{A}$.
        else
          Choose action $a := \pi(s) = \text{argmax}_a Q(s,a)$
        Perform action $a$ and observe $r, s'$.
        $Q(s,a) = Q(s,a) + \alpha\Big(r + \gamma \max_a Q(s',a) - Q(s,a)\Big)$
        Let $s := s'$.
```

2.3.3 Deep Q-learning

Methods like temporal difference learning avoid repeatedly traversing the entire state space, but tabular storage of state-action values can quickly become computationally inefficient. The remedy to this lies in modification of the algorithms using *function approximation* techniques. To this end, and owing to its success in the field of supervised machine learning, *deep learning* quickly emerged as a bright star in the reinforcement learning community. This section is devoted to *Deep Q-learning* (DQL), an approach combining Q-learning with deep learning.

Deep Q-learning algorithms (e.g. Mnih et al. (2013)) use deep neural networks as non-linear function approximators. Instead of explicitly storing each state-action value, experience gathered by the agent is used to train a deep neural network to generate state-action values from the input states. Such a network is called a Deep Q-Network (DQN), which for each state $s \in \mathcal{S}$ outputs a state-action value vector $Q(s, \cdot; \theta)$ parametrized by θ.

Deep learning belongs to the class of supervised learning techniques in which a generalized mapping between input-target pairs is learned. To adopt this approach, we define the *temporal difference target* y_i according to (2.21) for each iteration i, where r is the reward received upon transition from state s to subsequent state s'. Hence, unlike classic supervised learning, the target values are not fixed, ground-truth values, but improves with the network parameters during the training process. The next step is to minimize the loss, which at each iteration i is defined according to (2.22). Here, two particular features are added to increase stability and data efficiency; *experience replay* (Mnih et al., 2013), and the inclusion of a *target network* (Mnih et al., 2015).

$$y_i = r + \gamma \operatorname*{argmax}_a Q(s', a; \theta_i^-) \tag{2.21}$$

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)} \Big[\Big(y_i - Q(s, a; \theta_i) \Big)^2 \Big] \tag{2.22}$$

θ_i and θ_i^- denote the weights of each network at iteration i. The parameters of the target network θ^- are fixed between iterations i, only allowing for updates according to the current weights θ of the primary neural network in fixed intervals. Such more infrequent updates of the target network yields a reduction in data correlations, leading to increased stability (Mnih et al., 2015). Alternative approaches showing promising results have recently been developed, such as using an alternative softmax operator in place of adding a target network (Kim et al., 2019), but we settle for presenting the target network approach here.

Experience replay is another feature included to randomize the data set and reduce sample correlations, shown to have a significant positive effect on the agent performance. Using experience replay, each training sample (s, a, r, s') is uniformly drawn from a circular experience buffer D, where experience samples are stored as the training progresses. This is an important development, as nonlinear function approximation has been previously known to cause significant instability due to effects of for example sequence and sample-target correlations. These issues have been discussed by e.g. Dai et al. (2018). The DQL algorithm using experience replay and a target network (Mnih et al., 2015) is summarized in Algorithm (4), under an ϵ-greedy policy as before.

Deep Q-learning extends the reach of reinforcement learning algorithms to complex environments with high-dimensional state spaces. The DQN developed by Mnih et al. (2015) was able to outperform previous algorithms on multiple Atari 2600 games despite requiring high-dimensional sensory input data, proving the potential of reinforcement learning methods in complex situations. Moreover, Mnih et al. (2015) sheds light on the intimate relationship between reinforcement learning and neurobiological learning processes, motivating important algorithm components in recent biological findings.

Since the first successes of deep Q-learning, the algorithm has been ameliorated in different ways. This includes Double Deep Q-learning (van Hasselt et al., 2016) and prioritized experience replay (Schaul et al., 2016), the former improving DQN performance by decoupling action selection from state-action value evaluation, and the latter by exchanging uniform experience sampling for weighted sampling in favor of important transitions. More recently, Kapturowski et al. (2019) used recurrent neural networks with distributed prioritized experience replay for deep Q-learning, exceeding previous state-of-the-art performance on a range of Atari games. Other advances in deep reinforcement learning include dueling network architectures (Wang et al., 2016) and multiple agent asynchronous learning methods (Mnih et al., 2016), exemplifying the potential in combining deep learning with a variety of different reinforcement learning methods.

Recently, Zahavy et al. (2016) investigated the underlying ways in which feature learning is progressed using *Deep Q-Networks*, finding that these networks indeed capture hierarchical structures of the target task.

Generalizing a state-to-value mapping is not the only way deep learning mitigates the curse of dimensionality. Often, sensory input provides the agent with an unnecessarily high-dimensional observation state, in which case *state representation learning* (e.g. Lesort et al. (2018); Jonschkowski and Brock (2014)) can greatly reduce the effective dimensionality of the problem. Deep learning methods are often used to learn such an observation-state mapping.

Algorithm 4: Deep Q-Learning

```
Result: The optimal state-action value function $Q^*(s, a)$
1. Initialization
   Initialize the primary network with random weights $\theta$ and let $i := 0$.
   Let $D$ be the empty replay buffer, $\theta^- := \theta$, $\gamma \in [0, 1)$ and $\epsilon \in [0, 1]$.
2. Deep Q-Learning
   for each episode do
      Choose an arbitrary starting state $s \in \mathcal{S}$.
      while $s \neq$ terminal state do
         Choose $x \sim U(0, 1)$
         if $x < \epsilon$
            Choose a random action $a \in \mathcal{A}$.
         else
            Choose action $a := \pi(s) = \operatorname{argmax}_a Q(s, a)$
         Perform action $a$ and observe $r, s'$.
         Add $< s, a, r, s' >$ to the replay buffer $D$.
         Sample random minibatch of transitions $< s_j, a_j, r_{j+1}, s_{j+1} >$.
         if $s_{j+1} \neq$ terminal state
            $y_j := r_{j+1}$.
         else
            $y_j := r_{j+1} + \gamma \operatorname{argmax}_a Q(s_{j+1}, a; \theta_i^-)$
         Perform gradient descent step on $L_{\theta_i} = \Big(y_j - Q(s, a; \theta_i)\Big)^2$
         If $i \ \% \ c = 0$ for some $c$, set $\theta^- := \theta$.
         Let $s := s'$, $i = i + 1$.
```

2.3.4 Policy Gradient Methods

As we have seen, Deep Q-learning algorithms require repeatedly maximizing the state-action value function over all legal actions. This is computationally expensive for large action spaces, and can quickly become infeasible in the continuous case, where discretization of the action space is necessary which further reduces performance. As a result, another class of methods, *policy gradient methods* (e.g. Sutton et al. (1999a)), has become the go-to technique in contexts of continuous action reinforcement learning problems. These methods have achieved great success in contexts including robot manipulation tasks and games like Go

(Li, 2018).

Instead of parametrizing the state-action value function, policy gradient methods are based on direct parametrization of the policy itself. In the following derivation, $\pi_\theta(a|s) = \pi(a|s;\theta)$ is a stochastic policy parametrized by θ and $R(\tau)$ is the finite horizon discounted return following the policy π_θ along a trajectory τ, where τ refers to the sequence of states, actions and rewards generated by the policy. T is the horizon length. Given an initial state probability distribution $\mathcal{I}(s_0)$ and transition probability distribution $\mathcal{T}(s,a,r,s')$, the trajectory probability distribution $p(\tau|\theta)$ is given by (2.23).

$$p(\tau|\theta) = \mathcal{I}(s_0) \prod_{t=0}^{T-1} \mathcal{T}(s_t, a_t, r_{t+1}, s_{t+1}) \pi_\theta(a_t|s_t) \tag{2.23}$$

The learning objective is to maximize the expected return (2.24). This is done by updating the parameters θ in the direction of the policy gradient (2.25) throughout the learning process.

$$J(\theta) = E_{\tau \sim p(\tau|\theta)} \Big[R(\tau) \Big] = \int p(\tau|\theta) R(\tau) d\tau \tag{2.24}$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \int p(\tau|\theta) \nabla_\theta \log p(\tau|\theta)(\tau) R(\tau) d\tau \\ &= E_{\tau \sim p(\tau|\theta)} \Big[\nabla_\theta \log p(\tau|\theta) R(\tau) \Big] \end{aligned} \tag{2.25}$$

Here, the gradient $\nabla_\theta \log p(\tau|\theta)$ can be computed directly from (2.23) according to (2.26), since both distributions $\mathcal{I}(s_0)$ and $\mathcal{T}(s,a,r,s')$ are independent on θ. Our expression for the policy gradient can then be simplified according to (2.27).

$$\begin{aligned} \nabla_\theta \log p(\tau|\theta) &= \nabla_\theta \log \Big(\mathcal{I}(s_0) \prod_{t=0}^{T-1} \mathcal{T}(s_t, a_t, r_{t+1}, s_{t+1}) \pi_\theta(a_t|s_t) \Big) \\ &= \nabla_\theta \Big(\log \mathcal{I}(s_0) + \sum_{t=0}^{T-1} \Big[\log \mathcal{T}(s_t, a_t, r_{t+1}, s_{t+1}) + \log \pi_\theta(a_t|s_t) \Big] \Big) \\ &= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \end{aligned} \tag{2.26}$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau|\theta)} \Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \Big] \tag{2.27}$$

This is the result of the *policy gradient theorem*, which lays the foundation for several celebrated policy gradient methods. Once the policy gradient is obtained the policy can be optimized, e.g. through gradient ascent according to (2.28), where α denotes the step size.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta)|_{\theta_k} \tag{2.28}$$

In practice, Monte Carlo sampling is often used to practically compute the expectation (2.27). Given a set $\mathcal{N} = \{\tau_j\}_{j=1}^N$ of N on-policy trajectory samples such that $\tau_j \sim p(\tau|\theta)\ \forall j$, estimators of the expected return and its policy gradient can be obtained according to (2.29) and (2.30).

$$\widetilde{J}(\theta) = \sum_{\tau \in \mathcal{N}} R(\tau) \tag{2.29}$$

$$\nabla_\theta \widetilde{J}(\theta) = \frac{1}{|\mathcal{N}|} \sum_{\tau \in \mathcal{N}} \Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \Big] \tag{2.30}$$

The policy gradient theorem result (2.27) can be written on a more general form given in (2.31), where Φ is not locked to be the return $R(\tau)$. If Φ_t is defined as the discounted, accumulated reward after the current time step t, i.e. the reward-to-go, this does not affect the obtained expected value of the policy gradient. It does, however, affect the variance which has a negative impact on convergence properties. In fact, one problem with our current definition, and with policy gradient methods in general, is the high variance in the policy gradient sample estimates (Wu et al., 2018). There are several ways to reduce this effect, though some are prone to introduce bias.

If we let $\Phi_t = R_t - b(s_t)$, where $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_t$ is the reward-to-go and b is a θ-independent baseline, we arrive at the *REINFORCE algorithm with a baseline*, first introduced by Williams (1992). The resulting expectation is presented in (2.32), in which the variance is reduced without introducing bias in the empirical evaluation. This is shown by e.g. Wu et al. (2018), and can easily be motivated by (2.33). Wu et al. (2018) also provide a derivation of the optimal baseline. The two aforementioned variance reduction tricks, using the reward-to-go instead of the full trajectory return and adding a baseline, are two of the most common techniques for variance reduction (Greensmith et al., 2004). Recently, a lot of research has been aimed at finding better variance reduction techniques. Examples include *Generalized Advantage Estimation (GAE)* (Schulman et al., 2016), combining GAE with a linear baseline (Gu et al., 2017) and using action-dependent baselines (Wu et al., 2018).

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau|\theta)} \Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t \Big] \tag{2.31}$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau|\theta)} \Big[\sum_{t=0}^{T-1} (R_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] \tag{2.32}$$

$$E_{\tau \sim p(\tau|\theta)} \Big[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \Big] = \nabla_\theta E_{\tau \sim p(\tau|\theta)} \Big[b(s_t) \Big] = 0 \tag{2.33}$$

Another option is to define Φ_t as the state-value function $Q^{\pi_\theta}(s_t, a_t)$ (Schulman et al., 2016). This can be shown without exhaustive mathematics using the law of iterated expectations. This is done by letting $b(s_t) = 0$ in (2.32) and denoting τ_t the trajectory from step t to $T-1$, from which (2.34) is derived. Then, $E_{\tau_t \sim \pi_\theta}\Big[R_t|\tau_t\Big] = E_{\tau_t \sim \pi_\theta}\Big[R_t|s_t, a_t\Big] = Q^{\pi_\theta}(s_t, a_t)$, and the claim is true.

$$\begin{aligned}
\nabla_\theta J(\theta) &= E_{\tau \sim p(\tau|\theta)}\Big[\sum_{t=0}^{T-1} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\Big] \\
&= \sum_{t=0}^{T-1} E_{\tau \sim p(\tau|\theta)}\Big[R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\Big] \\
&= \sum_{t=0}^{T-1} E_{\tau_t}\Big[E_{\tau_t}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t) R_t|\tau_t\Big]\Big] \\
&= \sum_{t=0}^{T-1} E_{\tau_t}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t) E_{\tau_t}\Big[R_t|\tau_t\Big]\Big]
\end{aligned} \tag{2.34}$$

A common baseline choice is $b(s_t) = V^\pi(s_t)$. In fact, $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, known as the advantage function, is a close to optimal Φ_t in terms of variance reduction (Schulman et al., 2016). This is reasonable, since it measures the value difference induced by taking an action a_t compared to acting on-policy. Thus, the probability of taking action a_t is only increased for positive advantages $A^\pi(s_t, a_t)$. However, since it is unknown, estimation is required. Such an estimation often introduces some bias, which needs to be considered. Many policy gradient algorithms make use of different advantage estimators. For example, Schulman et al. (2016) introduces Generalized Advantage Estimation, mentioned briefly above. In this case, the advantage function depends on an estimation of the TD error (2.35), previously defined in our discussion on temporal difference learning. The generalized advantage estimator derived by Schulman et al. (2016) is presented in (2.36), where the GAE parameter $\lambda \in [0, 1]$ controls credit assignment and determines the bias-variance trade-off, and γ allows for control of the value estimation trust.

$$\delta_t^{\widetilde{V}} = r_t + \gamma \widetilde{V}(s_{t+1}) - \widetilde{V}(s_t) \tag{2.35}$$

$$\widetilde{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^{\widetilde{V}} \tag{2.36}$$

Of course, this approach relies on estimates of the value function to determine $\delta_t^{\widetilde{V}}$. If $\widetilde{V} = V^\pi$, then $\delta_t^{\widetilde{V}}$ is an unbiased advantage estimator in itself (Schulman et al., 2016). Approximation of the value function can be obtained through non-linear function approximation techniques, using methods such as the TD(1) method for value estimation (Sutton and Barto, 1998) or trust region methods

(Schulman et al., 2016). The latter uses trust region optimization for both the policy neural network and the value function neural network.

The above is an example of an *Actor-Critic* method, combining the policy gradient approach with value iteration. In such algorithms, the parametrized policy is referred to as the *actor*, updating its parameters based on feedback from the *critic*, in this case the parametrized value function. An example of an actor-critic method that has received a lot of praise is *Asynchronous Advantage Actor-Critic* (A3C), proposed by Mnih et al. (2016).

Despite the many successes made possible by the combination of policy gradient methods and deep learning, these algorithms often display poor sample efficiency and high sensitivity to the policy update step size. If the latter is too large, the signal-to-noise ratio is low, often resulting in an unstable learning process. On the other hand, if it is not large enough, the learning process may not progress at all. To remedy this, methods like Trust Region Policy Optimization (Schulman et al., 2015) use techniques to limit the policy update. A related method is Proximal Policy Optimization, which takes a novel approach to resolving these issues. This method is discussed in detail in the following section. Concluding this section, the REINFORCE algorithm with a parametrized state-value baseline $V_\varphi(s_t) = V(s_t;\varphi)$ is presented in Algorithm (5).

Algorithm 5: REINFORCE with State-Value Baseline

```
Result: An optimized policy $\pi(a|s;\theta)$
1. Initialization
   Initialize the primary network with random weights $\theta$ and the
   baseline network with random weights $\varphi$.
   Initialize $\gamma \in [0,1)$ and step sizes $\alpha^\theta > 0$ and $\alpha^\varphi > 0$.
2. REINFORCE algorithm
   for each episode do
       Generate a trajectory $\tau \sim \pi(a|s;\theta)$ of length $T$.
       for $t = 0, 1....T-1$ do
           $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_t$
           $G = R_t - V_\varphi(s_t)$
           $\varphi = \varphi + \alpha^\varphi G \nabla V_\varphi(s_t)$
           $\theta = \theta + \alpha^\theta G \nabla \log \pi_\theta(a_t|s_t)$
```

2.3.5 Proximal Policy Optimization

The algorithm on which our results are based belongs to a class of policy gradient algorithms known as *Proximal Policy Optimization* algorithms, introduced by Schulman et al. (2017). They improve on preceding policy gradient methods by allowing policy updates in multiple epochs, addressing the issue of large policy updates and increasing sample efficiency. These algorithms, which are referred to as PPO in the remainder of this thesis, rely on alternation between data sampling and optimization of a surrogate objective function through stochastic gradient descent.

To begin our derivation, we repeat the gradient estimator (2.31) as an expected value over time t with an advantage estimator, i.e. $\Phi_t = \widetilde{A}_t$, according to (2.38). The corresponding objective function is presented in (2.37). Multi-step optimization of $J(\theta)$ often results in policy updates too large for stable convergence, a problem PPO aims to evade by limiting the policy update using a clipped surrogate objective function.

$$J_t(\theta) = E_t\Big[\log \pi_\theta(a_t|s_t)\widetilde{A}_t\Big] \tag{2.37}$$

$$\nabla_\theta J_t(\theta) = E_t\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\widetilde{A}_t\Big] \tag{2.38}$$

Schulman et al. (2017) truncated the generalized advantage estimator (2.36) to trajectory length or horizon T, yielding the advantage estimator presented in (2.39). λ is the GAE parameter, γ is the discount factor and $\delta_t^{\widetilde{V}}$ is defined as before, see (2.35).

$$\widetilde{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}^{\widetilde{V}} \tag{2.39}$$

Next, we introduce the probability ratio $\eta_t(\theta)$ according to (2.40), where π_θ denotes the current policy, and $\pi_{\hat{\theta}}$ denotes the policy under the parameters of the previous update. The objective function (2.37) can then be reformulated according to (2.41). On this form, first introduced by Kakade and Langford (2002), straightforward optimization does not avoid the instability issues induced by large policy ratios.

$$\eta_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\hat{\theta}}(a_t|s_t)} \tag{2.40}$$

$$J_t^{CPI}(\theta) = E_t\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\hat{\theta}}(a_t|s_t)}\widetilde{A}_t\Big] = E_t\Big[\eta_t(\theta)\widetilde{A}_t\Big] \tag{2.41}$$

Schulman et al. (2015) imposes a trust region constraint based on Kullback–Leibler (KL) divergence to constrain the size of parameter updates up to a parameter ε by enforcing $E_t\Big[\mathrm{KL}\Big[\pi_{\hat{\theta}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\Big]\Big] \leq \varepsilon$. One PPO-algorithm addresses the issue using an adaptive KL penalty in place of the constraint, but the main PPO-algorithm, which we are concerned with in this section, takes a different approach, penalizing large policy updates by introducing the *clipped surrogate objective function* according to (2.42), for some clipping threshold ϵ.

$$J_t^{CLIP}(\theta) = E_t\Big[\min\Big(\eta_t(\theta)\widetilde{A}_t, \mathrm{clip}(\eta_t(\theta), 1-\epsilon, 1+\epsilon)\widetilde{A}_t\Big)\Big] \tag{2.42}$$

The *clip*-function prevents the policies from diverging by confining the probability ratio to the interval $1-\epsilon \leq \eta_t(\theta) \leq 1+\epsilon$. The objective function $J^{CLIP}(\theta)$

then takes the minimum value between the objective function $J^{CPI}(\theta)$ and the clipped objective function. Thus, $J^{CLIP}(\theta)$ is a lower bound on $J^{CPI}(\theta)$.

Schulman et al. (2017) implements PPO in a deep learning setting, with shared parameters for the policy and state-value functions. As such, the objective function is extended with an error term on the value function, in this case the squared error loss. Additionally, as originally proposed by Williams and Peng (1991), an entropy regularization term is added to encourage continued exploration. The final objective function then takes the form of (2.43), where V_t^* denotes the target state-value function, H denotes the entropy bonus and c_1, c_2 are the tunable value function- and entropy regularization coefficients.

$$J_t^{PPO}(\theta) = E_t\Big[J_t^{CLIP} + c_1\big(\widetilde{V}_\theta(s_t) - \widetilde{V}_t^*\big)^2 + c_2 H(s_t, \pi_\theta)\Big] \tag{2.43}$$

The form of the objective varies slightly depending on the solution method. For example, (2.42) can be optimized directly using automatic differentiation and multi-step stochastic gradient ascent. Algorithm (6) presents a PPO-pseudocode in which separate neural networks are learned to represent the policy and the value function, the latter parametrized according to $V_\varphi(s_t) = V(s_t; \varphi)$ as before. In this case, learning proceeds for K iterations, R_t^τ is the reward-to-go at time step t on trajectory τ, and $\widetilde{A}_{t,\varphi}^\tau$ is the advantage estimate at time step t under $V(s_t; \varphi_\kappa)$, in this case the truncated generalized advantage estimate (2.39).

The PPO algorithm outlined by Schulman et al. (2017) optimizes (2.43) with the truncated generalized advantage function using minibatch stochastic gradient descent with N parallel actors, and their work shows promising results on an array of benchmark tasks. The main takeaway, however, is the combination of a stable learning process, performance, sample complexity and implementation simplicity, which have made PPO a dominating algorithm on a wide range of reinforcement learning tasks for problems demanding either discrete or continuous action spaces.

Algorithm 6: PPO with Clipped Surrogate Objective

```
Result: An optimized policy $\pi(a|s;\theta)$
1. Initialization
    Initialize the policy network with random weights $\theta$ and the
    state-value function network with random weights $\varphi$.
    Initialize $\gamma \in [0, 1)$ and clipping threshold $\epsilon$.
2. Proximal Policy Optimization
    for each $\kappa = 0, 1...K$ do
        Generate a set of trajectories $\mathcal{N}_\kappa = \{\tau_j\}_{j=1}^{N}$ under current policy
        $\pi(a|s;\theta_\kappa)$, each trajectory of length $T$.
        for each $\tau \in \mathcal{N}_\kappa$ do
            Collect rewards-to-go and advantage estimates:
            for $t = 0, 1....T-1$ do
                $R_t^\tau = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_t$
                $\widetilde{A}_{t,\varphi}^\tau = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}^{V_{\varphi_\kappa}}$
        Update policy parameters $\theta$ by optimizing the PPO objective:
        $\theta_{\kappa+1} = \text{argmax}_\theta \left( \frac{1}{|\mathcal{N}_\kappa| T} \sum_{\tau \in \mathcal{N}_\kappa} \sum_{t=0}^{T-1} J_t^\tau(\theta) \right),$
        $where\ J_t^\tau(\theta) = \min\Big( \eta_t(\theta) \widetilde{A}_{t,\varphi}^\tau, \text{clip}(\eta_t(\theta), 1-\epsilon, 1+\epsilon) \widetilde{A}_{t,\varphi}^\tau \Big)$
        Update the state-value function parameters $\varphi$ by regression:
        $\varphi_{\kappa+1} = \text{argmin}_\varphi \Big( \frac{1}{|\mathcal{N}_\kappa| T} \sum_{\tau \in \mathcal{N}_\kappa} \sum_{t=0}^{T-1} \left(V_\varphi(s_t) - R_t^\tau\right)^2 \Big)$
```

2.4 Multiple Skill Acquisition & Sparse Rewards

The reinforcement learning solution methods discussed so far often result in slow learning processes when applied to complex tasks. This includes contexts characterized by long-term temporal dependencies, in which reward signals are naturally sparse. This type of reward signal is tempting to consider since it only requires a definition of success in relation to the goal. The drawback, however, is that such an approach forces the agent to rely on random exploration of the environment, which, if at all successful, is slow in complex environments.

In this section, we will explore different ways of adjusting the standard reinforcement learning framework to successfully complete a complex task by means of breaking it down into sub-tasks. In our example setting, the goal of a reinforcement learning agent constituted by a robotic arm is to find an object and place it on top of another object. This is a multifaceted problem that can be broken down into several sub-tasks, i.e. locating the first object, grasping it, moving it to the second object, and stacking it onto said object. This can be viewed as a simple analogy to the forestry crane problem constituting the objective of this thesis. Intuitively, simply rewarding the agent when accomplishing this task, i.e. when the objects are finally stacked onto each other, would require endless exploration with rare to no feedback from which it can learn.

To begin our discussion, we will dive into techniques that can be applied to the standard reinforcement learning framework in settings where sparse rewards pose a particular challenge.

2.4.1 Dealing with Sparse Extrinsic Rewards

In our example setting, the goal of the agent is to place one object on top of another. In a setting only allowing for very sparse rewards, the agent only receives feedback signals when this goal is achieved. Depending on the dimensionality of the problem, this can prove to be a very challenging task. In fact, this is often the case even in the comparatively simple sub-task of locating the first object.

The most common technique used to tackle the credit assignment problem in complex settings is known as *reward shaping*, in which reward functions yielding intermediate rewards are carefully designed to encourage incremental progress. (Ng et al., 1999) shows that basing the reward transformation on potentials does not affect the optimal policy compared to the corresponding extrinsic-reward setting. In many cases, however, the reward function is carefully shaped by domain experts, which can lead to increased implementation complexity and decreased generalizability in the final model. Moreover, reward shaping can introduce local optima, and while tedious reward function design can prevent illegal solutions, it may also yield a loss in solution innovation, removing the potential for harvesting unexpected solutions to problems facing the reinforcement learning agent. These limitations have been discussed by e.g. Zou et al. (2019) and Trott et al. (2019), two among several recent studies aimed at developing improved reward shaping techniques.

Besides reward shaping, other approaches have utilized *curiosity-driven learning* based on *intrinsic rewards* to circumvent the credit assignment problem. For example, Pathak et al. (2017) shows that combining an A3C baseline model with their intrinsic curiosity module significantly improves both performance and learning speed in VizDoom 3D-navigation tasks. This is true regardless of extrinsic-reward sparsity, but especially in settings displaying sparse or very sparse extrinsic rewards. Curiosity-driven methods encourage efficient exploration and are inspired by ideas derived from developmental human psychology (Oudeyer and Kaplan, 2007).

Recently, Burda et al. (2018) introduced *Random Network Distillation* to encourage exploration of novel states, succeeding in multiple Atari games in sparse extrinsic reward settings. The method did, however, struggle in settings displaying long-term temporal dependencies. An example of another recent technique used to aid exploration is *Uncertainty Bellman Exploration* (O'Donoghue et al., 2017), which later inspired an extension of PPO known as *Optimistic Proximal Policy Optimization* (Imagawa et al., 2019).

Before proceeding, a final, novel approach to dealing with sparse extrinsic rewards is introduced, namely *Hindsight Experience Replay* (Andrychowicz et al., 2017). This method relies solely on sparse, extrinsic rewards, but makes use of the fact that in certain tasks, each state can be considered a (pseudo) goal state. When this applies, episodes can be replayed with a goal state reached during the episode, streamlining updates during training. This method avoids reward shaping, and is useful when homogeneity in *visited* states is not what prevents sufficient exploration, but rather dimensionality itself. Andrychowicz et al. (2017) successfully demonstrates this approach on simple robot manipulation tasks using a simulated robotic arm, and shows that the results transfer to a physical robot.

For very complex problems these methods may not prove sufficient, especially if the agent is expected to learn skills that vary significantly between sub-tasks, such as locating, grasping and moving an object to a specific target object, as in our example setting. Thus, the following subsections discuss methods that can be applied to more complex problem settings.

2.4.2 Transfer Learning

The conventional reinforcement learning techniques previously discussed usually rely on an untrained agent learning to solve a task from scratch. If a parametrized policy is utilized, this means initializing the policy with random weights, as we have done in the algorithms summarized up to this point. This is, however, not a necessity. On this note, *transfer learning* is a paradigm which, in the context of reinforcement learning, refers to the transfer of knowledge from an agent previously trained to perform one task, to another agent faced with a different but similar task, in order to ease learning on the second task. In other words, the agent first learns to perform a simpler *source task*, and uses its acquired knowledge to bias learning on the *target task*.

Depending on the implementation, transfer learning techniques are not limited to parameter initialization. For example, Fernández et al. (2010) combines probabilistic reuse of policies with action- and state space mapping, removing the need for identical state- and action spaces in the source and target tasks. Moreover, the transfer of knowledge is not restricted to policies, but can also be carried out through other means, such as the value functions (e.g. Taylor and Stone (2005), Taylor et al. (2007)).

Upon evaluation of the success of transfer learning, it is common to distinguish between *weak* and *strong transfer*. The former refers to comparing the agent's performance and learning speed on the target task directly against the corresponding performance and learning speed in a setting without transfer learning, and the latter refers to a way of evaluation where the source task training process is taken into account (Narvekar et al., 2020).

Returning to our example setting, transfer learning by training an agent on the source task of locating the first object may ease learning on the target task of grasping the object. For a more in-depth coverage of transfer learning in reinforcement learning domains, surveys such as that of Taylor and Stone (2009) can be consulted.

2.4.3 Curriculum Learning

Building on the idea of transfer learning, recent reinforcement learning research has explored the possibility of improving an agent's performance on a target task by allowing the agent to gain and transfer knowledge on a *sequence* of source tasks. Narvekar et al. (2016) introduces this concept, referred to as *curriculum learning*, and shows how a tailored sequence of source tasks can significantly improve performance on a target task in two reinforcement learning domains. Moreover, their results confirm that the order in which each source task is learned impacts the performance on the target task. This suggests that complex reinforcement learning problems can be approached by means of applying transfer learning to a sequence of source tasks that optimizes performance on the target task. A classic example motivating the concept of curriculum learning is the game of Quick Chess, following a step-by-step approach introducing children to the advanced game of Chess.

The research by Narvekar et al. (2016) proposes functions for semi-automatic source task generation, while relying on domain knowledge to optimize the curriculum. A further extension was introduced by Narvekar et al. (2017), automating the sequence generation by formulating the process as an MDP of its own.

The curriculum learning framework was recently formalized by Narvekar et al. (2020) in the domain of reinforcement learning, further providing an extensive overview on recent accomplishments within the field. Not contrary to intuition, early work on the periphery of curriculum learning suggests that transferring knowledge between a sequence of source tasks organized in an order of increasing complexity accelerates learning and optimizes performance on the target task.

Finally, as elaborated on by Narvekar et al. (2020), the domain of curriculum learning is not limited to learning through a linear sequence of source tasks. As an example, a curriculum-guided approach to the method of Hindsight Experience Replay (Andrychowicz et al., 2017) was recently developed by Fang et al. (2019). Here, the core contribution is adaptive selection of pseudo goals, gradually trading curiosity for proximity to the true goal as the training progresses. Directly adopting the curriculum learning concept through optimization of sample order in the target task in this way has also sparked progress in supervised machine learning: For example, Bengio et al. (2009) shows that gradually increasing sample complexity in supervised tasks increases generalization in the final model.

As we have seen, curriculum learning brings with it the potential to ease learning and improve performance in complex reinforcement learning problems by breaking them down into simpler sub-tasks. In our example, the target task is for the robotic arm to stack two objects onto each other. We proposed four subtasks, which in theory may be broken down into an even larger sequence of source tasks forming a complete curriculum. Each of the four primary sub-tasks may also be performed by different agents, each learning one skill through a smaller curriculum. This nears the domain of *hierarchical reinforcement learning*, in which a hierarchical policy can be learned to decide when to activate previously learned policies, opening up for multi-task reinforcement learning agents acting through an hierarchy of sub-policies. To this the following subsection is devoted.

2.4.4 Hierarchical Reinforcement Learning

As noted, conventional reinforcement learning algorithms often struggle in solving complex problems with sparse extrinsic rewards or other long-term dependencies. As an advanced alternative to the approaches previously discussed, early work (e.g. Hauskrecht et al. (1998), Sutton et al. (1999b)) looked to temporally abstract actions, so-called *macro actions*, to remedy this through *hierarchical policy representations.* In the beginning of the 21^{st} century, Barto and Mahadevan (2003) gave an overview of advances on this topic. Since then, the field of *hierarchical reinforcement learning* has remained an active field of research. The promise of this approach lies in the temporal and spatial abstraction of complex reinforcement learning problems, and is reflected through its potential to decompose such problems into multiple sub-tasks while retaining the ability to reuse skills previously learned, consequently reducing the total computational complexity and increasing learning speed.

In this context, we are viewing the hierarchical reinforcement learning problem as a high-level abstraction of the complete problem, with actions that can be temporally extended and represented by sub-policies learning specific sub-tasks. While multiple solutions in terms of specific architectures exist, hierarchical reinforcement learning problems are typically formalized through *semi-Markov decision processes* (SMDP). The Bellman Equation of the discrete-time state-value function under high-level fixed policy π such that $\pi(s) = a$ can then be expressed according to (2.44), where N is a random variable corresponding to the temporal extension of macro action a (Dietterich, 2000).

$$V^{\pi}(s) = \sum_{s',N} \mathcal{T}_N(s,a,s')\Big(\mathcal{R}_N(s,a,s') + \gamma^N V^{\pi}(s')\Big) \tag{2.44}$$

Here, the transition probability function $\mathcal{T}_N(s,a,s')$ refers to the transition probabilities between states $s = s_t$ and subsequent states $s' = s_{t+N}$ upon performing action $a = a_t$, and $\mathcal{R}_N(s,a,s')$ refers to the expected sum of (discounted) rewards over the corresponding time horizon. These are then joint distributions over s' and N. Using this formulation, conventional reinforcement learning algorithms can be used to solve the temporally, and possibly spatially, abstracted problem.

To clarify the concept of hierarchical reinforcement learning, we return to our example problem of stacking two objects onto each other. Figure 2.2 shows a task graph illustrating a simple hierarchical representation of this problem, the solution to which in this case is the high-level policy π. The action space, then, consists of the two sub-policies π_1 and π_2, each a macro action of either picking up an object or stacking an object onto another. These sub-policies, in turn, consist of an action space composed of one or more of the macro actions or sub-policies π_3, π_4 and π_5, corresponding to the sub-tasks of either finding an object, grasping an object, or releasing an object. Depending on the complexity of the sub-problem, these sub-policies then calls a set of primary actions, controlling the movement of the robotic arm to perform the specified low-level task. Together, this hierarchy of policies solves the overall problem of stacking two objects onto each other. This can, of course, be extended to the more complicated problem of fitting an entire set of objects initially spread out over some space into a tower, for example.

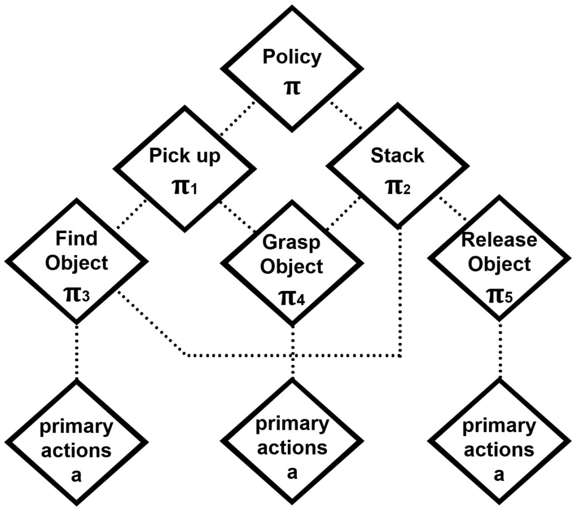


Figure 2.2: *A schematic illustration of a hierarchical reinforcement learning problem represented as a task graph.*

In general, a hierarchical policy may invoke either a sub-policy, as illustrated, or some primary action. Additionally, a hierarchical policy representation is of course not guaranteed to converge to the same behavior as a flat policy representation, since the sub-task design guides the learning process similarly to reward shaping. For example, the single-policy motion of a robotic arm successfully grasping an object may not coincide with the motion of a robotic arm first navigating to an object to *then* attempt to grasp it. A thorough introduction to the optimality problem in hierarchical reinforcement learning is provided by Hengst (2011).

An early method for hierarchical reinforcement learning is *Hierarchies of Abstract Machines* (Parr and Russell, 1997), which was soon followed by a framework for temporal abstraction using *options*, as well as the *MAXQ*-method based on value function decomposition (Dietterich, 2000). Examples of progress in hierarchical reinforcement learning include partial-order task-hierarchies invoking multi-task actions. The study was conducted by Hengst (2008), also demonstrating automatic decomposition into such task-hierarchies, one of multiple attempts at automating sub-task division in hierarchical reinforcement learning.

More recently, research in hierarchical reinforcement learning has made a series of notable achievements, e.g. Levy et al. (2017), Frans et al. (2018) and Nachum et al. (2019b). The latter involves a hierarchical architecture where goals are iteratively communicated between high- and low-level policies. On a similar note, Nachum et al. (2019a) shows that hierarchical reinforcement learning methods can yield great results on quadrupedal manipulation tasks using such a goal-conditioned hierarchical approach. The foundation, then, is a low-level policy learning basic locomotion behavior, from which a high-level, more abstract policy can make use of the learned skill to reach more demanding goals. The authors show that simulation success in the context of hierarchical reinforcement learning is transferable to a physical robot.

Furthermore, Levy et al. (2017) introduces a novel *Hierarchical Actor-Critic* approach in which a multi-level policy hierarchy is learned in parallel. What is more, the architecture adopts hindsight transitions allowing for success using solely sparse rewards in robot simulation settings. Moreover, Tessler et al. (2017) recently proposed the *Hierarchical Deep Reinforcement Learning Network* architecture, enabling *lifelong learning* through selective transfer of reusable skills, and Shu et al. (2018) proposed another framework for efficient multi-task reinforcement learning using a multi-level hierarchical policy structure, both demonstrating their success in the game of Minecraft.

While these kinds of empirical demonstrations of success are encouraging, Nachum et al. (2019c) recently investigated the theoretical foundation for the intuition behind the concept of hierarchical reinforcement learning, concluding that it is actually more efficient exploration that lays the groundwork for many achievements. Interestingly, Zahavy et al. (2016) simultaneously shows that *Deep Q-Networks* indeed works by capturing hierarchical structures of the target task.

Finally, for the interested reader, a brief but comprehensive overview of the fundamentals of hierarchical reinforcement learning is given by Hengst (2011).

Chapter 3

Method

The main objective of this thesis is to automate the grasping motion of a forestry crane manipulator in a simulated environment. This is done using deep reinforcement learning, in which the policy is parametrized by a feedforward neural network and optimized using the widely applied state-of-the-art on-policy algorithm Proximal Policy Optimization (PPO) (Schulman et al., 2017) outlined in Section 2.3.5. The simulation setup consists of the Unity 3D simulation platform together with the high-accuracy simulation engine AGX Dynamics and a reinforcement learning framework built on the PPO-algorithm available through the open source Unity ML-Agents Toolkit, an algorithm that has displayed promising performance and stability on a range of different tasks of varying complexity (Juliani et al., 2020). We investigate two approaches to this grasping task; a multi-agent approach and a single-agent approach using curriculum learning.

This section begins with an introduction to the simulation platform and the tools used in this research, followed by a formal problem formalization and a description of the forestry crane simulation model used in our simulations. Finally, details of each approach investigated in this thesis are described.

3.1 Problem Formalization

For our purposes, the grasping problem of the forestry crane manipulator is formulated as a Markov decision process (MDP); see Section 2.1 for theoretical details. The *agent*, which controls the forestry crane simulation model (see Section 3.3), is at a given time t in state $s_t \in \mathcal{S}$ making an observation $o_t \in \mathcal{O}$, where the observation space $\mathcal{O}$ refers to the environment observable to the agent. Following the policy $\pi(a|s)$, the agent performs action $\pi(a_t|s_t)$, whereupon it receives the reward $r_t = \mathcal{R}(s_t, a_t)$, where $\mathcal{R}(s_t, a_t)$ is some specified reward function. The problem is then reduced to finding the optimal policy $\pi^*(a|s)\ \forall s \in$

$\mathcal{S}$, for which the total expected return over an episode is maximized.

3.2 Simulation Environment

In this work, the simulation platform *Unity* is used as a training environment for the agent, here constituted by the forestry crane simulation model. Unity is interfaced with the simulation engine *AGX Dynamics*[1] and the Unity plugin *ML-Agents Toolkit*[2], which together enable advanced physics simulation with Unity scenes as machine learning training environments. Figure 3.1 shows an overview of the simulation setup. A more detailed description of the three main components, Unity, AGX Dynamics and the ML-Agents Toolkit, are provided in the following subsections.

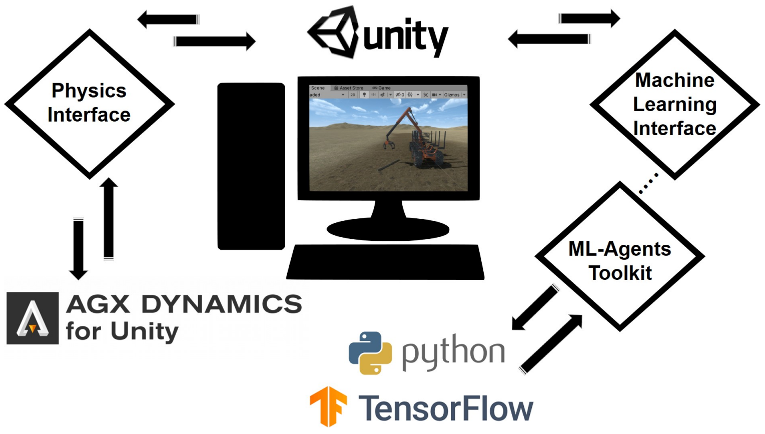


Figure 3.1: *An overview of the simulation setup.*

3.2.1 Unity

The simulation platform used in this research is *Unity*; a general platform supporting advanced 3D simulation development in real-time (Juliani et al., 2020). While used in various domains, ranging from the movie industry to mechanical engineering, Unity is perhaps most well-known for its popularity in the gaming industry.

The Unity platform consists of an incorporated multi-purpose engine offering not only physics simulation through the default engines Nvidia PhysX and Havok Physics, but also advanced rendering. In this work we use the AGX Dynamics plugin for Unity, *AGX Dynamics for Unity*, to employ advanced

[1]https://www.algoryx.se/agx-dynamics/
[2]https://github.com/Unity-Technologies/ml-agents

multi-body dynamics; see Section 3.2.2 for details. In addition to comprehensive physics integration, real-time graphical rendering and programmatic control through its C# scripting features, Unity offers an easy-to-use graphical user interface. This combination has made Unity a leading platform in game development and similar domains built on interactive 3D content, with increasing potential in research and engineering through its platform for building complex machine learning and simulation environments, in particular since the launch of the Unity ML-Agent Toolkit detailed in Section 3.2.3. This work is carried out using Unity 2019.3.11f1.

3.2.2 AGX Dynamics

Simulation-to-reality transfer is a requirement in the development of autonomous systems using simulated environments. This makes reliable physics simulation in complex systems a necessity in order to ensure correct system evolution over time. AGX Dynamics is a physics-based simulation and modeling tool for research- and industry purposes, enabling realistic simulations of constrained, multi-domain, multi-body mechanical systems under nonsmooth dynamics, including systems rich in for example impact and dry frictional contacts.

The discrete time integration employed is based on SPOOK; an implicit, fixed-step time-stepping scheme derived from the discrete variational principle, which ensures preservation of fundamental symmetries, and for which linear stability has been shown (Lacoursière, 2007). An advantage of this treatment of dynamical system evolution is that it enables the solutions to remain stable over large integration steps.

In a system of nonsmooth dynamics, the nonlinear complementary problem can be linearized to a mixed complementary problem. AGX Dynamics invokes both a direct and an iterative solver for such problems, as well as a hybrid of the two. This flexibility yields control of the accuracy-scalability tradeoff, allowing for varied precision between subsystems. This offers a desired imbalance between machine precision and efficient approximation where so is required, such as in systems displaying both nonsmooth granular dynamics and large vehicle dynamics (Servin and Brandl, 2018). The AGX Dynamics SDK includes a joint- and constraint library, providing common and easy-to-use joint configurations for which the technical specifications can be adjusted depending on the model requirements. For example, joints can be relaxed to introduce linear elasticity, non-holonomic constraints can be used to model simple motors and range limits on the force or motion can be used to induce complex joint behavior. Moreover, the joints can be parametrized to mirror physical systems in the real world. These properties together allow for the use of joints to model complicated multi-body systems.

The demonstrably high performance of the dynamical solutions attained by AGX Dynamics can also be attributed to its sophisticated contact detection and contact reduction techniques. Overall, it is an advanced simulation engine ensuring high, scalable performance in simulations of multi-domain mechanical systems. In the current work, an integration of AGX Dynamics with Unity, *AGX Dynamics for Unity*[3], is used, enabling reliable multi-body dynamics to be simulated in real-time or faster, while retaining the high-grade simulation graphics provided by Unity. In contexts of interactive 3D content, AGX Dynamics is sometimes referred to as the *physics engine*, here overriding the default physics engines offered by Unity. This work is carried out using AGX Dynamics for Unity 2.29.0.0.

3.2.3 Unity ML-Agents Toolkit

The Unity ML-Agents Toolkit (Juliani et al., 2020) is an open source toolkit for machine learning implementation, enabling the use of Unity scenes as training environments for reinforcement learning agents, and the setup makes the deployment of trained models into a scene straightforward through the neural network inference engine Barracuda. Two TensorFlow-based (Abadi et al., 2015) reinforcement learning algorithms are offered, including the PPO-algorithm used in this work. These machine learning algorithms are provided through the *mlagents* Python package[4]. Interaction between the low-level Python API and Unity is carried out through an external communicator. In addition to these components, an OpenAI gym wrapper is included to facilitate research in the machine learning community. The ML-Agents Toolkit enables several advanced machine learning concepts to be deployed, such as curriculum learning (see Section 2.4.3), curiosity-driven exploration (see Section 2.4.1), imitation learning and domain randomization.

The hierarchical composition of scene objects and components in Unity offers flexible multi-agent reinforcement learning solutions. The ML-Agents C# SDK constitutes the core of the toolkit, providing a learning environment built on the interaction between agents and a global academy managing the simulation environment. The model, used for training and inference, is denoted with a specific *behavior*, which can be shared between multiple agents. A final important property is that physics simulations can be run independently from the graphical rendering, significantly accelerating training when visual sensory input is not required.

We conclude this section with a summary of the reinforcement learning pipeline using the Unity ML-Agents Toolkit. See Figure 3.2 for an illustration. During training, an *agent* in a *Unity* scene collects observations of its environment, which are communicated to the Python API via the external communicator. The observations are processed by the Python API, which communicates with the Python trainers, sending back the next action to take. When training has converged, the final model can be embedded directly into the Unity scene. The Python API is no longer part of the workflow, and the inference observations are fed directly into the model network, which generates the optimal action. This work is carried out using the Unity ML-Agents Toolkit 1.0.2 (Release 5) and Python 3.7.7.

[3]https://www.algoryx.se/agx-unity/

[4]https://pypi.org/project/mlagents/

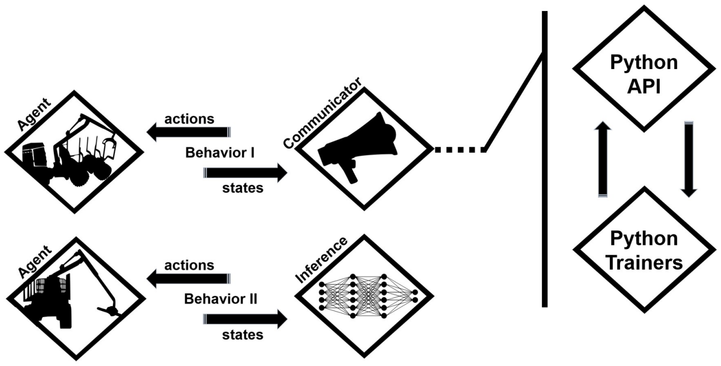


Figure 3.2: *An overview of the ML-Agents Toolkit.*

3.3 Simulation Model

In this work, a 3D model of a forestry crane manipulator constitutes the reinforcement learning agent. The complete simulation model of the forwarder is depicted in Figure 3.3 a), with a close-up view of the forestry crane manipulator provided in Figure 3.3 b). It is a slightly modified version of the XT28 simulation model (Vesterlund and Servin, 2020), consisting of 52 rigid bodies and 60 constraints in total. The physical parameters of each rigid body and constraint are estimated to simulate a realistic crane behavior, but have not been validated by domain experts. This does not reduce the complexity of the problem, and the methods used and conclusions drawn therefore also apply to a validated model.

The kinematical configuration can be described using seven degrees of freedom, six of which are actuated joints equipped with velocity-controlled motors, q_1, q_2, q_3, q_4, q_5 and q_6, and one of which is composed of two additional non-actuated joints, here jointly denoted q_f. Each motor is internally controlled to reach a given reference velocity while obeying current dynamics and constraints, such as maximum motor torques or maximum motor speeds. See Section 3.4.3 for details on how the motors are controlled. Figure 3.4 shows a schematic image of the forestry crane manipulator, marking each degree of freedom. Here, the position and orientation of the forwarder is disregarded. q_{2f} and q_{3f} are non-actuated joints kinetically constrained by q_2 and q_3, respectively. Moreover, we see that the model exhibits the redundant kinematical structure of conventional forwarders through links q_2, q_3 and q_4.

As depicted in Figure 3.4, the global frame of reference is a Cartesian coordinate system with y-axis in the direction opposing gravity and the x-axis along the direction of the vehicle's axis of symmetry. The coordinate system is fixed and centred at the base of the crane, and only moved outwards to increase visibility in the sketch. When describing the motion of each joint, we refer to local coordinate systems moving with the crane. The kinematics is easily visualized by moving the depicted global coordinate system to the centre of the

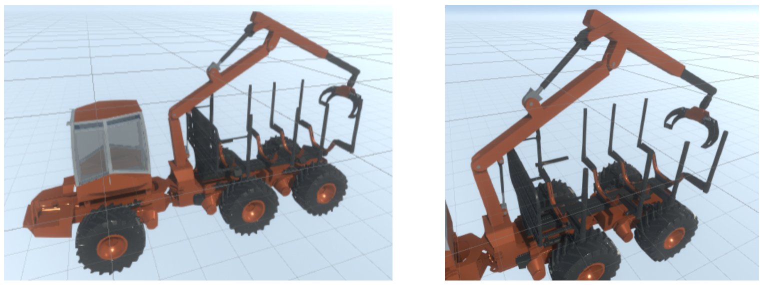

(a) ***(b)***

Figure 3.3: *a) Full view of the forwarder simulation model in Unity. b) Close-up view of the simulation model of the forestry crane manipulator in Unity.*

corresponding joint, aligned with the crane in its current position.

The first link, q_1, is a *revolute joint*, or *hinge*, with an angular motion motor constraint. A revolute joint constrains five degrees of freedom, in this case restricting q_1 to rotation around the positive y-axis centred at the joint. Similarly, the second and third links, q_{2f} and q_{3f}, are both revolute joints allowing for rotational motion around the positive z-axis. Together, these joints provide the kinematics of a 3-link elbow manipulator, controlling the motion of the *pillar* (mounted on the *fixed base*), the *inner boom* and the *outer boom* of the crane. q_{2f} and q_{3f} are non-actuated joints, to which the crane kinematics are transferred from the actuated *prismatic* joints q_2 and q_3, respectively. Prismatic joints constrain motion to translation along one axis, in this case along the axis of the hydraulic cylinders depicted in the figure. The fourth joint, q_4, is another actuated prismatic constraint which allows for translation of the crane's *telescope* along its axis.

The remaining joints, q_5, q_f and q_6, control the motion of the end-effector, here referred to as the *grapple*. q_5 is a revolute joint restricting motion to rotation around the grapple's axis of symmetry, whereas q_6 is a prismatic joint controlling the grasping motion of the grapple. q_f denotes two free revolute joints allowing for none-actuated rotation around the positive z-axis and the positive x-axis, respectively. This imposes additional control complexity, as exact crane maneuver requires adjusting control of the actuated joints to compensate for grapple motion in the xy-plane and the zy-plane due to the dynamics of the crane and the environment. The compliance of both free joints are set to 0.5 rad/Nm and the damping coefficients are set to 0.034 Nms/rad. In Figure 3.4, q_f is located at the *boom tip*, which we use to define the position of the grapple.

In addition to the *motor*, each actuated revolute and prismatic joint has secondary constraints in terms of a *range* and a *lock*, the former limiting the working range of the constraint and the latter forcing the constraint to a fixed position at zero-velocity.

A summary of the actuated joints can be found in Table 3.1. The angle range of each constraint is included, as well as its force range and maximum

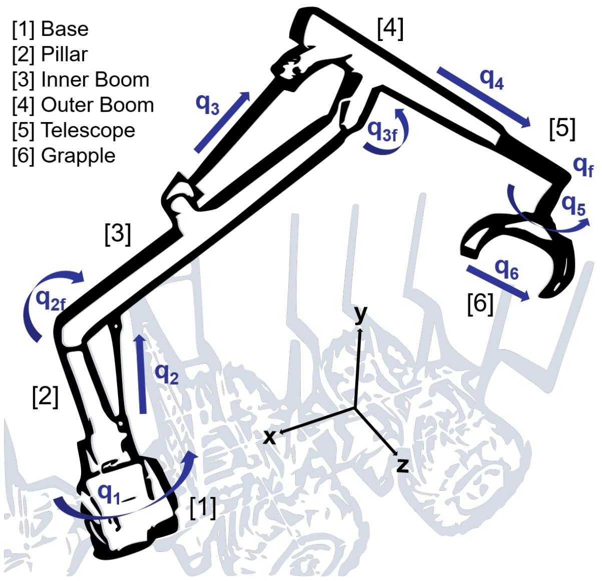


Figure 3.4: *A schematic illustration of the forestry crane manipulator, marking crane parts (black) and joints (blue). Revolute joints are marked with curved arrows in the direction of rotation, and prismatic joints are marked with straight arrows in the direction of translation.* q_1, q_2, q_3, q_4, q_5 *and* q_6 *are actuated, whereas* q_f *denote two non-actuated revolute joints.* q_{2f} *and* q_{3f} *are non actuated joints, kinematically constrained by* q_2 *and* q_3*. A more elaborate model description is provided in the text.*

speed. The position of the crane depicted in Figure 3.3 and 3.4 is the reference position for the positions and angles of the joints, i.e. at this position, each joint's position or angle is 0.

The crane and the target log are placed on a horizontal, static floor. For simplicity, the target log is modelled as a uniform cylinder of length 3 m, radius 0.08 m and mass 50 kg. The properties of the contact material between the gripping claws and the surface material of the target log include a surface uniform friction coefficient of 0.7, a uniform surface viscosity of $5 \cdot 10^{-8}$ Pa $\cdot$ s, a damping coefficient of 0.075 Nms/rad and a Young's modulus of magnitude $4 \cdot 10^{8}$ N/m^2.

	Constraint	**Min Angle**	**Max Angle**	**Max Speed**	**Max Force**
q_1	Revolute	$\frac{\pi}{2}$ rad	$\frac{\pi}{2}$ rad	1.0 rad/s	$\pm$ 30 kNm
q_2	Prismatic	-0.17 m	0.19 m	0.1 m/s	$\pm$ 800 kN
q_3	Prismatic	-0.01 m	0.84 m	0.4 m/s	$\pm$ 400 kN
q_4	Prismatic	-1.25 m	0.94 m	1.0 m/s	$\pm$ 400 kN
q_5	Revolute	$-\infty$	∞	0.2 rad/s	$\pm$ 8 kNm
q_6	Prismatic	-0.02 m	0.22 m	0.5 m/s	$\pm$ 400 kN

Table 3.1: *Summary of actuated joints, including the constraint type, maximum and minimum angles, maximum motor speed and motor force (or torque) range for each actuator.*

3.4 Multi-Agent Approach

We have reduced the complete forwarding and grasping problem to a problem in which the agent grasps a single log from a static vehicle on a horizontal surface. This limits the number of degrees of freedom to seven, as discussed in the previous subsection. The multi-agent approach splits the complete grasping task into two distinct tasks; a navigation task and a reduced-complexity grasping task, where grasping is performed from close proximity to the target log. The motivation is that performing the complete grasping task in a system dynamically controlled by six actuators requires learning complicated behavior in a natural sparse-reward setting. Exploring the environment efficiently enough to grasp the target log, and thus beginning to learn task completion, is difficult, especially in continuous state- and action spaces.

The agents, or sub-policies, performing each task are referred to as *Agent 1* and *Agent 2*. The goal of Agent 1 is to navigate to and remain still with the boom tip at a point 3.0 meter above the log centre of mass, and the goal of Agent 2 is to grasp the target log and lift it from the ground starting from the end-configuration of Agent 1. Agent 1 makes use of five degrees of freedom controlling actuators q_1, q_2, q_3, q_4 and q_5, i.e. all actuated joints except q_6. q_6 corresponds to the grasping motion of the grapple, which is considered redundant in the navigation task. In the case of Agent 2, actuators q_5 and q_1 are locked, thereby restricting the rotation angle of the grapple to the current rotation angle at the end-configuration of Agent 1, and restricting motion of the crane to the xy-plane in the local coordinate system moving with q_1.

3.4.1 Initial Condition

The initial condition of Agent 1 places the grapple above the vehicle's load bunk, with actuator angles according to Table 3.2. At the beginning of each episode, the angles are slightly perturbed according to a uniform distribution over the displayed ranges. The initial conditions should be viewed with the reference position of the crane in mind, see Figure 3.3 and 3.4.

	Constraint	**Initial Angle**
q_1	Revolute	0.000 ± 0.150 rad
q_2	Prismatic	0.145 ± 0.003 m
q_3	Prismatic	0.158 ± 0.005 m
q_4	Prismatic	0.000 ± 0.150 m
q_5	Revolute	0.000 ± 0.150 rad
q_6	Prismatic	0.000 ± 0.000 m

Table 3.2: *The initial condition ranges of the angle (or position) of each actuated joint. The grapple prismatic q_6 is initially open. The resulting initial Cartesian position of the boom tip in the global coordinate system is* $(-6, 96, 5, 33, 0.00) \pm (0.15, 0.07, 0.73)$ *m.*

To allow the agent to generalize its behavior to target logs at different positions, the initial position of the target log is randomly distributed over the area highlighted in grey in Figure 3.5. This is referred to as the *legal* target log area. The distribution of target log positions is uniform in the $r\theta$-plane. The area is designed such that the target log remains well within grasping reach of the forestry crane manipulator. The target log position refers to the centre of mass.

The rotation of the target log about the y-axis is also drawn from a uniform random distribution. To prevent the log from rotating beneath the vehicle, the possible rotation angles linearly increase with the distance to the load bunk on the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, reaching the upper limit only for logs no closer than the right-angled distance of half a log length to the vehicle.

Agent 2 continues where Agent 1 leaves off. In the training of Agent 2, the transition between the agents occurs when the boom tip of the crane constituting Agent 1 reaches within a distance of 0.2 m from the target position.

3.4.2 State Space

An agent can observe the states of each actuated joint, as well as the Cartesian target log position and the target log rotation about the y-axis. Thus, the state space is continuous and assumes an external perception system. Each actuator state consists of the actuator position or angle, the current force applied to the

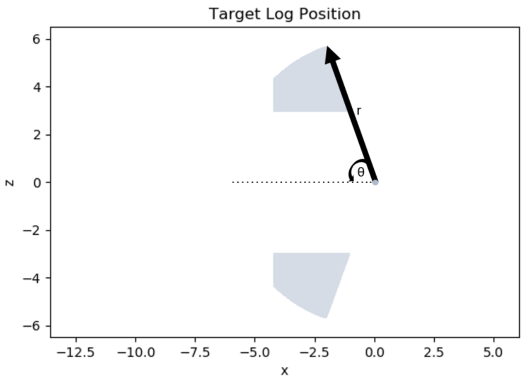


Figure 3.5: *The legal target log area is highlighted in grey. The origo of the polar coordinate system is located at the centre of mass of the fixed base of the forestry crane manipulator. The dotted line marks the symmetry axis in the direction of the vehicle, the end marking the initial position of the grapple in the xz-plane.*

motor, as well as the current motor speed. Observations from the 7 previous time steps are stacked to simulate short-term memory, and each observation is normalized based on the running mean and standard deviation of previous observations.

Agent 1 observes the states of actuators q_1, q_2, q_3, q_4 and q_5. q_6 is omitted since information about the closing-opening motion of the grapple is redundant in the navigation task. Accounting for the stacking of observations, the resulting size of the state vector is then 152. For the first 7 steps, the state vector is padded with zeros.

Agent 2 observes the states of the joints q_2, q_3, q_4 and q_6, as well as the Cartesian target log position. q_1 and q_5 are locked, since control of these joints are redundant for completion of the Agent 2 task, and the corresponding actuator states are not observed. With 8 stacked observations, the size of the Agent 2 state vector then amounts to 120.

3.4.3 Action Space

The agent is controlled by action signals given to the velocity-controlled motor of each actuated joint. Agent 1 is controlled by the five actuated joints q_1, q_2, q_3, q_4 and q_5, and receives action signals corresponding to the velocity changes of the corresponding motors. Likewise, Agent 2 receives action signals controlling the motor velocity changes of the four joints q_2, q_3, q_4 and q_6. The size of the action vector is then 5 for Agent 1 and 4 for Agent 2. Upon receiving an action signal, the motor tries to apply a force such that the corresponding velocity is reached in the degree of freedom it controls. The force of the motor is, however, limited, as described in Section 3.3.

The agents make a decision every other time step of each episode. Each action a is normalized to the interval $a \in [-1, 1]$ of a continuous action space, and the actions are then transformed to the corresponding velocity change of the motor constraints. The maximum velocity change at each time step for joints q_1, q_2, q_3, q_4 and q_5 corresponds to a thirtieth of the maximum speed of that particular joint. Thus, at zero-velocity, each joint requires 30 time steps to reach maximum speed, which enforces a smoother velocity profile and limits the acceleration of the crane. The limit is chosen to simulates the physical restrictions of the machine more realistically, but has not been validated by experts.

3.4.4 Reward Structure

The reward function of Agent 1 is shaped to incrementally reward the agent for reaching a point 3.0 m above the target log centre of mass, and remain at this point with the grapple in a grasping position, i.e. its rotation aligning with the rotation of the log. The target point is referred to as the *target position*. The reward function is presented in (3.1), where r_t is the reward received at time step t.

$$r_t = p_t^e \cdot p_t^r \cdot r_t^d (1 + r_t^s) \tag{3.1}$$

Here, r_t^d is a reward factor exponentially decreasing with the distance from the boom tip to the target position. Similarly, when the boom tip is within 0.5 m from the target position, the factor r_t^s awards the agent an exponentially increasing reward inversely dependent on the extension speed of the telescope and the distance between the boom tip and the target position. This is necessary to prevent unnatural solutions in which the agent utilizes the kinematic redundancy to hold the boom tip at the target position while still moving q_2, q_3 and q_4 synchronously. r^s is illustrated in Figure 3.7 e). It can be decomposed into a speed dependent part r^{s_1} and a distance dependent part r^{s_2}, such that $r^s = r^{s_1} r^{s_2}$. The two components are illustrated in Figure 3.7 c) and d). Figure 3.7 a) illustrates the distance reward r^d.

p_t^e is a penalty factor scaling down the reward if the telescope extension exceeds the angle range of q_4, specified in Table 3.1. This avoids illegal solutions where the grapple is held still by continuously applying force at the maximum or minimum telescope extensions. Finally, p_t^r is a factor scaling down the reward depending on the relative rotation of the grapple and the target log. Figure 3.7 a) and b) illustrate p^e and p^r, respectively.

Moreover, the agent is penalized for illegal behavior. These include the grapple colliding with the load bunk, the ground or the wheels of the vehicle, q_1 moving beyond 98% of its working angle range, and q_2 or q_3 reaching within 98% of the maximum or minimum limit of their respective force ranges (see Table 3.1). If an illegal move is detected, the agent is awarded a zero-reward and the episode is terminated prematurely.

The reward function of Agent 2 relies less on reward shaping. Initially, the reward function (3.1) applies with the target position 1.3 m above the centre of

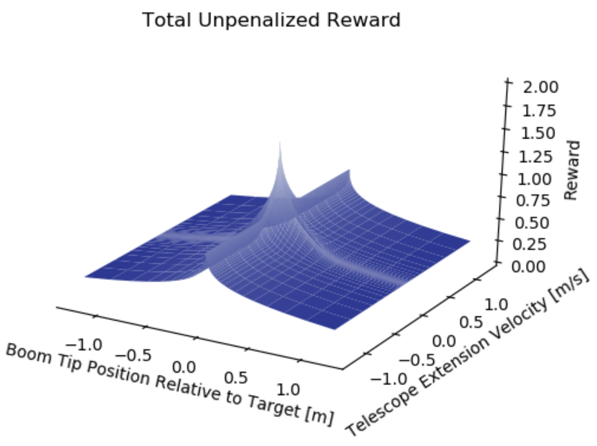


Figure 3.6: *The total unpenalized reward function $r = r^d(1 + r^s)$. The reward at each time step is dependent on the distance between the boom tip of the crane and the target position, as well as on the telescope extension speed.*

mass of the target log, such that the grapple claws are placed slightly above the surface of the log. The reward increases when the log is secured in the grapple, which defines a successful grasp. When the log is in contact with both grapple claws, the current reward increases by a factor of 2. If the target log is moved one radius in the y-direction, while in contact with both grapple claws and the upper part of the grapple, the grasp is considered a success and the reward instead increases by a factor of 10.

Agent 2 is penalized for the same illegal behaviors as Agent 1, the only difference being the endorsement of collisions between the grapple claws and the ground for Agent 2.

3.5 Single-Agent Approach

The single-agent approach solves the forwarding problem identical to the problem solved by the two agents of the multi-agent approach, but, as the name suggests, uses a single agent to perform the task of both navigating to and grasping the single log from a static, horizontal surfaced vehicle. This is done using *curriculum learning*; see Section 2.4.3 for theoretical details. With this approach, an additional experiment is carried out in which energy optimization is added to the goal of the agent. The remainder of this section details the state space, action space and reward structure used, as well as the different curricula investigated. In this approach, no actuated joint is locked and all six actuated joints are controlled by the agent.

3.5.1 State Space

The observation space is similar to that of Agent 1 in the multi-agent approach, see Section 3.4.2. The only difference is that the states of all actuated joints (q_1, q_2, q_3, q_4, q_5 and q_6) are observed. In addition to the actuator states, the Cartesian target log position and the target log rotation about the y-axis is observed. Thus, accounting for the stacked observations, the state vector length is 176.

3.5.2 Action Space

Identical to the multi-agent approach, the agent makes a decision every other time step, the action space is continuous and each action corresponds to a velocity change of each motor constraint. The action space is similar to that of Agent 2 in the multi-agent approach (see Section 3.4.3), although in this case all actuated joints (q_1, q_2, q_3, q_4, q_5 and q_6) are controlled. Thus, the length of the action vector is 6.

3.5.3 Reward Structure

The reward structure is similar to that of Agent 2 in the multi-agent approach, see Section 3.4.4, the difference being that the agent is given a total cumulative reward of 10 000 when the log is secured in the grapple, upon which the episode is terminated. In practice, this means that the shaped reward function incrementally increasing the agent's reward as it navigates to the target position is negligible in the total return after the agent has learned a grasping behavior.

An additional experiment is carried out in which the reward received upon successful grasps is scaled by the inverted ratio between the current total energy consumed by the agent, and the average total energy consumed by an agent based on the corresponding curriculum without energy optimization. Thus, the agent is given incentive to perform the grasping task as energy efficiently as possible. The total energy is defined as the energy consumed by constraints $q_1 q_2, q_3$ and q_4 up until the grasping initiation, i.e. only the energy consumed as the crane moves towards the target log is measured.

3.5.4 Curriculum

In order to gradually increase the difficulty of the problem, the height of the floor on which the target log is resting relative to the grapple varies with the *lessons* constituting the *curricula* of the curriculum learning training process. We investigate three different curricula, where the height of the floor is decreased with intervals of 0.5 m, 0.1 m and 0.01 m, respectively.

The initial condition in terms of the actuator states is identical to that of Agent 1 (see Section 3.4.1), and the legal target log area at the final lesson in each curriculum is identical to that of Figure 3.5. As such, the final task solved by each approach is the same. A major difficulty lies in the sparse reward setting accompanying the grasping task. Using the multi-agent approach, the combined behavior of the agents may not converge to the optimal grasping behavior of a single agent. Using curriculum learning, the difficulty is gradually increased which, ideally, increases learning speed, performance and the probability of finding the optimal grasping behavior.

Under the hypothesis that the learning process using a single agent is slow due to the few grasping attempts occurring during the training, the curricula investigated initially places the target log in close proximity to the grapple, gradually increasing the distance. Since no path planner is implemented, this is realized by gradually changing the height of the floor on which the target log is placed, and varying the legal area for the initial target log position.

Three curricula is investigated. The first four lessons are common to all three of them with a floor height of 3.55 m, placing the log directly beneath the grapple and above the vehicle's load bunk. In lesson 2-4 the legal target log area gradually shifts to the side of the load bunk. This is illustrated in Figure 3.8 a) to d). Since the area where the manipulator can successfully grasp the target log varies depending on the height of the target log relative to the ground, a further gradual shift of the area occurs with the floor height. Thus, in succeeding lessons, the inner and outer radii of the circles enclosing the blue area in e.g. Figure 3.8 c) varies linearly with the floor height throughout the curricula. This is where the three curricula diverge: The floor of the first curriculum is lowered 0.5 m between later lessons, while the corresponding number is 1 dm for curriculum 2 and 1 cm for curriculum 3. While the increase in difficulty between lessons is greater in the first curriculum compared to the second and third curricula, the number of lessons is significantly lower.

The agent advances to the succeeding lesson after obtaining an average return of 3000 over 20 subsequent episodes. With the reward structure defined in the previous subsection, this corresponds to a grasping success rate of 30%. The comparatively low threshold is chosen to prevent specialisation on the tasks of early lessons, since this may be destructive to the learning process of the final task. Table 3.3 shows the total number of lessons and the floor height variation throughout each curriculum. Figure 3.8 e) and f) illustrate the gradually expanding legal target position area at two later lessons of the first curriculum.

Collisions between the grapple claws and the static height-adjustable floor are disabled throughout the curriculum, while collisions between the grapple claws and the ground remain. This reduces the task complexity in earlier lessons, and forces the agent to learn to adjust its motion relative to the ground level at the final lessons, after mastering the grasping behavior.

3.6 Training Configuration & Architecture

This section details the neural network architecture and the hyperparameters of the PPO algorithm used, as well as important global simulation parameters. These are common to both the multi-agent and the single-agent approach.

The simulation time step is 0.02 s. The agent makes a decision every other

Curriculum 1		**Curriculum 2**		**Curriculum 3**	
Lesson	Height [m]	Lesson	Height [m]	Lesson	Height [m]
1	3.55	1	3.55	1	3.55
2	3.55	2	3.55	2	3.55
3	3.55	3	3.55	3	3.55
4	3.55	4	3.55	4	3.55
5	3.30	5	3.50	5	3.54
6	3.00	6	3.40	6	3.53
7	2.50	7	3.30	7	3.52
8	2.00	8	3.20	8	3.51
⋮	⋮	⋮	⋮	⋮	⋮
12	0.00	42	0.00	361	0.00

Table 3.3: *The number of lessons until ground level in each curriculum, and the floor height corresponding to each lesson.*

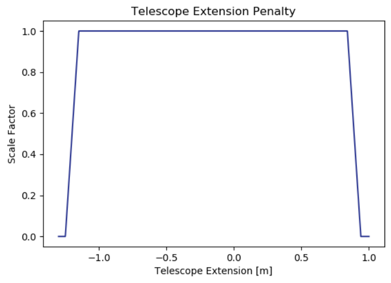
time step and the maximum length of an episode is 2000 time steps for Agent 1, and 400 time steps for Agent 2. The deep learning architecture consists of a 3-hidden layer feedforward neural network with 256 neurons in each layer.

The PPO hyperparameters used are the linear learning rate $\alpha = 0.001$, the clipping parameter $\epsilon = 0.3$, the entropy regularization coefficient $\beta = 0.01$, the GAE parameter $\lambda = 0.95$ and the discount factor $\gamma = 0.995$. See Section 2.3.5 on PPO for details on these hyperparameters. The number of epochs is 3, the mini-batch size 2024 and the buffer size 20240. Each training session is run using 8 parallel environments.

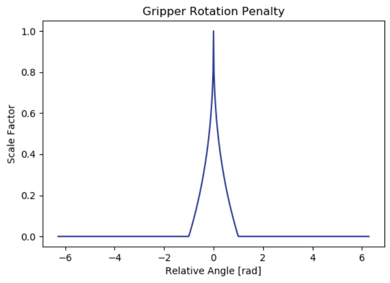
3.7 Safety, Ethics & Responsibility

With today's progress in the development of artificially intelligent (AI) systems, ongoing discussions regarding responsible development of trustworthy AI systems in terms of ethics and safety aspects have become increasingly important. To this end, the European Commission has established an independent High-Level Expert Group on Artificial Intelligence (AI HLEG), which recently released a collection of ethics guidelines for *Trustworthy AI* (AI HLEG, 2019). These guidelines state a number of requirements for responsible AI development covering various areas, including ethics (e.g. non-discrimination and data protection) and safety (e.g. social and technical robustness to avoid unintentional harm).

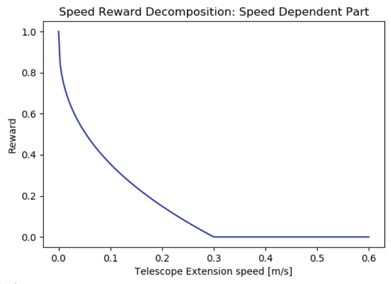
Our work is exclusively conducted in simulated environments and the immediate implications are therefore small. While the current application area likely provides natural adherence to ethical imperatives, it is important to be aware of safety aspects at this stage, as these kinds of autonomous systems require advanced safety systems to be deployed during operation in order to avoid unintentionally harming bystanders, the environment, or the physical machine. In this case, the agent is given the positional coordinates of the target log directly, which assumes the existence of an external perception system that can map the scene to these coordinates. Since the current network is blind to external obstacles and unexpected environmental changes, and thus cannot be trained to react to them, a deployed model needs to be monitored by an operator that can intervene in case of an upcoming safety breach. A separate network for visual perception of the scene allows for greater flexibility in the incorporation of more sophisticated and autonomous safety systems, but even without visual sensors it may be possible to train the network to detect and react to unexpected behavior. This is an important perspective for future research.



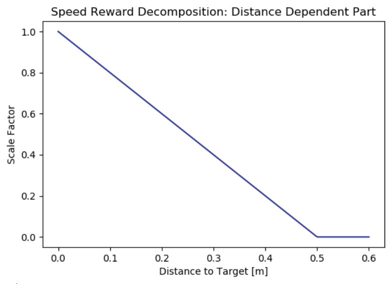
(a) *The agent is penalized if constraint q_4 exceeds the angle range specified in Table 3.1, at which point the scale factor exponentially decreases.*



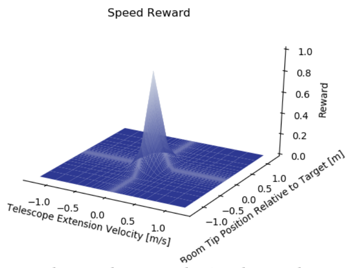
(b) *The agent is penalized if the grapple rotation at all deviates from the rotation of the target log, and the reward is 0 if the relative rotation exceeds 1 rad.*



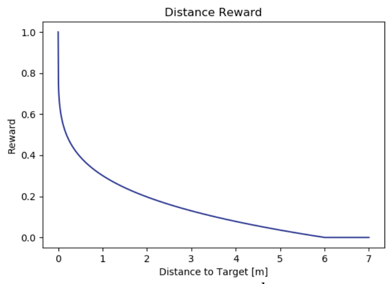
(c) *The speed dependent part r^{s_1} of the speed reward r^s increases exponentially with the decreasing telescope extension speed below a speed of 0.3 m/s, beyond which threshold the reward factor is 0.*



(d) *The distance dependent part r^{s_2} of the speed reward r^s increases linearly with decreasing distance from the boom tip of the crane to the target position. The scale factor is 0 for distances exceeding 0.5 m.*

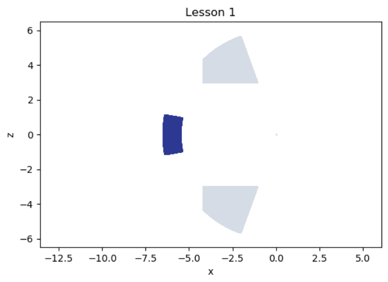


(e) *The total speed reward r^s, depending on both the telescope extension speed and the distance between the boom tip of the crane and the target position.*

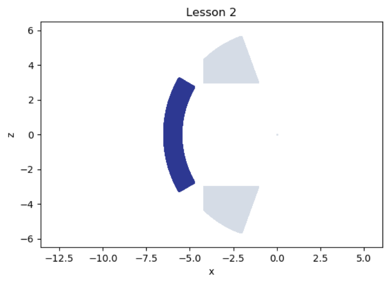


(f) *The distance reward r^d decreases exponentially with the distance between the boom tip of the crane and the target position, and becomes 0 beyond a distance of 6 m.*

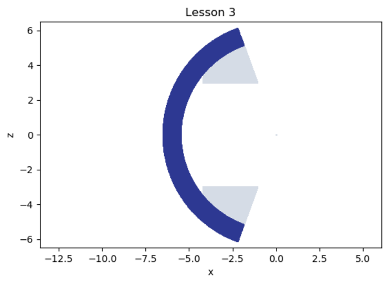
Figure 3.7: *Illustrative descriptions of the components of the total reward function 3.1 used during training of Agent 1.*



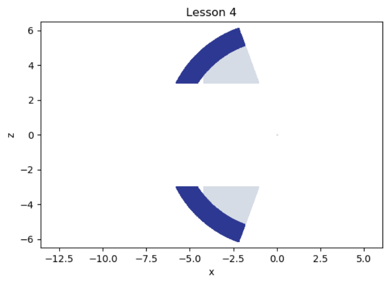
(a) *Legal target log area (blue) during lesson 2, with floor height 3.55 m. Also visible is the legal area at the final lesson (grey).*



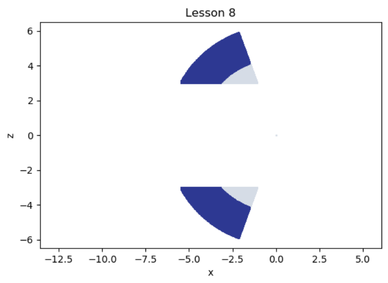
(b) *Legal target log area (blue) during lesson 2, with floor height 3.55 m. Also visible is the legal area at the final lesson (grey).*



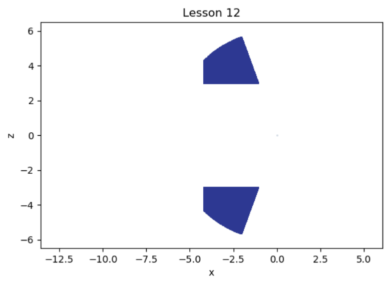
(c) *Legal target log area (blue) during lesson 3, with floor height 3.55 m. Also visible is the legal area at the final lesson (grey).*



(d) *Legal target log area (blue) during lesson 4, with floor height 3.55 m. Also visible is the legal area at the final lesson (grey).*



(e) *Legal target log area (blue) during lesson 8, with floor height 2.0 m. Also visible is the legal area at the final lesson (grey).*



(f) *Legal target log area (blue) during the final lesson. The floor height is 0.0 m.*

Figure 3.8: *Legal target area for different lessons of one of the curricula. The total number of lessons is 12 and the floor is lowered 0.5 m each lesson after lesson 4.*

Chapter 4

Results

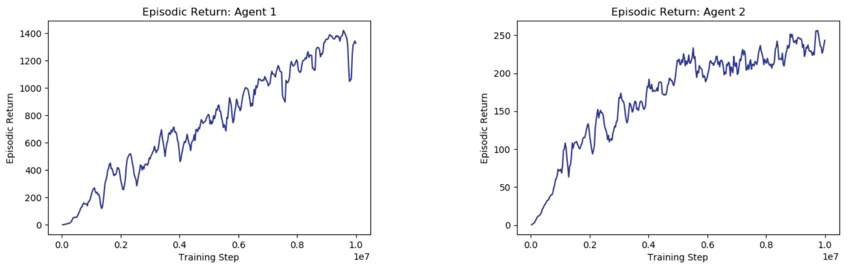
This section summarizes the results obtained using the multi-agent approach and the single-agent approach, as well as results obtained in the evaluation of the robustness of the final models of the single-agent approach. A more thorough discussion of the results presented here is found in Chapter 5.

4.1 Multi-Agent Approach

Figure 4.1 a) shows the Agent 1 learning progress in terms of average episodic return over time for the best model obtained. The corresponding results for Agent 2 are presented in Figure 4.1 b). See Section 3.4.4 for details on the reward structures and goals of the agents. Both agents are trained for 10 million training steps.

The final Agent 1 model is able to reach the target position with a success rate of 99.6% over 1000 consecutive episodes. Episodes constituting the remaining 0.4% are prematurely interrupted due to illegal behavior, in this case exceeding the working range of q_1. In 94.8% of the successful episodes, the distance between the boom tip and the target position is within 0.2 m at the final time step, the maximum distance from which Agent 2 is initialized. The following evaluation of the success of Agent 1 is based on the 99.6% successful episodes.

Figure 4.2 a) shows the Kernel Density Estimate (KDE) of the absolute distance between the boom tip and the target position at the final time step of each episode. This illustrates the precision with which Agent 1 is able to reach its goal, and the mean distance is 0.08 m. The deviation between the boom tip and target position is generally larger in the $z-$direction, according to Figure 4.3. Besides minimizing the distance between the boom tip and the target position, Agent 1 is rewarded for minimizing the telescope speed when the

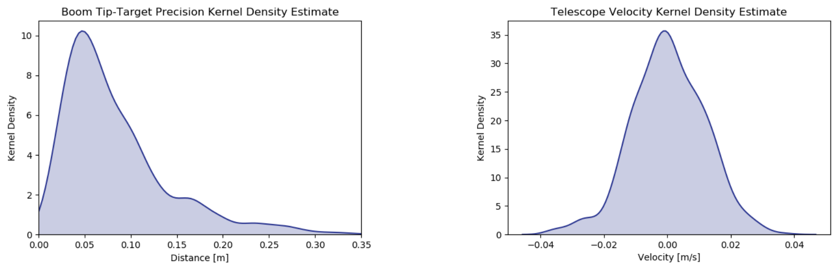


(a) ***(b)***

Figure 4.1: *a) The learning progress of Agent 1 in terms of time evolution of the average episodic return. b) Corresponding learning progress for Agent 2.*

boom tip is in close proximity to the target position. The q_4 velocity measured at the final time step is 0.00 ± 0.04 m/s, according to the distribution presented in Figure 4.2 b).

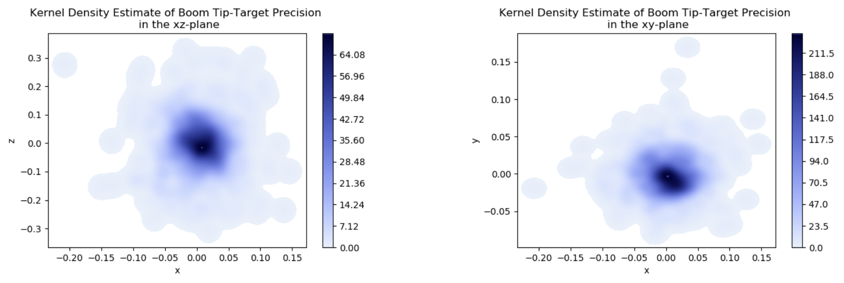
Agent 1 is also penalized for deviations between the global grapple rotation and the rotation of the target log about the global $y-$axis. Figure 4.4 a) shows the correlation between the global grapple rotation and the target log rotation. The q_5 rotation error is illustrated in the KDE of the relative rotation angle presented in Figure 4.4 b).



(a) ***(b)***

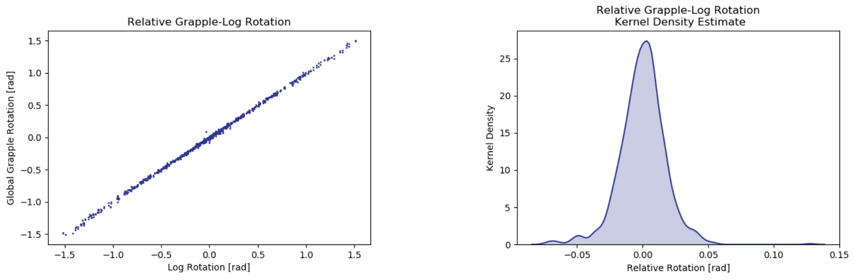
Figure 4.2: *a) KDE of the absolute distance between the boom tip and the target position at the final time step of successful episodes. b) KDE of the telescope velocity (q_4 velocity) at the final time step of successful episodes.*

Agent 2 is able to successfully grasp 76.3% of target logs over 1000 consecutive episodes. Of the failed grasping attempts, 54.7% fails due to illegal behavior, specifically collisions between the crane's inner boom and the load bunk. The remaining failed attempts occur when the grasping motion of q_6 fails to secure the log in the grapple as a result of unsuccessful coordination between the motion of each joint. In the case of both Agent 1 and Agent 2, no correlation between the success of the agents and the initial target log position is identified.



(a) *(b)*

Figure 4.3: *a) KDE of the distance between the boom tip and the target position in the xz−plane at the final time step of successful episodes. b) Corresponding KDE of the distance in the xy-plane.*



(a) *(b)*

Figure 4.4: *a) Correlation between grapple rotation and target log rotation in the global coordinate system. b) KDE of the relative grapple-target log rotation.*

4.2 Single-Agent Approach

Several models, with differences in curricula and reward structures, are obtained using the single-agent approach; see Section 3.5.3 and 3.5.4 for details. To ease comparison between models trained on different curricula, each model based on Curriculum 1, 2 and 3 are trained from lesson 4, inheriting knowledge from the best agent trained on lesson 1-4, since these lessons are common to all curricula. Figure 4.1 a) shows the learning progress, in terms of episodic return over time, for three independent agents trained on lesson 1-4. This illustrates examples of the deviation in the learning progress observed between training sessions. The corresponding lesson transitions are presented in Figure 4.5 b). The agents need a slightly different number of training steps to reach the cumulative reward threshold required to move to the next lesson. The agent trained using Training Session C completes lesson 4 in the least number of steps and arrives at the highest cumulative reward. This model is used to initialize the training sessions of agents trained using the different curricula presented below.

The best models obtained using Curricula 1, 2 and 3 are presented in Figure

4.6 a), with corresponding lesson transitions presented in Figure 4.6 b). The best average episodic return is achieved by the model based on Curriculum 2, and the energy optimized agent is trained on this curriculum. The learning progress for the energy optimized model is displayed in Figure 4.7. The former models are trained for 30 million time steps, at which point the episodic return shows a stabilizing tendency close to the theoretically maximum total return of 10000. This corresponds to training sessions consisting of approximately 150 000 episodes, depending on the length of each episode. The energy optimized model has no theoretical return limit and the average return has still not stabilized after 80 million training steps, at which point training is interrupted and the model is evaluated. The following results should be viewed in light of these training results.

To show the variation in the learning progress between different training sessions of agents trained on an identical curriculum, the learning progress of three different training sessions using Curriculum 2 is presented in Figure 4.8. The corresponding lesson transitions are presented in Figure 4.8 b).

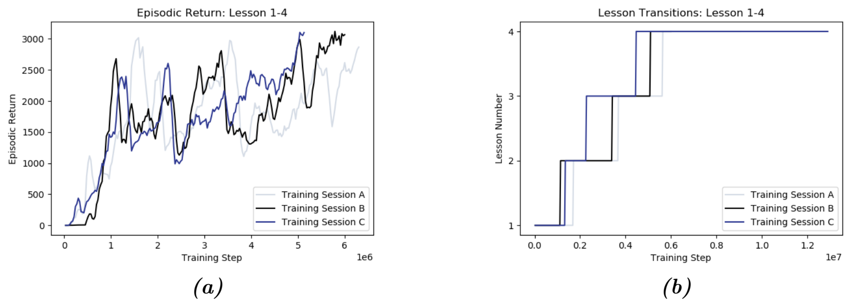


Figure 4.5: *a) The learning progress during the first four lessons for three different agents. b) Corrected lesson transitions.*

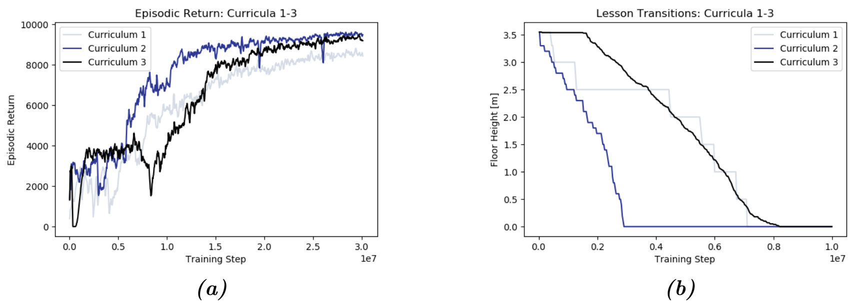


Figure 4.6: *a) The learning progress for agents trained on Curriculum 1,2 and 3, respectively. b) Corresponding lesson transitions.*

The definition of a successful grasp, i.e. when the agent is able to secure

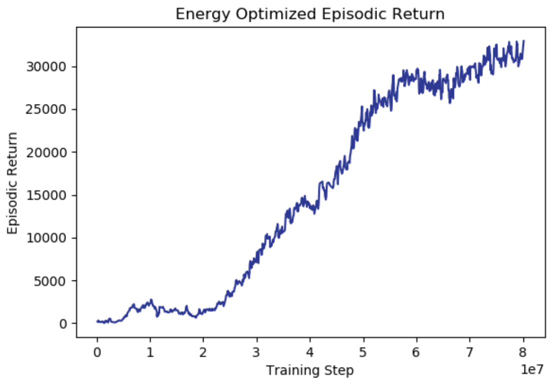
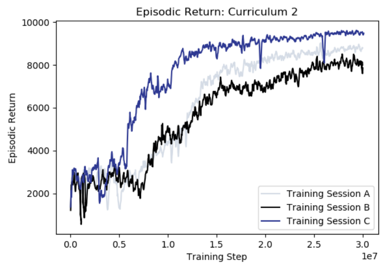
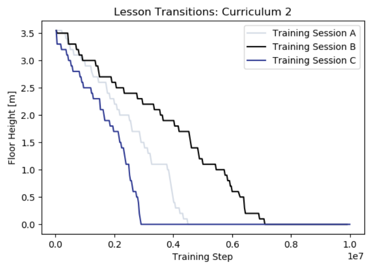


Figure 4.7: *The learning progress of the agent trained on Curriculum 2 with energy optimization.*



(a)



(b)

Figure 4.8: *a) The learning progress for three agents trained on Curriculum 2. b) Corresponding lesson transitions.*

the target log in its grapple, is given in Section 3.4.4. The best model in terms of grasping success rate is obtained using Curriculum 2, for which the grasping success rate is 97.4% over 1000 episodes. This can be compared to the success rates of 88.4% and 88.0% for the best models based on Curriculum 1 and 3, respectively. At 80 million training steps, the energy optimized Curriculum 2-based model reaches a success rate of 80.7% over 1000 episodes. Here, the scale factor used in the reward function (see Section 3.5.3) is 8.3 kJ; the mean episodic energy consumption of the Curriculum 2-based model without energy optimization.

The following evaluation of the control policies of each model is based on episodes during which successful grasping occurs. To compare the solutions of different models, the end-configuration of the crane is defined as the crane configuration at grasping initiation, up to which point the energy consumption is measured, as described in Section 3.5.3. The reason for this is that a large part of the total energy is consumed during grasping, and our goal is to measure the difference in energy consumption depending on how the crane reaches the target log depending on the learned control policy.

Figure 4.9 shows a KDE of the end-configuration angle (or position) for each constraint q_1, q_2, q_3 and q_4, over successful episodes for the best models based on Curriculum 1, 2 and 3, as well as the Curriculum 2-based energy optimized model.

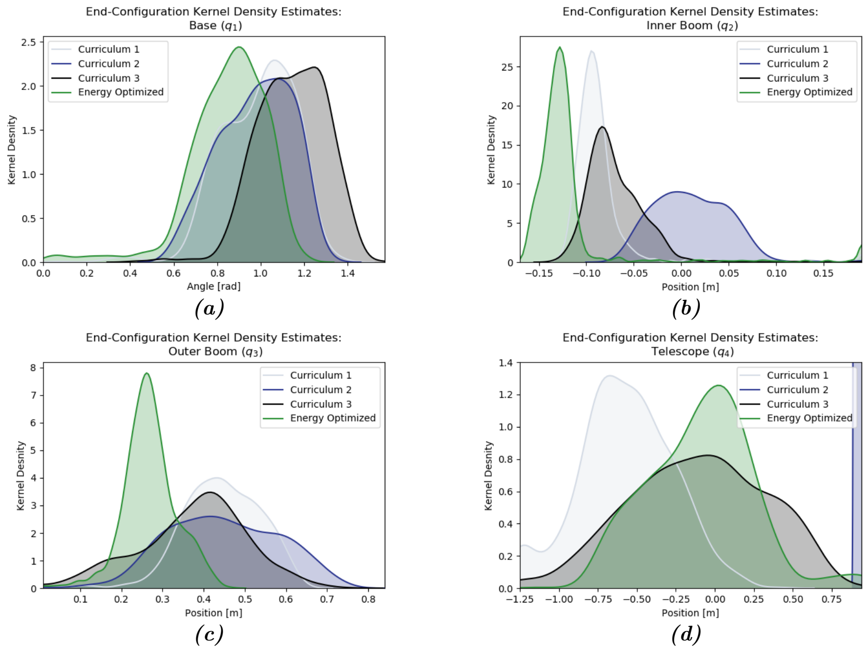


Figure 4.9: *Distribution of end-configuration angles for constraints q_1, q_2, q_3 and q_4 using the best models based on Curriculum 1, 2 and 3, as well as the energy optimized model based on Curriculum 2.*

A KDE of the total energy consumption of constraints q_1, q_2, q_3 and q_4 is displayed in Figure 4.15 a) - d) for the four models. The corresponding distribution of the total energy consumption is presented in Figure 4.15 e). As can be seen, the mean energy consumed using the energy optimized model is significantly lower than the corresponding energy consumption of all non-energy optimized models. In this case, the mean energy consumed using the energy optimized model is 3.46 kJ, compared to 8.31 kJ for successful episodes using the same curriculum without energy optimization. This corresponds to a decrease in the total energy consumption of 58.4%. The mean energy consumption for the best model obtained using Curriculum 1 and 3 is 5.87 kJ and 6.79 kJ, corresponding to an energy decrease of 41.1% and 49.0%, respectively, in the energy optimized model.

On average, a grasping cycle takes 3.5 seconds for the model based on Curriculum 2, compared to 4.6 seconds for the energy optimized model. For the models based on Curriculum 1 and 3 the corresponding grasping cycle time is 3.8 and 3.9 seconds, respectively. Figure 4.10 shows the boom tip speed during five different grasping cycles using the model based on Curriculum 2 and the energy-optimized model, illustrating the smoother and lower boom tip speed profiles generally obtained by the energy-optimized model. The log position and orientation for each of the five example cycles are distributed across the target domain.

Observations suggest a general reduction in the magnitude of the grapple oscillations relative to the boom tip in the energy optimized model compared to the corresponding non-energy optimized model. An example illustrating this for one of the grasping cycles is presented in Figure 4.11 a). The boom tip acceleration profile during the corresponding grasping cycle is presented in Figure 4.11 b).

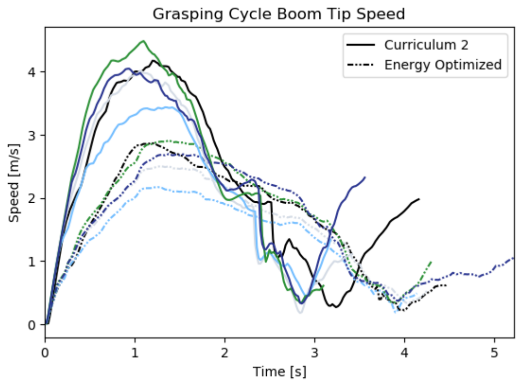
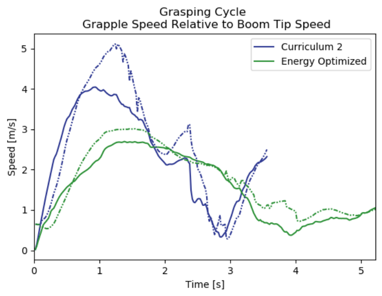
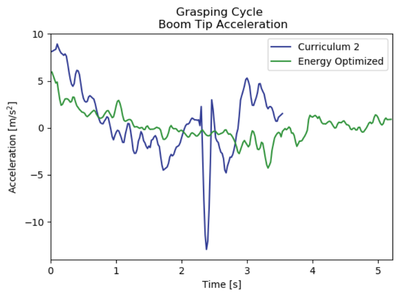


Figure 4.10: *Boom tip speed variation during five different grasping cycles using the Curriculum 2-based model (solid) and the corresponding energy-optimized model (dashed). Each color refers to cycles with a specific log position and orientation.*

Figure 4.12 a) shows the log distribution for successful grasps in the legal target log area for the best Curriculum 2-based model. The log distribution is similar for the models based on Curriculum 1 and 3, and across these models no clear correlation between failed or successful grasping attempts and the target log position is found. Figure 4.12 b) shows the corresponding target log distribution for successful episodes during different stages of the training session for the energy optimized model. Green dots represent the target log position after 40 million training steps, black dots represent the corresponding target log position after 60 million training steps, and blue dots represent the target log position for successful grasps after 80 million training steps. Following the notation of the polar coordinate system introduced in Figure 3.5, we can see that the final distribution in the $r\theta-$plane is similar to that of Figure 4.12 a), and that the energy optimized model initially fails at grasping logs at low-radii positions. As the training progresses, the agent successively learns to master the grasping of logs at lower radii at the same rate as logs positioned at higher radii.



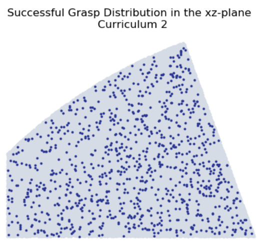
(a)



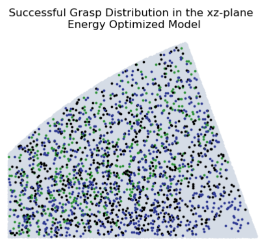
(b)

Figure 4.11: *a) The grapple speed (dashed), defined as the speed of a point at the centre of the grapple claws, and the boom tip speed (solid) during a grasping cycle using both the Curriculum 2-based model (blue) and the corresponding energy-optimized model (green). b) The boom tip acceleration during the grasping cycle selected in a) for the Curriculum 2-based model (blue) and the corresponding energy-optimized model (green).*

Figure 4.13 shows that for the energy optimized model, there is a correlation between the radial position of the target log and the total energy consumption of the crane, which may explain this observation.



(a)



(b)

Figure 4.12: *Distribution of the log position (blue) over the legal target area (grey) in the xz-plane for successful grasps using the best Curriculum 2-based model. The legal target log area is highlighted in grey. b) Corresponding distribution of the log position of successful grasps using the energy optimized model trained for 40 million steps (green), 60 million steps (black) and 80 million steps (blue).*

To investigate the robustness of the models to unexpected environmental changes, the success rate is measured for each model with target logs resting on floors at heights above ground level. This simulates the model robustness to uneven terrains. The robustness of each model to varying target log heights is demonstrated in Figure 4.14.

Moreover, the model robustness is evaluated by analysing the model response to additional crane oscillations. This is done by measuring the impact on the

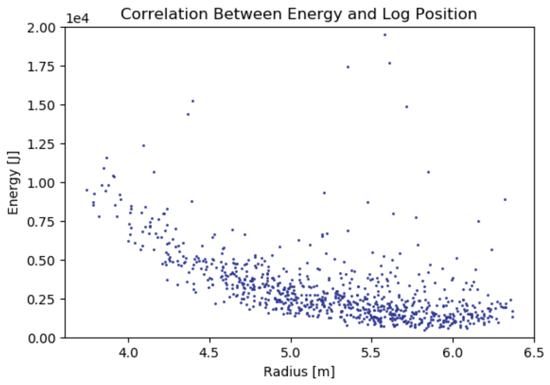


Figure 4.13: *Correlation between the radial target log position in the $r\theta-$plane and the total energy consumption for the energy optimized model.*

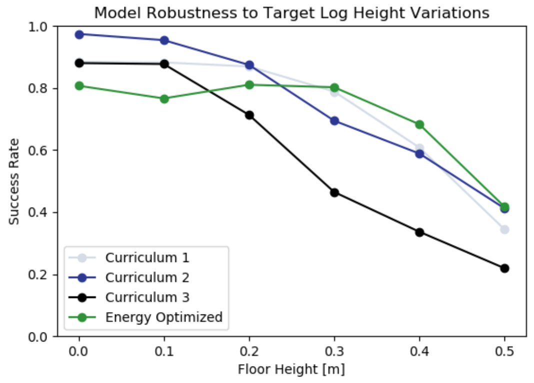
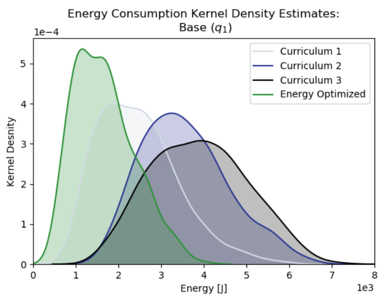


Figure 4.14: *Success rate variation with the target log height above the ground for the best models based on Curriculum 1, 2 and 3 without energy optimization, as well as for the Curriculum 2-based energy optimized model.*

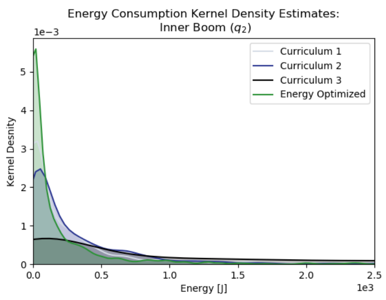
success rate of each model when the crane is exposed to increased oscillations due to the natural dynamical motion of the vehicle during grasping, despite being trained to grasp logs with a crane mounted on a static vehicle. The increased oscillatory crane motion induced by a dynamic vehicle are referred to as *crane oscillations* or *crane vibrations*. The magnitude of these are estimated by the episodic mean and standard deviation of the positional variation of the rigid body on which the base of the crane is mounted. This rigid body is never static, but its motion is negligible under a static vehicle. The mean of these measurements over 1000 episodes is presented in Table 4.1, along with the impact on the success rate of the increased crane vibrations. The positional offset of the rigid body in the static and the dynamic case is -0.77 cm in the $x-$direction, -10.9 cm in the $y-$direction and -3.4 cm in the $z-$direction.

Model	**x [cm]**	**y** [cm]	**z** [cm]	**Success Rate**
Curriculum 1	0.74 (0.81)	5.5 (2.3)	0.05 (0.78)	0.562 [0.884]
Curriculum 2	-0.18 (0.75)	5.4 (2.4)	0.07 (0.82)	0.810 [0.974]
Curriculum 3	0.15 (0.74)	5.6 (2.2)	0.11 (0.73)	0.274 [0.880]
Energy Optimized	0.70 (0.65)	5.4 (2.3)	0.09 (0.59)	0.270 [0.807]

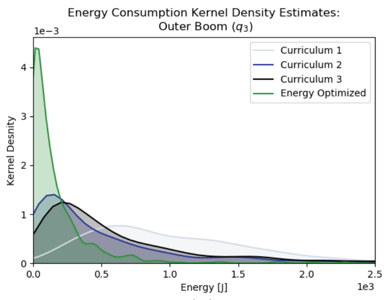
Table 4.1: *Size of crane vibrations in terms of the positional variation of the rigid body on which the base of the crane is mounted. Displayed is the mean episodic positional variation in the $x, y-$ and $z-$direction along with the corresponding mean standard deviation (parenthesis) and the success rate for each of the four models (original success rate in brackets).*



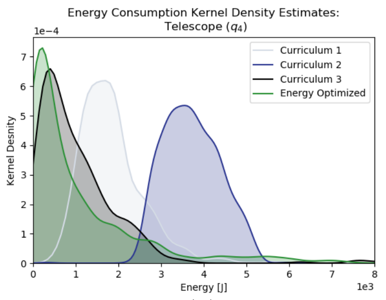
(a)



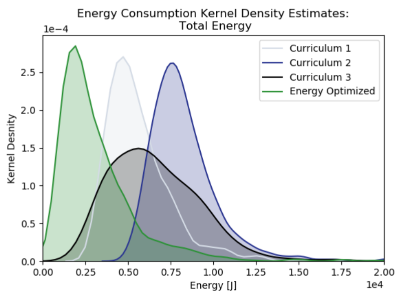
(b)



(c)



(d)



(e)

Figure 4.15: *Distribution of the total energy consumption and the energy consumption for constraints q_1, q_2, q_3 and q_4 using the best models based on Curriculum 1, 2 and 3, as well as the energy optimized Curriculum 2-based model.*

Chapter 5

Discussion

The results presented in Chapter 4 show that the grasping motion of a forestry crane manipulator can be automated in a simulated environment using both the multi-agent and the single-agent reinforcement learning approaches described in Chapter 3. This section aims to discuss and analyse these results.

5.1 Multi-Agent Approach

In the multi-agent approach, the applied reward functions yield stable training processes for both Agent 1 and Agent 2, as displayed in Figure 4.1. Agent 1 reaches a high success rate, with a mean distance of less than 10 cm between the boom tip of the crane and the target position. The true distance is often much smaller than this, as can be seen in Figure 4.2 a). A higher precision is not required in order to accomplish the task at hand. In fact, our results show that an Agent 1 precision of 0.2 m is enough to reach a grasping success rate exceeding 76% for Agent 2, and while no such measurements have been carried out in this work, it is likely unnecessary for a human operator to reach a higher navigation precision than what is reached by Agent 1. Since our implementation prevents transitions between the agents with an Agent 1 precision of less than 0.2 m, it is possible that utilizing the full precision of Agent 1 would generate a higher grasping success rate of Agent 2 with the current setup and reward structure. Likewise, the target position of Agent 1 is fairly arbitrarily chosen, and an optimal subtask-division of the global task in the multi-agent approach is a welcomed subject for future work.

It should be noted that not all training sessions result in a stable learning process and that some agents never learn using the setup and reward structure investigated in this thesis. Similar learning behavior can be observed using the single-agent approach. While no statistics on the success and stability of learning processes over multiple independent training sessions have been investigated, it is an interesting perspective for future work as the corresponding analyses could prove useful in future development of autonomous systems in simulated environments.

The Agent 1 results presented in Figure 4.2 and 4.3 suggest that the final q_4 velocity and the final boom tip position of the crane are oscillating slightly about zero-velocity at the target position, with the boom tip and telescope practically remaining still very close to the target position. As a completely static crane motion at the target position cannot be expected by an agent for which all actuated constraints remain free, these results are promising. As can be seen in Figure 4.3 a), the deviation between the boom tip and the target position at the final time step is slightly larger in the $z-$direction than in the $x-$ and $y-$direction. While the deviation in the $y-$direction is expected to be smaller than the deviation in the $xz-$plane since the crane motion in this direction is dependent on one less degree of freedom, the relative deviation in the $x-$ and $z-$direction may simply result from the stochastic nature of the learning process.

The goal of Agent 1 could only be accomplished by shaping the reward function to prevent the agent from finding illegal solutions. Such illegal behavior include colliding with the load bunk, exploiting the working range of certain joints and inefficiently taking advantage of the redundant kinematics of the crane. While the agent may be able to learn to avoid collisions even without such penalties, this may in theory lead to actual collision-based strategies, which can, of course, not be applied to a physical machine. The same is true for the illegal solution in which the agent comes up with a strategy to remain close to the target position by applying maximum motor force to q_4, pushing the telescope towards either working range limit. Other illegal behavior observed in early control policies are connected to the redundant kinematical crane configuration, expressed through coordinated motion of q_2, q_3 and q_4 in order for the boom tip to remain at the target position. This may not damage the physical machine but can be destructive in other ways, for example in terms of energy efficiency. In our work, the goal of Agent 1 is to navigate to and reach zero-velocity at the target position. In the practical application, the forestry crane manipulator is not required to remain at a target position for any amount of time before the transition to Agent 2 can occur, but the results are interesting nonetheless, and the conclusions drawn can be applicable to other systems exhibiting similar redundancy.

The learning response to the grapple rotation goal of Agent 1 shows highly promising results, with the mean deviation between the rotation of the grapple and the rotation of the target log amounting to a mere 0.73 degrees. The high rotational precision is independent on the target log rotation, as can be seen in Figure 4.4. These results show that scaling down the reward to reach a sub-goal can be efficient, and that adding positive reward factors is not always necessary. In itself, the sub-goal of rotating the grapple correctly is not trivial, as it requires coordination between q_1 and q_5, but it is not as complex as coordinating all six actuated joints correctly to perform the complete grasping task.

Early results showed that solving the complete grasping task with reward structures similar to those investigated here was not possible without either dividing the main task into subtasks and learning one policy for each subtask, or using a curriculum to ease learning of a single policy to complete the grasping task in its entirety. As presented in Chapter 4, the success rate of Agent 2 is high but not perfect, and significantly lower than the success rates obtained using the single-agent approach. This is not necessarily an approach-specific conclusion, and the comparatively low success rate of Agent 2 can likely rather be explained by the solution favoured by the control policy learned by the agent. Observations suggest that the solution of Agent 2 relies heavily on the motion of the crane's inner boom, controlled by q_2. Indeed, failed grasping attempts often result from collisions between the inner boom and the load bunk. As Figure 4.1 suggests, there may be room for improvements in the policy learned by Agent 2, as it cannot be ruled out that a longer training time could increase the performance further. Due to time-limitations, training Agent 2 more excessively was not possible, but it is not implausible that the agent may be able to master the grasping task with greater precision and adjust its solution to avoid collisions and increase coordination if allowed more time to learn. Moreover, the initial position of Agent 2 is dependent on the final control policy learned by Agent 1. Thus, it is not unlikely that other solutions to the navigation task, or other transitions between the agents, may ease learning and improve the performance of Agent 2 in the multi-agent approach.

A second observation regarding the solution of Agent 2 is that it only attempts to grasp the log *once* each episode. This is contrary to the resulting behavior of several control policies in the single-agent approach. According to our results, these agents often initialize a second attempt at grasping the target log after failing the initial grasping attempt, though this is not the case for all models and is likely dependent on the agent's state after the initial grasping attempt along with the particular experience accumulated during training. The major difference between the reward structures is that the agent in the single-agent approach is given a very high, discrete reward for succeeding to grasp the target log, whereas Agent 2 can accumulate rewards by remaining in close proximity to the target log even after failing to grasp it. Of course, the reward function theoretically incentivizes Agent 2 to re-grasp the log, but this is currently not reflected in the resulting behavior of the learned control policy. Since Agent 2 is not free to move q_1, a plausible explanation is that it is impossible to complete a successful second attempt if the initial attempt leads to a positional relocation of the target log. Locking several degrees of freedom reduces the number of parameters to learn during the training process, but simultaneously limits the flexibility of the agent to respond to unforeseen circumstances.

In light of this, the single-agent approach may be better suited for the unstructured forest environment of the current context, since the learning of flexible behavior requiring cooperation between multiple agents further complicates the learning process and our results show that success in the grasping task is possible to achieve with a single agent. However, dividing the entire forwarding task into subtasks is likely inevitable, and the results obtained using the multi-agent approach show that transition between multiple agents is possible in this context, but that it is important to investigate how the subtask-division is best implemented in practice.

5.2 Single-Agent Approach

The results obtained by the single-agent approach show that it is possible to achieve high success rates for agents trained to perform the entire grasping task using a single control policy. The best single-agent model in terms of success rate is the non-energy optimized model trained on Curriculum 2, which achieves the nearly perfect success rate of 97.4%. The best models obtained using each curriculum exceeds 88%, while the energy optimized model reaches a success rate of just above 80%.

5.2.1 Learning Process

Figure 4.6 and 4.7 illustrate that the learning process is stable in all cases, but as discussed, occurrences of training sessions in which the agents fail to learn altogether, or in which learning is slow or unstable, have also been observed. Moreover, no extensive hyper-parameter search has been performed, and a set of parameters generating higher performance may be possible to obtain in future research.

The learning process of the lessons common to all curricula, presented in Figure 4.5 for three independent training sessions, shows that there is not much deviation between the learning progress of each training session. Thus, it is likely that training agents from the first lesson yields differences specific to the current curriculum. Figure 4.8 shows that deviations between training sessions are larger for models trained on the full curriculum. In this case, the differences in the average episodic return obtained are larger between different models trained on the same curriculum than between the best models obtained using the various curricula. Though the final average episodic return obtained during the training session directly relates to the success rate of the final model, this is, however, not directly correlated with the behavior resulting from the learned control policy. In general, the solutions of Curriculum 2-based models show common traits differing from the solutions obtained by agents trained on Curriculum 1 and 3, and also from the solution obtained by the energy optimized model.

It is interesting to note that the best model overall, i.e. the best model obtained using Curriculum 2, reaches the final lesson faster than agents trained using Curriculum 1 and 3, which can be seen in Figure 4.6 b). Though Figure 4.8 b) shows that the time required for an agent to reach the final lesson varies between agents within a curriculum, it can be concluded based on our observations that agents trained on Curriculum 2 reaches the final lesson faster than agents trained on the remaining curricula. This may partly explain the high success rate obtained by the best Curriculum 2-based model, since this agent has spent more time learning the specific task on which it is evaluated. The success rates obtained by the best agents trained on Curriculum 1 and 3 are similar to one another, reaching slightly above 88%, and Figure 4.6 b) shows that these agents have spent a similar amount of time on ground level. This is not a general rule, however, since Figure 4.8 suggests a lower success rate for the other models trained on Curriculum 2, despite reaching the final lesson in a smaller amount of training time.

5.2.2 Solution

Due to the redundant kinematics of the crane configuration, there exists an infinite number of different solutions in order for the agent to successfully grasp a single log at a fixed position. Figure 4.9 illustrates some of the differences between the solutions obtained by the best agents trained on the different curricula in terms of the joint angle distributions of their respective end-configuration, where the end-configuration is defined as the configuration of the crane at the moment of grasping initiation.

It is evident that the best Curriculum 2-based model relies excessively on the use of q_4, unlike the other models. Thus, it reaches far out with the telescope of the crane, nearing the maximum working range independent of the log position. In contrast, the model based on Curriculum 1 has found a solution in which the telescope is contracting in the general case. As a consequence, it compensates by higher link activation of q_2, which moves the crane's inner boom. The trade-off for the agent trained on Curriculum 2 is the minimal use of q_2 compared to the other agents, which reduces the risk of collisions between the inner boom and the load bunk.

The agent trained on Curriculum 3 has found a solution in which it alternates between the usage of q_2, q_3 and q_4 using a larger portion of the working range for each joint. These observations show possible effects of using different curricula, but further research should be carried out in order to conclude whether these differences are statistically dependent on the curriculum or arise due to the stochastic nature of the learning algorithms. Comparisons to the solutions obtained by the energy optimized model is perhaps the most interesting. This is discussed in the following subsection.

5.2.3 Energy Optimization

As we have seen, the total energy consumption decreases by almost 60% in the energy optimized model compared to the non-energy optimized model based on the corresponding curriculum, and over 40% compared to the agents trained on Curriculum 1 and 3. Analysing the end-configuration distributions presented in Figure 4.9, it is evident that, in general, the energy optimized model relies far more on q_2 and far less on q_3 than the other models. This is reasonable, as the consumed energy can be minimized by taking advantage of gravity to the largest extent possible. The energy optimized model also restricts usage of q_4. While it covers the necessary part of the working range, the mean angle remains close to the initial condition. This is also expected, as the energy consumed by

q_4 is directly correlated with the high energy consumption of the Curriculum 2-based model, which relies heavily on the use of q_4. By adopting this strategy, the energy optimized model relies on the most energy efficient joint, q_2, while minimizing the energy consumed by q_3 and q_4 by using these links to a smaller extent. Moving the inner boom with gravity reduces the energy consumption the most, since q_2 carries the largest total crane weight and can take advantage of gravity independent on the activation of the other links. Both the outer boom (q_3) and the telescope (q_4) can, however, also move in ways that take advantage of the gravitational effects, and with certain coordination q_4 can also utilize the centripetal force of the rotating motion of the crane due to activation of q_1 to reduce the total energy consumption. Observations show that this is indeed the case for the energy optimized model. For example, it is explicitly observed that the learned control policy often results in the agent adjusting the telescope position upon coordination of the remaining joints such that this can be done in the direction of gravity at the end of the episode.

In general, the energy optimized model also uses q_1 slightly less than the other models, despite the fact that the motion and energy consumption of q_1 is not affected by the redundant kinematics or gravitational pull on the crane. Instead, its motion is largely restricted to the position of the target log, limiting the available energy optimization strategies for q_1. However, due to the large inertia of the crane, q_1 activation has a large effect on the total energy consumption and the reduction in q_1 activation may result from a strategy aiming to grasp the log from the minimal q_1 angle possible. The difference in the energy consumed by q_1 compared to the other models is indeed significant, as can be seen in Figure 4.15 a).

Of course, the energy consumption is not necessarily directly correlated with the end-configuration, as the energy consumption is dependent on for example the acceleration profile throughout the episode. This can be observed in the overall smoother motion enforced by the energy optimized model compared to the non-energy optimized models, and the smaller magnitude of the grapple oscillations that occur before grasping. As shown in Figure 4.11 a), the deviation between the speed profiles of the grapple and the boom tip during a randomly selected grasping cycle is smaller for the energy optimized model compared to the corresponding non-energy optimized model. This suggests a smoother acceleration profile resulting in reduced grapple oscillations, which is confirmed by the acceleration profiles of the two models during the grasping cycle, as presented in Figure 4.11 b).

A consequence of the calmer behavior observed in the energy optimized model is that the grasping cycle time exceeds the cycle time required by the corresponding non-energy optimized model by 30%. Though the absolute time differences are small, there is a positive correlation between the mean time cost and energy efficiency of the models analysed here, suggesting an overall smoother crane motion in the energy optimized model. This is confirmed by Figure 4.10, showing the boom tip speed profiles of five randomly selected grasping cycles. The boom tip speed and acceleration is generally lower in magnitude using the energy-optimized model. We emphasize that the energy consumption is only measured up until the point of grasping initiation, i.e. the grasping of the logs and the energy consumption of q_5 and q_6 are not included.

As previously mentioned, it is observed that agents trained using the single-agent approach often attempts at re-grasping the logs if the initial grasping attempts fail. Thus, outliers in terms of energy consumption may correspond to agents succeeding to secure the log in its grapple during a re-grasping attempt, as these have been recorded as successful grasps. It is also worth noting that the energy optimized strategy for the sub-task investigated in this thesis is not necessarily the energy optimal strategy for a grasping task including transporting the logs back to the load bunk, in which case the crane is required to also work against gravity and another grasping end-configuration may be preferable. Indeed, human operators are instructed to rely on the use of the telescope to increase precision as well as energy and time efficiency. Nonetheless, the results show that invoking an energy optimization goal in the reward function by a simple scale factor can yield significant changes to the final control policy and smooth the acceleration profile of an agent of this complexity. A more extensive analysis of the differences in the obtained behaviors between the energy-optimized model and the non-energy optimized model is left for future research.

5.2.4 Grapple Rotation

No clear correlation between the rotation of the grapple and the rotation of the target log can be found in most single-agent models. The reason for this is simply that the forestry crane manipulator is able to secure a single log in its grapple almost independent on the grapple rotation due to the small log radius relative to the maximum grapple capacity. Thus, there is no incentive for the agent to learn to rotate the grapple with high precision.

The Curriculum 1-based model does exhibit a lower success rate for target logs with large rotations such that the grapple needs to rotate in the direction opposite to the rotational direction of q_1. The best control policy results in a solution in which the crane uses the claws of the grapple to instead rotate the *logs* with these rotations on the ground before grasping. Since the terrain is modelled by a flat surface without obstacles, these kinds of strategies are likely not as successful in reality. This highlights the importance of addressing innovative solutions that the agent may find due to simplifications in the simulation model and environment. In this case, incentive can be given to the agent to prohibit accumulation of high rewards without the agent learning to coordinate the grapple rotation with the rotation of the log. Though a more realistic simulation model of the ground is preferable, adding an energy optimization goal including the energy consumption of q_5 and q_6 to the reward function should lead to the agent learning a control policy relying more on the rotation of the grapple, thus reducing the log-rotating behavior observed in some models.

5.2.5 Failed Grasping Attempts

For the best models based on Curriculum 1, 2 and 3, no correlation between failed grasping attempts and the log position is identified. However, Figure 4.12 shows that there is a radial dependency between failed and successful grasps throughout the training session for the energy optimized model, where the agent successively masters the grasping of logs at small radii. This may directly relate to the energy optimized solution in which q_2 is heavily used. Figure 4.13 shows that in the final energy optimized model, there is a correlation between high energy consumption and a small radial position of the target log. If the use of q_2 in the energy optimized solution makes it difficult to reach logs closer to the forwarder in the radial direction, the agent has to adjust its behavior to a less energy efficient strategy in order to succeed at grasping these logs.

Except for the aforementioned observations, no specific behavior behind or common reasons for failed grasping attempts are identified. In the case of Agent 1, it is observed that for a small fraction of episodes, the model exhibits a strange behavior in which it exceeds the working range of q_1. Similar behavior can be seen in 4% of episodes in which the energy optimized model fails to grasp the target log. In these episodes, the crane simply accelerates to maximum q_1 speed until the working range of q_1 is reached, without trying to grasp the target log. No reason for this behavior can be identified, but it is important to mention since removing this kind of behavior before applying the model to a physical machine is vital, whether it be related to the learned policy or the simulation model.

5.2.6 Robustness & Generalization

As we can see in Figure 4.14, all models remain robust to changes in target log heights of 0.1 m. This is enough to ensure a high success rate under slightly more uneven terrains than those the agent is exposed to during training. The model most robust to varying target heights is the energy optimized model, for which the success rate remains approximately constants up to relative target log heights of 0.3 m above the ground. This must be viewed as a consequence of the learned control policy, since the energy optimized model has trained far longer than the remaining models on the ground floor level. It is otherwise reasonable that the curriculum used may ease adaptation to varying target log heights, since the agents are trained to perform grasping under these circumstances in previous lessons. However, no such tendency can be observed, likely due to the low reward threshold required to move to the succeeding lesson of the curriculum, and the comparatively long training time spent on the final lesson. This motivates the use of this measurement in the evaluation of robustness.

Since transitions between the lessons are relatively smooth in all training sessions, there is no indication that training an agent to adjust to varying target log heights, rotation and inclination would pose an unattainable challenge. This is an important indication, since the simplified problem setting that has been the focus of this thesis is too trivial to provide efficient solutions in the real-world forwarding task, where uneven terrains are constantly affecting the work of the operator.

The decreased success rate of models exposed to increased crane oscillations are perhaps less encouraging across models, but nonetheless show that it is possible for well-trained models to generalize to a more dynamic environment than the one present during training. As Table 4.1 shows, the best Curriculum 2-based model is the most robust to increased crane oscillations, with a success rate of 81.0% with a dynamic vehicle. This corresponds to a decrease of 16.4 percentage points compared to the static case. The decrease is expected, since the models have learnt to utilize the expected grapple oscillations in the moment of grasping, and these oscillations are affected by the changed dynamics. The only other model for which the success rate remains higher than 50% is the one based on Curriculum 1 without energy optimization. Figure 4.15 suggests that the solutions of these models use q_4 to a larger extent than remaining models, but otherwise see large differences in link activation during the course of an episode.

The magnitude of the differences between the agents' abilities to adapt to increased environmental dynamics is not surprising, as the resulting robustness likely depends on the acceleration profile and the particular coordination between the joints favoured by the control policy learned by each agent. Observations suggest that the dominating effect of the added dynamics is the increased oscillating motion of the grapple at the moment of grasping. In the case of the models based on Curriculum 1 and 2, this sometimes causes the grapple to close before it has secured the target log in its claws, whereas the agent trained on Curriculum 3, which has learnt to move closer to the target log before initializing the grasping attempt, often pushes the target log away with its grapple before having a chance to grasp it. Thus, the effects of the increased crane oscillations are similar in all three cases and relate to the increased grapple oscillations, but the decrease in success rate is larger in the latter case due to the differences in the control policy found by the agents during training.

Another reason for the decreased success rate with a dynamic vehicle is that the agent experiences difficulties in estimating the crane position relative to the load bunk, resulting in an increased collision risk for agent's relying on solutions where the crane moves closer to the load bunk. Indeed, in several episodes displaying failed grasping attempts due to increased oscillations, this is caused by collisions between the crane and the load bunk. This is particularly the case for the energy optimized model, for which the success rate decreases from 80.7% with a static vehicle to 27.0% with a dynamic vehicle. The solution obtained by this model largely relies on the use of q_2, contrary to the excessive use of q_4 in the non-energy optimized model trained on the same curriculum; a difference that may account for the comparatively high robustness to increased crane oscillations in the non-energy optimized model. Table 4.1 shows that the system oscillations are largest in the $y-$direction, and that the same is true for the offset between a fixed position on the static and dynamic vehicle. In addition, the added dynamical flexibility leads to an increased bending motion of the crane due to the bending moment at its base, causing the effective initial condition of the grapple to move closer to the load bunk. This is more likely to affect the success rate of an agent for which the solution is more dependent on the use of the inner boom (q_2) or solutions which otherwise lower the crane and/or grapple early in the episode since these motions reduce the margins between the load bunk and the crane.

Based on these observations, the control policy robustness to these kinds of oscillations may increase if the definition of illegal behavior extends to a safe zone around the vehicle. This can be incorporated into the reward function during training to prevent solutions where the margins between the crane and the vehicle are very small. Such solutions may not only increase the success rate under increased oscillations, but are perhaps even more important to prevent damage due to dynamical differences occurring during transfer from simulation to reality. Indeed, even if the simulation physics is a perfect representation of reality, the model needs to be robust to the inevitable environmental changes that, in practice, cannot be perfectly modelled for very unstructured environments. Moreover, the simulation model parameters are difficult to specify to high precision, especially since the parameters of a physical machine are prone to uncertainties, temporal variations and variations due to weather and temperature conditions.

Despite the difficulties of several models to handle increased crane oscillations, the fact that the best model remains largely robust to increased oscillations is auspicious, and suggests that adapting to differences between the simulated and real world is possible depending on the learned control policy. This is encouraging in terms of prospects for simulation-to-reality transfer in the development of simulated autonomous systems. Moreover, the induced grapple oscillations are smaller for the energy optimized model compared to the non-energy optimized models, likely due to an energy optimized acceleration profile, which suggests that the robustness to changes in the dynamics can be expected to increase significantly in the energy optimized model if the added collision risk can be avoided. It should also be emphasized that neither of these agents are trained under oscillations of this magnitude, and that they have all successfully learnt to compensate for the crane oscillations present under a static vehicle. Thus, given the success rates obtained and the observed behavior of models displaying low success rates under a dynamic vehicle, the agents are likely able to adapt their solutions when trained on similarly dynamic environments.

5.3 Concluding Thoughts

Overall, the agents trained using the single-agent approach reaches a higher grasping success rate than the agents trained using the multi-agent approach. The best model reaches a success rate of 97.4% in the single-agent approach, compared to 76.3% for Agent 2 using the multi-agent approach. Though the success rates of the two approaches are not intended to be directly compared, both approaches show encouraging results. To perform the complete forwarding task, including transporting the log back to the log bunk, a combination of the two approaches is likely necessary in practice. This can be realized through the use of curriculum learning to achieve complex subtasks, combined with efficient transitions between agents specialized in each specific subtask. Since the overall forwarding task requires significant strategic planning and consists of multiple different subtasks, a hierarchy of sub-policies may be the best approach.

Our observations suggest that the control policies found by agents trained using the single-agent approach result in crane behaviors that are, in essence, similar to the ones achieved by manual operators, and that the crane motions are generally smooth enough not to damage a physical machine. An especially promising observation is that the agents have learnt to utilize the grapple dynamics during grasping, which is typical for well-trained and experienced operators. One advantage to applying reinforcement learning in this context is that simultaneous coordination of multiple joints is difficult for a human operator, whereas this is the default for a reinforcement learning agent that receives action signals for each joint at every time step. Thus, the optimal solution from the perspective of an agent may not be the optimal solution from the perspective of an operator. Additional experiments can be carried out to compare link activation between the solutions obtained by the agents and the link activation resulting from manual operation.

Of course, there are many steps remaining before such approaches can be applied to physical machines. One such step is related to the observation space of the agent. While a completely autonomous system must be able to detect target logs by for example visual sensory observations of the environment, our setup assumes an external perception system which is not developed at this stage. These questions are left for future research, but our results provide an encouraging foundation showing that deep reinforcement learning can be used to successfully train agents to perform simple forms of log grasping in simulated environments. See Section 3.7 for a brief discussion on safety aspects of the current and future work.

Chapter 6

Conclusions & Recommendations

The aim of the work presented in this thesis has been to automate the grasping motion of a hydraulic-actuated forestry crane manipulator in a simulated environment using deep reinforcement learning methods, in particular the state-of-the-art reinforcement learning algorithm Proximal Policy Optimization. Two strategies have been investigated; a multi-agent approach dividing the main task into a navigation task and a grasping task, and a single-agent approach using curriculum learning to enable the learning of a single control policy to perform the entire task. Our work is limited to automation of the single log grasping motion using a crane mounted on a static vehicle. The agent observes the state of its six actuated joints together with the spatial position and rotation of the target log, and is controlled by continuous action signals given to the velocity-controlled motor of each actuated joint.

The results are promising using both the multi-agent and the single-agent approach. According to our observations, the joint activation profiles of our models yield a grasping behavior similar to that produced by a human operator. Further experiments can investigate this in detail. The best model obtained using the multi-agent approach reaches a grasping success rate of 76.3%. The corresponding success rate for the best model obtained using the single-agent approach is 97.4%. Our results show that energy optimization incentivized in the reward function reduces the mean energy consumption up until the point of grasping by 58.4%. The joint activation profile of this model takes advantage of gravity to a large extent, while also smoothing the overall acceleration profile and reducing grapple oscillations.

Three different curricula are investigated, where the distance between the initial position of the grapple and the target log increases discretely over a curriculum-specific number of lessons. The conclusion is that this affects the policy of the agent and thus the joint activation profile over the course of an episode, though further research is needed to conclude whether or not this is strictly dependent on the curriculum. The best model obtained using each curriculum reaches success rates exceeding 88%. All models are robust to target logs placed 0.1 m above the log position on the ground, with the energy optimized model managing to grasp target logs placed up to 0.3 m above the ground position without significant performance loss. Moreover, exposure to increased crane oscillations induced by a dynamic vehicle reduces the success rate of the best model from 97.4% to 81.0%. These results are encouraging, since they imply potential for success in uneven terrains and increasingly dynamic environments even before exposing the agent to such environments during training.

This thesis has provided the first implementation of deep reinforcement learning control of a forestry crane manipulator in a simulated environment. Our results show that both the multi-agent and the single-agent approach can reach high success rates in a simulated environment using this method. While the performance is higher using a single policy, this is likely not approach-specific. Overall, our results show potential for end-to-end automation of forestry cranes, including robustness encouraging for transfer from simulation to reality, and provide a starting point for future research. To automate the entire task of forest forwarding, a combination of multi-agent and single-agent approaches is likely necessary. Next steps include investigating the potential for reinforcement learning to be applied to more complex grasping scenarios in environments mirroring the dynamics of the real forest environments, as well as to other subtasks, such as releasing the log in the log bunk, grasping logs from a moving vehicle or grasping multiple logs simultaneously. Other interesting investigations left for future research include analysing the model sensitivity to physical parameters as well as using for example off-policy methods, physics informed networks or differentiable physics to augment training.

While our models require an external perception system providing the position and orientation of the target log, it is also important to investigate the possibility for reinforcement learning automation based on visual observations of the environment, which is necessary to obtain a completely autonomous system. Such systems could also be designed to address safety issues to prevent accidents and physical damage upon real-world model deployment. Further research may also investigate the differences induced by the use of different curricula in the grasping task, as it is vital to understand its impact on the joint activation profile in the final policy. This is important to any subtask in the forwarding process that requires the use of curriculum learning.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

AI HLEG (2019). Ethics guidelines for trustworthy AI. Report, European Commission, Brussels.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *Advances in Neural Information Processing Systems 30*, pages 5048–5058.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press, Princeton, New Yersey.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Bertsekas, D. P. and Tsitsiklis, J. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. In *Proceedings 2000 ICRA. Millennium Conference. IEEE Interna-*

tional Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), volume 1, pages 348–353.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, (11):1–94.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *International Conference on Learning Representations.*

Caldera, S., Rassau, A., and Chai, D. (2018). Review of deep learning methods in robotic grasp detection. *Multimodal Technologies Interact*, 2(3):57.

Dai, B., Shaw, A., Li, L., Xiao, L., He, N., Liu, Z., Chen, J., and Song, L. (2018). Sbeed: Convergent reinforcement learning with nonlinear function approximation. *35th International Conference on Machine Learning, ICML 2018*, 3:1809–1818.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Fang, M., Zhou, T., Du, Y., Han, L., , and Zhang, Z. (2019). Curriculum-guided hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 12602–12613.

FAO and UNEP (2020). *The State of the World's Forests 2020. Forests, biodiversity and people.* The State of the World's Forests (SOFO). Author.

Fernández, F., García, J., and Veloso, M. (2010). Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871.

Fodor, S. (2017). *Towards semi-automation of forestry cranes: automated trajectory planning and active vibration damping.* PhD thesis, Umeå University.

Frans, K., Ho, J., Chen, X., and Abbeel, P. (2018). Meta learning shared hierarchies. *International Conference on Learning Representations.*

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press. http://www.deeplearningbook.org.

Gosavi, A. (2004). Reinforcement learning for long-run average cost. *European Journal of Operational Research*, (155):654–674.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, (5):1471–1530.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2017). Q-prop: Sample-efficient policy gradient with an off-policy critic. *Proceedings of the International Conference on Learning Representations (ICLR).*

Hansson, A. and Servin, M. (2010). Semi-autonomous shared control of large-scale manipulator arms. *Control Engineering Practice*, 18(9):1069–1079.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L., and Dean, T. (1998). Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229.

Hengst, B. (2008). Partial order hierarchical reinforcement learning. In *Australasian conference on artificial intelligence*, pages 138–149, Auckland, New Zealand.

Hengst, B. (2011). Hierarchical reinforcement learning. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*, pages 495–502. Springer, Boston, MA.

Hera, P. L., Mettin, U., Manchester, I. R., and Shiriaev, A. (2008). Identification and control of a hydraulic forestry crane. *IFAC Proceedings Volumes*, 41(2):2306–2311.

Hera, P. L. and Morales, D. O. (2019). What do we observe when we equip a forestry crane with motion sensors. *Croatian Journal of Forest Engineering*, 40(2):259–280.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, Massachusetts.

Imagawa, T., Hiraoka, T., and Tsuruoka, Y. (2019). Optimistic proximal policy optimization. *ArXiv*, abs/1906.11075.

Jonschkowski, R. and Brock, O. (2014). State representation learning in robotics: Using prior knowledge about physical interaction. In *Robotics: Science and Systems.*

Joshi, S., Kumra, S., and Sahin, F. (2020). Robotic grasping using deep reinforcement learning. *ArXiv*, abs/2007.04499.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2020). Unity: A general platform for intelligent agents. *ArXiv*, abs/1809.02627v2.

Kaelbling, L. P., Littman, M. L., and Cassandra., A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, (4):237–285.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning*, 2:267–274.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *ArXiv*, abs/1806.10293.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2019). Recurrent experience replay in distributed reinforcement learning. *Proceedings of the 7th International Conference on Learning Representations.*

Kim, S., Asadi, K., Littman, M., and Konidaris, G. (2019). Deepmellow: removing the need for a target network in deep q-learning. *Proceedings of the 28th International Joint Conference on Artificial Intelligence.*

Kumra, S. and Kanan, C. (2017). Robotic grasp detection using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 769–776.

Lacoursière, C. (2007). Regularized, stabilized, variational methods for multibodies. *Proc. 48th Scandinavian Conf. Simulation and Modeling (SIMS '07)*, pages 40–48.

Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. *International Journal of Robotics Research*, 34(4–5):705–724.

Lesort, T., Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural networks : the official journal of the International Neural Network Society*, 108:379–392.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1–40.

Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2017). Learning multi-level hierarchies with hindsight. *ArXiv*, abs/1712.00948.

Li, Y. (2018). Deep reinforcement learning. *ArXiv*, abs/1810.06339.

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. *Proceedings of the National Conference on Artificial Intelligence*, pages 394–402.

Mettin, U., Westerberg, S., Shiriaev, A. S., and Hera, P. X. L. (2009). Analysis of human-operated motions and trajectory replanning for kinematically redundant manipulators. *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 795–800.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning.*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *Computing Research Repository*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Antonoglou, A. S. I., King, H., Kumaran, D., Wierstra, D., Legg1, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, (518):529–533.

Nachum, O., Ahn, M. J., Ponte, H., Gu, S., and Kumar, V. (2019a). Multi-agent manipulation via locomotion using hierarchical sim2real. *ArXiv*, abs/1908.05224.

Nachum, O., Gu, S., Lee, H., and Levine, S. (2019b). Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations.*

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. (2019c). Why does hierarchy (sometimes) work so well in reinforcement learning? *ArXiv*, abs/1909.10618.

Narvekar, S., Peng, B., Leonetti, M., Jivko Sinapov, M. E. T., and Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *ArXiv*, abs/2003.04960.

Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, Singapore.

Narvekar, S., Sinapov, J., and Stone, P. (2017). Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI).*

Ng, A. Y., Harada, D., and Russel, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Proceedings of the 34th International Conference on Machine Learning*, 99:278–287.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. (2017). The uncertainty bellman equation and exploration. *ArXiv*, abs/1709.05380v4.

Oudeyer, P.-Y. and Kaplan, F. (2007). What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1(6).

Park, Y., Shiriaev, A., Westerberg, S., and Lee, S. (2011). 3d log recognition and pose estimation for robotic forestry machine. In *2011 IEEE International Conference on Robotics and Automation*, pages 5323–5328.

Parr, R. and Russell, S. J. (1997). Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 conference on Advances in neural information processing systems*, pages 1043–1049.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *Proceedings of the 34th International Conference on Machine Learning*, 70:2778–2787.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3406–3413.

Puterman, M. L. (1994). *Markov Decision Processes - Discrete Stochastic Dynamic Programming.* John Wiley Sons, Inc., New York.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted markov decision processes. *Management Science*, (24):1127–1137.

Quillen, D., Jang, E., Nachum, O., Finn, C., Ibarz, J., and Levine, S. (2018). Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *ArXiv*, abs/1709.10087.

Royal Swedish Academy of Agriculture and Forestry (2015). *Forests and forestry in Sweden.* https://www.skogsstyrelsen.se/globalassets/in-english/forests-and-forestry-in-sweden$_2$015.*pdf*(accessed May 30, 2020).

Sanner, S. and Boutilier, C. (2009). Practical solution techniques for first-order mdps. *Artificial Intelligence*, (173):748–788.

Santos, M. S. and Rust, J. (2004). Convergence properties of policy iteration. *Society for Industrial and Applied Mathematics*, 42(6):2094–2115.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. *Computing Research Repository*, abs/1511.05952.

Scherrer, B., Gabillion, V., Ghavamzadeh, M., and Geist, M. (2012). Approximate modified policy iteration. *Proceedings of the 29th International Conference on Machine Learning.*

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *Proceedings of the International Conference on Learning Representations (ICLR).*

Schulman, J., Sergey Levine, P. A., Jordan, M., , and Moritz, P. (2015). Trust region policy optimization. *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *Computing Research Repository*, abs/1707.06347.

Servin, M. and Brandl, M. (2018). Physics-based virtual environments for autonomous earthmoving and mining machinery. In *Commercial Vehicle Technology 2018. Proceedings.*, pages 493–504. Springer, Vieweg, Wiesbaden.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosofical Magazine*, 41(314):256–275.

Shu, T., Xiong, C., and Socher, R. (2018). Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *ArXiv*, abs/1712.07294.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, (39):287–308.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, (3):9–44.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 1057–1063.

Sutton, R. S., Precup, D., and Singh, S. (1999b). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211.

Taylor, M. E. and Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685.

Taylor, M. E., Stone, P., and Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, (8):2125–2167.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1553–1561.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.

Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Conference on Neural Information Processing Systems*, pages 10376–10386.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 16:2094–2100.

Vesterlund, F. and Servin, M. (2020). Xt28 simulation model. *GitLab repository at Umeå University.* https://git.cs.umu.se/umit/xt28 (accessed May 14, 2020).

Viereck, U., Pas, A. T., Saenko, K., and Platt, R. (2017). Learning a visuomotor controller for real world robotic grasping using easily simulated depth images. *ArXiv*, abs/1706.04652.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learnin.*

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, (8):279–292.

Westerberg, S. (2014). *Semi-Automating Forestry Machines Motion Planning: System Integration, and Human-Machine Interaction.* PhD thesis, Umeå University.

Wiering, M. and van Otterlo, M. (2012). *Reinforcement Learning: Sate-of-the-Art (Adaptation, Learning, and Optimization (12)).* Springer Science & Business Media.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256.

Williams, R. J. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268.

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. *Computing Research Repository*, abs/1803.07246.

Zahavy, T., Baram, N., and Mannor, S. (2016). Graying the black box: Understanding dqns. In *Proceedings of the 33rd international conference on machine learning*, volume 48, pages 1899–1908.

Zou, H., Ren, T., Yan, D., Su, H., and Zhu, J. (2019). Reward shaping via meta-learning. *Computing Research Repository*, abs/1901.09330.