Karlsruhe Institute of Technology

# Vision-based object grasping with Deep Reinforcement Learning in simulation

Bachelor thesis

Anna Fedorchenko

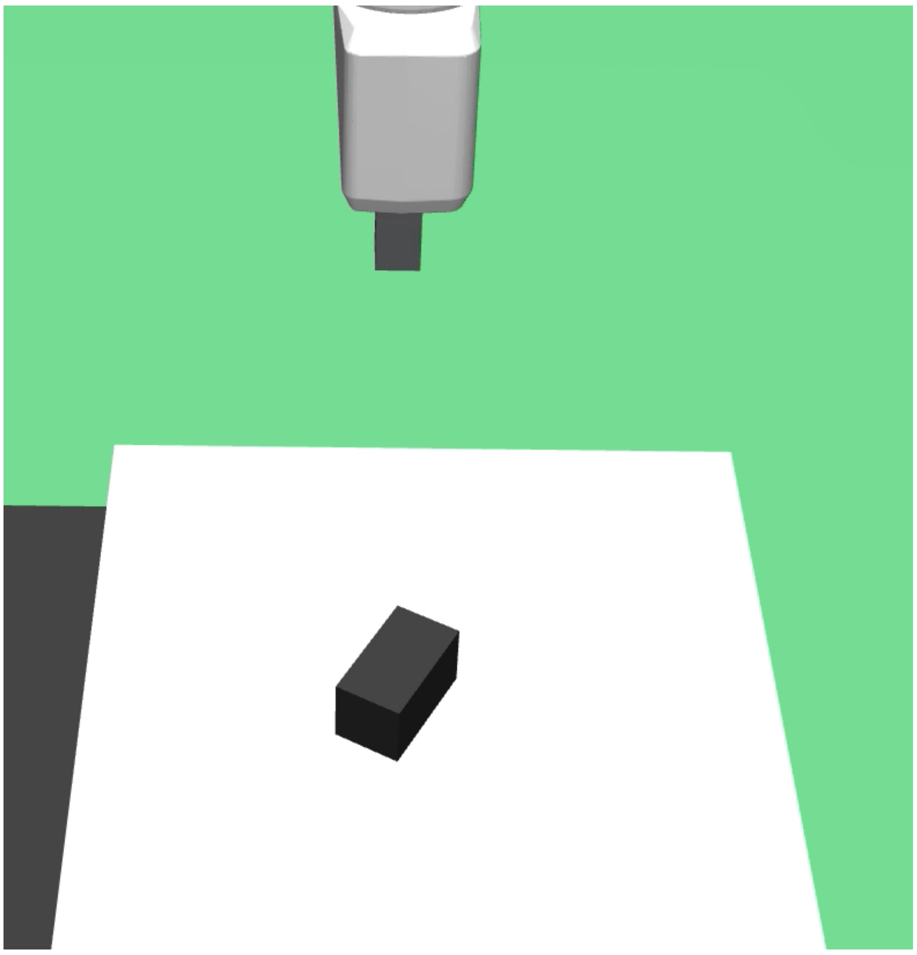
Department of Computer Science
Institute for Anthropomatics
and
FZI Research Center for Information Technology

| | |
|---|---|
| Reviewer: | Prof. Dr.-Ing. R. Dillmann |
| Second reviewer: | Prof. Dr.-Ing. R. Stiefelhagen |
| Advisor: | M. Sc. Atanas Tanev |

Research Period: 1st January 2020 – 11th June 2020

# Vision-based object grasping with Deep Reinforcement Learning in simulation

by
Anna Fedorchenko



**Bachelor thesis**
June 2020

Bachelor thesis, FZI
Department of Computer Science, 2020
Reviewers: Prof. Dr.-Ing. R. Dillmann, Prof. Dr.-Ing. R. Stiefelhagen

Abteilung Technisch Kognitive Assistenzsysteme
FZI Research Center for Information Technology

## Affirmation

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, *Anna Fedorchenko*
June 2020

# Contents

## Zusammenfassung

Greifen ist ein wichtiges und komplexes Problem. Aktuelle Lösung in der Industrie basiert sich auf bereits im Voraus bekannten Objekten und vordefinierten Bewegungen. Allerdings eine kleine Änderung bei dem Objekten oder in der Umgebung kann dazu führen, dass der Roboter nicht mehr in der Lage sein wird das Objekt erfolgreich zu greifen.
Ein System mit höherem Grad der Autonomität ist erwünscht. Solches System soll in der Lage sein sich an Objekte von unterschiedlichen Formen und Größen anzupassen, sowie an diverse Umfeldbedingungen, Lichtverhältnisse und andere mögliche Störungen. Der Roboter soll seine Entscheidungen auf der Wahrnehmung der aktuellen Umgebung basieren und nicht auf dem preprogrammierten Verhalten. Solches System könnte mithilfe von Reinforcement Learning erzeugt werden.
Das Ziel dieser Arbeit ist ein System zum Greifen in der Simulation basierend auf Deep Reinforcement learning zu entwickeln. Die Herangehensweise muss datengetrieben und selbstüberwacht sein. Der Roboter soll ohne Informationen über das Zielobjekt und Teilnahme des Menschen durch Versuch und Irrtum lernen. Basierend auf der binären Belohnung ob das Objekt erfolgreich gegriffen wurde oder nicht und auf dem RGB-Bild der Umgebung soll der Roboter eine Policy entwickeln.
Abschnitt 2 gibt Übersicht über verwandte Arbeiten im Bereich Greifen, sowie Reinforcement learning Algorithmen. Im Abschnitt 3 wird die entwickelte Herangehensweise erklärt und die Implementierung wird im Abschnitt 4 vorgestellt. Die Ergebnisse werden im Kapitel 5 beschrieben. Abschließend folgt der Ausblick auf die zukünftige Arbeit.

# Abstract

Grasping is an important and difficult problem. The current approach in the industry is mostly based on knowing in advance what objects are to be grasped and predefined movements of the robot. However, a small modification of the object or in the environment can lead to the robot not being able to grasp the object anymore.
A system with a higher degree of autonomy is desired. Such system should be able to generalize to objects of different shapes and sizes, as well as various settings, light conditions and other possible noise. The robot should base its decisions on the perception of the current environment rather than on the preprogrammed behavior. Such system can de developed with the help of reinforcement learning.
The goal of this thesis is to develop a robotic grasping system in simulation with the help of reinforcement learning. The approach should be data-driven and self-supervised. Without any information about the target object and human involvement the robot should learn to grasp by trial-and-error. Based on binary reward whether the object was grasped or not and the RGB-image of the environment the robot should develop a grasping policy.
Chapter 2 gives an overview of related work in the field of robotic grasping as well as reinforcement learning. In chapter 3 the approach of the thesis is explained with detailed information on the implementation in chapter 4. The results are represented in chapter 5.

# 1 Introduction

## 1.1 Motivation

Nowadays there is a big variety of use cases in the field of robotics in everyday life, starting from smart home devices to autonomous driving vehicles and space ship robots. Due to the significant development of software and hardware in the last decades it is becoming possible to create new robots that could considerably impact humans' lives.
In high number of various scenarios robots have to grasp an object and then perform different actions on it: lift, shift, put on a desired position. The robot will not be able to complete its task unless the target object is grasped successfully. Therefore grasping is a crucial problem in robotics that requires a precise solution.
The current approach for the grasping problem in the industry is mostly based on knowing the exact positions of the robotic gripper and the object, what kind of the object that is, its size, shape and position. So the grasping motion is also already predefined. If the characteristics of the object or its position slightly change, the robot might not be able to grasp it anymore. Such grasping systems function under a set of limitations for the objects and the setup.
The goal solution of the grasping problem would be the one that reduces the number of required limitations. The robot should be able to generalize to objects of any size and shape, regardless of their positions and surrounding environment. It should be able to adapt to new surroundings, light conditions, noise and any other constraints. The higher degree of autonomy is desired, so that the robot could base its decisions on the perception of the current environment rather than according to a preprogrammed behavior.
Due to the development of machine learning in recent years it has become possible to create intelligent systems that show high success rates in coping with different tasks: image and speech recognition, self-driving vehicles, game playing. Machine learning techniques have been also used in various approaches that aim to solve the grasping problem. Reinforcement learning(RL) is one of the types of machine learning which in combination with deep learning results in the deep reinforcement learning(DRL) approach that could effectively be used for the grasping problem. DRL is a general purpose algorithm that is based on learning from experience and can help combine perception and control. It has been used to create self-supervised grasping systems that are trained end-to-end with minimal to none human involvement. These systems have shown high success rates and proved to be able to generalize to novel objects. [1], [2].
In order to train a system based on the reinforcement learning approach, the robot has to complete a big number of trial-and-error steps before it finds a good policy. Training the robot in simulation is helpful for fast collecting large quantities of data with little cost. After verifying that the suggested algorithm is indeed successful at solving the defined problem in simulation, it can be transferred to the real world environment.

## 1.2 Problem statement

The focus of this thesis is to develop a method for vision-based robotic object grasping in simulation using reinforcement learning approach. The learning-based approach must be data-driven and self-supervised: there is no information about the target object and no human involvement in the training process. The robot must learn how to grasp by trial-and-error. Using only the RGB image of the workspace, it must decide which action to take. During training it will get binary reward from the environment whether the object was successfully grasped or not and based on this information it must train a policy.
The robot consists of a 7-DOF robotic arm with a two-finger jaw gripper. The executed grasps are planar. The end-to-end method must combine vision with robotic control. The training and testing will be conducted in a simulated environment.

# 2 Related work

## 2.1 Approaches for Solving the Grasping Problem

Sahbani et al. [3] divided different grasp synthesis algorithms in analytical and empirical. Analytical approaches investigate the physics, kinematics and geometry of the object and the robot in order to determine contact points and gripper position[4], [5]. Shimoga et al. Schimoga [6] defined a force-closure grasp as the one that guarantees that "the fingers are capable of resisting any arbitrary force and/or moment that may act on the object externally". Four independent properties that are crucial for a successful force-closure grasp were listed: dexterity, equilibrium, stability, dynamic behavior. Analytical approaches concentrate on developing algorithms that satisfy these properties.
[7] of Ding et al. is an example of the analytical approach. Having a multi-fingered gripper with n fingers, an object and the position of m of n fingers that do not form a form-closure grasp, the position of the remaining $(m-n)$ fingers have to be determined to satisfy the requirement of the form-closure grasp. There is a Coloumb friction between object and fingers 2.1. In order for fingers not to slip while executing the grasp, the finger force must have a certain direction (lie in a friction cone), which can be expressed as a set of non-linear constraints. Calculations consider the center of mass of the object, combination of grasping force and torque, center of the contact points. A sufficient and necessary condition for form-closure grasps was formulated.

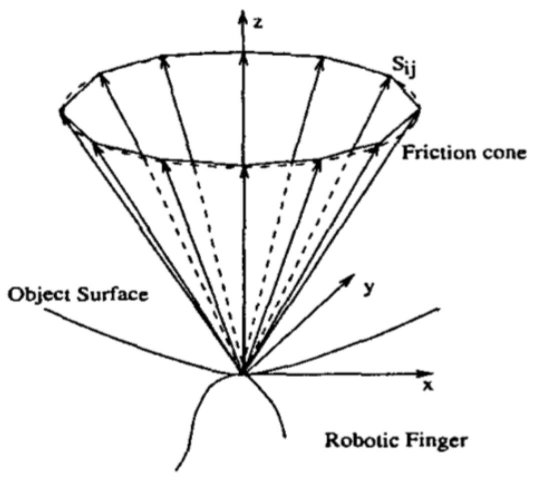


Figure 2.1: Outtake from "Computing 3-D Optimal Form-Closure Grasps" [7] by Ding et al. The image shows the position of the friction cone at a grasping point on the object's surface. In order for fingers not to clip while executing the grasp, the finger force must lie in the friction cone.

As Bohg et al. [8] stated, analytical approaches usually require exact 3D models of the object,

rely on the knowledge of the object's surface properties, its weight distribution and are not robust to noise.
In the same paper a detailed overview of the data-driven grasp synthesis approaches was presented. Grasp synthesis is defined as "the problem of finding a grasp configuration that satisfies a set of criteria relevant for the grasping task". Data-driven or also called empirical approaches sample various grasp candidates and then rank them using different strategies. The overview of aspects that influence grasp hypothesis is presented in figure 2.2.

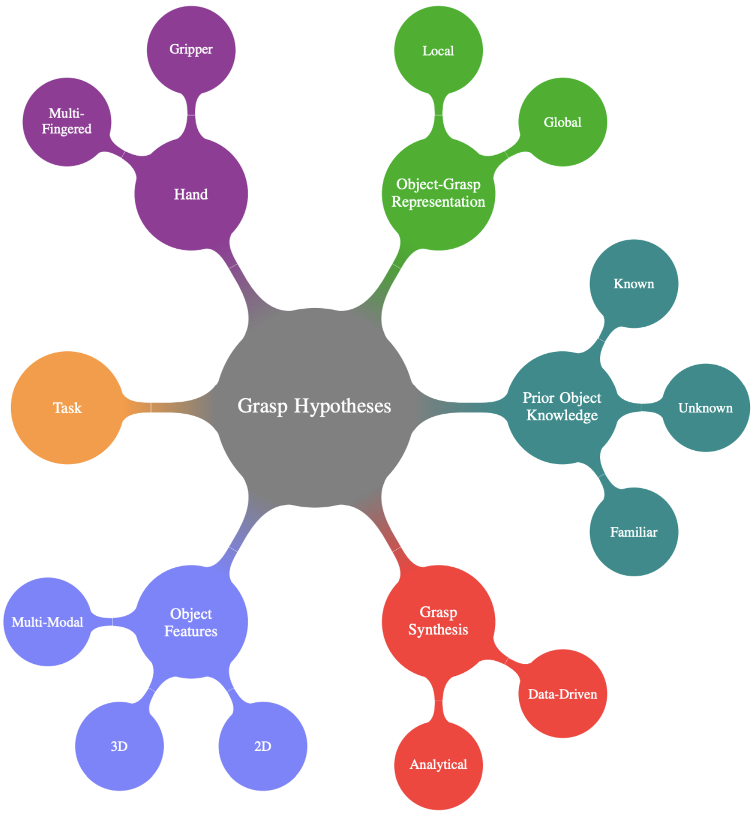


Figure 2.2: Classification of different aspects that influence the grasping problem according to [8] of Bohg et al. The most important aspect is the prior knowledge of the object.

Human demonstrations can be used to generate data for learning. In [9] magnetic trackers were placed on the hand of the human who was grasping and moving objects on the table. The robot recognized which object was moved and which grasp type was used, it then reproduced the task using this information.
Another strategy is to compare the candidate grasp to labeled examples. Redmon et al. [10] use the Cornell grasping dataset [11] to compare the sampled grasps to the "ground truth" grasps from the dataset. The rectangle metric is used for filtering good grasps. The metric includes two requirements: the grasp angle must be within 30° of the ground truth grasp and the Jaccard Index $(J(A,B) = \frac{|A \cap B|}{|A \cup B|})$ of the predicted grasp and the ground truth is greater than 25 %.
Empirical approaches can use analytical metrics for ranking grasp candidates or labeling robust grasps. Mahler et al. [12] introduced a synthetic dataset that includes 6.7 million point clouds

with parallel-jaw grasps and analytic grasp quality metrics. Objects in the dataset come from the database containing 1500 3D object mesh models (129 of them come from KIT object database [13]). For every object robust grasps covering the surface were sampled, using the antipodal grasp sampling approach developed in Dex-Net 1.0 [14]). For stable poses of each object planar grasps (grasps perpendicular to the table surface) are chosen, as well as corresponding rendered point clouds (depth images). The introduced dataset was used to train a neural network, that was also introduced in [12]. The network accepts a depth image of the scene and outputs the most robust grasp candidate.
Human labeling of possible grasps for objects has some significant disadvantages. First of all, it is time consuming. Secondly, there exist a number of possible grasps for every object, it is hard and even impossible to tag every one of them. Pinto et al. [2] also remarked the fact that human labeling is "biased by semantics": as an example they describe usual human labeling of handles of cups, because that is how a person would most likely to grasp a cup, although there are more possible configurations for successful grasp.
This reasoning led to development of self-supervised systems, where a robot learns from its own experience during the trial and error process. This approach is inspired by the way humans learn. Pinto et al. [2] used a CNN for assessing planar grasp candidates. The grasp candidate is represented as $(x, y)$ coordinates - the center of a chosen part of the image(patch) - and as an angle q - the rotation of the gripper around the z-axis. The CNN estimates a 18-dimensional likelihood vector. Each component of the vector represents the probability whether the grasp will be successful at the center of the patch with one of the 18 possible rotation angles of the gripper. The grasp with the highest probability of the success is executed. Depending on the result of the grasp, the error loss is back-propagated through the network. This way, the system does not rely rather on analytical metrics nor on human labeled examples, since it learns from its own experience.
Following the classification of [8], the objects that have to be grasped can be divided into three categories: known, familiar and unknown objects.
For known objects the data-driven approaches for finding a successful grasp often consist of estimating the pose of the object and then choosing the suitable grasp candidate from a precalculated grasp database, known as "experience database". In [15] of Tremblay et al. the deep neural network takes as input only one RGB-image of the scene with known household objects and outputs belief maps of 2D keypoints of these objects. After that a standard perspective-n-point (PnP) [16] algorithm uses these belief maps to estimate the 6-DOF pose of each object. For grasping the robot moves its arm to a point above the object and then completes a top-down grasp.
Another category of objects to grasp is familiar objects. Objects can have similarities in various aspects(texture, shape, etc.). Familiar objects might be grasped in similar ways, the difficulty consists in detecting these similarities between objects and then applying grasping experience on them. Manuelli et al. [17] researched category-level object manipulation with the help of using semantic 3D keypoints as object representation. The two object categories examined were shoes and cups. The goal of robotic manipulation was not only grasping, but also positioning the objects in a specific predefined way (i.e. place shoes on a shelf, hang cups on the hook by the handle). First the database of manually labelled keypoints on 3D reconstruction of the objects in different

training scenes was created. Then a neural network was used to detect these keypoints from an RGB-D image of the scene. The detected keypoints together with the RGB-D image were used to calculate a suitable grasp using a similar to the baseline learning-based algorithm demonstrated in [18].
In case of unknown objects, the object was never seen by the robot beforehand and never used in training. Many approaches are based on approximating the shape of the object and then choosing the grasp for it [19]. In recent years the approaches that have been most successful at solving this type of grasping problem are based on the trial-and-error learning-based approach [2], [20], [18].

## 2.2 Simulation

In data-driven approaches training data is required to learn for a successful grasp. Levine et al. [21] used 14 robotic arms that sampled 800 000 grasp attempts. Pinto and al. [2] used one robot manipulator to conduct 50 000 grasp attempts in course of 700 hours. However, it is expensive to collect such amount of data and it is very time consuming, this is where the arguments for learning in simulation come to a point.
Goldfeld et al. [22] created a grasp planner containing database of grasps for 3D models of different objects, that were generated using GraspIt! [23] simulation engine. Kasper et al. [13] introduced the system for digitizing objects. In year 2010 the OpenGRASP [24] simulation toolkit for grasping and dexterous manipulation was created.
James et al. [25] developed an approach that used only a small amount of real-word training data in addition to simulation, which helped to reduce the real-world data by more than 99%. Bousmalis et al. [26] implemented a grasping method that helps to significantly reduce the amount of additional real-world grasp samples. In their experiment 50 times less real-world samples were required to achieve the same level of performance compared to their previous system.
Another advantage of using simulation is the ability to pretrain the network. Redmon et al. [10] stated that pretraining large convolutional neural networks significantly decreases training time and also helps to avoid overfitting". However there are some drawbacks to synthetic data. Most significant one is the reality gap: the network trained in a simulated environment shows much worse performance in real world. An effective way to trying to close the reality gap is to use domain randomization [27]. Tremblay et al. [15] used photorealistic data in addition to domain randomization. It added variation and complexity to the training, which helped for the trained neural network to be able to successfully operate in the real world scenario without any additional fine-tuning. Another technique to overcome the reality gap is domain adaptation [28]. Bousmalis et al. [26] used domain adaptation to extend the grasping system trained in simulation.

## 2.3 End-To-End Learning

In robotics the classical control over the robot consists of several steps, each one of them usually represented by a separate module which are connected with each other and pass the data to each other. For example: after retrieving an observation of the scene using a calibrated camera another

module detects objects in the image. The next module might create a physics model of the scene to plan an action that needs to be executed. The next step would calculate the positions of robot's joints and after that the actual action would be executed. If an error is made in one of these steps, it might become even more significant in the next steps resulting in incorrect output. End-to-end approach suggests making such connection between modules, that can be changed and adapted during the learning. In that way mistakes that would happen in one of the modules would be corrected in the following ones so that in the end the output would not be influenced by them.
Levine et al. [29] discovered that training perception and control systems end-to-end shows better results and consistency than training each component separately.

## 2.4 Vision-based Learning

Grasping novel objects is challenging as the robot has no information about them. The task is even more complicated when there are multiple objects in a cluttered environment that partially cover each other. In the past several attempts were made to create systems that can locate good grasping points from images of the scene [30]. Nowadays most data-driven grasping approaches use Convolutional Neural Networks(CNNs) to detect and grasp objects. CNNs can effectively learn to extract features from an image that are essential for grasping.
A variety of types of CNNs exist, for example VGG [31], Residual neural networks [32], Densely connected neural networks [33]. [20] uses ResNet to extract spacial features of the image which are then inputted to another Res-Net. [34] and [1] use a DenseNet. Custom architecture for the CNN that delivers best results for the task can be designed [35].
Multiple researches were aimed at developing the grasping system where the robot must calculate a grasping movement based only on the RGB or RGB-D image of the working space [20], [1], [26]. Levine et al. [21] developed a grasping approach that is based on collecting large-scale data using multiple real robots and no human involvement in the training process, and getting continuous feedback on visual features using only over-the-shoulder RGB camera. This end-to-end approach uses raw pixel images as input and directly outputs task-space gripper motion. The paper addresses the issue of the need of massive analysis and planning in robotic manipulation tasks. The suggested methods avoids it by providing the system with continuous feedback from the setup, fragmenting the grasping attempt in several timesteps, getting an RGB-image and letting it decide what action to take at each timestep. This way the robot can correct its mistakes using previous experience, at the same time not requiring exact camera calibration.

## 2.5 Deep reinforcement learning for grasping problems

One of the most important goals of learning-based approaches is for system to be able to perform effectively on previously unseen objects - generalization. Another essential aspect that is considered during development of the approach is the intention of minimizing human's participation in the training of the system. In recent years several approaches based on reinforcement learning have been successfully applied for solving the grasping problem [2], [1], [35]. With the help of reinforcement learning it is possible to create a self-supervised system that can learn through trial-and-error. This way the human involvement can be minimized as the robot collects training data by itself. Systems trained with reinforcement learning are also able to generalize to novel objects and scenarios.

Kaelbling et al. [36] made following definition for the reinforcement learning problem: "Reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment." The trial-and-error interactions are the factor that is crucial for a self-supervised system: it supervises itself by committing an action, getting the result of this action and learning from it.

Quillen et al. [37] compared different reinforcement learning off-policy algorithms for vision-based robotic grasping. The authors state that on-policy algorithms struggle with diverse grasping scenarios as the robot has to go back to previously seen objects in order to avoid forgetting the gained experience. Therefore off-policy methods might be preferred for the grasping problems. The criteria for evaluating the RL algorithms were: overall performance, data-efficiency, robustness to off-policy data and hyperparameter sensitivity. These factors are important when applying the grasping algorithms to robotic systems in real life.

Boularias et al. [38] used reinforcement learning approach to grasp objects in dense clutter. The task was formulated as a Markov Decision Process (MDP). The state is represented as the RGB-D image of the scene. Action space consists of two types of actions: pushing and grasping, each action is an according vector. In a cluttered environment with a variety of objects sometimes the position of an object makes it hard to grasp it. Executing a pushing action on this or another object might help to gain better access to the object for grasping. The reward is 1 if the robot managed to successfully grasp an object, otherwise 0. The RGB-D image is primarily segmented into objects using spectral clustering [39].

Several of recent researches have achieved impressive results at solving the grasping problem with the extended version of Deep Q-Learning based on Convolutional Neural Networks. In [1], [20] and [34] in the grasp planning part the robot estimates the state of the environment through RGB-D image of the workspace, it is used as input to a CNN. The workspace is represented as a discrete action grid with the same resolution as the input RGB-D image. The CNN is used as a Q-function approximator. The network outputs a probability map of the same size as the input image, each value $(x,y)$ in the probability map is an estimated Q-value of completing a top-down planar grasp in the middle of pixel $(x,y)$ which corresponds to grasping in the middle of the cell $(x,y)$ of the action grid. The action is defined as the number of the cell $(x,y)$ in the action grid and the rotation angle. The number of possible rotation angles is predefined: in [34] there are three possible rotation angles: $0°$, $45°$ and $90°$. [20] and [1] used 16 possible rotations (different multiples of $22.5°$). Before inputting the image to the network, it is rotated by an according angle, altogether resulting in $n$ input images for one state, with $n$ - the number of possible rotation angles. For each input image the probability map is calculated. Then the cell of the map with the biggest value across all output maps is greedily chosen and an according top-down grasp with the gripper rotated as the input image containing the biggest Q-value is executed.

This pixel-wise parameterization of state and action provides several advantages: the actions have a spatial locality to each-other, CNNs are efficient for pixel-wise computations, the training can converge with less data.
It is necessary to point out that in these researches [1], [20], [34] grasping was studied in combination with other actions: pushing in [1], sliding to wall in [34] and throwing in [20]. The approaches were trained end-to-end, which helped to reach high success rates. The reward function in [34] is binary: if the object is grasped 1, otherwise 0. In [1] additionally the pushing action that has a noticeable change to the system is rewarded with 0.5. In [20], the grasping part was trained with the help of obtaining a success label. Binary signal whether the throwing part succeeded was more efficient than calculating the antipodal distance between gripper fingers directly after the grasping part.
RBG-D image as state representation, action space corresponding to image resolution, binary rewards, extending Deep Q-Learning based on CNN and end-to-end training is an effective way to train the robot to reach a high success rate in grasping and to quickly generalize to new objects and scenarios.

## 2.6 Reinforcement learning

Reinforcement learning is one of the types of Machine Learning. Combined with deep learning, it is a powerful technique for solving complex problems. Deep reinforcement learning has been used in different areas, such as games [40], [41] and robotics [42], [1].

The core idea of reinforcement learning is having an agent that can perform actions in a specified environment. The agent gets observations from the environment as an input, the output is an action. At all times the agent is in one of the predefined states, the actions that are possible in the current state are a subset of all actions set. For every performed action the agent receives a feedback in form of a reward from the environment. According to the reward the agent can decide whether the chosen action was the right decision. The core idea of reinforcement learning is modeled as Markov decision process (MDP), which consists of 4-tuple $(S, A, R, P)$ [36]:

- set of states S
- set of actions A
- a reward function $r_t = R(s_t, a_t, s_{t+1}), R : S \times A \times S \to \mathbb{R}$
- state transition function $P : S \times A \to P(S)$, where $P(S)$ is a probability distribution over the set S (i.e. it maps states to probabilities)

In some notations the starting state distribution $\rho_0$ is defined as the fifth element of the MDP. The name of the process comes from the concept of Markov chains: having a sequence of events the probability of each event depends only on the state that was achieved in the previous event, ignoring all events before. Markov state: $P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t)$.

The policy $\pi$ determines which action must be taken in each state [43]. The action might be deterministic:

$$a_t = \mu(s_t) \tag{2.1}$$

or stochastic:

$$a_t \sim \pi(\cdot|s_t) \tag{2.2}$$

Same for the state transition function: deterministic $s_{t+1} = f(s_t, a_t)$ or stochastic $s_{t+1} \sim P(\cdot|s_t, a_t)$. Often there are multiple possible ways for the agent to accomplish a task. For example, there might be a short and a long path to a destination point, both of them conforming the requirements. However, the short path is better because it takes less time and less actions, so the agent should choose this path. This can be expressed as following: goal of the optimal policy is to maximize the sum of rewards during action sequence that leads to completing the task. There are multiple ways to define the sum of the rewards:

- finite-horizon undiscounted return: $R(t) = \sum_{t=0}^{T} r(t)$, a straightforward sum of all rewards for all taken actions.
- infinite-horizon discounted return: $R(t) = \sum_{t=0}^{\infty} y^t r(t)$, where y is a discount factor $\in$ (0,1). The discount factor makes sure the actions that are taken soon are more relevant that the ones that are taken many steps later. From mathematical point of view, the discount factor is one of the conditions that the infinite sum converges to a finite value.

Value function of the state $s$ is an expected return that the agent will get if it starts in state $s$ and acts according to the policy. Action-value function adds dependency to an action $a$: expected return after starting in state $s$ and taking action $a$, continuing by acting forever according to the policy $\pi$:

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{E}[R(\tau)\,|s_0 = s, a_0 = a] \tag{2.3}$$

The optimal action-value function $Q*(s,a)$ gives the expected return if the agent starts in state $s$, takes an arbitrary action $a$, and then forever after acts according to the optimal policy in the environment:

$$Q^{*}(s,a) = \max_{\pi} \underset{\tau \sim \pi}{E}[R(\tau)\,|s_0 = s, a_0 = a] \tag{2.4}$$

where policy $\pi$ is optimal.
There is a connection between value and action-value functions, that is helpful for some calculations:

$$V^{\pi}(s) = \underset{a \sim \pi}{E}[Q^{\pi}(s,a)] \tag{2.5}$$

- the value function of the state is an expected return of the Q-function in this state if the action a taken comes from the policy $\pi$.
and

$$V^{*}(s) = \max_{a} Q^{*}(s,a) \tag{2.6}$$

## 2.7 Reinforcement Learning Algorithms

There is a wide variety of reinforcement learning algorithms. They can be categorized in model-based and model-free. As the name suggests, model-based algorithms have a model of the environment and use it to find an optimal policy. Bansal et al. [44] state that model-free reinforcement learning approaches are effective at finding complex policies, however they sometimes take very long time to converge. Model-based algorithms might be better at generalizing and reduce the number of steps to find an optimal policy, however without exact model the learned policy might be far from an optimal one. Also the model must be updated together with the policy.

Model-free reinforcement learning algorithms can also be divided in being on- and off-policy. Policy is used in reinforcement learning to decide which action to perform in the current state. While learning and building the policy up, the algorithm does not necessarily need to always chose the action that the latest version of the policy suggests. The chosen action might be for example the one that will maximize the value function for the state as in Q-learning. In that case the algorithm is off-policy. In the opposite case, when algorithm strictly follows the policy while learning, it is an on-policy one.

### 2.7.1 Soft Actor-Critic

Soft Actor-Critic(SAC) is a off-policy state-of-the art reinforcement learning algorithm [45]. As the name suggests, it is based on the actor-critic approach. The key idea of SAC is the actor trying to maximize not only the expected return but also the entropy. The algorithm is stable and also sample-efficient, it is capable of solving high-dimensional complex tasks.
Entropy is a measure of randomness in the policy. The term 'entropy' comes from the area of information theory and means the amount of information or 'surprise' of the possible outcome of the variable. When the outcome of some source of data is rather unexpected because it has less probability, it's entropy is high.
Entropy $H$ of $x$ from its distribution $P$:

$$H(P) = \underset{x\sim P}{E}[-\log P(x)] \qquad [2.7]$$

Increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum.
At each timestep the agent gets additional to the immediate reward $R(s_t, a_t, s_{t+1})$ a bonus reward proportional to entropy: $\alpha H(\pi(\cdot|s_t)$.

In SAC the goal of the algorithm is to find a policy that will maximize the objection:

$$J(\pi) = \underset{\tau\sim\pi}{E}[R(\tau)] = \underset{\tau\sim\pi}{E}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))] \qquad [2.8]$$

The target policy is:

$$\pi^* = \underset{\pi}{argmax} J(\pi) = \underset{\pi}{argmax} \underset{\tau\sim\pi}{E}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))] \qquad [2.9]$$

$\alpha$ is the temperature parameter that controls the influence of the entropy term on the reward function.

The implementation of SAC includes five neural networks which are aimed to enable learning of three functions: Parameterized state value function $V_\psi(s_t)$ , soft Q-function $Q_\theta(s_t, a_t)$, and a tractable policy $\pi_\phi(a_t|s_t)$. The parameters of these networks are $\psi$, $\theta$, and $\phi$. Additionally there is a target value network $V_{\bar{\psi}}$. The algorithm uses two Q-functions $\theta_1$ and $\theta_2$ which helps to avoid overestimations and speed up the training.
First the agent collects some experience which is saved in the replay buffer D (distribution of previously sampled states and actions). After that all five networks are updated based on the information from the replay buffer.

**Algorithm 1** Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
    for each environment step do
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
    end for
end for
```

Figure 2.3: Description of the SAC algorithm from the original paper [45]. First the agent collects some experience which is saved in the replay buffer D. After that all five networks are updated using information from the replay buffer.

### 2.7.2 Proximal Policy Optimization Algorithm

Proximal Policy Optimization (PPO) [46] is the state-of-the-art on-policy algorithm. The main idea of the algorithm is following: after computing the update, the new policy should be not to far away from the old one. This idea is shared with another on-policy algorithm TRPO [47], however PPO algorithm is easier to implement and in has shown better performance on multiple reinforcement learning problems [46].

The policy update rule of the PPO algorithm:

$$\theta_{k+1} := \underset{\theta}{argmax} \underset{s,a \sim \pi_{\theta_k}}{E} [L(s,a,\theta_k,\theta)] \qquad [2.10]$$

where $\theta$ is the policy parameter,

$$L(s,a,\theta_k,\theta) = min(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), g(\varepsilon, A^{\pi_{\theta_k}}(s,a))) \qquad [2.11]$$

$\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is the ratio of the probability under the new and old policies, $A^\pi$ - advantage function, $\varepsilon$ is a hyperparameter which sets a limit how far can an new policy be far away from the old one. The advantage function $A^\pi(s,a)$ helps to determine how much better the action $a$ in state $s$ is in comparison to the random selected action according to $\pi(\cdot|s) : A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon)A & A \geq 0 \\ (1-\varepsilon)A & A < 0 \end{cases} \qquad [2.12]$$

# 3 Approach

In order to combine vision with robotic grasping, the DRL approach is introduced. As described in section 2.6, the goal is to develop a policy $\pi$ according to which the agent will decide what action $a_t$ he will take at the current state $s_t$. The policy is trained through trial-and-error. After getting information about the current state $s_t$, the agent takes an action $a_t$. The environment yields reward $r_t$ for this action and the next state $s_{t+1}$, that the action lead to. With the help of this information the agent builds up its policy. The approach is visualized in figure 3.1. At all times there is only one object on the table that is the same during the whole training.

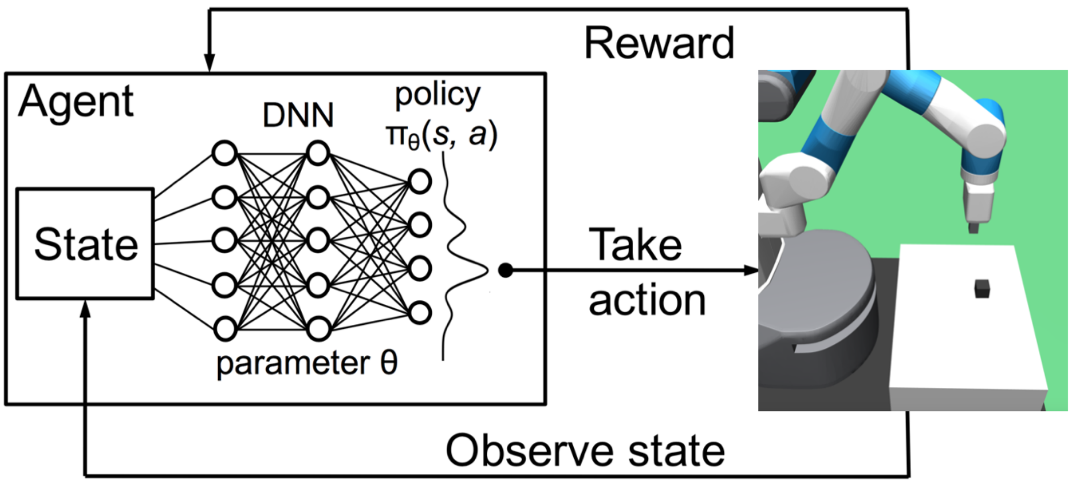


Figure 3.1: The goal of the DRL approach is to develop a policy that decides which action at which step the agent should take. In order to do that the agent interacts with the environment by observing environment's state, taking actions and getting rewards for them. This way the agent determines through trial-and-error the correct behavior. Source of the image: [48]

## State representation

The state is represented as an RGB-image of the work surface. The camera is positioned to capture only the part of the table where the object can be located. The image does not include the gripper.

## Action space

The robot movements are modeled as planar grasps. In the beginning of the iteration the gripper has a predefined fixed height $z$ over the table. The actions are target $(x, y)$ coordinates. The gripper goes to $(x, y)$ preserving its height $z$. Afterwards it goes down, changing only its $z$-coordinate and

having maximal jaw opening. Then the gripper closes and goes back up.
In some experiments the gripper yaw-rotation is part of the action space $(x, y, \alpha)$. After reaching target $(x, y)$ position, the gripper rotates the angle $\alpha$ and then completes a top-down planar grasp.

### Reward function

The reward function is sparse:

$$r_t = \begin{cases} 1 & \text{if the grasp was successful} \\ 0 & \text{otherwise} \end{cases} \qquad [3.1]$$

This type of reward is intuitively logical as it grants 1 only if the whole task of picking the object is completed successfully. However, it is also challenging as in cases when the agent gets 0 as reward it does not know how far he was from the correct action and how it should change its policy to achieve success.

# 4 Implementation

In order to implement the system several components were used, which are described in following sections. The approach was implemented for several action spaces in combination with different RL algorithms.

## 4.1 System components

Mujoco [49] was chosen as simulation environment as it has shown faster and more accurate performance than other simulators [50]. As a base for the simulation the gym "FetchPickAndPlace-v0" environment [51] was used. Implementation of the RL algorithms SAC and PPO were taken from the stable baseline [52] library.

### 4.1.1 Mujoco

Mujoco (Multi-Joint dynamics with Contact) is a physics engine developed for model-based control [49]. It was developed at "Movement control laboratory", University of Washington for research projects in the area of "optimal control, state estimation and system identification", as none of already existing tools delivered a satisfying performance. Mujoco has shown more stable, fast and accurate performance for robotic tasks in comparison to other physics simulators [50].
Mujoco is user-convenient as well as computation efficient. The model of the environment can be specified in a an XML-file format. The visualization is interactive and done by the rendering library OpenGL [53]. Some of the key features of Mujoco include:

- Generalized coordinates combined with modern contact dynamics
- Soft, convex and analytically-invertible contact dynamics
- Separation of model and data

and many more. Further description of Mujoco and its documentation can be found on the official website [54].

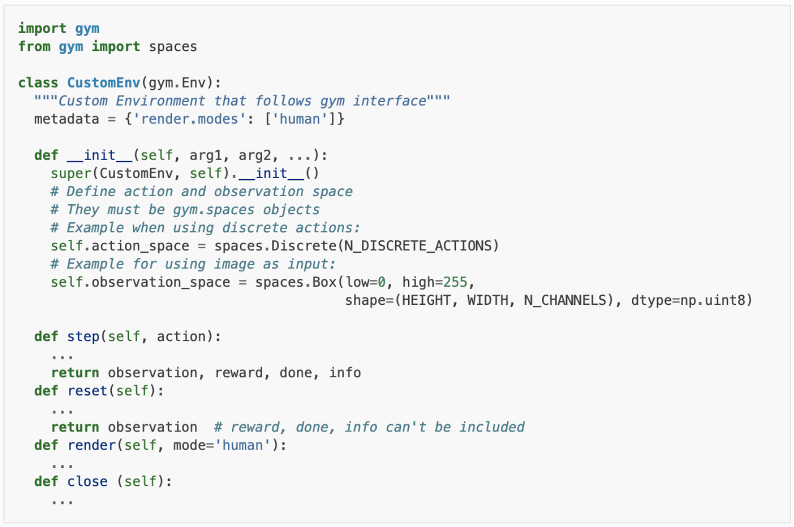
### 4.1.2 OpenAI Gym

Gym[55] is toolkit developed by OpenAI [56] for researching in the area of reinforcement learning. Gym is an open-source library which includes collection of environments for different reinforcement learning problems. They include Atari (i.e. Breakout-v0) and Board games (Go), Robotics(HandManipulateBlock-v0) and many more [55]. The user can modify the environment and construct its own agent.

### 4.1.3 Stable Baselines

Baselines [57] is a library developed by OpenAI [56] containing implementations of different RL algorithms. Stable baselines [52] is a collection of improved RL algorithms based on OpenAI baselines. The algorithms of stable baselines have unified structure and are well documented. They have been validated through a bigger number of tests with high percentage of code coverage. Moreover, some new reinforcement learning algorithms were represented. They can be used in combination with gym environments. Stable baselines also include a set already pretrained agents [58].

In order to train the agent using RL algorithms from stable baselines, the environment must follow the gym interface. The environment is required to implement standard gym methods __init__(), step(), reset(), render(), close() and inherit from gym.Env.



```
import gym
from gym import spaces

class CustomEnv(gym.Env):
  """Custom Environment that follows gym interface"""
  metadata = {'render.modes': ['human']}

  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()
    # Define action and observation space
    # They must be gym.spaces objects
    # Example when using discrete actions:
    self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
    # Example for using image as input:
    self.observation_space = spaces.Box(low=0, high=255,
                                        shape=(HEIGHT, WIDTH, N_CHANNELS), dtype=np.uint8)

  def step(self, action):
    ...
    return observation, reward, done, info
  def reset(self):
    ...
    return observation  # reward, done, info can't be included
  def render(self, mode='human'):
    ...
  def close (self):
    ...
```

Figure 4.1: Custom environment example for using stable baselines. The CustomEnv inherits from gym.Env and implements methods __init__(), step(), reset(), render(), close() which is a requirement to be able to train using one of the RL algorithm's implementation from stable baselines.

## 4.2 Simulation Setup

As a base for the simulation the gym "FetchPickAndPlace-v0" environment [51] was used. The goal task in this environment is for the robot to pick up a box from the table and place it to the desired specified position.

- Observation space:
  In "FetchPickAndPlace-v0" the observation space is represented as a list of different values, such as positions, rotations, velocities of the gripper and the object. For the current environment it was changed to consist only of an RGB-image.

- Action space:
  The action space in "FetchPickAndPlace-v0", that consisted of changes of grippers coordinates and rotations, was adjusted to have only two values: target $x$ and $y$ coordinates, as well as the target rotation of the gripper in some experiments.
- Reward function:
  The reward function was modified from the dense to a sparse one: in "FetchPickAndPlace-v0" environment it is represented as the distance between achieved and desired goal, whereas in the modified version it is binary: $r = \begin{cases} 1 & \text{if the grasp was successful} \\ 0 & \text{otherwise} \end{cases}$

The step() function was adjusted in order to modulate the planar grasp: the gripper goes to $(x, y)$ preserving its height $z$. Afterwards the planar grasp takes place: the gripper goes down, changing only its $z$-coordinate and having maximal jaw opening. Then the gripper closes and goes back up. Each training episode lasts at most 50 steps which is defined by the *max_episode_steps* parameter. It means the robot has 50 attempts to grasp the object before the algorithm switches to the next episode.

## 4.3 Observation Space

The camera is adjusted to take an RGB-image of the part of the table where the object can be located. The image represents the state of the environment. The $81 \times 81$ pixels with 3 channels for RGB result in a $81 \times 81 \times 3$ input to the neural network.

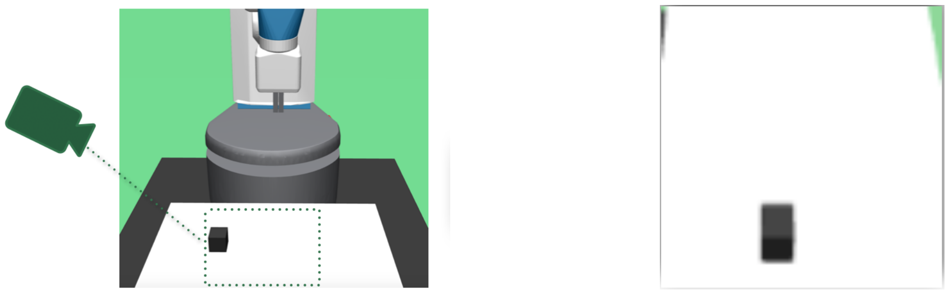


Figure 4.2: The camera on the side of the table takes an RGB-image of the part of the table where the object can be located. The second image is an example of the observation that is used as state representation and input to the neural network

## 4.4 Reward Function

The agent is granted with the reward 1 if he successfully grasps the object, otherwise 0. In order to determine whether the grasp was successful, the distance between the gripper jaws is compared to a threshold value that corresponds to the width of the object. If the distance exceeds the threshold, the object is located between the jaws at the end of the manipulation, which means it was grasped successfully: $r = \begin{cases} 1 & d > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$

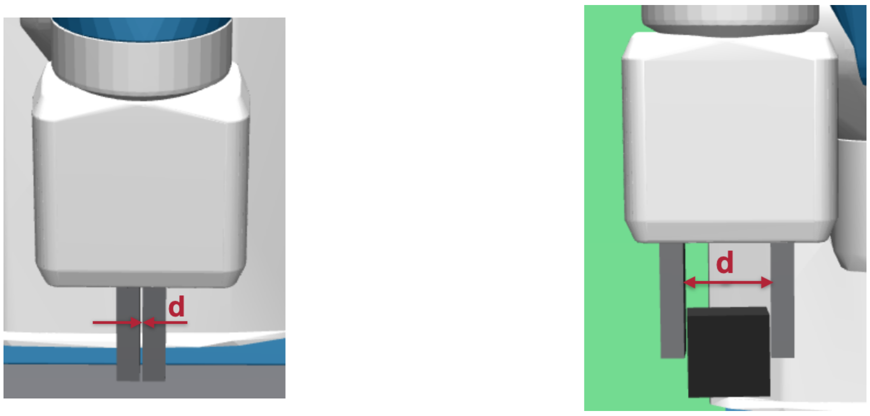


Figure 4.3: In the first image the gripper's jaws are fully closed after the grasped attempt - the object was not grasped. In the second image the attempt was successful, the object is between the jaws preventing them from closing, so the distance between the jaws is slightly greater or equals the width of the object.

## 4.5 Action Space

Three possible implementations are used to represent the action space. Apart from using the straightforward continuous action space, the idea of discretizing action space as in [1], [20] is tested. With the help of these multiple approaches can be determined whether the representation of the action space can influence performance of some reinforcement learning algorithms.

### 4.5.1 Continuous Action Space

The action $(x, y)$ is target Cartesian coordinates of the gripper: x ∈ [1.1, 1.45] and y ∈ [0.55, 0.95]. Continuous spaces are represented through the Box class of the gym library:

```
self.action_space = spaces.Box(np.array([1.1, 0.55]), np.array
    ([1.45, 0.95]), dtype='float32')
```

Example of the action: [1.2, 0.7].

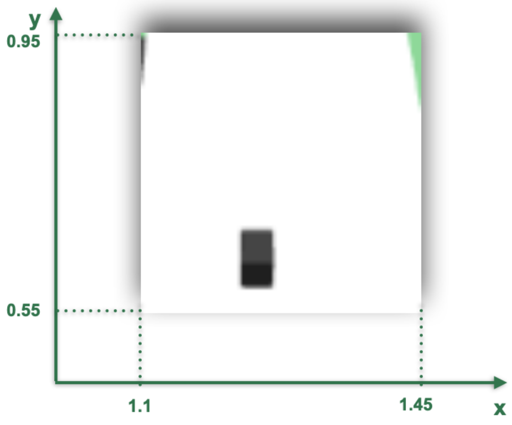


Figure 4.4: Continuous Action Space: the action $(x, y)$ is target cartesian coordinates of the gripper.

### 4.5.2 Discrete Action Space without Rotation

For this approach the workspace is represented as $50 \times 50$ grid. The action $(x, y)$ means going to the middle of the cell number $(x, y)$ and completing a planar grasp. Discrete action space consists of several discrete values: $x$ and $y$ are integers, $x \in [0, 49]$ and $y \in [0, 49]$. Discrete action spaces are represented through the MultiDiscrete class of the gym library:

```
self.action_space = spaces.MultiDiscrete([49, 49])
```

Before executing the action, it is translated to cartesian coordinates. The width and length of the cell of the grid is calculated based on the parameters of the workspace $x \in [1.1, 1.45]$ and $y \in [0.55, 0.95]$ and the grid size $50 \times 50$:

```
# the width of the cell
stepX = (1.45 - 1.1) : 50 = 0.007
# the length of the cell
stepY = (0.95 - 0.55) : 50 = 0.009
# (x,y) coordinates of the middle of the first cell
first_cell_middle_X = 1.1 + 0.007:2 = 1.1035
first_cell_middle_Y = 0.55 + 0.009:2 = 0.5545
```

Using these parameters, the action is converted from descrete to cartesian coordinates.

```
def convert_discrete_action(self, action):
        new_action = [0, 0]
        new_action[0] = first_cell_middle_X + action[0] * stepX
        new_action[1] = first_cell_middle_Y + action[1] * stepY
        return new_action
```

Example of the action: $(x, y) = (2, 40)$ which corresponds to $(1.1175, 0.9145)$.

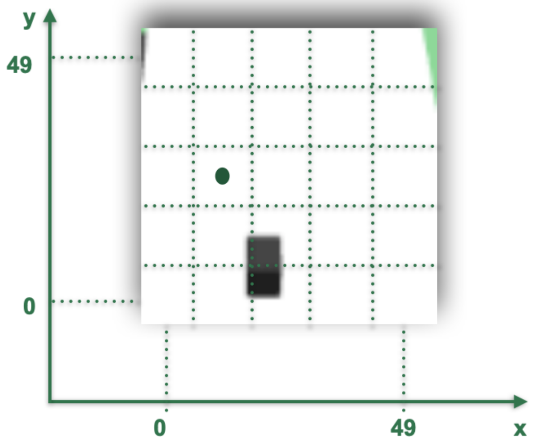


Figure 4.5: Discrete Action Space. The workspace is represented as $50 \times 50$ grid, the action $(x, y)$ means going to the middle of the cell number $(x, y)$ and completing the planar grasp.

### 4.5.3 Discrete Space with Rotation

As in the previous method, described in section 4.5.2, the workspace is represented as $50 \times 50$ grid, action $(x, y)$ means going to the middle of the cell number $(x, y)$. In this method one more variable is added to the action space - the target yaw gripper angle $\alpha$. $\alpha$ is an integer, $\alpha \in [0, 6]$. Before completing the top-down planar grasp, $\alpha$ is translated to degrees and the gripper is rotated in a degrees value that corresponds to the $\alpha$ angle.
The angle range for the gripper is limited to [0°, 90°] with the 15° step, resulting in 7 possible rotations which are expressed by the $\alpha$ value.

```
def convert_discrete_action(self, action):
        stepA = 15
        new_action = [0, 0, 0]
        new_action[0] = first_cell_middle_X + action[0] * stepX
        new_action[1] = first_cell_middle_Y + action[1] * stepY
        new_action[2] = action[2] * stepA
        return new_action
```

Example of the action: $(x, y, \alpha)$ = (2, 40, 3) which corresponds to (1.1175, 0.9145, 45°).

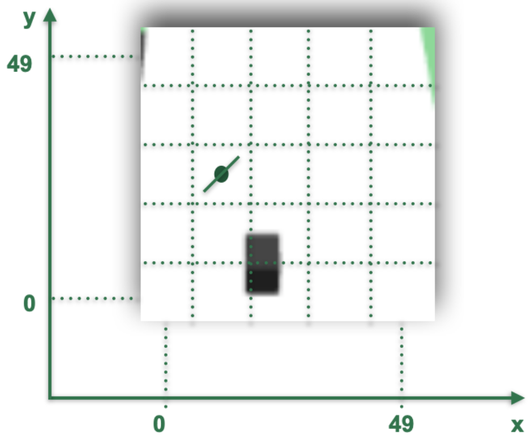


Figure 4.6: Discrete Action Space including rotation. The workspace is represented as $50 \times 50$ grid, the action $(x, y, \alpha)$ means going to the middle of the cell number $(x, y)$, rotating the gripper value in degrees that corresponds to $\alpha$ and completing the planar grasp.

## 4.6 Learning Algorithms

Two algorithms are tested for solving the task: Soft-Actor Critic and Proximal Policy Optimization. Both are leading reinforcement learning algorithms.

### 4.6.1 The Soft-Actor Critic Algorithm

Soft-Actor-Critic (SAC) is the state-of-the-art off-policy algorithm that can be used for environments with continuous action spaces, so it is used in combination with the action space approach described in section 4.5.1.

If not explicitly stated otherwise, the hyperparameters that were applied during experiments are:

| **Hyperparameter** | **Value** |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| learning rate | 0.0003 |
| replay buffer size | 50000 |
| learning starts (how many steps of the modelto collect transitions for before learning starts) | 100 |
| train_freq (update the model every train_freq steps | 1 |
| batch size | 64 |
| $\tau$ (the soft update coefficient: “polyak update”) | 0.005 |
| Entropy regularization coefficient | auto (learned automatically) |
| target_update_interval (update the target network every target_network_update_freq steps) | 1 |
| gradient_steps (how many gradient update after each step) | 1 |
| target_entropy | auto |
| action_noise | None |
| random_exploration | 0.0 |

Table 4.1: Hyperparameters for for the applied SAC algorithm. These are the default parameters set for SAC in the stable baselines library.

### 4.6.2 Proximal Policy Optimization Algorithm

The Proximal Policy Optimization(PPO) is the state-of-the-art off-policy algorithm that can be used for environments with continuous and discrete action spaces. It was tested in combination all three approaches for representing action space.
The hyperparameters that were applied during experiments are:

| **Hyperparameter** | **Value** |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| timesteps_per_actorbatch (timesteps per actor per update) | 256 |
| clip_param (clipping parameter $\varepsilon$) | 0.2 |
| entcoeff (the entropy loss weight) | 0.01 |
| optim_epochs (the optimizer’s number of epochs) | 4 |
| optim_stepsize | 0.001 |
| optim_batchsize | 64 |
| lam (advantage estimation) | 0.95 |
| adam_epsilon | 1e-05 |
| schedule (type of scheduler for the learning rate update ) | linear |

Table 4.2: Hyperparameters for for the applied PPO algorithm. These are the default parameters set for PPO in the stable baselines library.

## 4.7 CNN structure

The network consists of three constitutional layers, followed by a fully connected layer. The activation function is rectified linear unit (ReLU). The weights were randomly initialized through orthogonal initialization (as an orthogonal matrix).

| **Layer** | **Number of filters** | **Filter size** | **Stride** |
|---|---|---|---|
| Conv1 | 32 | 8 | 4 |
| Conv2 | 64 | 4 | 2 |
| Conv3 | 64 | 3 | 1 |
| Fc1 | 512 hidden neurons | | |

Table 4.3: Detailed description of the convolutional neural network that was used in the approach.

# 5 Evaluation

In order to train and evaluate the algorithms several test case scenarios were designed. By starting from a simple scenario and increasing difficulty it could be determined at what stage and whether the algorithm is experiencing difficulties and its performance overall.
Three lists of random object positions within the limits of the workspace were randomly generated. The first list contained only one position, the second - 20, the third - 50, each list was used in different scenarios. During one iteration in each scenario the object could be located on one of the positions of the corresponding list. The first scenario is considered to be the easiest one, as the object is always located on the same spot. In the second scenario the object can be located on one of the 20 possible positions, in the third one - on one of the 50 possible positions.
In the fourth scenario the object's position is randomly determined for each iteration - which corresponds to a list with infinite object positions, making it the most difficult scenario. At the same time it is the most important case as it does not include any limitations for the object's position (apart from being located within the camera view on the workspace) and if the agent can perform well on this scenario, it should be able to cope with the first three easier ones.

## 5.1 Experiments with the Soft-Actor-Critic Algorithm

Several experiments using the Soft-Actor-Critic(SAC) algorithm described in section 4.6.1 in combination with continuous action space were conducted. The results were unstable: repeating the same experiment with same parameters delivered different results. In some cases the training was successful. However sometimes the network was showing good performance which dropped after some training steps, resulting in 0% success rate at the end. Saving the model several times during training helped to retrieve the model that was giving a good performance. In some trainings the agent never achieved good performance. There might be several reasons for such unstable behavior, they will be discussed later on.

For all four test scenarios the best agents are represented in the table below. The agents show high success rates and relatively low variance. These best agents were retrieved during successful training runs, however if the experiments will be repeated again, the results might be worse because of instability and inconsistency of the algorithm.

| Test case | Success rate, % | Variance, % |
|---|---|---|
| 1 object position | 100 | 0 |
| 20 object positions | 100 | 0 |
| 50 object positions | 90 | 30 |
| Random position | 97 | 17 |

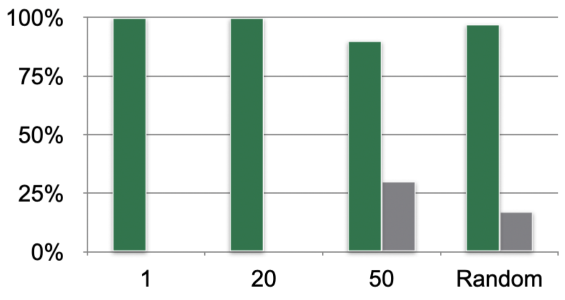


Figure 5.1: Results of the evaluation of best agents for each test scenario. These agents show high success rate and low variance. However if the trainings for each experiment will be repeated, the resulting agents might be not as successful as the ones represented in the table.

Sections 5.1.1 and 5.1.2 show statistics of several trainings with the SAC algorithm. Section 5.1.1 illustrates the unexpected behavior of success rate dropping during training and inconsistency of the training. Section 5.1.2 disclosures more information about one of the successful runs.

### 5.1.1 One Constant Positions Of The Object

Among successful experiment runs, where the position of the object was learned even under 10k training steps, there were some that were not stable. As an example a training that lasted 30k steps. After 14k steps the agent was performing with 100% sucessrate. However shortly after the performance dropped and never recovered.

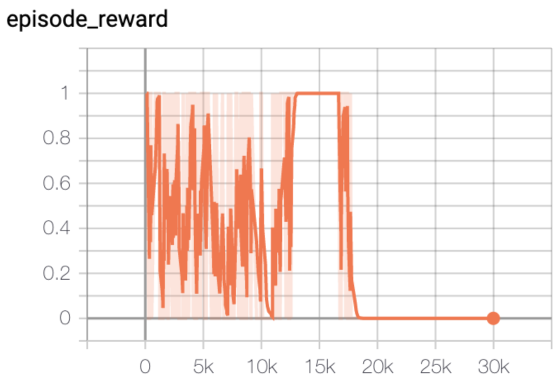


Figure 5.2: An example of unstable behavior of the algorithm based on the reward value: after 14k training steps the agent learned correctly where the object is located. Then the performance dropped to 0 and did not recover after that.

### 5.1.2 20 Constant Positions Of The Object

A list of 20 $(x, y)$ coordinates in the range of robot's action space were randomly generated. During each episode the object is located on the table with its $(x, y)$ coordinates one random position from the list. The goal of the experiment was to determine whether the network is able to learn how to grasp the object.

The training statistics is:

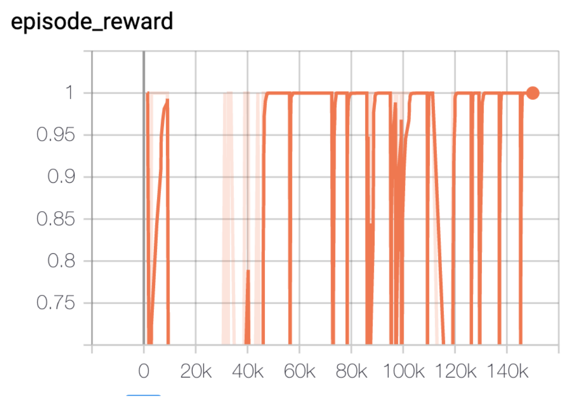


Figure 5.3: After about 45k training steps the success rate was almost always 1. The reason it dropped in some episodes might be due to inaccuracy in actions or some object positions occurred for the first time - the agent never saw them before which lead to bad performance, they were learned after that

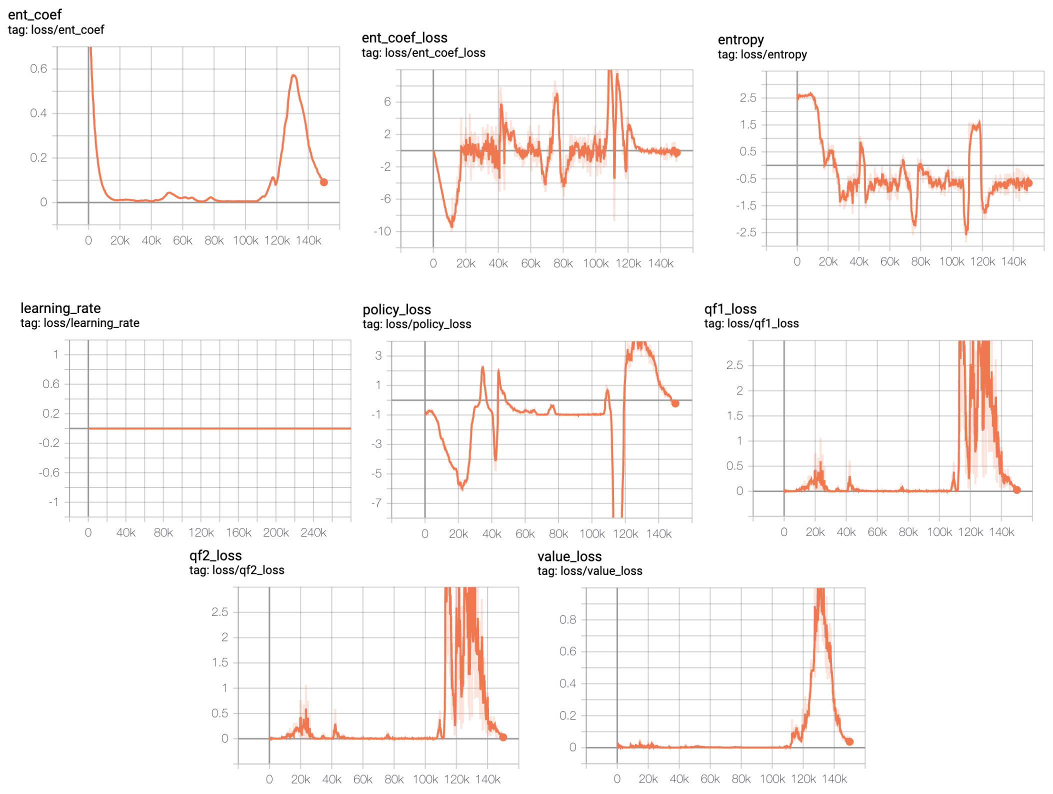


Figure 5.4: Statistics of the training: object's position is one of 20 possible ones, learn to grasp the object in 150k steps. At the end of the training the entropy is becoming constant which means that the algorithm is sure which action to take. The entropy coefficient is going down as well because the agent does not need to do much exploration anymore. Policy loss is going to zero, as well as the losses from both Q-networks and the target value network, which means the weights of the network are adjusted in an optimal way to achieve success.

The evaluation consisted of 300 episodes, 100% success rate - the task was completed success-

fully.
The 150000 steps of the training resulted in 43998 episodes. As max_episode_steps value was set to 50, in the beginning the majority of episodes lasted 50 steps, however at the end of the training the value shrank to 1, occasionally going up.

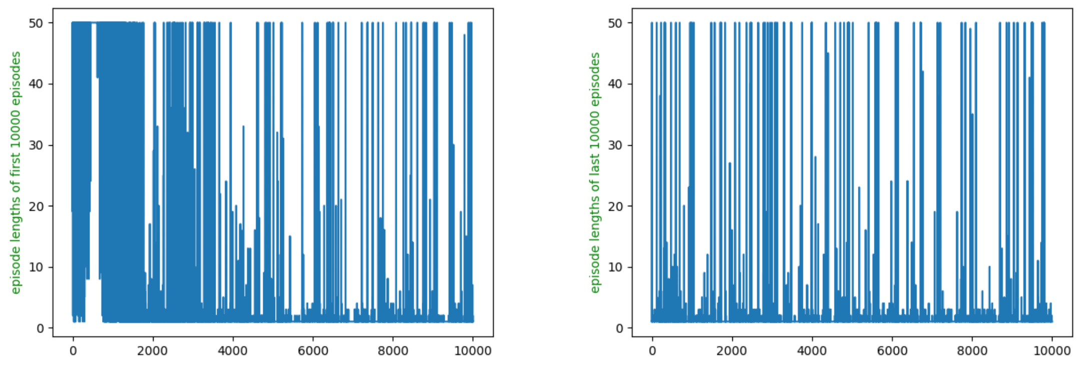


Figure 5.5: The first image shows the statistics of the length of first 10k episodes of the training, the second one - of the last 10k. It is noticeable, that in the second case the majority of episodes were short because the agent has already learned most of positions of the objects. In case of inaccuracy of actions or incorrect assumptions about the location of the object the episode lasted longer - up to 50 steps

### 5.1.3 Reasons for Unstable Training with the Soft-Actor-Critic Algorithm

Soft-Actor-Critic is a state-of-the-art reinforcement learning algorithm, it showed impressive results on such gym tasks as Humanoid-v2 and Ant-v2 [45]. The current grasping task is much more simple with a smaller action space, which is why the instable training results are very unlikely to be caused by the algorithm. Another reason could be the instablity of the environment. However, training with PPO was stable: repeating the experiments delivered same expected results, which means the task can be trained in the created environment. Another assumption about the cause of unstable behavior is the implementation of SAC by stable_baselines. The actual reason should be discovered in future work.

## 5.2 Experiments with the Proximal Policy Optimization Algorithm

### 5.2.1 Continuous Action Space

The algorithm performed very poorly in the continuous action space. It failed to learn even the simplest task where the object's position does not change.

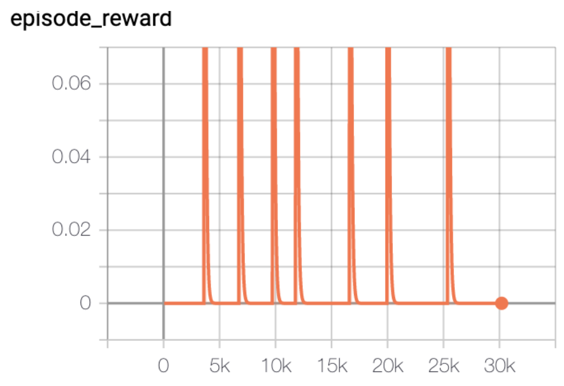


Figure 5.6: PPO performed extremely bad on the task without being able to learn the simplest set-up where the position of the object is constant during the whole training. In the first graphics there are some cases of reward 1 - there are so rare that they are most likely accidental.

### 5.2.2 Discrete Action Space without Gripper Rotation

Following the idea of Zeng et al. in [1], [20], the action space was discretized to consist of a 50x50 grid. The action (x,y) would mean the gripper would execute a planar grasp in the middle of the cell (x,y).
The results of the experiments with 1, 20 and 50 constant positions of the object were successful: during evaluation for all these cases the agent performed with almost 100% success rate.

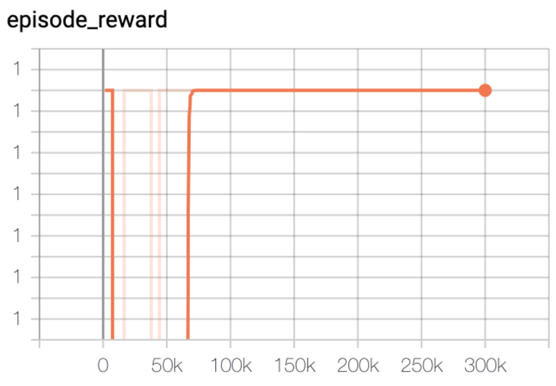


Figure 5.7: 300k step training of PPO on random position of the object. After approximately 60k steps the agent managed to establish a good policy and cope with 100% success rate on training cases.

For random object position after 300k training steps the success rate was 98% with 11% variance. The model created in another training with 500k steps showed 99.5% success rate with 7% variance during 1000 evaluation steps.

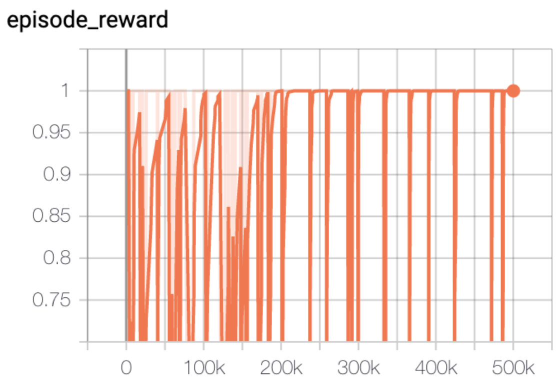


Figure 5.8: 500k step training of PPO on random position of the object. The evaluation of the model at the end showed 99.5% success rate with 7% variance.

For all four test scenarios the best agents are represented in the table below. In the first three scenarios the agent coped perfectly with the task: 100% success rate with 0 % variance. The success rate in the forth scenario is 99.5% with higher variance. It might be due to the fact that during evaluation some positions of the object were never seen by the robot in the training, which is why it was not able to grasp it or it took several attempts to grasp the object, as opposed to the first three scenarios, where the object was grasp from the first attempt.

| **Test case** | **Success rate, %** | **Variance, %** |
|---|---|---|
| 1 object position | 100 | 0 |
| 20 object positions | 100 | 0 |
| 50 object positions | 100 | 0 |
| Random position | 99.5 | 7 |

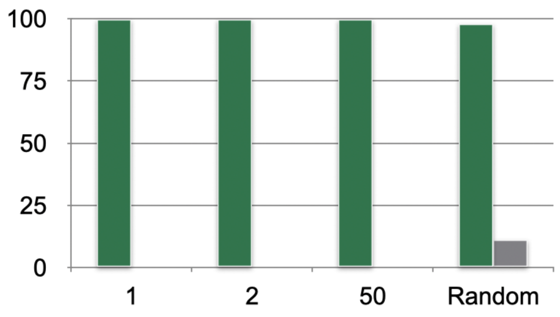


Figure 5.9: The PPO algorithm in combination with the discrete action space showed high success rates for all four test scenarios. In the fourth scenario during evaluation some of the object's positions were new to the robot, which is why he did not manage to grasp them right away or in 2% at all. Increasing the number of training steps might close this gap.

The idea of discretizing the action space was formulated in [59]. Although the idea is simple, it can drastically improve the performance of baseline on-policy algorithms.

### 5.2.3 Discrete Action Space with Gripper Rotation

In this scenario the action space is discrete including gripper rotation. The test object has a random rotation from 0° to 90°, however it is possible for the gripper to grasp it at any times if the correct gripper position is determined.

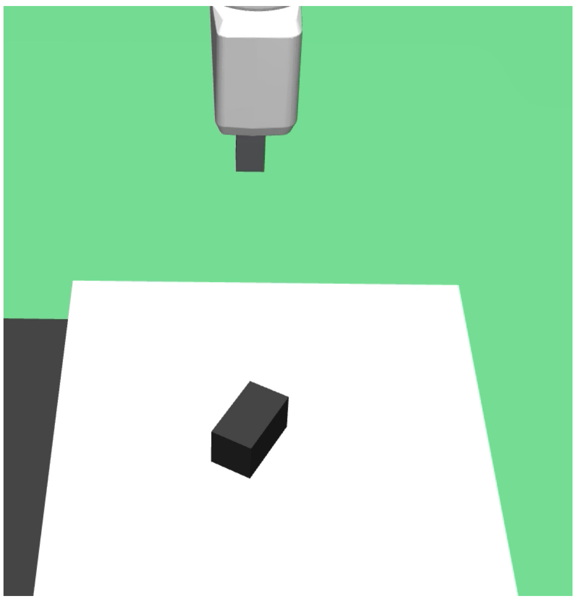


Figure 5.10: The target object has a random rotation from 0° to 90°, however it is possible for the gripper to grasp it at any times if the correct gripper position is determined.

For all four test scenarios the best agents are represented in the table below. The results are not as good as in the previous case, where there was no rotation of the object and the gripper. The reason might be due to the fact that the task was more complicated with the bigger action space.

| **Test case** | **Success rate, %** | **Variance, %** |
|---|---|---|
| 1 object position | 100 | 0 |
| 20 object positions | 98 | 14 |
| 50 object positions | 90 | 30 |
| Random position | 98 | 15 |

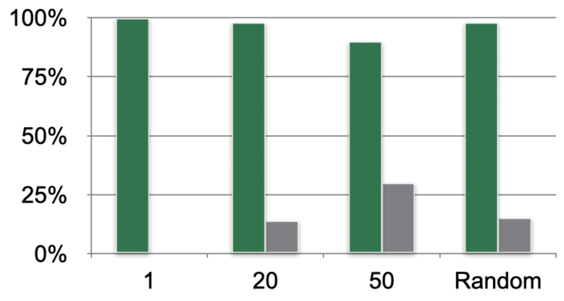
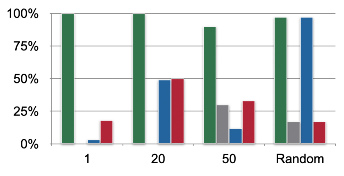


Figure 5.11: The PPO algorithm in combination with the discrete action space and gripper rotation showed high success rates for all four test scenarios. In the third scenario with 50 possible positions the success rate is lower and the variance is higher than in the fourth case, which is a not expected behavior as the third scenario is meant to be easier than the fourth one. Increasing the number of training steps might be helpful to get better results for the third case as the agent will experience more object positions and thus can better learn.

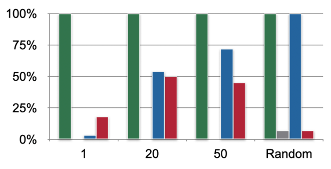
During evaluation it was noticeable that the gripper made several attempts before successfully grasping the object. Often during these attempts the object was slightly shifted and rotated, accidentally resulting in more convenient for the gripper positions to be grasped. This kind of approach might be effective in simulation, however it is not desired in the real-world setup - the gripper's actions should be precise and be aimed at grasping the object at first attempt.

## 5.3 Generalization Test

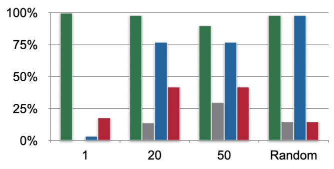
For each experiment four different training scenarios were conducted: one, 20, 50 and random object positions. For first three scenarios the agents were evaluated using the same list of object positions that they were trained on. This brings up the following question: how well would these agents perform on the general case with the object position being randomly assigned on the working space?
The generalization test was conducted. The results showed, that the agents that were trained on specific object positions were not able to show good performance on the general case where the object position was random.
The results of the generalization test together with evaluation results are shown in figure 5.12. The expected behavior is shown in figure 5.12b: with the rising number of object positions seen in the training the success rate for the general case rises and the variance becomes lower.
In figure 5.12a this idea is violated by the case with 50 possible positions of the object. This can be explained by the the fact that the trained agent for this case performed worse during its own evaluation: lower success rate and higher variance as in the case with 20 objects. Which suggests that the agent was not good trained overall, which is why he performed very poorly at the generalization test.
In figure 5.12c the agents trained for 20 and 50 object positions showed a high success rate for the general case - over 75%. However the variance for both cases is over 40%, which means the agents were often not entirely sure which action to take and after several attempts succeeded to chose the right one. Such high variance signifies that these trained agents cannot be used for the general case as they are too uncertain in their actions.



(a) SAC with continuous action space



(b) PPO with discrete action space without rotation



(c) PPO with discrete action space with rotation

🟩 Success rate during evaluation
⬜ Variance during evaluation
🟦 Success rate at generalization test
🟥 Variance at generalisation test

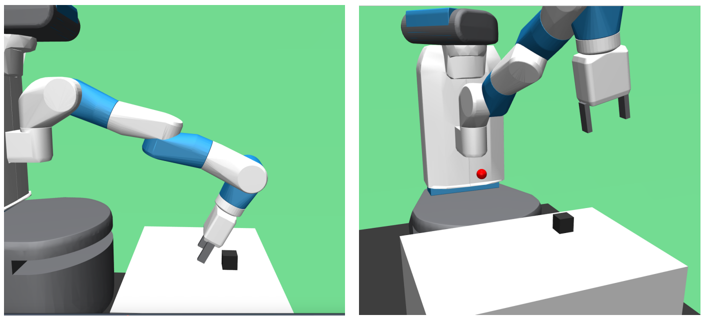
Figure 5.12: The diagrams show results of evaluation and generalization tests for three experiments: SAC with continuous action space, PPO with discrete action space without rotation, PPO with discrete action space with rotation. The x-axis represents the number of object positions that the agent was trained on, with "Random" being random object position during training. Green represents the success rate of the agent during the evaluation on the scenario that it was trained with, grey is the corresponding variance. Blue is the success rate at the generalization test, red is its variance.

## 5.4 Simulation Inaccuracies

During the setup of simulation some unexpected behaviors occurred that had a significant influence on the results of the experiment. Until these difficulties were eliminated, it was impossible to train a successful agent.

### 5.4.1 The Angle Of The Gripper

The action space consists of target $(x,y)$ coordinates of the point that the gripper has to achieve before going down. The step() function computes the difference between target and current positions, the difference is then applied to the simulation using inverse kinematics. The inverse kinematics is calculated by solving the convex optimization problem. The optimization problem is defined combining description of bodies that are used in the simulation and physical constraints such as forces, friction, etc.
One of the challenges that was faced during the set-up is that there can be many solutions to the problem of gripper achieving the target $(x,y)$ position. In some cases the gripper developed an angle in different plains, successfully reaching target position but the angle made it impossible to compute a successful grasp.



Setting the robot to the starting position for every step helped to correct that inaccuracy for the current task. However, this approach is time-consuming and might be not applicable for the similar problems in different set-ups. The possible better solution to this problem might be to set additional constraints for the position of joints of the robot that would effect the solution of the inverse kinematics problem.

### 5.4.2 Slipping Of The Object

The first version of grasping consisted of completely closing the gripper and going up with an object. However, the object did not stay in the gripper - it slipped down. This behavior was not expected as physical forces such as friction were set to the default values. The solution to the problem was to keep closing the gripper while going up - it helps to prevent the object from slipping.

# 6 Conclusion

In this thesis the system for robotic vision-based grasping with reinforcement learning in simulation was presented. Two reinforcement learning algorithms were tested in combination with different representations of action spaces. The reward for taken action was sparse. Discretizing action space can be a powerful tool that helps to achieve high success rate of a task. The generalization tests showed that with the increasing amount of randomness during experiments, the trained agents learn to perform better on the general case that does not include limitations.
Deep reinforcement learning helps to combine perception and control. The DRL approach can be applied to develop end-to-end self-supervised grasping systems that are data-driven and require little to none human involvement during training.
Training the robot in simulation is helpful for fast collecting large quantities of data with little cost.

### 6.0.1 Discussion

Two reinforcement learning algorithms PPO and SAC were applied. The SAC algorithm in combination with continuous action space showed instable results, which might be due to a bug in the implementation from the stable baselines library. PPO for the case without rotation of the gripper in combination with discrete action space showed good consistent results, with success rate over 99% for all test scenarios. PPO applied to continuous action space showed poor results not being able to solve even the simplest task where the object is always located on the same spot. PPO in combination with discrete action space including rotation of the gripper showed stable behavior with success rate over 90% for all test scenarios. However in this experiment during evaluation the gripper often needed several attempts to grasp the object, slightly adjusting the position of the object during each attempt in order to make it more convenient to grasp, which is not a desirable behavior for the real-world setup.
Additionally the generalization test was conducted. The results showed that the agents trained for a specific case of predefined finite small number of positions do not perform well on the general case where the object position is random. The more randomness the agent experiences during training, the better its performance is on the general case.

### 6.0.2 Future Work

The conducted approach included a set of limitations which suggest directions for future work. First of all, the robot should be able to grasp target objects of different shapes, sizes, forms and colors. What is more, the scenario of cluttered environment where there are multiple objects on the workspace partially covering each other should be researched. For more general problems planar grasps might not be sufficient to grasp the object, so pitch- and roll-angles of the gripper should be

included in the action space of the robot. Other reinforcement leaning algorithms should be tried out as they might be more effective for the grasping problem.
The ultimate goal is to create a real-world system for robotic grasping. In future researches the transition from simulation to the real world environment must be analyzed in order to effectively transfer the agent pretrained in simulation, so that either the reality gap problem is eliminated during training in simulation or as little as possible additional real-world training is required.

# A List of Figures

# B List of Tables

# C Bibliography

[1] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[2] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[3] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

[4] Van-Duc Nguyen. Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, 1988.

[5] Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[6] Karun B Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[7] Dan Ding, Yun-Hui Liu, and Shuguo Wang. Computing 3-d optimal form-closure grasps. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 4, pages 3573–3578. IEEE, 2000.

[8] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2013.

[9] Staffan Ekvall and Danica Kragic. Learning and evaluation of the approach vector for automatic grasp generation and planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4715–4720. IEEE, 2007.

[10] Joseph Redmon and Anelia Angelova. Real-time grasp tection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1316–1322. IEEE, 2015.

[11] `http://pr.cs.cornell.edu/grasping/rect_data/data.php`.

[12] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps

with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[13] Alexander Kasper, Zhixing Xue, and Rüdiger Dillmann. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012.

[14] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1957–1964. IEEE, 2016.

[15] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*, 2018.

[16] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[17] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[18] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[19] Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergström, Danica Kragic, and Antonio Morales. Mind the gap-robotic grasping under incomplete observation. In *2011 IEEE International Conference on Robotics and Automation*, pages 686–693. IEEE, 2011.

[20] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019.

[21] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[22] Corey Goldfeder and Peter K Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011.

[23] Andrew T Miller and Peter K Allen. Graspit! a versatile simulator for robotic grasping. 2004.

[24] Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, Sami Moisio, Jeannette Bohg, James Kuffner, et al. Opengrasp: a toolkit for robot grasping simulation. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 109–120. Springer, 2010.

[25] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[26] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[27] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[28] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE signal processing magazine*, 32(3):53–69, 2015.

[29] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[30] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[31] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13, 2019.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[34] Hengyue Liang, Xibai Lou, and Changhyun Choi. Knowledge induced deep q-network for a slide-to-wall object grasping. *arXiv preprint arXiv:1910.03781*, 2019.

[35] Lars Berscheid, Thomas Rühr, and Torsten Kröger. Improving data efficiency of self-supervised learning for robotic grasping. *arXiv preprint arXiv:1903.00228*, 2019.

[36] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[37] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE, 2018.

[38] Abdeslam Boularias, James Andrew Bagnell, and Anthony Stentz. Learning to manipulate unknown objects in clutter by reinforcement. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[39] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[40] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[42] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[43] `https://spinningup.openai.com/en/latest/`.

[44] Somil Bansal, Roberto Calandra, Kurtland Chua, Sergey Levine, and Claire Tomlin. Mbmf: Model-based priors for model-free reinforcement learning. *arXiv preprint arXiv:1709.03153*, 2017.

[45] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[47] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[48] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[49] 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*, pages 5026–5033. IEEE, 2012.

[50] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[51] OpenAI. Fetchpickandplace-v0. `https://gym.openai.com/envs/FetchPickAndPlace-v0/`, 2018.

[52] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[53] `https://www.opengl.org/`.

[54] `http://www.mujoco.org/book/index.html`.

[55] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[56] `https://openai.com/`.

[57] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[58] Antonin Raffin. Rl baselines zoo. `https://github.com/araffin/rl-baselines-zoo`, 2018.

[59] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*, 2019.