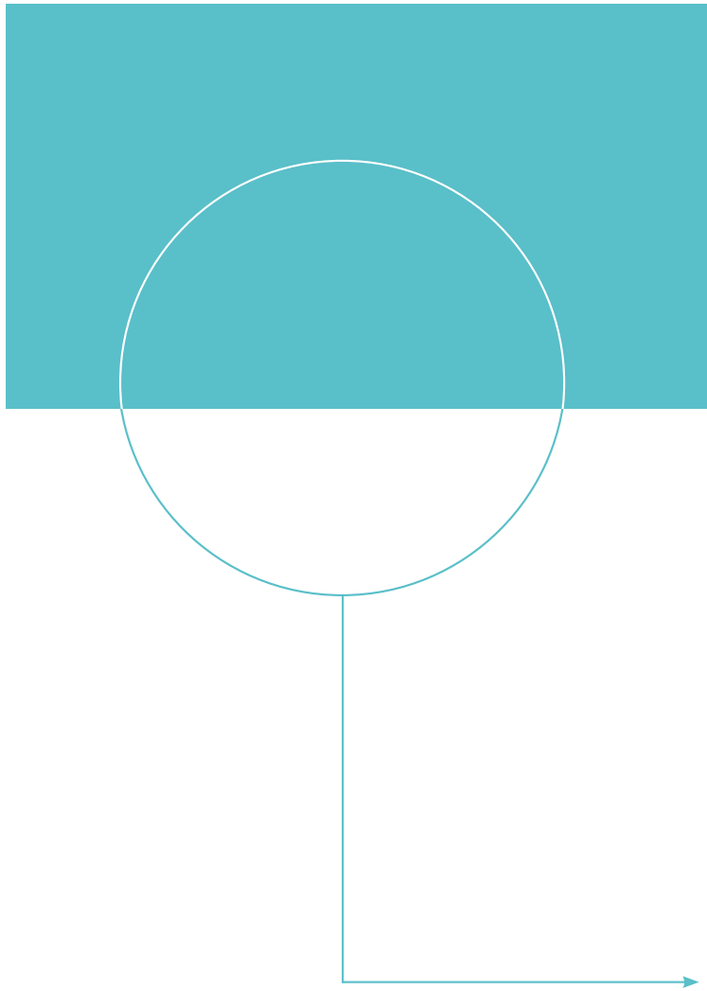


Master's thesis

2022

Sigmund Hennum Høeg

**NTNU**
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Sigmund Hennum Høeg

# Learning to grasp

## A study of learning-based methods for robotic grasping

June 2022

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

# Learning to grasp

A study of learning-based methods for robotic grasping

**Sigmund Hennum Høeg**

Engineering and ICT
Submission date: June 2022
Supervisor: Lars Tingelstad
Co-supervisor: Eirik Bjørndal Njåstad

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

# Learning to grasp

**A study of learning-based methods for robotic grasping**

Sigmund Hennum Høeg

2022-06-10

# Abstract

Robots today are primarily used for tasks that are repetitive and predictable. However, many tasks that create value for our society include a high degree of randomness, for example, picking up objects. Because of the immense diversity of different objects, robots are traditionally restricted to picking in scenarios where the object's geometry is known and no disturbances occur.

Considering the general problem of real-world manipulation tasks, model-free reinforcement learning appears to provide a suitable family of algorithms, as they make no assumptions about of dynamics of the system. Instead, they find the action necessary to solve the task through experience. This thesis will study these methods, focusing on their application to robotic grasping, evaluating their performance and limitations when picking previously unseen objects. A series of experiments are performed in a simulated environment, where two popular model-free reinforcement learning algorithms, Soft Actor-Critic and Proximal Policy Optimization are tasked with picking up a cube with a robotic manipulator.

It is shown that model-free reinforcement learning algorithms provide a framework that can be applied to various manipulation problems. The results show that such algorithms can solve the picking task in the simulator without task-specific refinement. The thesis also highlights significant challenges that hinder the algorithms from solving general manipulation problems in practice, most importantly the need for large quantities of data. Finally, potential solutions to the identified challenges are identified and presented.

# Sammendrag

Roboter i dag brukes først og fremst til oppgaver som er repetitive og forutsigbare. Men mange oppgaver som skaper verdi for samfunnet vårt inkluderer høy grad av tilfeldighet, for eksempel å plukke opp gjenstander. På grunn av det enorme mangfoldet av forskjellige objekter, er roboter tradisjonelt begrenset til å plukke i miljøer der objektets geometri er kjent og ingen forstyrrelser oppstår.

Tatt i betraktning det generelle problemet med manipulasjonsoppgaver i den virkelige verden, ser modellfri reinforcement learning ut som svært passende familie av algoritmer, siden de ikke gjør noen antagelser om systemets dynamikk. I stedet finner de sekvensen av handlinger nødvendig for å løse oppgaven gjennom erfaring. Denne oppgaven vil studere disse metodene, med fokus på deres anvendelse på griping av objekter med roboter, samt evaluere deres ytelse og begrensninger når de plukker objekter som ikke er sett tidligere. En serie eksperimenter utføres i et simulert miljø, der de to populære modellfrie reinforcement learning-algoritmene Soft Actor-Critic og Proximal Policy Optimization får i oppgave å plukke opp en kube med en mekanisk manipulator.

Det er vist at modellfrie reinforcement learning-algoritmer gir et rammeverk som kan brukes på ulike manipulasjonsproblemer. Resultatene viser at slike algoritmer kan løse plukkeoppgaven i simulatoren uten oppgavespesifikk tilpasning. Oppgaven belyser også betydelige utfordringer som hindrer algoritmene i å løse generelle manipulasjonsproblemer i praksis, viktigst av alt behovet for store datamengder. Til slutt identifiseres og presenteres mulige løsninger på de identifiserte utfordringene.

# Preface

I want to thank Eirik Bjørndal Njåstad for advising me through the process of writing this thesis. I would also like to thank Lars Tingelstad for his support and for pointing me to some highly influential research that ignited a spark for me to research the topic of robotic manipulation further. I am thankful for the support from my parents and the surprising interest they have shown in my discussions of the topic of the thesis. Finally, I would like to thank NTNU and ETH Zürich for allowing me to incorporate a highly fruitful exchange into my degree.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1.

# Introduction

Robots are getting more intelligent. In different areas, they seem to manage superhuman tasks, such as doing backflips with drones [16] or beating the best human player consistently in the game of Go. [46] However, how come robots cannot help us with more everyday tasks, like washing the dishes? What about clearly significant tasks, like healthcare and taking care of the elderly? Shouldn't these intelligent machines be helping us personally in our everyday lives? Why don't we see them on the front lines in the case of emergencies and natural disasters, where help is desperately needed?

This absence of robots in everyday scenarios comes down to the fact that tasks that are easy for humans are not necessarily easy for robots. Roughly speaking, robots have traditionally been exceptional compared to humans in following rules that solve the problem. On the other hand, humans are masters of tasks that require experience.

Consider the task of picking up a previously unseen object. Seen through human eyes, this is no considerable achievement. However, when it comes to specifying a set of rules that apply to every object, the task grows complicated, and it is considered one of the most challenging tasks within robotics. Even with knowledge of scene geometry, determining where to grasp an object is a significant challenge.

The introduction will start by providing a set of scenarios to motivate the improvement of autonomy of robots. It will then go on to describe a fundamental challenge of robotic autonomy, namely manipulation, before defining the main topic of this thesis, robotic grasping. Section 1.4 will provide a helpful framework by introducing the see-think-act cycle. Next, results from deep learning and deep reinforcement learning will be presented, to motivate an emerging trend of applying these method to solve robotic problems. Finally, it will describe the specific problem considered in the report and list the contributions. The overview in section 1.9 will outline the rest of the thesis.

## 1.1. Motivating scenarios

The following paragraphs will provide motivating examples of why robotic manipulation is essential to improve, a skill critical in many scenarios.

### 1.1.1. E-commerce

In E-commerce, consumers typically order custom orders, and the products are picked from a highly diverse assortment. In contrast to in-house production, keeping track of every possible product, its shape, and its weight, for example, is impossible. Many suppliers may supply the products, and the assortment changes rapidly. Therefore, humans are often tasked with picking as robots fail to pick the vast array of products consistently. However, companies mention that finding human resources for this monotonous work is challenging, and robots can work longer hours and go without sensitivity to temperature, air quality, and lighting. [38]

Amazon was ranked as the third company globally by total revenue in 2021 [5], with retail as its primary source of income. [11] In 2015, they offered $ 20 000 as the winning prize in their Amazon Picking Challenge, where top universities applied robotics research to the task of picking. Such an initiative highlights the demand for using robotics in E-commerce.

### 1.1.2. Manufacturing

Advances in robotics are historically due to demand in large-scale industries, and it is indeed a significant driving force behind technological development today. Automation of standard processes such as assembling parts into a product and bin-picking components are active research areas. [49] Human-robot collaboration also demands a high level of intelligence, including assertion of human intentions and subtle communication.

In a competitive market, new products will be introduced frequently. A robot executing a pre-programmed trajectory in a production cycle will need to be re-programmed upon introducing a new product. Humans will be able to take brief instructions to produce the new product quickly. Improving robotic manipulation can heighten abstraction, pushing down the time needed to set up production for a new product, as instructions can be given more concisely.

### 1.1.3. Human assistance

In our lives, we are surrounded by objects that are important to us, and the utility robots have for us will primarily be based on their ability to handle everyday objects.

In robot-human collaboration, tasks involving picking and handing objects are abundant. A motivating example, although futuristic, is a robotic assistant in the kitchen. The need to pick up food products or utensils will be a large part of the assistant's work. Another example of assistance is picking up and bringing personal objects on-demand in healthcare or assisting disabled and older people. As in E-commerce, there are seldom any restrictions on what objects the robot might encounter.

### 1.1.4. Inaccessible and dangerous environments

Environments where human access involves danger, for example, in space or underwater, have called for the help of robots. There, the need for robust autonomy is strong because of the impossibility of human intervention. In some cases, communication is not possible, prohibiting teleoperation. Examples of tasks in such environments are inspection and maintenance of subsea installations or exploration of other planets.

During the Fukushima disaster in 2010, several robots were sent to assist the dangerous work at the nuclear power plant contaminated by radiation. The harsh conditions, as well as lack of communication, made the robots of limited use. Four years later, DARPA hosted a challenge where autonomous robots should exit a vehicle, open a door knob, and turn a valve. The competition was inspired by the dire need for improvement that the Fukushima disaster revealed. [6]

### 1.1.5. Teleoperation

Achievements of human teleoperating dextrous robots set a high benchmark for the capabilities of state-of-the-art hardware. For example, the ANA Avatar X-price semifinal winners, NimbRo, manage to grasp objects and sort them by weight with their avatar system. [42] Their showcase of dexterity and presence shows that humans can use similar actuation and sensors to operate a robot to grasp novel objects, among other many other tasks. However, latency is an issue even in situations where teleoperation is possible. An onboard algorithm would be able to act faster on sensory input.

## 1.2. Robotic manipulation

The motivational examples mentioned above call for advances in many regards; however, this report will focus on the general manipulation problem. Mason [27] gives several definitions of the term, including the following:

**Definition 1** *Manipulation refers to an agent's control of its environment through selective contact.*

This class of problems includes a highly diverse set of tasks; examples are opening doors, building Lego, folding clothes, and pouring liquids. These are typically natural tasks to humans, involving our hands, and happen ubiquitously in everyday activities. The following paragraphs will provide specific challenges in empowering robots to perform manipulation tasks.

#### Sparse rewards

Many robotic manipulation problems are often binary marked as solved or not solved; however, obtaining a solved state can require a complex sequence of actuation signals to the robot. When opening a door, measuring whether the door is open at the end of a trial is simple and well defined. However, the sequences that the agent has to perform to succeed are unlikely to encounter, given no reward signal before task completion. Ibarz et al. [14] describe this as finding the high reward needle in the zero reward haystack.

#### Generalization

Compared to a lab setup, the diversity of encountered problems is enormous, even within a single manipulation problem in the real world. Opening a door at the level of humans, for example, involves being able to cope with any conceivable door design. The step from solving the problem for a specific door in a lab to opening doors in the real world is substantial. [7]

#### Sensing

Manipulation tasks often imply understanding the scene geometry and the poses of the objects; however, the complete state of the scene is seldom directly observable. Onboard sensors usually consist of cameras providing 2D images or 3-D depth images, leaving the agent with the task of extracting useful information from images about the scene to complete the task.

**Robustness**

Unlike a simulated environment or a lab setup, executing tasks in the real world implies being robust to external disturbances. Having the ability to recover and replan in the case of unforeseen factors is crucial, especially in inaccessible environments.

## 1.3. Robotic grasping

Kalashnikov et al. [15] argue that grasping includes a large part of the challenges shared with the much broader robotic manipulation problem. They point to the challenge of generalization as a critical shared factor. The abundance of geometries, appearances, weights, and materials makes robotic grasping an ideal scenario to develop and evaluate methods that can point to solutions to the general manipulation problem.

Robotic grasping is the task of using a gripper and reasonable sensory inputs to pick up an object from the environment, fixing it rigidly to the end-effector of the robot, enabling the robot to use it further.

In many manipulation tasks, robotic grasping arises as a sub-problem. Opening a door, for example, includes the robot rigidly grasping the door handle such that it can be used to open the door. In collaborative manufacturing, handing objects to humans or receiving them can be viewed as an extension of the grasping problem; the object is no longer stationary in a container or on a surface but in the hand of a human.

### 1.3.1. Challenges of robotic grasping

Robotic grasping tasks can take many forms. The complexity of the task and the resulting solutions heavily rely on specific factors of the grasping task. The objects might be isolated on a table or buried in a container among other objects. In addition, object with specific properties prove to be especially challenging to grasp.

**Deformable objects** pose a challenge in several ways. Firstly, the object's appearance changes under forces, making the degrees of freedom not restricted to 6, as for a rigid object. For image-based manipulation, the actor has to generalize and recognize the object under all deformations. Deformability also challenges finding a good grasp, as the geometry changes during contact.

**Specular and transparent** objects also pose a similar challenge, as the appearance can change rapidly depending on the viewing angle, background, and

surroundings

## 1.4. The see-think-act cycle

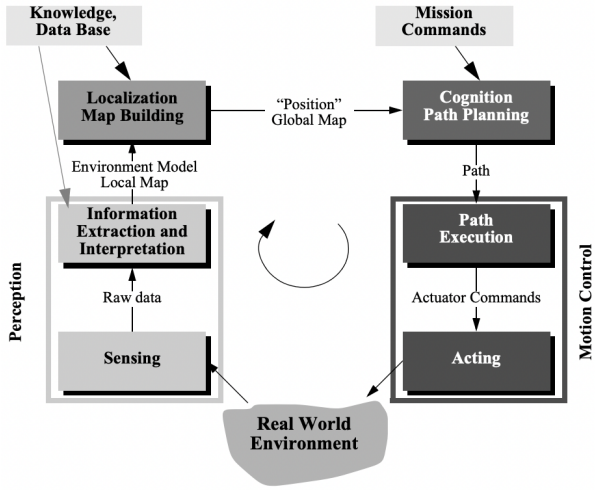



**Figure 1.1.:** The see-think-act cycle, as described by Siegwart et al. in their book "Introduction to Autonomous Mobile Robots" [45]

The see-think-act cycle describes a framework for robotic control:

1. See: Use exteroceptive and interoceptive sensors to get information about the robot's state and its surroundings.
2. Think: Using this information, determine what actions to take to reach a specified goal. This goal can be more than a location; more generally, it can be a desired state of the robot and the environment.
3. Act: Execute the actions in the environment. For example, move the end-effector or the whole robot to a location using actuators and motion control.

This cycle is visualized in figure 1.1, showing the cycle of perception, planning, and motion control. The approach is also termed deliberative control. [44]

In the context of this cycle, this thesis will restrict its view to the "think"-process. Choosing the actions to take can be done using various techniques. For example, a map of the environment can be estimated. Then, a planning algorithm can find a suitable trajectory for the robot to reach a given goal.

## 1.5. Deep learning

In the context of generalization, there is a parallel to classification problems in computer vision. Image classification is a task highly intuitive for humans, building on experience and understanding of the world around us. However, formulating a set of rules a machine can perform to infer whether a picture contains a cat is inconceivable.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [36], a classification challenge consisting of millions of images and hundreds of classes, aims to benchmark state-of-the-art algorithms on the task of object classification. Work by Krizhevsky et al. [20] outperformed the competitors with the AlexNet-architecture, a deep convolutional neural network. Their report concludes that the depth of the network is crucial to obtaining such high results. This fact proves that this seemingly intuitive task for humans demands a high computing complexity.

In the case of image classification, deep neural networks can capture the complicated relationship between appearance and object categories. The intractability of stating these rules is evident by the sheer size of these networks; the AlexNet has 60 million learned parameters. [20]

Image classification has seen significant advancements; however, recent progress has flattened. In terms of accuracy on the ImageNet dataset, the best performance rose from 51 to 80 between 2011 and 2016. From 2016 to 2022, it rose to 90. The last instance of ILSRVC took place in 2017, further suggesting that the necessary breakthroughs have been achieved. Robotic grasping, however, still prevails as an unsolved challenge.

## 1.6. Deep reinforcement learning

In recent years, the field of deep reinforcement learning (deep RL) has produced several success stories. In contrast to supervised learning, as is typically used in image classification, RL is learning how to act in situations to maximize a numerical reward signal. [50, p. 1]

In 2015, Mnih et al. [28] devised a method capable of performing above human experts on an extensive range of Atari games. Their method observes only the screen output of the games and learns complicated tactics to achieve high scores. Shortly after, in 2016, Silver et al. [46] used Deep RL to construct AlphaGo, an algorithm beating the best human Go-player. This goal has achievement has been marked as outstanding and previously thought to be ten years away.

### 1.6.1. Deep RL for robotics

Deep RL has shown many promising results within robotics, learning complex locomotion gaits [41][8][13] and performing several manipulation tasks such as solving a Rubik's cube. [31] Specifically for grasping, Kalashnikov et al. [15] demonstrated that a model free-approach could generalize to grasp unseen objects using only observations from an RGB camera. Their method, termed QT-Opt, will be further described in the related works. Their results hint that model-free methods are promising in the challenge of generalizing in terms of grasping and the general manipulation tasks.

## 1.7. Problem statement

The fundamental motivation for this thesis is the need to develop robots that can perform manipulation tasks at a human level or above, as exemplified by the motivational examples in section 1.1.

Referring to the see-think-act cycle, see figure 1.1, this thesis will focus on the "thinking"-stage, taking place after extraction and processing of sensory data and before acting in the environment.

The thesis will narrow its focus to robotic grasping, as it is a challenging task inheriting many of the challenges common to manipulation tasks, such as generalization.

Within the restrictions, this thesis looks at promising methods to tackle the grasping problem's challenges and, ultimately, general robotic manipulation. Due to the recent breakthroughs using deep RL, we will explore how such algorithms can be used to solve a grasping problem.

Specifically, we will assume a setup similar to that described in appendix A as a system for deployment, and the task of picking up a cube. The thesis will study the possibilities and challenges of using model-free methods by running several experiments in a simulated environment.

Limitations in time motivate us to look for open-source implementations and lightweight frameworks. This pragmatic approach can prove helpful to practitioners who do not have access to large-scale computing power and lack the time and skill necessary to implement specific methods.

## 1.8. Contributions

This report will contribute the following:

1. An overview and discussion of several relevant works on the topic of robotic grasping
2. An evaluation of two popular model-free deep RL algorithms applied to robotic grasping in a simulated environment
3. A discussion of apparent challenges and benefits of deep RL in the context of robotic grasping and general robotic manipulation.

## 1.9. Thesis overview

Chapter 2 will present preliminaries supporting the related work and the methods used in the experiments. Chapter 3 will discuss the related work and literature on robotic grasping, and chapter 4 will present the setup and results from a series of experiments conducted in a simulated environment. The discussion in chapter 5 will regard results from the related work and the experiment in the light of the problem of robotic grasping and the manipulation problem in general. The conclusion of chapter 6 will summarize the observations made in the discussion and point toward exciting topics to explore in further research.

# Chapter 2.

# Preliminaries

This section will describe the concepts relevant to the methods discussed in chapter 3 on related work.

First, section 2.1 will briefly define fundamental concepts within probability theory. Then, turning to classical reinforcement learning, section 2.2 will define Markov decision processes (MDPs), which serve as a framework for defining a wide range of problems. Then, foundational methods for solving MDPs will be defined in section 2.3. Section 2.4 will move towards problems with continuous action and state spaces, where formulating the MDP is impossible. Finally, section 2.5 and 2.6 will briefly define relevant concepts within optimization and deep learning, respectively.

## 2.1. Probability theory

Stochasticity arises when modeling robotic tasks. The environment is often modeled as stochastic, compensating for unavoidable modeling errors. As we will see in section 2.3, having a stochastic actor has important benefits, allowing for exploration and robustness. Therefore, the concepts of entropy and Kullbach-Leibler divergence will be briefly defined.

### 2.1.1. Entropy

The entropy of a distribution is a measure of the uncertainty of the random variable. Formally, it is the average bits needed to encode an outcome of the variable. Intuitively, if the random variable has a high probability of realizing to a small set of values, each of these high-probability outcomes can be encoded with a small number of bits. If, on the other hand, the random variable can take on a wide range of values with equal probability, the average number of bits needed to

represent the outcome is higher. The expression of entropy is analytically defined as: [35]

$$H[p(\mathbf{x})] = -\int p(\mathbf{x}) \log_2 p(\mathbf{x}) d\mathbf{x}$$

For Gaussian distributions in one dimension, this evaluates to:

$$H[\mathcal{N}(\mu, \Sigma)] = \frac{1}{2} \log(\log 2\pi\sigma^2) + \frac{1}{2},$$

which we can see only depends on the variance $\sigma$ of the distribution.

### 2.1.2. Kullback - Leibler divergence

The Kullback-Leibler (KL-) divergence is a measure between two probability distributions, $p$ and $q$, and a natural interpretation is as a measure of difference. It is defined by the following expression:[35]

$$\mathrm{KL}(p||q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}$$

The KL-divergence is always $\geq 0$, and for two identical distributions it evaluates to 0. In addition, it is not symmetrical, $\mathrm{KL}(p||q) \neq \mathrm{KL}(q||p)$.

## 2.2. Markov decision processes

As we will see in the Related Works, it is usual to model problems as Markov decision processes (MDPs)[15] [10], and it provides an essential foundation for classical and state-of-the-art methods.

### 2.2.1. Definition

MDPs are defined by:

- a set of states $\mathcal{S}$
- a set of actions $\mathcal{A}$
- a set of transition probabilities $P^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$
- a set of rewards $r^a_{ss'} = \mathbb{E}[r_t | s_t = s, a_t = a, s_{t+1} = s']$

It is essential to note the Markovian property of MDPs. The transition probabilities only depend on the current state and action: it does not depend on previous

actions and states. Therefore, they must include all information necessary to calculate the distribution of the following state $s'$.

Also important to note is the expression for the reward. It implies an interest only in reward expectation and not a dependency on the variance of the reward distribution. This assumption might be relevant in cases where the notion of risk adversity is of interest.

Finite MDPs are MDPs with a finite set of states and actions, which will be assumed in the following sections.

### 2.2.2. Policies

MDPs offer the possibility of choice through the action set $\mathcal{A}$. Because of the Markovian property, only the current state is needed to determine which action to take in an MDP. Policies are therefore defined as functions from states $\mathcal{S}$ to actions $\mathcal{A}$. This mapping is denoted $\pi : \mathcal{S} \to \mathcal{A}$.

Non-deterministic policies instead output a distribution over the possible actions $\pi : \mathcal{S} \to P(a), a \in \mathcal{A}$. For simplicity of notation, deterministic policies will be considered for the rest of the discussion on MDPs.

### 2.2.3. Discounted reward

Consider a policy applied to an MDP. This results in a Markov reward process (MRP), where the state transitions only depend on the current state, because the action is determined by the policy: $P_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = \pi(s_t))$. A sample path in this MRP starting in any initial state $s_0$, can be denoted $(s_0, a_0, r_0, s_1, ..)$. The expected value of timestep-discounted rewards of this trajectory distribution gives the expression for discounted reward:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))\right], \gamma \in [0, 1] \tag{2.1}$$

It incorporates a measure of short-sightedness through the parameter $\gamma$. A value close to 0 gives a higher relative weighting to the earlier returns. Constructing the policy $\pi$ to maximize $J(\pi)$ will result in a shortsighted or myopic behavior. On the contrary, maximizing the same expression with a $\gamma$ close to 1 will result in far-sighted behavior, where rewards in the future will have the same relative weight as early rewards.

The optimal policy $\pi^*$ is defined to maximize $J$.

### 2.2.4. Value function

Given the discounted reward 2.1 and a policy $\pi$, the corresponding value function can be constructed. For every state $s$, it assigns the expected reward of starting a trajectory in state $s$ in the resulting MRP. It is defined as

$$V^{\pi}(s) = J(\pi|s_0 = s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s\right]$$

The optimal value function $V^*$ is defined to be the value function for the optimal policy $\pi^*$.

### 2.2.5. Action value function

Another useful quantity is the action-value function. It represents the value of taking an action in a state s, and thereafter following the policy:

$$Q(s, a) = \mathbb{E}[r(s_t, a_t) + \gamma V(s') \mid s_t = s, a_t = a]$$

### 2.2.6. Bellman equation

An important result in the context of MDPs is the Bellman equation. Reformulating the equation for the discounted reward 2.1 gives the following relation:

$$V^{\pi}(s) = J(\pi|s_0 = s) = \mathbb{E}\left[r(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s\right]$$

$$= \mathbb{E}\left[r(s_0, a_0) + \gamma V^{\pi}(s')|s_0 = s\right]$$

The Bellman equation connects the value function at state $s$ with the value functions for all the possible successor states $s'$. It also poses a requirement for the optimal value function:

$$V^*(s) = \max_a \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a)V^*(s')\right] \tag{2.2}$$

#### TD-Error

Observing the transition $(s, a, r, s')$, we can define the Temporal Difference (TD) error $\delta^{TD}$ for the action-value function $Q$:

$$\delta^{TD} = Q(s, a; \theta) - r - \gamma \max_{a'} Q(s', a'; \theta) \tag{2.3}$$

This is closely related to equation 2.2, and will be zero for all state action pairs for an optimal Q-function.

### 2.2.7. Optimal policy

Every MDP inherit a fundamental question: "*What actions can be taken to maximize the expected reward?*" As mentioned, the optimal policy associated with an MDPs is the answer to this question. With the definition of the value function, we can obtain the optimal policy as a greedy policy utilizing the optimal value function:

$$\pi^*(s) = \max_{a \in A} \mathbb{E}[r(s,a) + \gamma V^*(s')p(s'|s,a)]$$

### 2.2.8. Maximum entropy MDPs

Maximum entropy RL can be formulated as not only maximizing the discounted reward of the policy but also the entropy of the policy.

Tang and Haarnoja [43] describe the benefits of maximum entropy learning through the example in figure 2.1. When maximizing discounted reward, the agent would tend to find the shortest path as shown in the left-hand figure. It would, however, not learn the possibility of the lower route, as it is not optimal. When jointly rewarding the entropy of the policy, the policy will incorporate the near-optimal behavior of taking the lower route, as it adds to the entropy of the algorithm.

**(a)** Single path **(b)** Both paths

**Figure 2.1.:** A example problem to illustrate the robustness that maximum entropy RL provides. From Tang and Haarnoja [43]

#### Maximum entropy objective

To maximize for entropy, a new term is added to the discounted reward. For each timestep, the entropy of the policy is rewarded: [10]

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi}[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t))] \tag{2.4}$$

Here $\alpha$ is called a temperature parameter. Setting this to 0 results in the original discounted reward. $\rho_\pi$ is the distribution of action and states visited, assuming trajectories are generated by policy $\pi$.

In the context of maximum entropy reinforcement learning, the term "soft" is often used. This means that all actions have a nonzero probability in every state [50, p. 100]:

$$\pi(\mathbf{a}|\mathbf{s}) \geq 0, \forall \mathbf{a} \in \mathcal{A}, \forall \mathbf{s} \in \mathcal{S}$$

## 2.3. Solving MDPs

This section will describe policy- and value iteration, which are procedures for finding the optimal policy for a given MDP. Lastly, it will describe the solution of maximum entropy MDPs, where maximizing the entropy of the policy is an added objective.

### 2.3.1. Policy iteration

Using the fact that the optimal policy is simply the greedy policy associated with the optimal value function, the policy gradient algorithm incrementally improves the policy by updating to the greedy policy of its value function in each iteration. The algorithm outlined in algorithm 1, will converge to the optimal policy $\pi^*$.

**Algorithm 1** Policy Iteration

**procedure** POLICY ITERATION
  $\pi \leftarrow randomPolicy()$
  **while** $\pi$ not converged **do**
    $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$ ▷ Policy evaluation
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$ ▷ Policy improvement
  **end while**
  **return** $\pi$
**end procedure**

Each iteration requires updating value function approximation for the current policy, called policy evaluation. This is done by iterating over all states and updating $V(s)$ in the following manner:

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$$

The policy improvement step aims to update the policy greedily on current value function $V^\pi$. This is done by looping over all states and updating the policy:

**Algorithm 2** Value Iteration

**procedure** VALUE ITERATION
  $V_0(s) \leftarrow \max_a r(s,a)$
  **while** $||V_t - V_{t-1}||_\infty \leq \epsilon$ **do**
    **for** each $s \in \mathcal{S}$ **do**
      $V_t(s) \leftarrow \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_{t-1}(s)$
    **end for**
  **end while**
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
  **return** $\pi$
**end procedure**

$$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \tag{2.5}$$

### 2.3.2. Value iteration

Value iteration, in contrast, updates the value function directly. It updates the next estimate $V_t$ greedily on itself. The algorithm is outlined in algorithm 2.

Note: $||V_t - V_{t-1}||_\infty = \max_s |V_t(s) - V_{t-1}(s)|$.

### 2.3.3. Solving maximum entropy MDPs

Solving MDPs, given the maximum entropy objective of equation 2.4, can be done using soft policy iteration. In the same manner that policy iteration alternates between policy evaluation and improvement, soft policy iteration alternates between soft policy evaluation and improvement.

#### Soft Policy improvement

Instead of acting greedily on the action-value function, the entropy must also be considered. This additional objective entails reformulating the policy improvement step from policy improvement, equation 2.5.

To motivate the derivation of the maximum entropy policy improvement, consider a one step problem with a discrete policy distribution $\pi(\mathbf{a})$. [32] The objective is defined as:

$$\max_{\pi(\mathbf{a})} \mathbb{E}[r(\mathbf{a})] + \alpha \mathcal{H}(\pi(\mathbf{a})) \tag{2.6}$$

This can be posed as an constrained optimization problem over $\pi$, which can be solved using the Lagrangian. The solution to this optimization problem is

$$\pi^*(\mathbf{a}) = \frac{1}{Z} \exp\left(\frac{1}{\alpha} r(\mathbf{a})\right),$$

where $Z$ is a normalizer, to ensure that the elements of $\pi$ sums up to 1.

$$Z = \sum_a \exp\left(\frac{1}{\alpha} r(a)\right)$$

The intuition behind the temperature parameter $\alpha$ is illustrated by looking at the limit when $\alpha \to 0$. We then get that the ratio of the term $\frac{1}{\alpha} r(\mathbf{a})$ associated with the highest reward will grow. The result is that all probability is assigned to this action, simplifying the greedy policy in the classical policy improvement from equation 2.5.

When $\alpha \to \infty$, the ratios between the terms $\frac{1}{\alpha} r(\mathbf{a}), \mathbf{a} \in \mathcal{A}$ goes toward 1, resulting with all elements of $\pi^*$ being weighted by the same probability. This is the other extreme case, where entropy only is maximized without taking into account the objective of maximizing reward.

Extending this result to multi step MDPs, we write down the Bellman equation for the value function, incorporating the entropy objective:

$$V_k(\mathbf{s}) = \max_\pi \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + \alpha \mathcal{H}(\pi(\mathbf{a}|\mathbf{s}) + V_{k-1}(\mathbf{s}'))\right] \tag{2.7}$$

The action value function is defined as:

$$Q_k(\mathbf{s}, \mathbf{a}) = \mathbb{E}[r(\mathbf{s}, \mathbf{a}) + V_{k-1}(s')]$$

Plugging this into equation 2.7 for the Bellman equation for the soft value function gives:

$$V_k(\mathbf{s}) = \max_\pi \mathbb{E}[Q(\mathbf{s}, \mathbf{a}) + \alpha \mathcal{H}(\pi(\mathbf{a}|\mathbf{s}))]$$

Comparing this expression 2.6 of the objective for the one-step problem, we see that it is similar, as the only difference is that $r(\mathbf{s}, \mathbf{a})$ is changed by $Q(\mathbf{s}, \mathbf{a})$. Consequently, the policy maximizing this objective is:

$$\pi_k(\mathbf{a}|\mathbf{s}) = \frac{1}{Z} \exp\left(\frac{1}{\alpha} Q_k(\mathbf{s}, \mathbf{a})\right) \tag{2.8}$$

And the value function update for this policy is:

$$V_k(\mathbf{s}) = \alpha \log \sum_a \exp\left(\frac{1}{\alpha} Q_k(\mathbf{s}, \mathbf{a})\right)$$

## 2.4. Model-free RL with function approximators

The techniques discussed in the previous sections assume that the MDP can be represented in a tabular form. From this follows that the action-value function $Q(s, a)$ can be represented by a matrix (or a tensor) with each element corresponding to taking action $a$ in state $s$. In many settings, it is infeasible to explore the state space and the entire action space for each state. The need to generalize across the state spaces and actions spaces by using assumptions such as continuity, smoothness, and periodicity is apparent.

For robotic grasping, it is imperative to represent the values and action-value functions with approximators, thus turning the problem of learning the value function and the Q-function into regression. This section will describe Q-learning, which uses function approximators to learn the Q-function. It will then look to policy search methods, where the parameters of the policy are learned directly.

### 2.4.1. Q-learning for continuous state and action spaces

In practice, the Q-function can be represented in a tabular form, listing a value for each possible pair of states and actions. This representation quickly becomes unpractical with large state and action spaces; typically, parametric functions are used instead. A Neural Network (NN) is a popular example and is described in section 2.6 in greater detail.

The TD-error for the Q-function, equation 2.3 provides an objective which can be used as for fitting the parameters. To simplify the computation of the gradient with respect to $\theta$, the previous parameters $\theta_{old}$ could be used to obtain an estimate of the target:

$$\delta = Q(s, a; \theta) - r - \gamma \max_{a'} Q(s', a'; \theta_{old}) \tag{2.9}$$

Note: Following the notation of Sutton and Barto [50], we denote the TD-error with $\delta$, and its expectation, called the Bellman error, with $\bar{\delta}$.

For example, the following loss can be used in the training of a neural network:

$$l_2 = (\hat{y} - y)^2 = \frac{1}{2}\left(Q(s, a; \theta) - r - \gamma \max_{a'} Q(s', a'; \theta_{old})\right)^2 = \frac{1}{2}\delta^2 \tag{2.10}$$

The gradient of this loss can be expressed as:

$$\nabla_\theta \delta = \delta \nabla_\theta Q(s, a)$$

Using back propagation, the gradient $\nabla_\theta Q(s, a)$ can be effectively obtained. The parameters of the network could be updated using gradient ascent:

$$\theta \leftarrow \theta + \alpha \delta \nabla_\theta Q(s, a; \theta),$$

where $\alpha$ is the step length. This algorithm is similar to stochastic gradient descent (SGD), widely used in supervised deep learning. However, $(s, a)$-pairs are not independent and identically distributed, as would typically be the case in the supervised setting. The system dynamics and the policy determine this correlation.

Another important difference, is that the "label" $y = r + \max_{a'} Q(s, a'; \theta_{old})$ is not a true label, but rather a bootstrap estimate obtained using the previous parameters $\theta_{old}$. A large part of recent research in RL is concerned with what this entails and how to make Q-learning methods work effectively despite this approximation.

### 2.4.2. Q-learning for the maximum entropy objective

The modification of the objective gives a new formulation of the action-value function $Q$ and the TD-error, giving rise to a new formulation for Q-learning. The soft Q-function [9] is defined as:

$$Q_{\text{soft}}(s, a) = r(s, a) + \mathbb{E}[\sum_{l=1}^{\infty} \gamma^l (r_{t+l} + \alpha \mathcal{H}(\pi(\,\cdot\,|s_{t+l})))]$$

The soft Bellman equation for the soft Q-function is defined as:

$$Q^\pi(s, a) = \mathbb{E}_{s', a'}[r(s, a) + \gamma(Q^\pi(s', a') + \alpha \mathcal{H}(\pi(\,\cdot\,|s')))]$$

Minimizing the soft Bellman error provides an objective for Q-approximators:

$$\bar{\delta}_{\text{soft}} = \mathbb{E}_{s'}\left[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi}[\tilde{Q}^\pi(s', a')] + \alpha \mathcal{H}(\pi(\,\cdot\,|s'))\right] - \tilde{Q}^\pi(s, a)$$

This error can be estimated without bias by a single sample. Recalling the definition of entropy $\mathcal{H}(\pi(\,\cdot\,|s')) = \mathbb{E}_a[-\log \pi(a|s)]$, we can estimate $\bar{\delta}_{\text{soft}}$ upon observing the transition $(s, a, r, s')$:

$$\tilde{\delta}_{\text{soft}} = r + \gamma(Q^\pi(s', \tilde{a}) - \alpha \log \pi(a'|s')) - Q^\pi(s, a),\ \tilde{a} \sim \pi(\,\cdot\,|s')$$

### 2.4.3. Policy gradient methods

Instead of learning the action-value function, policy search algorithms aim to find the policy directly. Parameterizing the policy with a set of parameters $\theta$, we can formulate a loss function to optimize wrt. $\theta$. Policy gradient methods are based on the idea of estimating a gradient of the expected discounted reward, and update the policy by stepping in the direction of this gradient. Policy gradients are widely used in combination with NNs, as gradients are easily obtainable using auto differentiation frameworks.

Policy gradients aim to maximize the discounted reward, discussed in 2.1. Defining $\tau$ as the episode, consisting of transitions and rewards, $\tau = (s_0, a_0, r_0, .....)$, we can write the discounted reward as the following:

$$r(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

This expression can be viewed as an evaluation of the policy which produced the trajectory. The expected return of a trajectory can be stated as an objective:

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)] := \arg\max_\theta J(\theta)$$

Here, the the trajectory distribution is parameterized by $\theta$, and can be expressed as:

$$p_\theta(s_0, a_0, ..., s_T, a_T) = p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) \tag{2.11}$$

The policy gradient methods aim to obtain the gradient $\nabla_\theta J(\theta)$, and use this to optimize the objective function over $\theta$. With equation 2.11, we can write this gradient as

$$\begin{aligned} \nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \\ = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)] \end{aligned} \tag{2.12}$$

In the second step of the derivation, we are using the fact that:

$$\nabla_\theta \log p_\theta(\tau) = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} \rightarrow \nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$$

The expected value in 2.12 can be approximated by a sample average:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=0}^{T} r(s_{i,t}, a_{i,t}) \right) \tag{2.13}$$

This method highly inefficient, and has a large variance. So large in fact, that this is never used in practice. A reason for this is that, here the probability of an action is associated with a positive gradient if the whole episode is successful. An important fact to minimize this variance, is to use a baseline. It can be shown that incorporating a baseline $b(s)$ does not affect the optimization objective, but can decrease the variance of the gradient estimate. [50, p. 329]

An early proposed method in this regard is the REINFORCE-algorithm [53]. As a baseline, it uses an estimated expected reward of the time steps remaning in the episode: $t \in [t, T]$, rather than the full trajectory:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \gamma^t G_t \nabla \log \pi(a_t|s_t; \theta) \right]$$

where $G_t$ is the discounted downstream reward of the episode from the current time step $t$:

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

This gradient can be estimated via sample average, as seen in 2.13. Using gradient ascent we can update the parameters for the policy:

$$\theta \leftarrow \theta \ + \alpha \nabla_\theta J_{REINFORCE}(\theta)$$

### 2.4.4. Advantage estimation

In the same way that the downstream rewards $G_t$ are used as a baseline in REINFORCE, a popular alternative baseline is the advantage function:

$$A(s, a) = Q(s, a) - V(s)$$

Schulmann et al. [40] report that this gives the lowest possible variance and offer the generalized advantage estimate (GAE) as a way to estimate the advantage function. It is used in the Proximal Policy Optimization algorithm discussed in section 3.6.

Before defining GAE, we define $\delta_t$ as the TD-error for the value function $V(s)$ at timestep $t$:

$$\delta_t = r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right)$$

This is analogous to the expression for the TD-error for the action value function $Q(s, a)$, equation 2.3, used in Q-learning. Consequently, the TD-error at timestep $t + 1$ is:

$$\delta_{t+1} = r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})$$

The generalized advantage estimate for the action taken at timestep $t$ is defined as the sum:

$$\hat{A}_t^{GAE} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} = \sum_{t'=0}^{\infty} (\gamma\lambda)^{t'} \delta_{t+t'}^{V}$$

This gives the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \hat{A}_t^{GAE} \nabla \log \pi(a_t|s_t;\theta) \right]$$

This gradient can be estimated by collecting transitions from the environment and truncating this sum to estimate the advantage. A more detailed description of this will follow in section 3.6, discussing Proximal Policy Optimization.

## 2.5. Optimization

This section will briefly cover relevant methods from the field of optimization. Firstly, the cross-entropy method will be discussed, as it is encountered in work by Kalashnikov et al. on QT-Opt. [15]

### 2.5.1. Cross-entropy method for continuous optimization

In the case of continuous and large discrete action spaces, the maximization of $Q(s, a)$ over the action $a$ is not trivial. It is impossible to check every possible action, as it can be readily done in small discrete action spaces. This introduction focuses on CEM as applied for continuous optimization in work done by Kalashnikov et al. [15].

CEM is a sampling-based method for function optimization, and this description will describe the case of a Gaussian distribution. It first samples the function at $N$ points. It then fits a Gaussian distribution to the best $M < N$ of these points, from which it samples the following $N$ points. This procedure can be repeated until the standard deviation of the fitted Gaussian is under a given threshold or simply after a fixed number of iterations. The overall algorithm is outlined in algorithm 3.

It is essential to set the initial mean $\mu_0$ and variance $\sigma_0$ suitably; a natural choice

**Algorithm 3** Cross-Entropy Method for Continuous Optimization

```
procedure CROSS ENTROPY METHOD(f, N, M, iterationLimit, μ_0, s )
    distr ← GaussianDistr(μ_0, σ_0)
    while i ≤ iterationLimit do
        pointsAndValues ← {}
        while j ≤ N do
            x_j ← distr.sample()
            f_j ← f(x_j)
            pointsAndValues.append((x_j, f_j))
            j ← j + 1
        end while
        x* ← sortOnF(pointsAndValues)[0 : M]          ▷ Get M best points
        distr ← fitGaussianToPoints(x*)
        i ← i + 1
    end while
    return distr.mean()                    ▷ Return mean of best points so far
end procedure
```

is a variance proportional to the size of the desired search space and a centered mean to ensure a broad search.

### 2.5.2. Trust policy optimization and line search

Given the unconstrained optimization problem

$$\min_x f(x)$$

Nocedal and Wright [30, p. 19] describe two categories for iterating from a point $x_k$ to $x_{k+1}$:

- **Line search methods** choose a direction from the current iterate $x_k$, and then search for the step length.
- **Trust region methods** first choose a region around the current iterate $x_k$, and then finds the direction.

#### Trust region method using a quadratic approximation

As an example of a trust region method, the following algorithm finds an optimum by iteratively maximizing a quadratic approximation around $\hat{\theta}_{max}$:

1. Approximate the function around the current guess $\tilde{f}(\theta) \approx f(\theta)$.

2. Choose a trust region
3. Optimize the approximation in this region: $\tilde{\theta}^* = \arg\min_{\theta \in \text{trustRegion}} \tilde{f}(\theta)$

A suitable approximation will be close to the real objective function $f$ in the chosen trust region.

## 2.6. Neural networks

Because of its importance in modern RL methods, this section will introduce artificial neural networks (NNs), and a subset of these, convolutional neural networks (CNNs).

Artificial neural networks (NNs) are inspired by biological compositions of neurons, as found in the human brain. An illustration of an NNs is shown in the figure as a computational graph. The left-hand side, labeled $x_{ni}$, are the inputs to the network, $z_{nj}$ are the hidden units, and $y_{nk}$ are the outputs. Each node except for the input layer has an activation dependent on the linear weighted sum of the activation of nodes pointing to it. Specifically, a nonlinear activation function $\phi$ is applied to the weighted sum to calculate the activation. In the figure, the weights are denoted $v_{ij}$ and $w_{jk}$. For example, the activation of $z_{nj}$ can be written as:

$$z_{nj} = \phi\left(\sum_{i=0}^{D} w_{ij} x_i\right)$$

Examples of activation functions $\phi$ are ReLU and the Sigmoid-function.

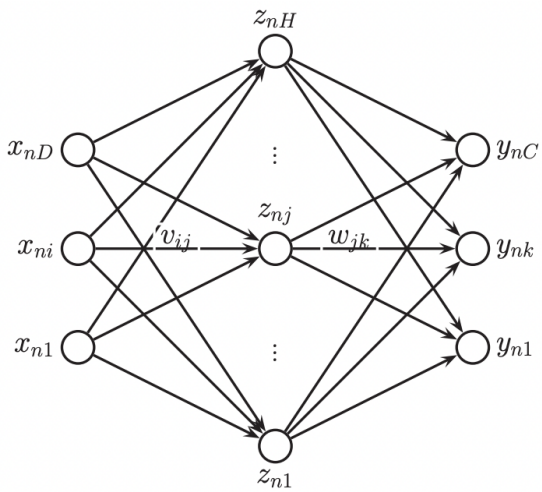



**Figure 2.2.:** Computational graph for a NN [29]

### 2.6.1. Convolutional neural networks

In image processing, convolutional operations with handcrafted filters have been used extensively for edge detection and feature extraction. [2] [24] Convolutional neural networks (CNNs) instead learn these filters, making them powerful for interpreting visual images. A figure of CNN architecture is shown in figure 2.3, an architecture featuring convolutional, subsampling, and fully connected layers.

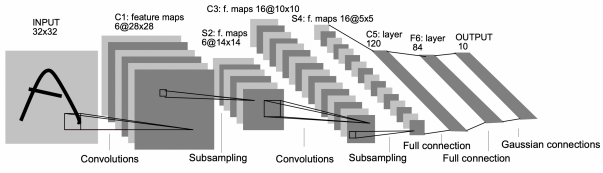



**Figure 2.3.:** The LeNet-5 arcitechture [22]

Convolutional layers are especially applicable to image-based object recognition, because of it's invariance to translations. Thus, one filter can recognize a feature regardless of its position in the picture. For example, a single filter can detect horizontal edges for the whole image due to the convolution over the whole image. This is different from traditional NNs, where this invariance instead has to be learned.

### 2.6.2. Auto differentiation

Neural networks are a popular representation of policies, value, and action-value function approximators. Typically, an objective is formed for each network, and auto differentiation is used to obtain the gradients of this objective wrt. to the network parameters. An example of this is the Soft Actor-Critic algorithm, described in section 3.7.

In broad terms, auto differentiation frameworks keep track of the computational graph involving NNs. An often reoccurring pattern is the following:

1. Sample data from a distribution $\mathcal{D}$
2. Compute predictions $\hat{y} = f(x_i)$ using NN
3. Compute loss $L(\hat{y})$
4. Compute gradients of loss wrt. to NN parameters
5. Step in the gradient direction

Stable baselines 3 [34], which is used in the experiments, utilize the Pytorch framework to obtain the gradients.

# Chapter 3.

# Related work

This chapter provides an overview of relevant work done on robotic grasping. A high-level overview of the methods and their theoretical background is given. In addition, the results and assumptions of the current methods are also presented to outline current challenges.

Firstly, it is important to mention that robotic grasping is a large and active area of research, and providing an exhaustive overview of the most successful methods in the field is a challenging task. Important sources for choosing the methods are the survey on learning-based robotic grasping done by Kleeberger et al. [19] conducted in 2020 and the overview provided by Ibarz et al. [14], in "How to Train Your Robot: Lessons We've Learned" published in 2021. At the time of writing, a search of "robotic grasping" using the service Google Scholar returns roughly 13000 publications on the topic since 2021. However, this chapter aims to point out current challenges within robotic grasping and give an indication of the progress that has been made to solve them.

Kleeberger [19] report that Dex-Net and QT-Opt methods have the highest reported grasp success rate, up to 98% and 76-96%, respectively. Section 3.2 will give an overview of the Dex-Net method and its results and limitations. Subsequently, 3.4 will explore the QT-Opt method. Section 3.3 will focus on the review article on robotic deep RL from Ibarz et al. [14], providing an overview of the challenges of robotic grasping. A technique to improve the sample efficiency of off-policy methods, Hindsight Experience Replay, [1] is discussed in section 3.8. Finally, the popular deep RL methods Soft Actor-Critic and Proximal Policy Optimization will be described in section 3.7 and 3.6, respectively.

## 3.1. Literature on grasping

A large part of the literature on robotic grasping divides the problem into two tasks. First, using the sensory information, find a grasp. Finally, move the end-effector to this grasp and perform the grasp. This separation of concerns is sometimes referred to as an open-loop approach to grasping. [15] [14]

The act of finding a proper grasp is termed grasp synthesis, and there exists a range of methods to solve this problem. A review article by Sahbani et al. [37] offers an overview, and the authors differ between analytic and empirical methods. The "Springer Handbook of Robotics" by Siciliano and Khatib [44] describes analytic models for finding contact points and forces necessary to ensure a stable grasp given an object and a gripper geometry.

Central terms within grasp analysis are force closure and form closure. Form closure is achieved when friction-less contact points between the manipulator and the object keep the workpiece rigidly attached to the manipulator. A 6-degree-of-freedom object needs seven friction-less contacts to achieve form closure. Force closure includes frictional forces to withstand external forces. If the gripper is modeled as two soft fingers, force closure of a 6 degrees of freedom object can be achieved with two contacts.

These methods provide essential tools to analyze the grasp stability given contact points and object geometry. However, a determining factor in many scenarios stated in section 1.1 is that the geometries of the objects to be grasped are seldom known to us. Furthermore, in cases where object geometries are known, such as in manufacturing, tolerances infer minor variations from unit to unit.

The concept of empirical grasp synthesis is based on evaluating grasp samples given a metric. Empirical grasp synthesis will be further explored in section 3.2 on work by Mahler et al. on their Dex-Net dataset.

## 3.2. Learning ambidextrous robot grasping policies (Dex-Net 4.0)

Work by Mahler et al. [26] with the Dex-Net 4.0 dataset achieve picking of novel objects with high reliability. Kleeberger et al. report this method to have the highest reported grasp success rate among all the methods they considered. [19] Mahler et al. devise a policy capable of grasping using two gripper modalities, a parallel-jaw and suction-cup gripper.

This section will first formally define the problem formulation used by Mahler et al. and subsequently outline their method, referred to as Dex-Net throughout the

rest of the thesis. Lastly, it will present a set of assumptions and limitations.

### 3.2.1. Objective

Mahler et. al define the grasping problem as a partially observed MDP (POMDP), meaning that the agent cannot observe the full state of the system:

$\mathbf{y}_t$: The observation received at grasp attempt $t$. On the form of depth images.

$\mathbf{u}_t$: The grasp action taken, upon receiving observation $\mathbf{y}_t$.

$\pi(\mathbf{y}_t) \in \mathcal{R}$: Policy that maps observations to grasp actions.

$\mathbf{x}_t$: Latent state of the whole scene, thereby objects and camera, unknown to the robot.

The state $\mathbf{x}$ describes all information about the objects, the camera intrinsic parameters and pose: $\mathbf{x} = (\mathcal{O}_1, \dots \mathcal{O}_m, \mathcal{C}, \mathbf{w}_1, \dots \mathbf{w}_m)$

These quantities are related according to the distributions defining the POMDP. After taking an action $\mathbf{u}_t$, a reward of $R_t = 1$ is given if the grasp is successful, and the agent is rewarded 0 otherwise. The chosen metric for evaluating a policy is the rate of reward, or mean picks pr. hour (MPPH):

$$\rho(\pi) = \mathbb{E}\left[\left(\sum_{t=0}^{T-1} R_t\right) \Big/ \left(\sum_{t=0}^{T-1} \Delta_t\right)\right]$$

Where T is the number of attempts, and $t$ is the time of executing time in hours. We can see that this expression considers both the grasp success and the execution time. The expectation is taken over the environment distribution:

$$p(\mathbf{x}_0, \mathbf{y}_0, \dots, \mathbf{x}_T, \mathbf{y}_T) = p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \pi(\mathbf{y}_t))$$

### 3.2.2. Method

Mahler et al. assume a constant time per grasp, thereby keeping $\Delta t$ fixed. What remains is to reach a perfect grasp success rate. Their approach is to train a grasp quality CNN (GQ-CNN) for each gripper type, predicting the probability of grasp success given a candidate grasp with the given gripper. During deployment, they maximize over both GQ-CNNs to determine which gripper to use and where to grasp.

#### Dataset generation

The dataset consists of observations, grasps attempts, and their reward: $D = \{R_i, \mathbf{y}_i \mathbf{u}_i\}$. Generating the data set is done by using an explorational policy based on an algorithmic supervisor that estimates the grasp success using the concept of force closure. [25] This data collection is done to create a balanced dataset, consisting of both successful and unsuccessful grasps. Synthetic depth images are captured using a simulated environment.

### 3.2.3. Results

#### Ambidextrous robot grasping

The resulting policy has a 95 % reliability when picking in heaps of 25 novel objects with 300 MPPH. It outperforms scripted policies on adversarial objects that are transparent, deformable, or geometrically challenging to grasp.

### 3.2.4. Assumptions and imitations

The policy trained on the Dex-Net 4.0 dataset, while scoring high on the reliability of an extensive range of novel objects, has some essential assumptions.

The modeling of the problem describes the actions as a grasp attempt, either a grasp with a two-fingered gripper or a pick using a suction cup. This one-step formulation implies that no pre-grasp manipulation is possible, which in some cases can be beneficial. It is worth noting that Mahler et al. add a non-prehensile pushing action if the policy fails to grasp several times consecutively; however, long-term planning is not possible. [3]

The algorithm always picks the object with the highest grasp success in the heap, involving that it is not possible to specify the object to be grasped to the policy. Further, it cannot search for and pick up objects that are occluded.

An analytic method is used to compute the stability given geometry and a proposed grasp in the dataset generation. This dependence restricts the Dex-Net method to the problem of grasping and is not readily transferable to pre-grasp manipulation. It also restricts the possibility of transferring results from Dex-Net to the broader robotic manipulation problem, where an analytic model to calculate the success probability is not necessarily available.

## 3.3. How to train your robot

The review article "How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned" by Ibarz et al. presents important works within robotic learning and describes fundamental challenges and methods to mitigate them. It provides a starting point for uncovering challenges concerning implementing deep RL methods for grasping.

### 3.3.1. The reality gap

The authors put forth the significant benefits of prototyping in simulation rather than in the real world, pointing to factors such as speed and safety. A significant obstacle, however, is the reality gap. It is caused by model inaccuracies in the simulator and differences between rendered and real images.

Several tactics are presented to battle the sim-to-real gap; one of them is domain randomization. The approach consists of forming distributions for parameters in the simulator and, upon resetting the environment, drawing a new set of parameters. The idea is to obtain robustness to a wide range of dynamics, and by forming appropriate distributions, the actual real-world parameters are sufficiently covered by the distributions.

### 3.3.2. Exploration and sparse reward

Addressing the topic of exploration in sparse reward problems, Ibarz et al. discuss the tactic of reward shaping. Reward shaping tries to tackle this by adding rewards to guide the agent to explore more fruitful action sequences. For example, the distance between the end-effector and the object could be incorporated into the reward, guiding the end-effector to approach the object. Ibarz et al. mention that this approach is vulnerable to stimulating greedy or sub-optimal behavior to occur.

## 3.4. QT-Opt

In their work "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation", Kalashnikov et al. [15] show the capabilities of large-scale deep RL trained on real-world data.

Their method achieves a 96% success rate in grasping previously unseen objects. They also show that the method achieves complex strategies, such as a non-prehensile motion to single objects before grasping and recovering if the object is lost or pushed out of the gripper due to external disturbances.

### 3.4.1. Method

QT-Opt is based on Q-learning and extended to be compatible with continuous actions spaces. Figure 3.1 gives an overview of the method. Several robotic cells, consisting of a manipulator and a bin of objects, collect experience using a Q-function approximation which is periodically updated using collected data. The following sections will describe these two processes in further detail.

#### Collecting experience

Deploying the algorithm is done by maximizing the learned Q-function. The Q-function approximator is a neural network with 1.2 million parameters, and the authors choose to maximize the function using CEM optimization described in algorithm 3.

The data collection is done by the following steps:

1. Take image of scene, termed the state $\mathbf{s}$
2. Find action $\mathbf{a}$ as: $\arg\max_{\mathbf{a}} Q_\theta(\mathbf{s}, \mathbf{a})$
3. Execute action $\mathbf{a}$, and repeat.

The authors mention that a scripted policy is initially used to guide the Q-function to successful states.

#### Updating the Q-function approximation

At a high level, the update of the Q-function approximator $Q_\theta$ can be divided into the following steps:

1. Collect a batch of transitions from the replay buffer
2. Estimate the value function
3. Compute the loss for $Q_\theta$
4. Step in gradient direction and repeat

The QT-Opt method aims to minimize Bellman error, defined as the expectation of the TD-error, equation 2.3. The authors use the cross entropy as the distance measure:

$$L(\theta) = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim p(\mathbf{s},\mathbf{a},\mathbf{s}')}[D_{\text{cross-entropy}}(Q_\theta(\mathbf{s},\mathbf{a}), Q_T(\mathbf{s},\mathbf{a}))] \tag{3.1}$$

The target Q-value is defined as:

$$Q_T(\mathbf{s}, \mathbf{a})) = r(\mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')$$

To estimate the value, Kalashnikov et. al. keeps two versions of the Q-network parameters, one exponential moving average $\bar{\theta}_1$ and the other a lagged version of this average, $\bar{\theta}_2$. First, the action $a^*$ is chosen by maximizing $Q_{\bar{\theta}_1}$:

$$a^* = \arg\max_{\mathbf{a}'} Q_{\bar{\theta}_1}(\mathbf{s}, \mathbf{a})$$

Then, a combination of the two networks are used to estimate the value function $V(\mathbf{s})$. To avoid overestimation, a typical complication within Q-learning, the minimum estimated Q-value is chosen to estimate the value:

$$V(\mathbf{s}') = \min_{i=1,2} Q_{\bar{\theta}_i}(s', a^*)$$

This process of updating the action-value function is done in an off-policy manner and can be run as a separate process from the experience collection—this overall process is illustrated in figure 3.1. The "Bellman Updater" produces labeled data by the procedure outlined above, paving the way for supervised learning. The "Training Worker" extracts a batch of labeled samples and updates the parameters of the Q-function.

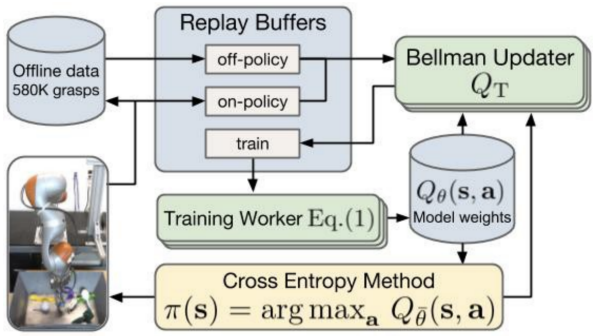



**Figure 3.1.:** Distributed learning using QT-Opt. [15]

### 3.4.2. Assumptions and limitations

The QT-Opt algorithm has some essential assumptions and limitations.

The action space of the algorithm is set to $\mathbf{a}_t = (\mathbf{t}_t, \mathbf{r}_t, g_{\text{close},t}, g_{\text{open},t}, e_t)$, where $\mathbf{t}_t, \mathbf{r}_t$ is the 3D-translation and the rotation around the vertical axis. $[g_{\text{close},t}, g_{\text{open},t}] \in \{0, 1\}^2$ is a one-hot vector indicating whether to close or open the gripper. The stopping criterion $e_t$ is a Boolean which allows the actor to terminate the episode

upon a successful grasp.

In their formulation of the problem, the authors aim to maximize grasp success. This problem formulation entails that the policy has no notion of what object it picks up, meaning that we cannot specify what object it should grasp next. Additionally, it is not able to find a specified object occluded by other objects.

The dataset used to train the QT-Opt algorithm consists of 580k grasp attempts and is collected using seven robotic manipulators for several weeks. While most of this data is collected with minimal human supervision, it restricts the QT-Opt algorithm to cases where such an abundance of data can be collected efficiently.

## 3.5. Robosuite

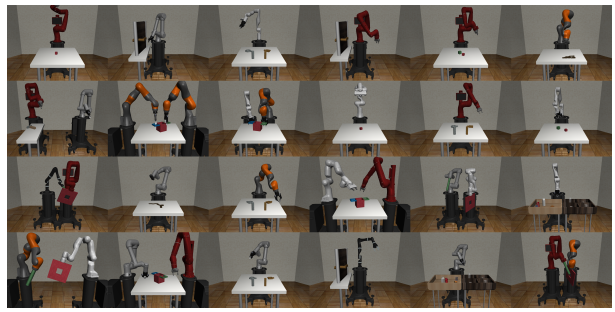


**Figure 3.2.:** An overview of different environments provided in the robosuite package. [54]

Robosuite is a benchmark and simulation framework for reinforcement learning for manipulators based on the Mujoco physics engine. Zhu et al. released Robosuite v1.0 in 2020 along with their publication [54], and at the time of writing, the latest release is v1.3.

The framework provides a set of environments as benchmarks and platforms for testing algorithms. The environments are focused on the manipulation of objects, often requiring long-term planning before completing the tasks. Examples of environments are lifting cubes, screwing bolts, and peg-in-hole operations.

### 3.5.1. Benchmarking results

The authors test the Soft Actor-Critic [10] algorithm in a lifting environment. There, the actor is tasked with picking up a cube using a robot manipulator. Zhu et al. [54] run two experiments with two different control interfaces for the agent. The operation space pose controller (termed OSC-POSE) allows the agent to set

the pose of the end-effector directly. The joint velocity interface, on the other hand, allows the actor to set a rotational velocity on the manipulator's joints.

The results show that the algorithm learns the task faster when controlling the end-effector pose directly, as opposed to the joint velocities. The authors argue that it allows for exploring the task space more efficiently. See figure 3.3. Panda and Sawyer are different types of manipulators, and we see that the algorithm learns the task faster with the pose controller, regardless of the manipulator.

The SAC algorithm solves this using the operation space controller using experience from 500 epochs with 500 steps per episode. For a desktop computer, this training took around two days.

The observation modalities available to the agent are fixed in all the experiments. It can directly observe the position of the cube, as well as the configuration of the robot.


**Figure 3.3.:** Episodic return of SAC deployed on different robots and control interfaces in the Lift-environment. The maximum reward possible is 500. [54]

## 3.6. Proximal Policy Optimization

The Proximal Policy Optimization (PPO) algorithms are a family of deep RL algorithms devised by Schulmann et al. [41]. The authors have focused on forming algorithms which are easy to tune, sample efficient, and straightforward to implement. It can run several actors in parallel, and the pipeline resembles traditional supervised learning, which allows for techniques such as dropout and parameter sharing between the networks.

Among the different algorithms described by Schulmann et al., this section will describe PPO-clip, which the authors report to perform the best over a wide range of environments. It is also the most popular algorithm in the PPO family and is the one used in the experiments.

### 3.6.1. Algorithm

The overall algorithm is similar to policy gradients described in section 2.4, as simplicity is one of the authors' goals. A high-level view of the algorithm is shown in algorithm 4.

**Algorithm 4** Proximal Policy Optimization, Actor-Critic Style

```
Initialize parameters for the Neural Networks
for iteration=1,2,... do
    for actor=1,2,...,N do
        Collect rollouts with π_θold for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize L wrt. θ for K epochs, and minibatch size M
    θ_old ← θ
end for
```

Collecting rollouts can be distributed across multiple workers, each deploying a recent version of the policy in the environment. The advantage estimates $\hat{A}$ are in turn calculated using the rollouts and the trained value function estimate $V(s)$. The authors use a truncated generalized advantage estimate (GAE):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} = \sum_{t'=t}^{T} (\gamma\lambda)^{t'}\delta^V_{t+t'}$$

Recall that $\delta^V_t$ is defined as the TD-error for the value function approximation:

$$\delta_t = r_t + V(s_{t+1}) - V(s_t)$$

After collecting T transitions, the worker calculates the truncated advantage estimates stored in the replay buffer.

### 3.6.2. Loss function

The loss function is comprised of 3 terms, the clipped surrogate loss $L^{\text{CLIP}}$ for the actor, $L^{VF}$ for the value function approximator and an entropy bonus term

$S[\pi](s_t)$.

The loss for the value function approximator is a squared loss error to rewards-to-go estimated using rollout data. [33]

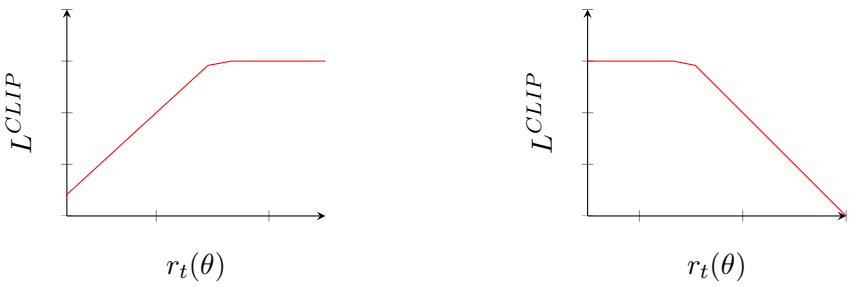
The policy loss is inspired by trust region methods, as it aims to keep the policy update close to the previous policy while still improving the performance. This is reflected in the design of the objective function for the policy, which is defined as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

Here, $\hat{A}_t$ is the GAE estimate for the timestep calculated by the workers. The probability ratio $r_t(\theta)$ is defined as:

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}$$

This is the ratio between the action probability of the new and old policy parameters. Both quantities are positive, so $r$ lies in the range $[0, \infty]$. If the action is more probable with the new policy, $r(\theta) > 1$, and if less likely for the new policy: $r(\theta) \in [0, 1]$.




**(a)** $\hat{A}_t > 0$ **(b)** $\hat{A}_t < 0$

**Figure 3.4.:** Plots of $L^{CLIP}$

Figure 3.4 shows $L^{CLIP}$ using a single point estimate. The leftmost figure shows the case when $\hat{A}_t > 0$. Intuitively, more advantageous actions should be made more probable, and gradients of the $L^{CLIP}$ will tend to shift $r$ in the positive direction. The graph bends off at the point $1+\epsilon$, restricting the update in shifting the probability of beneficial actions beyond this limit. The same is visible in the rightmost graph, showing the case when $\hat{A}_t < 0$, where the probability intuitively should be lowered. The gradient is only positive up until the point $1-\epsilon$, where further decreasing of the action probability does not yield any further increase in the objective value.

As described in the algorithm, the combined loss function is optimized using SGD or the Adam optimizer [18] on the data collected on policy.

## 3.7. Soft Actor-Critic

Soft-Actor Critic (SAC), devised by Haarnoja et al. [10] is a model-free, off-policy, maximum entropy actor-critic algorithm. It builds upon the theory presented in section 2.1.1 from maximum entropy RL. It can be likened to the policy iteration algorithm 1, alternating between updating $V^\pi$ and $\pi$, aiming to maximize the maximum entropy objective defined as equation 2.4.

### 3.7.1. Algorithm

An overview of the SAC algorithm is provided by algorithm 5.

**Algorithm 5** Soft Actor-Critic

Initialize parameters for the neural networks
**for** each episode **do**
  **for** each environment step **do**
    $\mathbf{a}_t \sim \pi(\mathbf{a}_t|(\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim \text{Environment}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1}\}$
  **end for**
  **for** gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ ▷ Update value function
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update action-value function
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\pi)$ ▷ Update policy
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  **end for**
**end for**

Three neural networks are trained: value function $V_\psi(s)$, action value function $Q_\theta(s, a)$ and the policy $\pi_\phi(a|s)$. They are updated in the following manner:

#### Action Value Function

The Q-networks aim to reduce the soft Bellman residual, a difference in predicted Q-value and a bootstrapped Q-value estimate:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[ \frac{1}{2} \left( Q_\theta \left( \mathbf{s}_t, \mathbf{a}_t \right) - \hat{Q} \left( \mathbf{s}_t, \mathbf{a}_t \right) \right)^2 \right],$$

where the Q-value estimate relies on the value network:

$$\hat{Q}\left(\mathbf{s}_t, \mathbf{a}_t\right) = r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[V_{\bar{\psi}}\left(\mathbf{s}_{t+1}\right)\right]$$

In the implementation, this expectation is estimated using samples. As the the action value function is parameterized as a neural network, auto differentiation software is used to obtain the gradients.

To avoid the problem of overestimation in the policy improvement step, the authors train two action-value networks, parameterized by $\theta_1$ and $\theta_2$.

### Value function

Having a separate value network is not necessary, as taking the expectation of the Q-network will produce a value estimate. However, the authors note that introducing a value network stabilizes the algorithm. The authors use the Q-network to compute targets that are used in the objective function for the value network.

$$\hat{\delta}_V = \mathbb{E}[\frac{1}{2}(V(\mathbf{s}_t) - \mathbb{E}[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t)^2]]$$

This expectation is estimated similarly to the action-value function, namely by sampling state-action pairs from the replay buffer.

### Policy

Recalling equation 2.8 for the soft policy update, note the inner expectation over the policy. As $\pi$ also is the distribution we want to optimize over, this complicates the procedure for calculating the gradients of this objective. The authors choose to parameterize the policy in the following way, essentially separating the randomness from the parameters of the model. The randomness is generated from independent noise $\epsilon_t$, and $f_\phi$ is a deterministic function, parameterized by $\phi$:

$$\mathbf{a}_t = f_\phi(\epsilon_t; \mathbf{s}_t) \tag{3.2}$$

The authors do not specify the function $f$ explicitly, however the implementation from Stable Baselines 3 [34] uses the following function [47]:

$$f_\phi(\epsilon_t; \mathbf{s}_t) = \tanh(\mu_\phi(\mathbf{s}_t) + \sigma_\phi(\mathbf{s}_t) \odot \epsilon_t), \epsilon_t \sim \mathcal{N}(0, I)$$

Where $\odot$ represents element-wise multiplication. The mean $\mu_\phi$ and the variance $\sigma_\phi$ are outputs from the policy network.

Plugging equation 3.2 into the equation for the soft policy update 2.8 gives:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}[\log \pi_\phi(f_\phi(\epsilon_t;\mathbf{s})|\mathbf{s}) - Q_\theta(\mathbf{s}, f_\phi(\epsilon_t;\mathbf{s}))]$$

In the same way as for the value and action value approximators, we can get the gradient of this using auto differentiation software and sampling from the replay buffer. After sampling states from the replay buffer, a set of actions $\tilde{\mathbf{a}}_i \sim \pi_\phi(\,\cdot\,|\mathbf{s}_i)$ is calculated using the policy. Together with the log probabilities, the loss for the policy can be computed as:

$$\frac{1}{|B|}\sum_{s\in B}\left(\min_{i=1,2} Q_{\phi_i}(\mathbf{s},\tilde{\mathbf{a}}) - \alpha\log\pi_\phi(\tilde{\mathbf{a}}|\mathbf{s})\right)$$

The gradients of this object with regard to the policy parameters $\phi$ are computed using auto-differentiation.

### 3.7.2. Stable Baselines implementation

The implementation used in the experiments deviates from the above description in the following ways:

1. In the original formulation of SAC, described above, the reward scaling is left as a hyperparameter to be tuned. This is due to the compromise between policy entropy and exploitation. Haarnoja et al. propose a modification to battle this sensitivity by dynamically adjusting the entropy parameter. It is updated along with the policy and value functions as:

   $$\alpha \leftarrow \alpha - \lambda\hat{\nabla}_\alpha J(\alpha)$$

   Where $J(\alpha)$ is defined as:

   $$J(\alpha) = \mathbb{E}_{\mathbf{a}_y\sim\pi_t}[-\alpha\log\pi_t(\mathbf{a}_t|\mathbf{s}_t) - \alpha\bar{\mathcal{H}}]$$

   Similalry to the other objectives, this is estimated using sample estimates, and predicting actions using the current policy. The "desired entropy" $\mathcal{H}$ is usually set to $-\dim(\mathcal{A})$ where $\mathcal{A}$ is the action space.

2. The implementation of SAC used in the experiments omits the value function network, and computes targets using the a set of target networks:

   $$\hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right) = r + \gamma\left(\min_{j=1,2} Q_{\phi_{\text{targ}},j}\left(s',\tilde{a}'\right) - \alpha\log\pi_\theta\left(\tilde{a}'\mid s'\right)\right), \quad \tilde{a}'\sim\pi_\theta\left(\cdot\mid s'\right)$$

3. The parameters for the target networks are updated through Polyak aver-

aging the parameters of the Q-networks. At each iteration $i$:

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i$$

4. At the beginning of collecting rollouts, actions are not sampled using the policy but instead sampled from a uniform distribution. This modification of early exploration is reported to improve performance.

## 3.8. Hindsight Experience Replay (HER)

In their work on Hindsight Experience Replay (HER), Andrychowicz et al. achieve higher sample efficiency in settings with sparse rewards. Furthermore, by adding the HER technique to DDPG, they enable it to solve tasks that the algorithm was previously unable to solve. They also verify their results on a physical robot, allowing it to pick up cubes from a table. They achieve this without fine-tuning in the real world by adding observation noise in the simulator training.

### 3.8.1. Motivation

The fundamental idea behind HER is intuitive. Consider the task of hitting a goal from some distance on a hockey field. The first attempt might result in missing the goal; the puck, perhaps, drifts too far off to the left. Traditional model-free algorithms would treat this episode as a failure, as the goal was not achieved. Instead of viewing this as a failure, the authors use the fact that this would be a perfect shot were the goal placed further to the left. This idea proves effective in cases of sparse rewards, making data from unsuccessful episodes helpful.

More concretely, consider the REINFORCE algorithm. The value of actions in a given state depends on the reward obtained in the remainder of the episode. In a sparse reward environment, an unsuccessful episode will not give us any information on what actions are beneficial.

### 3.8.2. Multi-goal RL

In work on "Universal Value Function Approximators", Schaul et al. train a universal value function $V(s, g : \theta)$, which aims to generalize over potential goals. [39] A possible intuition is that a state close to the goal is also a high reward state, i.e., it is possible to generalize $V$ in the vicinity of $g$, not only in the vicinity of $s$. Following this idea, HER passes not only the current state but the current goal $g$ to the agent, $\pi(s_t, g)$. HER assumes the following:

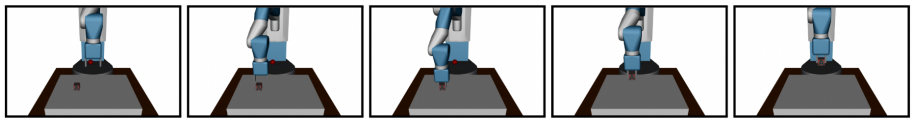


**Figure 3.5.:** The pick-and-place environment used by Andrychowicz et al. [1] to evaluate the performance of HER. The goal position is marked with a red dot.

- Every goal is associated with a function $f_g : \mathcal{S} \rightarrow \{0, 1\}$, which indicates whether a state satisfies the goal: $f_g(s) = 1$ .
- For every state s, there exists a goal which is satisfied in this state: $\exists_m : \mathcal{S} \rightarrow \mathcal{G}$ s.t. $\forall_{s \in \mathcal{S}} f_{m(s)}(s) = 1$.

### 3.8.3. Algorithm

HER can be combined with any model-free off-policy algorithm, for example, DQN [28] or DDPG [23]. Due to the multi-goal representation, tuples on the form $(s_t||g, a_t, r_t, s_{t+1}||g)$, are stored in the replay buffer. The goal $g$ is the goal provided to the actor in the relevant episode. Here, $||$ indicates concatenation. The distinctive feature of the HER method is that it also adds tuples on the form $(s_t||g', a_t, r_t, s_{t+1}||g')$ to the buffer, where $g'$ is not the original goal of the episode, but taken from the set of goals $\mathcal{G}$.

We can sample $g'$ from $\mathcal{G}$ in different ways, for example, the last state of the episode. The tuples will thus seem to be sampled from a successful episode. Through their experiments, the authors find the most successful tactic to randomly sample states encountered downstream in the current episode. Intuitively, this enables the actor to learn how to reach the termination state and encountered states.

### 3.8.4. HER for grasping

Their experiments evaluate HER in three different environments, pushing, sliding, and pick-and-place. Due to its relevance to the focus of this thesis, we will consider the pick-and-place environment.

In this environment, a box is placed at a random position on the table in front of the robot, and it shall move the cube to a specified position above the table. Figure 3.5 illustrates the environment.

Important are the choices for observation and actions spaces and the formulation of reward. Andrychowicz et al. defines the following for their pick-and-place environment:

**Observations:** The policy is given:

- the absolute position of the end-effector,
- the relative position of the cube from the end effector,
- the relative position of the goal from the cube and
- the distance between the fingers of the parallel-finger gripper.

**The action space** of the agent is the gripper's position; the rotation is fixed. In addition, the agent controls the finger distance of the gripper.

**The reward signal** is sparse, which is an assumption needed to use the HER technique. They return a reward of $-1$ as long as the cube is not within a fixed tolerance from the specified goal position and 0 once the goal is reached.

Figure 3.6 shows the results Andryschowicz et al. obtained by training DDPG combined with HER in the pick-and-place environment. We observe that the algorithm solves the task after experiencing approximately 50 epochs. This corresponds to 2M timesteps in the environment.

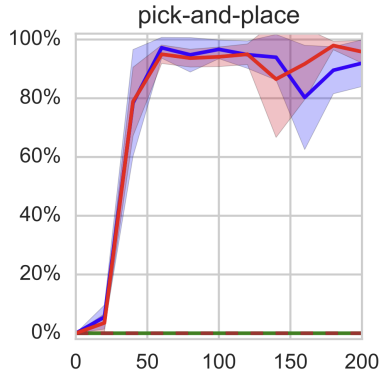



**Figure 3.6.:** The results of DDPG with and without HER in the pick-and-place environment. The green and red lines are the performances of two variants of DDPG without HER. The blue and the red line are two variants of DDPG combined with HER. The horizontal axis is the number of epochs the agent is trained for, and for each epoch, the agent collects 40 000 timesteps in the environment. The vertical axis is the success rate. [1]

### 3.8.5. Assumptions and limitations

In the observation space, the actor is provided information about the object's position, which is directly measured with a typical setup including a camera. To cope with this, the authors use a trained CNN to predict the cube position using

RGB-D images from a camera mounted on the robot in real-life experiments. Using simulations and domain randomization, they achieved stable performance on the real-life implementation.

Andrychowicz et al. report that it is necessary to provide a single state where the cube is grasped and start half of the episodes from this state. The algorithm could not learn the grasping task without this single demonstration state. Additionally, they mention that it can learn the task if some goals are on the table, not just in the air. They provide no experimental results using this method; whether the performance is comparable is unknown.

# Chapter 4.

# Experiments

The experiments in this section aim to describe how model-free deep RL algorithms might be used to solve real-world robotics tasks. Specifically, it will explore the performance of the PPO and SAC algorithm in a simulated environment with a robotic manipulator, where the goal is to pick up a cube.

The experiments will explore the benefits and challenges of using a model deep model-free RL algorithm. The choice of action space, observation space, and reward signal strongly influence the task's difficulty, and different combinations will be explored.

For the action space, work done by Zhu et al. [54] shows that the SAC algorithm learns significantly faster when directly controlling the end-effector pose than controlling the actuator's joint velocities. This simplification of the control is realistic, as all robotic manipulators typically offer this interface. Because of this, direct pose control will be used for the rest of the experiments.

## 4.1. Setup

The experiments will be performed in an environment from robosuite. [54] The package includes, among other manipulators, the KUKA LBR iiwa and predefined environments. It utilizes Mujoco [51] for the simulations and visualizations.

### 4.1.1. Lift environment

The lift environment is one of the predefined environments in Robosuite. It consists of a table with a cube placed randomly on it and a robotic manipulator.

### Reward

The dense reward at each step is defined as the sum of the following terms:

- Reaching: $1 - \tanh(10 \cdot ||p_{gripper} - p_{cube}||)$, where $p_{gripper}, p_{cube} \in \mathbb{R}^3$ is the position of the gripper and the cube.
- Grasping: 0.25 if the gripper is grasping the cube, 0 otherwise. Specifically, both fingers are in contact with the object.
- Lifting: 2.25 if $z_{cube} > z_{table} + 4\text{cm}$, 0 otherwise.

For the sparse reward, only the term associated with lifting is used, point 3, while the dense reward signal sum all of them. This gives a maximum possible reward per time step of $\frac{3.5}{2.25} \approx 1.56$ for dense and 1 for sparse reward. The episode length is fixed to 200 timesteps, meaning that the respective maximum total rewards are 330 and 200. This limit is theoretical, neglecting the time it takes to move the end effector, grip the object and lift it above the threshold.

### Observations

The experiments will two options for the observations:

- RGB-D images of a camera overlooking the table: $I \in \mathbb{R}^{128\times128\times4}$, $I_{ij} \in [0, 255]$.
- A combination of modalities including robot configuration and the cube pose.

The components of the observations of robot configuration and object pose is listed below:

1. The pose of the cube, represented by: $p_{\text{cube}} = [x, y, z] \in \mathbb{R}^3$ and the orientation quaternion: $\xi_{\text{cube}} \in \mathbb{R}^4$.
2. The relative position of the end-effector to the cube: $p_{\text{end-eff}} - p_{\text{cube}}$.
3. Several proprioceptive observations:
    a) For each of the seven joints of the KUKA-arm
        i. Joint position, cosines: $q_{\cos} \in \mathbb{R}^7$, $q_{\cos,i} \in [-1, 1]$.
        ii. Joint positions, sines: $q_{\sin} \in \mathbb{R}^7$, $q_{\sin,i} \in [-1, 1]$.
        iii. Joint velocities: rad/s: $\dot{q} \in \mathbb{R}^7$.
        iv. End-effector position, orientation: $p_{\text{end-eff}} \in \mathbb{R}^3$ , $\xi_{\text{end-eff}} \in \mathbb{R}^4$.

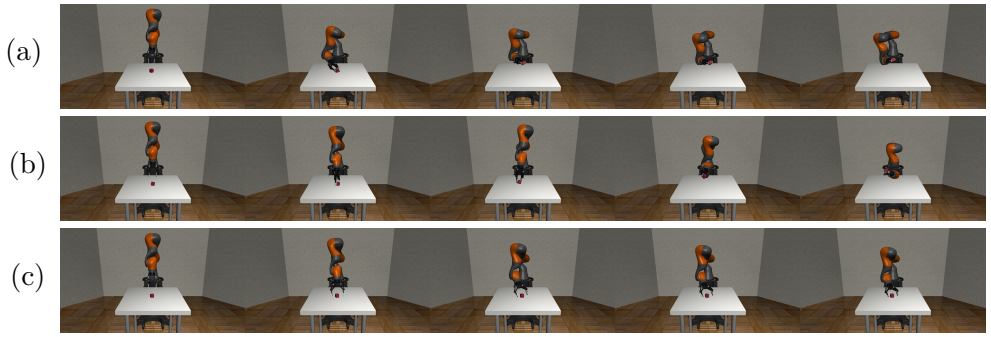



**Figure 4.1.:** Row (a) shows pictures of an episode with SAC trained with sparse reward. Row (b) shows the behavior of SAC trained with dense rewards. Row (c) shows an instance of PPO trained with sparse rewards, where the agent fails to learn the picking task.

v. Angles and velocities of the 6 joints of the gripper: $\dot{q}_{\text{end-eff}}, q_{\text{end-eff}} \in \mathbb{R}^6$.

These elements are concatenated into a 1-D list of 40 values, and all values are represented as 32-bit floats.

When using RGB-D images as observations, the algorithm uses the policy type "CnnPolicy" provided by Stable Baselines 3 [34]. Otherwise, the policy type "MlpPolicy" is used.

## 4.2. Reward shaping experiment

To explore the effect of reward shaping, we run an ablation study. Both PPO and SAC are trained in the lift environment with two settings: One where the algorithm receives the sparse reward signal and one where it receives the shaped reward. Considering the randomness of the environment and the algorithms, we run five instances where only the random seed is changed. In the case of both algorithms, they collect 2 million timesteps in the environment. Additional hyperparameters are listed in Appendix B. All policies are evaluated using sparse rewards to measure progress towards the true goal: picking up the cube. They are also set to exploit during evaluation, which in the case of PPO and SAC means picking the mode of the policy-distribution $\pi(\mathbf{a}|\mathbf{s})$ as the action.

### 4.2.1. SAC

Figure 4.2 shows the comparison of the SAC algorithm trained using a sparse and shaped reward signal. Instances corresponding to the same settings are grouped to indicate the average performance.

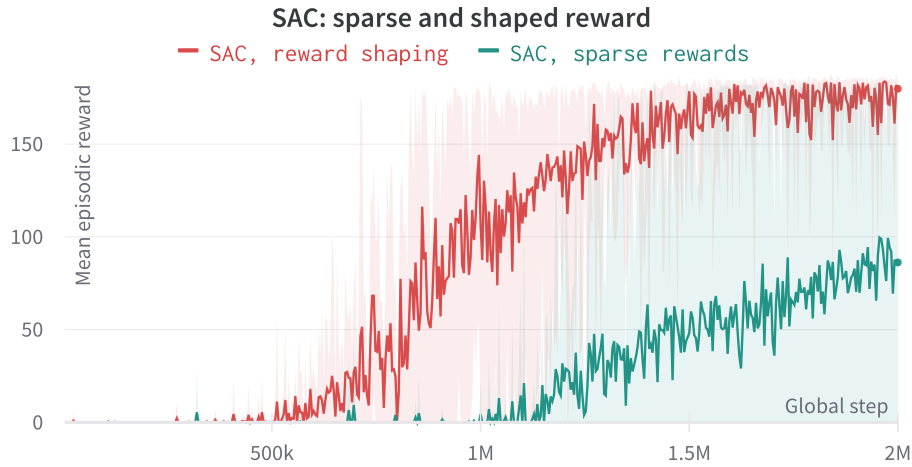



**Figure 4.2.:** Comparison of SAC performance using a sparse and shaped reward signal. The bold line is the mean of 5 runs, and the shaded area spans over the maximum and minimum of the runs.

Figure 4.3 shows the performance of 5 instances of SAC on the environment with shaped reward. Figure 4.1, row (b) shows a visualization of a trained algorithm's behavior.

Figure 4.4 shows the performance of the five instances of SAC trained with sparse rewards. A visualization of the behavior of a trained algorithm is shown in figure 4.1, row (a).

### 4.2.2. PPO

Figure 4.5 shows the performance of PPO with shaped and dense rewards. We can see that, in general, the algorithm learns the task more quickly using reward shaping than sparse reward; however, it suffers from high variability in both settings due to some instances not learning the task. The following sections will discuss the different scenarios.

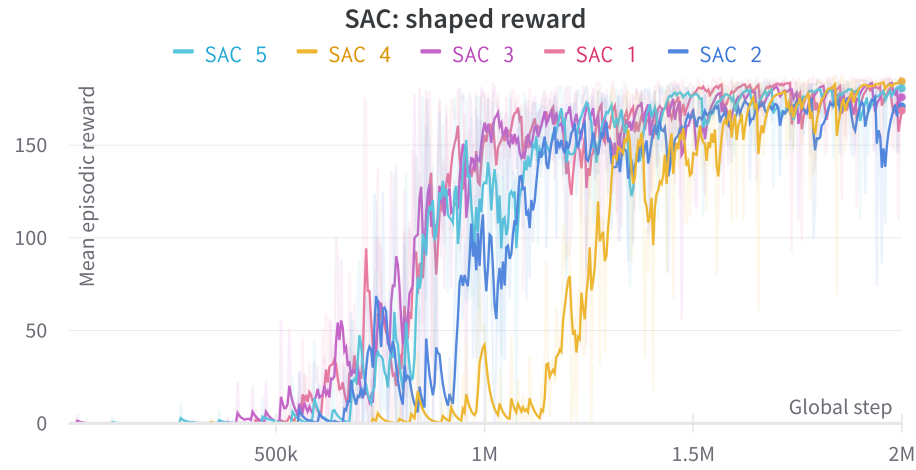



**Figure 4.3.:** Evaluation performance on a sparse reward environment of SAC instances trained in a shaped reward environment. For improved visibility, smoothing of the graphs is used used. The faint lines represent the original data.

#### Failure to learn from shaped reward

Figure 4.6 shows the evaluation of five different instances of PPO, trained with shaped reward, but evaluated with sparse rewards. We observe that two of the five instances, 5 and 2, fail to learn the task.

Figure 4.7 shows the shaped reward during exploration for the different runs. Here, the same runs that failed at the sparse evaluation plateau at a lower reward appear to be stuck in suboptimal behavior. The behavior of such an instance is shown in figure 4.1, row (c).

#### Failure to learn from shaped reward

Figure 4.8 shows the individual performance of 5 instances of PPO trained with a sparse reward signal.

## 4.3. Observation space

To study the effect of the observation space, we run a study of PPO with the two different observation spaces described in section 4.1. Because of computational restrictions, we set the algorithm to collect 1 million timesteps in the environment. In both cases, the algorithms are rewarded using a shaped reward signal.

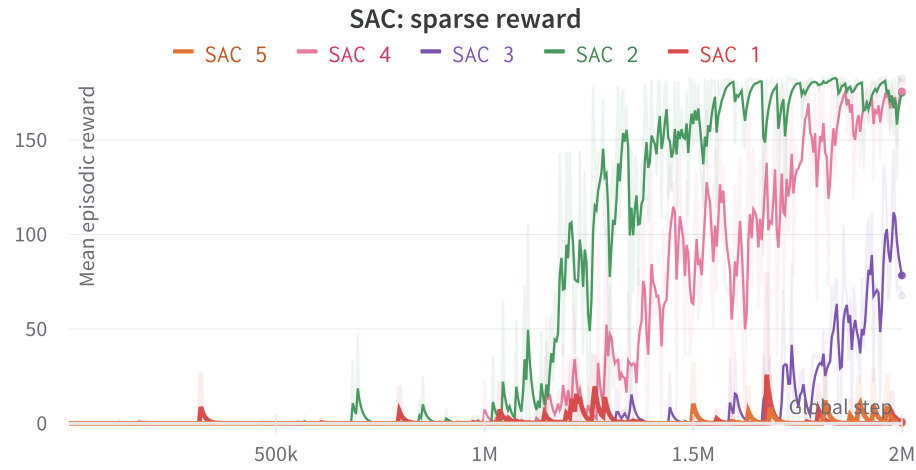



**Figure 4.4.:** Evaluation performance on a sparse-reward environment of SAC-instances trained on a sparse reward environment. For improved visibility, smoothing of the graphs is used. The shaded lines represent the original data. Instances 1 and 5 both fail to learn the task.

The policy using image-observation could not be trained in parallel environments, extending the training time drastically: Training the policy took about 100 hours, whereas the parallel training of non-image-based policies took 45 minutes. Storage is also a restriction, as the buffers add up to hundreds of gigabytes using SAC. Therefore, the experiments are run using PPO, an off-policy algorithm that does not require a large buffer.

Figure 4.9, showing the reward during exploration, illustrates the comparative performance of the two different sets of observation modalities. Due to computational constraints, only one run trained with RGB-D observations is shown. The average performance of two runs using the alternative observations is shown as a comparison. As opposed to the other experiments, episode length here is 1000, meaning that the maximum total reward is $\approx 1600$.

Figure 4.10 shows images from an episode with an agent trained with the two different observation modalities. For the policy observing RGB-D images, the policy parameters at 450 000 steps of the algorithm is used, as this is near the highest point of the rewards during training. The latest parameters are used for the policies observing the combined modalities, as they are also the best-performing.

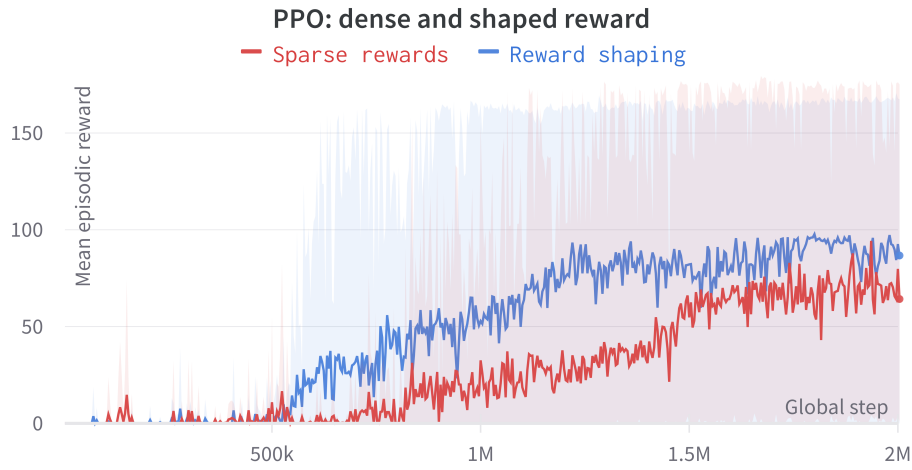



**Figure 4.5.:** Comparison of PPO performance using sparse and shaped reward. The bold line is the mean of 5 runs, and the shaded area spans over the maximum and minimum of the runs.

## 4.4. Comparison of SAC and PPO

Figure 4.11 shows the rewards of SAC and PPO during training. The algorithms are trained in the lift environment with shaped reward and the combination of modalities such as robot configuration and object pose. The hyperparameters are listed in appendix B.

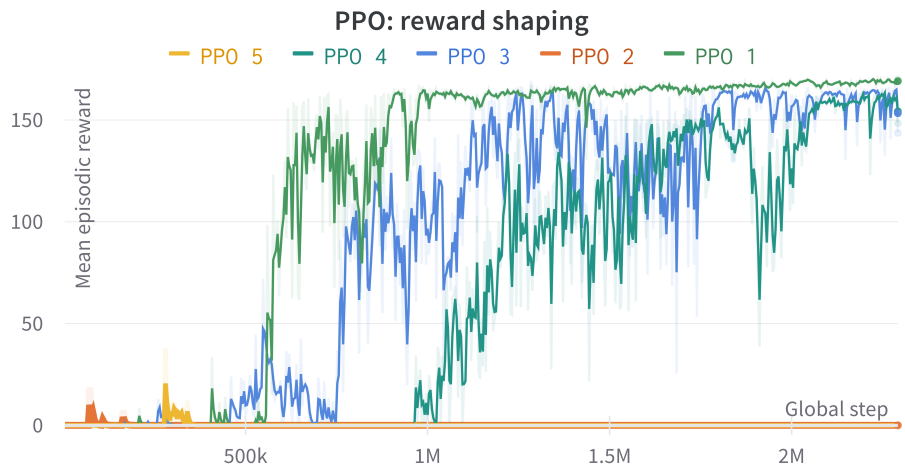



**Figure 4.6.:** Evaluation performance on a sparse-reward environment of PPO-instances trained on a shaped reward environment. For improved visibility, smoothing of the graphs is used used. The faint lines represent the original data. PPO instances 5 and 2 fail to learn the task and receive no reward after 500 steps.

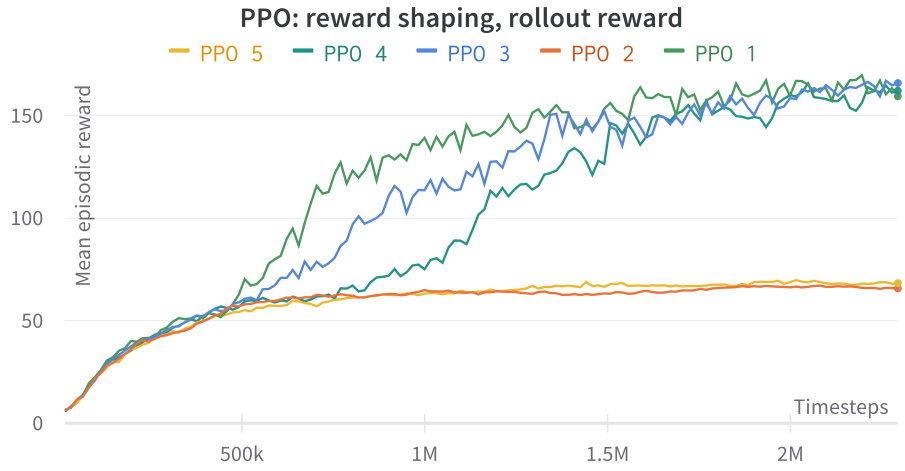



**Figure 4.7.:** Rollout reward of PPO-instances trained in a shaped reward environment. This is the reward that the algorithm receives during exploration, as opposed to the evaluation reward that is shown in other plots.

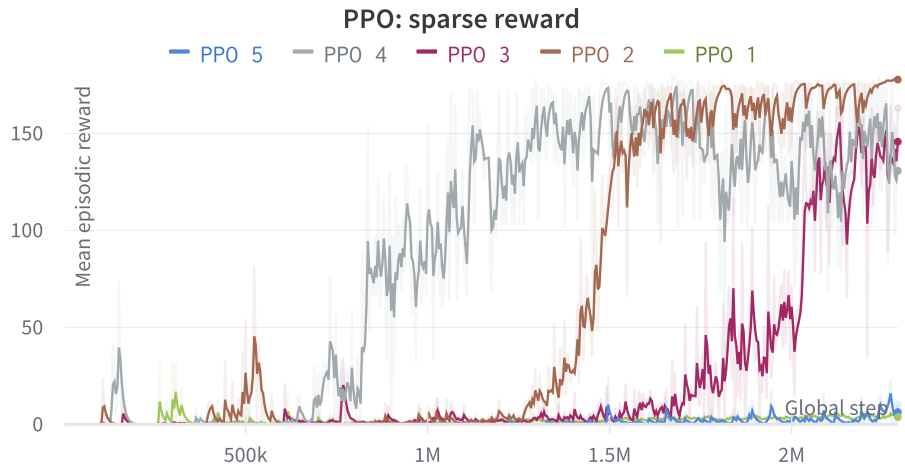



**Figure 4.8.:** Evaluation performance on a sparse-reward environment of PPO-instances trained in a sparse reward environment. For improved visibility, smoothing of the graphs is used. The shaded lines represent the original data. Instances 1 and 5 both fail to learn the task

.

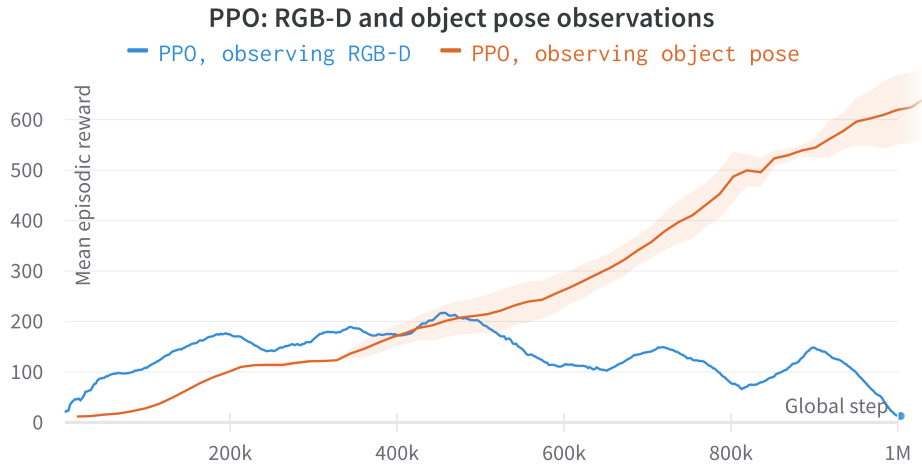



**Figure 4.9.:** Comparing PPO performance using RGB-D as opposed to robot configuration and cube pose as observations.

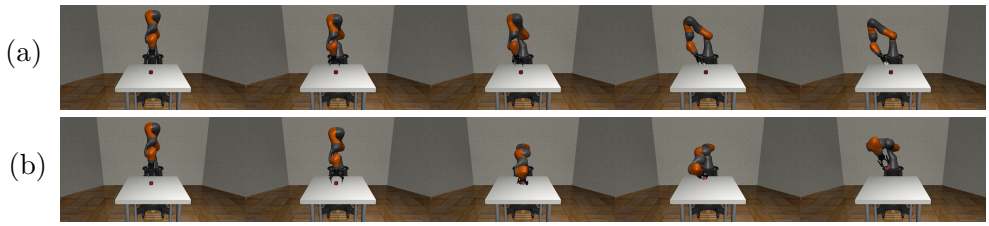



**Figure 4.10.:** Row (a) shows pictures of an episode with PPO trained with RGB-D observations. Row (b) shows the behavior of PPO trained with a combination of different modalities, such as robot configuration and cube pose.

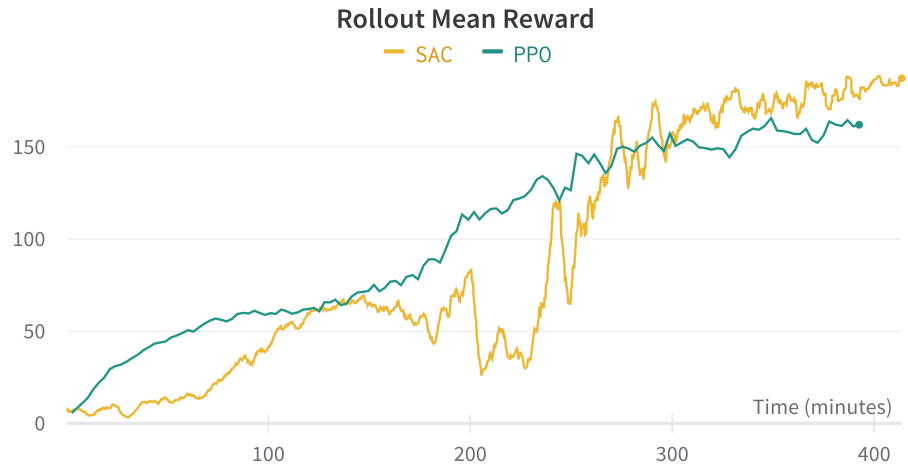



**Figure 4.11.:** Learning curves for an instance of both SAC and PPO. The horizontal axis is the run-time of the training-process

# Chapter 5.

# Discussion

This discussion will consider the problem of robotic grasping and the broader manipulation goal stated in the introduction. It will first discuss the results of the experiments in chapter 4. It will then go on to present some of the limitations of the experiments. Then, it will turn to the related works and their results and limitations in robotic grasping. Finally, it will discuss these findings in the context of general robotic manipulation.

## 5.1. Discussion of the experiment results

This section will discuss the results presented in the chapter 4 on the experiments. It will discuss the effect of different observations and reward signals provided. Finally, it will compare the performance of PPO and SAC and discuss the behavior of a policy trained with SAC.

### 5.1.1. Observation space experiment

The observation space is a large part of the complexity of a manipulation problem. This fact is suggested by the comparative experiment done in section 4.3. We see that the two instances of PPO observing modalities such as robot configuration and object pose learns the task with low variance and rise steadily in reward from the beginning. Although the learning curve is still rising at 1 million timesteps, we observe in figure 4.10, row (a), that the policy has learned a behavior sufficient to grasp the cube.

In comparison, the PPO instance trained by observing RGB-D images does not learn the task. Additionally, it degrades in performance after reaching a peak. The images of the behavior of the trained algorithm confirm that this policy does not learn the behavior, not even learning to exploit the “reaching” reward

which guides the end-effector towards the cube. This result suggests that learning suitable end-effector movements directly from RGB-D images is a significantly more complex task than learning using the other modalities.

The results might also suggest that knowledge of the cube pose simplifies the problem significantly. Intuitively, a scripted policy using this information is far more straightforward compared to using RGB-D images. In the simplest case, we might program the following steps: Move the gripper directly above the cube, lower the gripper, close the gripper, and lift.

Looking to the related work, we have discussed QT-Opt, which learns to grasp using solely RGB observations. In addition, the method can grasp unseen objects, a task far more complicated than the one considered in these experiments. However, in the experiments, the PPO algorithm fails to learn the task using RGB-D observations. There are many reasons for this gap between the success of the experiments, and we theorize that essential factors are the dataset quality, the network's architecture, and the training duration. The PPO algorithm in the experiments only experiences unsuccessful episodes, only performs 500 gradient steps, and no specific study has been performed on architecture design. QT-Opt, in comparison, trains for 15M steps and has a complex architecture for the Q-function approximator.

### 5.1.2. Reward shaping experiment

This subsection will discuss the results from the ablation studies conducted with sparse reward. It will first discuss the results from the training with sparse reward, then with reward shaping.

#### Sparse rewards

A typical failure case for both algorithms is to fail to learn the task in the sparse reward setting. See figure 4.4 and 4.8. For both algorithms, 2 of 5 instances fail to learn the task. This demonstrates the fundamental problem of sparse rewards, presented as a general challenge of robotic manipulation problems in the introduction. Each actor collects 2 million transitions in the simulator, hinting that the successful sequence of actions is stumbled upon by chance rather than sequential progress.

Empirically, we see that once high-rewarding action sequences are found, the algorithm manages to learn the task. Instances that achieve an episodic reward over 50 seldom degrade in performance drastically; however, they often undergo a jump from low rewards to near-optimal rewards. This effect of sudden learning

can be likened to learning the skill, and the results show that this is reliant on the actor experiencing episodes with higher rewards.

#### Reward shaping

The reward signal during training impacts how fast the algorithm can learn the task. This is shown in the experiments, where both algorithms learn faster on average in the shaped reward environment. This is seen in figure 4.2 and figure 4.5 showing the mean episodic reward in the evaluation environment during training. This is especially clear for SAC, as training on average results in higher reward behavior starting from around 500 algorithm steps, whereas this happens after 1M steps for the sparse reward setting.

#### Exploitation of shaped reward

Another important observation is that the average PPO performance lies substantially lower ($\approx 100$) than that of SAC ($\approx 175$). This deviation is explained by further investigating the separate runs of the PPO algorithm. We see in figure 4.6 that two of the instances fail to learn the task altogether. Looking at figure 4.7, which shows the episodic reward during exploration, we observe that these instances plateau on a lower level ($\approx 60$). This indicates that the algorithm learns to maximize a part of the dense reward unrelated to grasping.

Figure 4.1, row (c), shows the behavior of the policies that fail to learn the task while being trained with dense rewards. We see that the policy learns to position the gripper around the object but fails to learn to grasp the object and pick it up. We suspect that the policy learns to maximize the component of the shaped reward corresponding to reaching. It manages to minimize the distance between the gripper and the object; however, it does not explore the opportunity to close the gripper and receive a higher reward. Ibarz et al. [14] mention the same issue when discussing reward shaping and noting that it might lead to exploitation behavior. This behavior might be mitigated by tuning the weighting of this component of the shaped reward; however, Ibarz et al. point out that this can be challenging.

### 5.1.3. Comparison of SAC and PPO

Figure 4.11, showing the learning curve of an instance of each algorithm, reveals interesting characteristics. We noted that SAC typically has a more turbulent exploration reward during training, having more spikes and sudden drops. On the other hand, PPO tends to increase smoothly before stabilizing on a fixed reward level.

This coincides with the inner workings of the algorithms, as PPO aims to keep the updated policy close to the previous, resulting in a steady growth of the exploration reward. On the other hand, SAC uses a large replay buffer (500 episodes) to update its policy and shift it more quickly. An interesting parallel is a dichotomy between trust regions and line search, described in section 2.5 on optimization techniques. SAC, as it samples randomly from a buffer and takes a gradient step, resembles the line search algorithm stochastic gradient descent. PPO resembles trust-region methods, as it uses an on-policy replay buffer and aims to find the optimum within a region by doing several epochs on the same data.

Looking at the experiments done in the dense reward setting, we observe that PPO tends to get stuck at a lower-reward plateau, whereas SAC consistently manages to solve the task. It is evident that SAC inherits more exploration and is resilient to getting stuck in suboptimal behavior. While a study of hyperparameters, such as learning rate and reward scale is needed to conclude, these results indicate that the off-policy learning of SAC might help pull it out of local maxima and consistently discover the optimal behavior.

### 5.1.4. Resulting behavior

Looking at the agent's behavior in the experiment, we can see that it has significant variance and, in some cases, have an arguably suboptimal grasping strategy.

Intuitively, an optimal tactic to grasp the object is to lower the gripper vertically, close the gripper, and raise the box. In this case, no rotation around the horizontal axes is needed. Examining row (a) of figure 4.1, we see that the resulting policy learns to grasp in a way that rotates greatly before grasping the object.

This behavior results from the sparse rewards, and the agent is purely incentivized to lift the cube. If there is an interest in limiting unnecessary movement, such as the rotation mentioned above, this might be incorporated into the reward. However, incorporating an energy-term might lead to additional challenges. In the same way that policies might fail to learn the task given a shaped reward, penalizing actuation might result in the policy exploiting this term. For example, the agent might only learn to stand still, as moving will result in a penalty.

## 5.2. Limitations of the experiments

This section will list some of the limitations of the experiments, considering the application of deep RL algorithms to solve real-world grasping tasks.

### 5.2.1. Generalization

The experiments consider the case of one fixed object, specifically a red cube. They do not explore the challenges of learning to grasp several objects and extend that knowledge to grasping unseen ones. However, the environment does initialize the cube position randomly while keeping the rotation fixed, requiring the agent to generalize over the object position within a small range.

### 5.2.2. Simulated

The experiments are conducted in Mujoco, a simulated environment, where the physics of the system is computed using numerical methods. Ideally, the algorithms would be evaluated in the real world, and there might be significant discrepancies in the performance of the algorithms when deployed on a similar setup in real life.

However, several techniques exist to transfer algorithms trained in simulations to act on physical setups, proving that results in the simulation are relevant to solving real-world problems. Techniques such as domain randomization and random force injection can train policies to be deployed directly to a physical setup. [1] [52] Nevertheless, this usually entails the need for more experience as the policy needs to generalize over a more stochastic environment.

### 5.2.3. Hyperparameters and architectures

The experiments do not sweep over all hyperparameters. Even though SAC and PPO are designed to be easy to tune, parameters of the environment could favor the performance of one algorithm above another. For example, a different weighting of the terms in the shaped reward might have enabled PPO to learn the task more reliably.

The experiments are restricted to two considering architectures, namely "MlpPolicy" and "CnnPolicy", two options offered by Stable Baselines 3 [34]. Different architectures might result in faster learning, especially in the case of RGB-D observations.

The experiments are also restricted to two sets of observations, and a more thorough study might support the claim that observing object pose seems to simplify the problem significantly. Additionally, exploring different algorithms might reveal how image observations and proprioceptive information might be combined to ease the learning of the task when the object pose is unknown.

### 5.2.4. Choice of algorithms

As stated in the introduction, SAC and PPO were chosen as they are available open-source through Stable Baselines 3 [34]. However, they differ from the algorithms discussed in the related works, and these differences might give rise to gaps in performance. In other words, the experiments are not exhaustive in showing the capabilities of model-free RL algorithms.

Both PPO and SAC are Actor-Critic style algorithms, meaning that they are inspired by policy iteration, where the policy (actor) is updated using a value function approximator (critic). QT-Opt is based on Q-learning, where the policy is not explicitly defined but implicitly by maximizing a Q-function.

This difference in the Actor-Critic and Q-learning formulation might bring different advantages and challenges; however, the experiments reveal challenges common to all model-free RL algorithms, such as a dependency on data, observations, and action interface and reward signal.

## 5.3. In the context of robotic grasping

This section will discuss the results from related works and the experiments in the light of robotic grasping and present emerging challenges.

### 5.3.1. Comparison of QT-Opt and Dex-Net

Two grasping methods were explored in the section on related work: QT-Opt and Dex-Net. While they achieve a comparable grasp success, the methods differ substantially. Kleeberger et al. [19] categorize Dex-Net as using supervised learning, as it turns the problem of grasping into predicting grasp success for a candidate grasp. Therefore, the exploration problem arising with deep RL is instead converted to constructing a sufficiently large and diverse dataset.

QT-Opt is characterized as an RL algorithm, as it formulates the problem of grasping as a multi-stage MDP. It also incorporates the challenges of exploration and utilizes sparse rewards.

The different approaches result in highly different behavior, which is not revealed by inspecting their grasp success rate. QT-Opt can perform sequences of action within a single grasp, allowing for sequences of non-prehensile actions. Due to their approach, Dex-Net is incapable of sequences of actions within a single grasp attempt.

### 5.3.2. Data collection

QT-Opt inherits a restriction based on the need for a large and diverse dataset. In their work, Kalashnikov et al. use a dataset of 580k grasp attempts, collected mainly autonomously. Still, this dependence upon a large dataset limits the method from being applied to tasks where data collection can not be done efficiently. Haarnoja et al., from their experiments with SAC for locomotion on a physical quadruped, report being unable to capture a large amount of data due to the time-consuming collection process. [8]

The vital need for data is further exemplified by the results from the experiments, where the algorithms need to collect around 1 million timesteps before solving the grasping task. Moreover, this is in the setting where the algorithm observes perfect pose information of the object to grasp. To illustrate the timescales of conducting a similar setup in the real world: Assuming that one timestep corresponds to 1 second in real-time, 1 million timesteps corresponds to $\approx$ 28 hours of effective training time, neglecting the setup needed between episodes.

As shown in work on QT-Opt, data collection can be done largely autonomously in the case of robotic grasping; however, solving tasks where this is not necessarily possible calls for different techniques.

#### Collection of useful data

An important specification of data dependence is the dependence on valuable data, often successful episodes by a scripted policy or human demonstrations.

As discussed in the experiments, specifically regarding the experiments using sparse rewards, we observe that the algorithms quickly learn the task when the first successful episodes are collected. This benefit of experiencing reward is indicated by the shift from collecting purely low-reward episodes to consistently performing high-reward grasps. This hints that having access to valuable data is a trigger to allow the algorithm to learn the task.

Kalashnikov et al. [15] report that they have to use a scripted policy to initially collect useful episodes. Andrychowicz et al. [1] report resorting to similar methods, starting half of the episodes in a state where the gripper grasps the object.

### 5.3.3. Semantics in grasping

Both algorithms aim to grasp objects successfully; however, they do not allow choosing which object to pick up next. In both cases, when the algorithm picks up an object, it does not know what it might be. Furthermore, they cannot search for a pre-specified object in a heap of objects.

While not explicitly stated, both algorithms presumably grasp the objects with the same force. This restricts the application to objects where a light touch is necessary.

### 5.3.4. Dealing with sparse rewards

As argued by Ibarz et al. [14], specifying the sparse rewards is often the most natural way to specify a manipulation problem. This is also the case for grasping tasks where a sparse reward signal introduces no bias to choosing behaviors that might be suboptimal.

In the experiments, the bias towards sub-optimal behavior induced by reward shaping, is exemplified. In the discussion of the experiments using reward shaping, PPO fails to learn the task in some instances, instead learning to maximize a term of the shaped reward that does not correspond to grasping and lifting the object.

Hindsight Experience Replay, presented in section 3.8, offers a framework for dealing more efficiently with sparse rewards. Although their method drastically increases the performance of the DDPG algorithm, it requires around 1.6 million timesteps in the environment to solve the pick-and-place task, which is in many ways similar to the grasping problem of the experiments in chapter 4. This shows that HER, too, relies on a large dataset that might be impractical to collect in the real world.

### 5.3.5. Observation space

As seen through the experiments and the related work, the choice of observations available to the agent heavily influences the complexity of grasping. Therefore, using model-free RL algorithms on a physical robot demands considerations of what observations are given to the agent. For some setups, this might limit the practicality of deploying such algorithms.

In the discussion of the experiments regarding the observation space, we see that changing the observation space from RGB-D images to a combination of different modalities such as object pose and robot configuration drastically changes the algorithm's performance.

In their experiments, Andrychowicz et al. [1] offer similar observations to the algorithm, using a CNN to infer the object pose using RGB-D images and feeding this estimate to the trained actor.

QT-Opt and DexNet 4.0, however, depend only on depth image or an RGB image of the scene as observations, respectively. They reveal that learning to grasp

directly from image or depth observations is possible given a large enough dataset and a network of sufficient complexity.

In a robotic grasping system, several observation spaces are possible. We could construct a policy that can infer grasps from raw input, such as depth or images. This approach is often referred to as learning end-to-end [14]. Alternatively, the policy could receive observations that are processed, such as shown in work by Andrychowicz et al. [1] on HER, where they use an estimate of the pose of the object with a separate CNN. Such a preprocessing of the images can be connected to the "thinking" stage between the "seeing" and "acting" in figure 1.1 illustrating the see-think-act cycle. In the same manner that Ziegward et al. mention different stages of the "thinking" stage, such as building a map, steps to extract useful information such as object pose can be extracted by a separate algorithm at this stage.

The experiments show that providing object pose as an observation to the agent is highly beneficial to accelerate the learning using SAC and PPO. Therefore, an approach similar to that of Andrychowicz et al. [1] is a natural approach to implementing a grasping solution on a physical manipulation setup, similar to the system described in appendix A.

### 5.3.6. Action space

The action space offered to the actor also dramatically impacts the task's difficulty and affects the possibility of learning to solve the grasping task on a given physical system.

Experiments by Zhu et al. [54] show that the performance of the SAC algorithm is substantially higher when given direct control of the end-effector pose. Similar action space is also used in the setting of QT-Opt, and Dex-Net 4.0. Additionally, QT-Opt restricts the action space further, as the only rotation allowed is about the vertical axis.

The results from Zhu et al. might be explained by the fact that grasping is significantly simplified when both actions and observations are defined as task space poses. Moving toward the object to be grasped, for example, is solved by setting the gripper pose equal to the object pose, perhaps with some offset. Solving this problem by acting directly on the joints complicates this task significantly.

Setting task-space poses directly is an interface available at the large majority of robotic manipulators and is, therefore, a realistic formulation of the action space. Directly controlling the end-effector pose can, in some cases, cause unwanted behavior, such as singularities. It can also have ambiguities, as and-end effector pose might be reached through several robot configurations. However, this can

be circumvented by assuming that the set of poses that the actor can choose is well outside the configuration space where these problems arise.

## 5.4. In the context of robotic manipulation

This discussion will present a discussion on the related work and the experiments from the perspective of the general robotic manipulation problem.

### 5.4.1. From robotic grasping to general manipulation

The algorithms for robotic grasping explored in the related works vary in relevance to general robotic manipulation.

Dex-Net 4.0 constructs the database using an analytic model to calculate the grasp stability. This limits the method to the case of robotic grasping. However, it might be an inspiration for methods incorporating knowledge of the mechanics to solve the manipulation problem.

The HER technique can prove helpful in many manipulation problems, as sparse rewards can be specified easily. However, the method is restricted to tasks that can be specified in a multi-goal format. While this is highly relevant for tasks such as aiming for a goal, like throwing a dart or curling, many tasks are rewarded in a binary manner. For example, at what angle the door is opened is of little concern; the fact that it is opened is the main objective.

However, QT-Opt, SAC, and PPO are fundamentally model-free and can be applied to other manipulation tasks. Still, some aspects limit their application to general manipulation problems.

#### Collection of useful data

As seen in the experiments, as well as the results of QT-Opt, these model-free algorithms are dependent on a large set of successful episodes to learn the tasks. QT-Opt solves this problem partly by using scripted policies and partly with large-scale data collection.

Even though scripted policies might suffice to collect valuable data, this dependence on demonstrations can restrict model-free methods when considering general manipulation tasks. Consequently, they will not be able to efficiently solve tasks that cannot be solved by a scripted policy initially. Alternatively, human demonstrations can also be used; however, this might be elusive for some tasks.

### 5.4.2. Observations

The lessons learned from the experiments with different observations extend to general manipulation. Specific modalities are highly informative and help the agent learn the task quickly. The experiments show that knowledge of the object state and the robot configuration has this effect. Helpful representations might change from problem to problem; however, knowledge of the poses of different objects in the scene might be valuable in most manipulation tasks.

How informative the observations are, is restricted by the sensors. However, separating the task of extracting useful information from sensor data can also provide more informative observations to the agent. For example, using a pose-estimator as discussed for robotic grasping might be relevant to a large set of manipulation problems.

### 5.4.3. Actions

As discussed for robotic grasping, the agent learns the task more quickly when directly controlling the end-effector pose in the task space. This fact might extend to several manipulation problems for the same reasons given. The action of approaching an object is greatly simplified when formulating observations and actions as task-space poses.

While this is a natural formulation for robotic manipulators, such as described in the setup of appendix A, this does not extend to all physical robots. The popular quadruped platform used for robotic dogs is often controlled at the joint level. Other examples are articulated grippers, such as humanoid hands, which do not have a single end-effector to put in a desired pose. However, setting the fingertips in the task space is a possibility.

# Chapter 6.

# Conclusions and further work

As stated in the problem description, this thesis explored the capabilities and limitations of model-free RL for robotic grasping, with the motivation of solving the general manipulation problem. The experiments applied the model-free deep RL methods SAC and PPO to pick up a box in a simulated setup with a robotic manipulator, similar to the setup described in appendix A. The discussion examined the results from both related works and the experiments and highlighted emerging challenges of model-free RL for grasping and general manipulation.

Considering the discussion of the previous chapter, we can make the following conclusions:

- Model-free deep RL algorithms provide a framework for solving a large set of tasks with general algorithms; however, some significant challenges limit a practical application to all manipulation problems.
- Given the sufficient state information, open-source implementations of model-free algorithms can be used to solve a grasping problem in simulation without task-specific refinement.
- Open source implementations of model-free deep RL algorithms learn the task significantly faster when observing object pose and robot configuration rather than RGB-D images.
- Reward shaping accelerates the learning process for both SAC and PPO.
- Our experiments hint that SAC is robust to exploiting shaped reward, as it solved the task 5 out of 5 times. In contrast, PPO solved it 3 out 5 times.
- An emergent solution for performing a grasping task on a physical setup is to separate the concern of object pose estimation to a separate process, such as a CNN.

## 6.1. Further research

This section will identify areas of future research in light of the observations this thesis has made.

### 6.1.1. Deployment on an actual setup

The experiments were exclusively done in a simulated setup, and deploying algorithms on a physical setup calls for solutions to the sim-to-real gap. Comparing existing methods, such as domain randomization and random force injection [52], and further improving them will impact how practical algorithms are on real robotic systems.

### 6.1.2. Policy architectures

The experiments in this report did not explore potential architectures extensively, and further research on the benefits of different policy architectures might improve the performance.

Some architectures allow the network to build a memory of previous inputs and outputs, especially architectures used for natural language processing (NLP). Policies can benefit from this in many manipulation tasks. For example, digging through a heap for an object might benefit from knowing where the agent has looked previously. Another example is opening a locked door, where the agent has to unlock the door before opening the handle. With a direct mapping of visual inputs to actions, the algorithm cannot infer whether it has unlocked the door when it sees an image of the handle.

Exploring different architectures allows for a combination of different sensor modalities. In the experiments, a fully convolutional network is used for the RGB-D observations, and a fully connected network is used for the one-dimensional observations. Forming an architecture that applies convolutional operations on the images before merging the result with one-dimensional observations will allow for using all modalities and might improve performance.

### 6.1.3. Different optimization techniques

The methods discussed in this thesis have been confined to using variations of gradient descent to find the optimal policy. However, other optimization techniques can have different advantages, some of which can be highly fruitful in deep RL. For example, evolution-based methods. [48] [17]

### 6.1.4. Improving observations

While pose estimation is highly advantageous when manipulating rigid objects, this is not sufficient when the tasks include soft materials or liquids. In these scenarios, learning a scene representation from images using autoencoders might provide more informative observations to the agent.[4]

### 6.1.5. Sophisticated control

The experiments have assumed that the agent directly allows for commanding the end-effector position. However, how this is achieved is not explored. When contact between the environment and the end-effector occurs, the force exerted by the end-effector on the environment depends on the controller's stiffness. A correct controller design is necessary to ensure the safety of the physical robot and its surroundings. Additionally, challenges such as grasping fragile objects call for more sophisticated controllers than discussed here. Allowing the algorithm to control stiffness parameters or the torques on the system directly might allow higher safety and more sensitive manipulation.

### 6.1.6. Accelerate the collection of experience

Experiments show that the ability to parallelize environments has a massive effect on the training time, often cutting the runtimes in orders of magnitude. Exploring simulators that easily allow for generating realistic experiences from simulated parallel environments could have significant benefits in the efficiency of generating data and allow algorithms to learn more complicated tasks in a shorter amount of wall time.

### 6.1.7. Demonstrations and multi-goal learning

The discussion of sparse rewards in section 5.1.2 expresses that experiencing successful episodes primes the learning of the task. An evident challenge is to quickly guide the algorithm to fruitful action sequences in a sparse reward setting. Exploring techniques dealing with this problem, such as learning from demonstrations, is a clear direction to explore to improve algorithms for manipulation problems. Other directions are multi-goal learning, as demonstrated by HER, and inverse RL, where the reward function is learned from demonstrations. [14]

### 6.1.8. Predicting uncertainty

To help the collecting diverse datasets, allowing the policy to express uncertainty might be crucial. In machine learning, it is helpful to differentiate between

aleatoric and epistemic uncertainty in a model. They can be briefly described as [12] :

- Aleatoric (or statistical) uncertainty: Irreducible noise inherent to the process. For example, a dice has a uniform distribution of landing on all six sides. This randomness is irreducible, and more data on experiments with the dice will not decrease this uncertainty.
- Epistemic uncertainty: Uncertainty about the model due to lack of data. Consider a coin a weighting, making the probability of heads equal to 1. Experiments will reveal what the coin always will show; however, before experimenting, there is uncertainty about whether the coin will show heads or tails. This uncertainty is reducible by collecting more data.

The aleatoric uncertainty can be likened to the uncertainty that maximum entropy learning aims to nurture in the policies, as described in 2.3. Many robotic tasks, including grasping, are solved in various ways, and having this uncertainty in the policy is proven beneficial. For example, having the notion of multiple ways to grasp an object makes the policy more robust when it is forced to grasp the object in another way than most optimal.

Extending to robot grasping, we can connect the epistemic uncertainty to the random exploration of the model-free algorithms in the sparse reward environment. The uncertainty of what actions to take to obtain the sparse reward is highly reducible; one successful episode will limit the action-sequence space to be explored significantly.

As in machine learning, it is helpful to signify when models predict with significant uncertainty due to lack of data or inherent noise. Extending this to solving manipulation tasks, devising algorithms that can signify the epistemic uncertainty, and combining this with guiding in the form of shaped rewards or demonstrations might eliminate the need for highly random exploration that model-free algorithms are forced to do in sparse reward settings.

### 6.1.9. Estimating a model

Model-free algorithms start with highly random exploration; however, with more experience, they can seemingly learn complex behavior without forming an explicit model. Their lack of a model makes them highly bias-free; however, the high variance is visible in a large amount of exploration before discovering fruitful behavior.

On the other hand, having a model can allow for finding an optimal policy using no exploration. Given an MDP of the problem, policy- or value iteration can find

the highest performing policy purely by executing computations using the model. However, a fully model-based approach can suffer from discrepancies between the model and the real world, as modeling the simulator precisely as the real world is a nearly impossible task. Therefore model-based methods can be seen as having no variance but a significant bias in following the model.

Further research can follow this reasoning to strike a balance between model-free methods and model-based methods. As mentioned by Ibarz et al. [14], several successful works have been done where a model is learned through large datasets and is then used to choose actions. Including epistemic uncertainty allows for developing models that can express where they need more data. This can work towards obtaining robust models for an extensive range of scenarios.

# References


[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *arXiv:1707.01495 [cs]* (Feb. 23, 2018). arXiv: 1707.01495. URL: http://arxiv.org/abs/1707.01495 (visited on 04/26/2022).

[2] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 679–698. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1986.4767851.

[3] Michael Danielczuk, Jeffrey Mahler, Chris Correa, and Ken Goldberg. "Linear Push Policies to Increase Grasp Access for Robot Bin Picking". In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE). Munich, Germany: IEEE, Aug. 2018, pp. 1249–1256. ISBN: 978-1-5386-3593-3. DOI: 10.1109/COASE.2018.8560406. URL: https://ieeexplore.ieee.org/document/8560406/ (visited on 06/06/2022).

[4] Chelsea Finn and Sergey Levine. *Deep Visual Foresight for Planning Robot Motion*. arXiv:1610.00696. type: article. arXiv, Mar. 12, 2017. DOI: 10.48550/arXiv.1610.00696. arXiv: 1610.00696[cs]. URL: http://arxiv.org/abs/1610.00696 (visited on 06/04/2022).

[5] *Global 500*. Fortune. URL: https://fortune.com/global500/2021/ (visited on 05/14/2022).

[6] Larry Greenemeier. *Fukushima Disaster Inspires Better Emergency-Response Robots*. Scientific American. URL: https://www.scientificamerican.com/article/fukushima-disaster-inspires-better-emergency-response-robots/ (visited on 05/13/2022).

[7] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. *Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates*. arXiv:1610.00633. type: article. arXiv, Nov. 23, 2016. DOI: 10.48550/arXiv.1610.00633. arXiv: 1610.00633[cs]. URL: http://arxiv.org/abs/1610.00633 (visited on 05/28/2022).

[8] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. *Learning to Walk via Deep Reinforcement Learning*. arXiv:1812.11103. type: article. arXiv, June 19, 2019. DOI: 10.48550/arXiv.1812.11103. arXiv: 1812.11103[cs,stat]. URL: http://arxiv.org/abs/1812.11103 (visited on 06/01/2022).

[9] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. "Reinforcement Learning with Deep Energy-Based Policies". In: *arXiv:1702.08165 [cs]* (July 21, 2017). arXiv: 1702.08165. URL: http://arxiv.org/abs/1702.08165 (visited on 05/11/2022).

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *arXiv:1801.01290 [cs, stat]* (Aug. 8, 2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290 (visited on 04/26/2022).

[11] *How Amazon Makes Money*. Investopedia. URL: https://www.investopedia.com/how-amazon-makes-money-4587523 (visited on 05/14/2022).

[12] Eyke Hüllermeier and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods". In: *Machine Learning* 110.3 (Mar. 1, 2021), pp. 457–506. ISSN: 1573-0565. DOI: 10.1007/s10994-021-05946-3. URL: https://doi.org/10.1007/s10994-021-05946-3 (visited on 06/04/2022).

[13] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (Jan. 30, 2019), eaau5872. ISSN: 2470-9476. DOI: 10.1126/scirobotics.aau5872. arXiv: 1901.08652[cs,stat]. URL: http://arxiv.org/abs/1901.08652 (visited on 06/04/2022).

[14] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. "How to Train Your Robot with Deep Reinforcement Learning; Lessons We've Learned". In: *The International Journal of Robotics Research* 40.4 (Apr. 2021), pp. 698–721. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364920987859.

arXiv: 2102.02915.
URL: http://arxiv.org/abs/2102.02915 (visited on 05/11/2022).

[15] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *arXiv:1806.10293 [cs, stat]* (Nov. 27, 2018). arXiv: 1806.10293.
URL: http://arxiv.org/abs/1806.10293 (visited on 05/11/2022).

[16] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. *Deep Drone Acrobatics*. arXiv:2006.05768. type: article. arXiv, June 11, 2020.
DOI: 10.48550/arXiv.2006.05768. arXiv: 2006.05768[cs].
URL: http://arxiv.org/abs/2006.05768 (visited on 06/05/2022).

[17] Shauharda Khadka and Kagan Tumer. *Evolution-Guided Policy Gradient in Reinforcement Learning*. arXiv:1805.07917. type: article. arXiv, Oct. 27, 2018.
DOI: 10.48550/arXiv.1805.07917. arXiv: 1805.07917[cs,stat].
URL: http://arxiv.org/abs/1805.07917 (visited on 06/06/2022).

[18] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 29, 2017). arXiv: 1412.6980.
URL: http://arxiv.org/abs/1412.6980 (visited on 05/03/2022).

[19] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F. Huber. "A Survey on Learning-Based Robotic Grasping". In: *Current Robotics Reports* 1.4 (Dec. 1, 2020), pp. 239–249.
ISSN: 2662-4087. DOI: 10.1007/s43154-020-00021-6.
URL: https://doi.org/10.1007/s43154-020-00021-6 (visited on 04/26/2022).

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 05/13/2022).

[21] *LBR iiwa*. KUKA AG.
URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa (visited on 05/19/2022).

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner.
"Gradient-based learning applied to document recognition".
In: *Proceedings of the IEEE* 86.11 (Nov. 1998). Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256.
DOI: 10.1109/5.726791.

[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra.
*Continuous control with deep reinforcement learning*. arXiv:1509.02971. type: article. arXiv, July 5, 2019. DOI: 10.48550/arXiv.1509.02971.
arXiv: 1509.02971[cs,stat].
URL: http://arxiv.org/abs/1509.02971 (visited on 06/09/2022).

[24] D.G. Lowe. "Object recognition from local scale-invariant features".
In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Proceedings of the Seventh IEEE International Conference on Computer Vision. Vol. 2. Sept. 1999, 1150–1157 vol.2.
DOI: 10.1109/ICCV.1999.790410.

[25] Jeffrey Mahler and Ken Goldberg. "Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences".
In: *Proceedings of the 1st Annual Conference on Robot Learning*. Conference on Robot Learning. ISSN: 2640-3498. PMLR, Oct. 18, 2017, pp. 515–524.
URL: https://proceedings.mlr.press/v78/mahler17a.html (visited on 06/06/2022).

[26] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg.
"Learning ambidextrous robot grasping policies".
In: *Science Robotics* 4.26 (Jan. 16, 2019). Publisher: American Association for the Advancement of Science, eaau4984.
DOI: 10.1126/scirobotics.aau4984.
URL: https://www.science.org/doi/10.1126/scirobotics.aau4984 (visited on 05/13/2022).

[27] Matthew T. Mason. "Toward Robotic Manipulation". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018). _eprint: https://doi.org/10.1146/annurev-control-060117-104848, pp. 1–28.
DOI: 10.1146/annurev-control-060117-104848.
URL: https://doi.org/10.1146/annurev-control-060117-104848 (visited on 06/02/2022).

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie,

Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis.
"Human-level control through deep reinforcement learning".
In: *Nature* 518.7540 (Feb. 2015). Number: 7540 Publisher: Nature Publishing Group, pp. 529–533. ISSN: 1476-4687.
DOI: 10.1038/nature14236.
URL: https://www.nature.com/articles/nature14236 (visited on 05/07/2022).

[29] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*.
Red. by Francis Bach.
Adaptive Computation and Machine Learning series.
Cambridge, MA, USA: MIT Press, Aug. 24, 2012. 1104 pp.
ISBN: 978-0-262-01802-9.

[30] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. 2nd ed.
Springer series in operations research. OCLC: ocm68629100.
New York: Springer, 2006. 664 pp. ISBN: 978-0-387-30303-1.

[31] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang.
*Solving Rubik's Cube with a Robot Hand*. arXiv:1910.07113. type: article.
arXiv, Oct. 15, 2019. DOI: 10.48550/arXiv.1910.07113.
arXiv: 1910.07113[cs,stat].
URL: http://arxiv.org/abs/1910.07113 (visited on 06/04/2022).

[32] Pieter Abbeel. *L1 MDPs, Exact Solution Methods, Max-ent RL (Foundations of Deep RL Series)*. Aug. 25, 2021.
URL: https://www.youtube.com/watch?v=2GwBez0D20A (visited on 05/22/2022).

[33] *Proximal Policy Optimization — Spinning Up documentation*. URL:
https://spinningup.openai.com/en/latest/algorithms/ppo.html (visited on 06/08/2022).

[34] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann.
"Stable-Baselines3: Reliable Reinforcement Learning Implementations".
In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.
ISSN: 1533-7928. URL: http://jmlr.org/papers/v22/20-1364.html (visited on 05/11/2022).

[35] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Red. by Francis Bach. Adaptive Computation and Machine Learning series. Cambridge, MA, USA: MIT Press, Nov. 23, 2005. 272 pp. ISBN: 978-0-262-18253-9.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 1, 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y (visited on 05/13/2022).

[37] A. Sahbani, S. El-Khoury, and P. Bidaud. "An overview of 3D object grasp synthesis algorithms". In: *Robotics and Autonomous Systems*. Autonomous Grasping 60.3 (Mar. 1, 2012), pp. 326–336. ISSN: 0921-8890. DOI: 10.1016/j.robot.2011.07.016. URL: https://www.sciencedirect.com/science/article/pii/S0921889011001485 (visited on 06/05/2022).

[38] Adam Satariano and Cade Metz. "A Warehouse Robot Learns to Sort Out the Tricky Stuff". In: *The New York Times* (Jan. 29, 2020). ISSN: 0362-4331. URL: https://www.nytimes.com/2020/01/29/technology/warehouse-robot.html (visited on 05/14/2022).

[39] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, June 1, 2015, pp. 1312–1320. URL: https://proceedings.mlr.press/v37/schaul15.html (visited on 06/09/2022).

[40] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv:1506.02438. type: article. arXiv, Oct. 20, 2018. DOI: 10.48550/arXiv.1506.02438. arXiv: 1506.02438[cs]. URL: http://arxiv.org/abs/1506.02438 (visited on 05/24/2022).

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms".

In: *arXiv:1707.06347 [cs]* (Aug. 28, 2017). arXiv: 1707.06347.
URL: http://arxiv.org/abs/1707.06347 (visited on 05/02/2022).

[42] Max Schwarz, Christian Lenz, Andre Rochow, Michael Schreiber, and Sven Behnke. *NimbRo Avatar: Interactive Immersive Telepresence with Force-Feedback Telemanipulation*. arXiv:2109.13772. type: article. arXiv, Sept. 28, 2021. DOI: 10.48550/arXiv.2109.13772. arXiv: 2109.13772[cs].
URL: http://arxiv.org/abs/2109.13772 (visited on 06/01/2022).

[43] Daniel Seita.
*Learning Diverse Skills via Maximum Entropy Deep Reinforcement Learning*. The Berkeley Artificial Intelligence Research Blog.
URL: http://bair.berkeley.edu/blog/2017/10/06/soft-q-learning/ (visited on 05/11/2022).

[44] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. 2nd ed. 2016. Springer Handbooks.
Cham: Springer International Publishing : Imprint: Springer, 2016.
1 online resource (LXXVI, 2227 p. 1375 illu in color.)
ISBN: 978-3-319-32552-1.

[45] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. Red. by Ronald C. Arkin. 2nd ed. Intelligent Robotics and Autonomous Agents series.
Cambridge, MA, USA: MIT Press, Feb. 18, 2011. 472 pp.
ISBN: 978-0-262-01535-6.

[46] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis.
"Mastering the game of Go with deep neural networks and tree search".
In: *Nature* 529.7587 (Jan. 2016). Number: 7587 Publisher: Nature Publishing Group, pp. 484–489. ISSN: 1476-4687.
DOI: 10.1038/nature16961.
URL: https://www.nature.com/articles/nature16961 (visited on 05/13/2022).

[47] *Soft Actor-Critic — Spinning Up documentation*. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html (visited on 05/21/2022).

[48] Kenneth O. Stanley and Risto Miikkulainen.
"Evolving Neural Networks through Augmenting Topologies".
In: *Evolutionary Computation* 10.2 (June 2002), pp. 99–127.
ISSN: 1063-6560, 1530-9304. DOI: 10.1162/106365602320169811.
URL: https://direct.mit.edu/evco/article/10/2/99-127/1123
(visited on 06/06/2022).

[49] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham.
"Can robots assemble an IKEA chair?"
In: *Science Robotics* 3.17 (Apr. 18, 2018). Publisher: American Association for the Advancement of Science, eaat6385.
DOI: 10.1126/scirobotics.aat6385.
URL: https://www.science.org/doi/10.1126/scirobotics.aat6385
(visited on 05/19/2022).

[50] Richard S. Sutton and Andrew G. Barto.
*Reinforcement Learning: An Introduction*. Red. by Francis Bach. 2nd ed.
Adaptive Computation and Machine Learning series.
Cambridge, MA, USA: A Bradford Book, Nov. 13, 2018. 552 pp.
ISBN: 978-0-262-03924-6.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa.
"MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.
ISSN: 2153-0866. Oct. 2012, pp. 5026–5033.
DOI: 10.1109/IROS.2012.6386109.

[52] Eugene Valassakis, Zihan Ding, and Edward Johns. "Crossing The Gap: A Deep Dive into Zero-Shot Sim-to-Real Transfer for Dynamics".
In: *arXiv:2008.06686 [cs]* (Aug. 15, 2020). arXiv: 2008.06686.
URL: http://arxiv.org/abs/2008.06686 (visited on 05/07/2022).

[53] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning".
In: *Machine Learning* 8.3 (May 1, 1992), pp. 229–256. ISSN: 1573-0565.
DOI: 10.1007/BF00992696.
URL: https://doi.org/10.1007/BF00992696 (visited on 06/08/2022).

[54] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín.
"robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *arXiv:2009.12293 [cs]* (Sept. 25, 2020). arXiv: 2009.12293.
URL: http://arxiv.org/abs/2009.12293 (visited on 04/28/2022).

# Appendix A.

# Description of physical robot

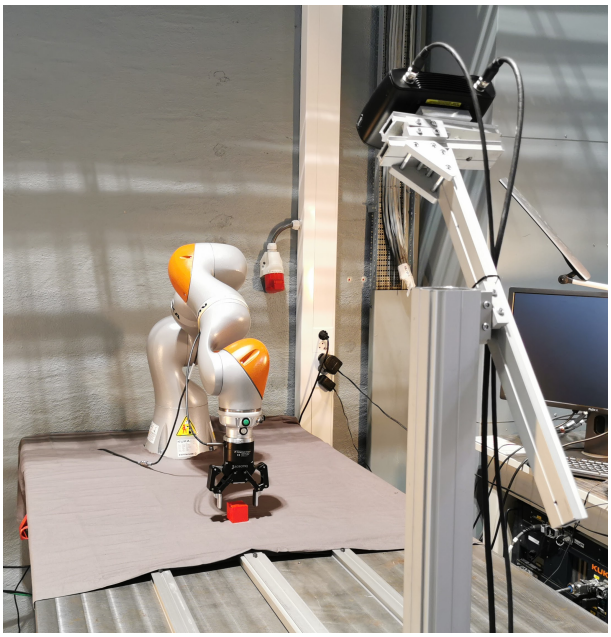


**Figure A.1.:** An overview of the setup used as an example throughout the thesis.

The setup consists of a robot arm (KUKA LBR iiwa) equipped with a gripper (Robotiq 2F 85) and a Zivid Two RGB-D camera facing the scene. An overview of the setup is shown in figure A.1.

## A.1. KUKA LBR iiwa

KUKA LBR iiwa is a collaborative robot designed to work alongside humans. It has safety mechanisms hindering forceful collisions with humans. It can carry

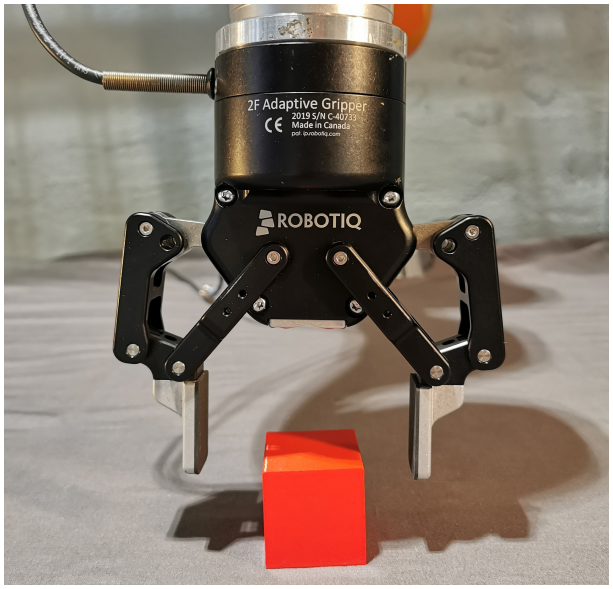



**Figure A.2.:** Robotiq 2F 85 mounted on the mechanical arm

.

payloads up to 14 kg and has a reach of 820 mm. It has seven joints, giving in one additional degree of freedom when keeping the end-effector pose fixed. This can enable it to avoid obstacles. It is controlled by joint torques applied at the joints, with software offering other control interfaces, such as directly controlling the end-effector pose. [21]

## A.2. Robotiq 2F 85

Attached to the manipulator is a Robotiq 2F 85, a parallel gripper. The width between the finger is 85 mm, and it can lift an object weighing up to 5 kg. It is depicted in figure A.2.

## A.3. Zivid Two

The Zivid Two camera provides the RGB-D images. This uses structured light to obtain high precision estimates of the depth of the image. It faces the robot and the workspace, as shown in figure A.1. Figure A.4 shows a closeup of the camera.

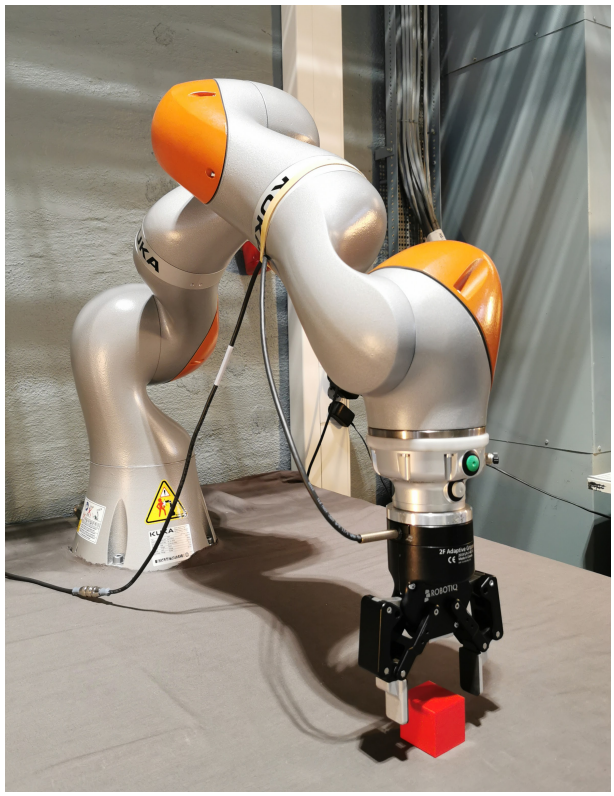


**Figure A.3.:** The KUKA LBR iiwa in the example setup


**Figure A.4.:** A closeup of the Zivid Two RGB-D camera

# Appendix B.

# Hyperparameters

The parameters for SAC and PPO used in the experiments are outlined in table B.1 and B.2, respectively. All other parameters are left to default values unless stated otherwise in the experiments.

| Parameter | Value | Explanation |
|---|---|---|
| T | 1 | Environment steps pr iteration |
| $n_{\text{gradient steps}}$ | 1 | Gradient steps pr. iteration |
| $\mathcal{H}$ | -7 | Desired entropy, $= -\dim \mathcal{A}$ |
| $tau$ | 0.005 | Polyak update parameter for the critic networks |
| Buffer Size | $10^6$ | The maximum size of the replay buffer |
| $\gamma$ | 0.99 | Discount factor |
| $n_{\text{explorational steps}}$ | 100 | Initial steps using the explorational policy |

**Table B.1.:** The SAC parameters used in the experiments

| Paramameter | Value | Explanation |
|---|---|---|
| N | 8 | Parallel environments for exploration |
| T | 2048 | Timesteps collected pr. environment before policy update |
| $\lambda$ | 0.95 | Bias-variance-tradeoff parameter of GAE |
| $\gamma$ | 0.99 | Variance reduction parameter in GAE (discount factor) |
| K | 10 | Number of epochs in policy update |
| M | 64 | Minibatch size |

**Table B.2.:** The PPO parameters used in the experiments