Deep Reinforcement Learning for Robotic Grasping from Octrees

Learning Manipulation from Compact 3D Observations

Andrej Orsula
M.Sc. in Robotics

Master's Thesis

AALBORG
UNIVERSITY

Department of Electronic Systems
Aalborg University
https://es.aau.dk

Master's Thesis

Title:
Deep Reinforcement Learning for Robotic Grasping from Octrees

Programme:
M.Sc. in Robotics

Author:
Andrej Orsula[1]

Supervisor:
Simon Bøgh

Number of Pages:
54

Submission Date:
May 28, 2021

Abstract:

TODO: Abstract
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tempus faucibus sodales. Nulla tempus sem eget tortor lacinia egestas. Nam ornare sed enim volutpat egestas. Nulla tristique in risus in congue. Morbi nec mauris cursus, lacinia elit et, sagittis lacus. Curabitur in erat lectus. Nunc in sapien rutrum mauris pretium ullamcorper eget id ante. Nunc at sem urna. Quisque gravida velit eget nulla varius varius. Vivamus elit lectus, suscipit a nunc at, pellentesque ullamcorper magna. Nullam gravida tellus metus, sed vehicula erat rutrum vel. Mauris varius ligula quis elit ornare viverra. In vitae erat metus. Aenean lectus diam, hendrerit at rutrum sit amet, vestibulum ut nulla. Etiam porttitor urna eget risus auctor molestie. Aenean convallis dignissim vulputate. Praesent maximus facilisis vehicula. Integer nec fermentum leo, et interdum tellus. Quisque porttitor hendrerit laoreet. Suspendisse aliquet ex a metus congue dapibus. Nulla libero mauris, iaculis cursus dui hendrerit, hendrerit feugiat ligula. Donec hendrerit magna leo, eget molestie orci placerat a. Maecenas viverra urna eget dolor tempor, ut lacinia elit eleifend. Nulla fermentum velit ac fringilla laoreet. Nullam scelerisque placerat mauris vitae pellentesque. Praesent et metus fringilla leo pretium convallis. Suspendisse tempor imperdiet felis eu tristique.

[1]aorsul16@student.aau.dk

Contents

Summary

Preface

This Master's Thesis is written by Andrej Orsula as his final work of M.Sc. programme in Robotics at Aalborg University during the academic year 2020/21.

Acknowledgements

Special thanks goes to Simon Bøgh for his supervision, guidance and numerous discussions throughout the whole process that helped shaping this project. Moreover, I must express a very profound gratitude to my mum, dad, sister and brother for their love and everlasting support.

Additional Resources

The source code developed during this project is freely available on the following GitHub repository.

- https://github.com/andrejorsula/drl_grasping

All readers interested in reproducing the results from this work are welcome to use pre-built Docker images that can be pulled from Docker Hub.

- https://hub.docker.com/r/andrejorsula/drl_grasping

This manuscript can be accessed on the following GitHub repository, together with additional resources such as raw data collected during the experiments.

- https://github.com/andrejorsula/master_thesis

Recordings associated with this work are available under the following YouTube playlist.

- https://youtube.com/playlist?list=PLzcIGFRbGF3Qr4XSzAjNwOMPaeDn5J6i1

Glossary

CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DOF	Degree of Freedom
DRL	Deep Reinforcement Learning
GAN	Generative Adversarial Network
IK	Inverse Kinematics
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Process
NN	Neural Network
PBR	Physically Based Rendering
PILCO	Probabilistic Inference for Learning Control
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROS	Robot Operating System
SAC	Soft Actor Critic
SDF	Simulation Description Format
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic
TQC	Truncated Quantile Critics

1 Introduction

Grasping is a fundamental manipulation skill that is essential for a variety of everyday tasks. Stacking, inserting, pouring, cutting and writing are all examples of such tasks that require an object or a tool to be firmly grasped prior to performing them. A hierarchy of subroutines can be assembled together in order to accomplish more complex goals, which in turn requires grasping of diverse objects that can differ in their appearance, geometry as well as inertial and mechanical properties. Despite the uniqueness this might bring to each individual grasp, a versatile robot should generalize over different objects and scenarios instead of treating them as distinct subtasks.

Task-specific algorithms are often analytically developed for a specific gripper on a set of objects via time-consuming approach. Despite effectiveness of such methods, they usually lead to a solution that lacks the required generalization and even slight differences in the process or manipulated objects might require manual reprogramming (Sahbani et al., 2012). Empirical approaches were introduced to overcome the difficulties with analytical grasping by progressively learning through sampling and training. In this way, supervised learning provides a way to learn grasp synthesis from a dataset that is labelled with analytical grasp metrics, however, this approach requires a large volume of data in order to achieve the desired generalization (Mahler et al., 2017). Although imitation learning allows robots to quickly learn simple grasps (Zhang et al., 2018), the amount of required human expert demonstrations can also become too costly and time-consuming before a general policy is learned. Reinforcement learning (RL) (Sutton and Barto, 2018) could offer a solution to this problem, as self-supervision provides the means for a robot to progressively become better at grasping via repeated experience and minimal human involvement. The popularity of RL has significantly increased in recent years, especially due to the noteworthy results obtained by deep reinforcement learning (DRL). Several publications demonstrated that DRL can be used to achieve human level performance in tasks such as playing Atari games (Mnih et al., 2015), or even beating world champions in the boardgame Go (Silver et al., 2017) and real-time strategy game StarCraft II (Vinyals et al., 2019). Moreover, Schrittwieser et al. (2020) established just how far DRL has come with a single algorithm that can achieve superhuman performance by learning a model without any prior knowledge of the game rules in multiple domains, i.e. Go, Chess, Shogi and 57 Atari games.

While games with a well-defined set of rules are popular benchmarks for developing and testing algorithms, RL has also been employed for several real-world applications such as finance (Fischer, 2018), industrial process control (Nian et al., 2020), scheduling (Shyalika et al., 2020) and robotic manipulation (Kroemer et al., 2021). Among these, RL has likewise gained popularity in robotic grasping due to its flexibility. However, there are many challenges in applying RL to solve robotics problems with high-dimensional continuous action and observation spaces (Kroemer et al., 2021). It can be difficult to design a suitable reward function because robotics tasks such as grasping require multiple objectives to be optimised simultaneously, e.g. grasp an object with as little energy while avoiding all obstacles. Furthermore, collection of training data on physical robots is a time-consuming and potentially unsafe process, therefore, robotics

simulators are commonly utilised because they provide a less expensive and much faster way to train RL agents. Unfortunately, this often introduces a reality gap between the virtual and real-world domain that needs to be addressed via sim-to-real approaches.

End-to-end DRL approaches for solving robotic grasping have become more attractive in recent years due to their ability to directly map raw observations into actions, where visual observations in form of 2D RGB and 2.5D RGB-D images are the most common. Features from these images are typically extracted by utilising 2D convolutions, which unfortunately do not provide the required level of generalisation over the depth and spatial orientation (Gualtieri et al., 2018). Since the underlying representation of the scene in which robot operates is 3D, representing observations with 3D data structures might provide benefits in terms of generalisation. Therefore, this work aims to investigate the advantages of utilising 3D representation for observations in the context of robotic grasping.

The primary focus of this work is to apply DRL to robotic grasping of diverse objects with the use of compact 3D observations in form of octrees. The key contributions are listed below.

- **Simulated Environment for Grasping with Domain Randomization** – A novel simulated environment for robotic grasping in the context of RL research is developed in this work. It utilises realistic 3D scanned objects and extensive domain randomization in order to enable sim-to-real transfer. The environment is developed on top of Ignition Gazebo[1] robotics simulator that is interfaced by the use of Gym-Ignition (Ferigo et al., 2020) to provide compatibility with other OpenAI Gym environments (Brockman et al., 2016).

- **Octree Observations for End-to-End Grasping with DRL** – This work introduces a novel approach for utilising octree-based visual observations for end-to-end robotic grasping with DRL. Octrees provide an efficient 3D data representation with a regular structure that enables the use of 3D convolutions to extract spatial features. Furthermore, the use of 3D representation promotes invariance to camera pose, which further improves sim-to-real transfer to various real-world setups.

- **Curriculum Learning** – A curriculum was developed for the grasping task in order to progressively increase its difficulty. Besides common techniques of increasing the workspace size and number of objects, this work investigates the effect of decomposing the full task into sequential sub-tasks with distinct termination states.

- **Comparison of Three Actor-Critic RL Algorithms** – Three off-policy actor-critic RL algorithms are compared on the developed grasping environment with the proposed octree observations. The compared algorithms are TD3, SAC and TQC.

This thesis has the following organisation. First, various approaches for solving robotic grasping are presented and compared in chapter 2, alongside 3D data representations and their applicability with deep learning. Chapter 3 presents relevant theory and notation that aids with understanding of this thesis. It is followed by chapter 4 that formulates the full problem that this work addresses. Chapter 5 then presents the concrete implementation steps that enable subsequent experimental evaluation that is reported in chapter 6. Finally, chapters 7 and 8 discuss the results and conclude the work presented in this thesis.

[1] https://ignitionrobotics.org

2 Related Work

Robotic manipulation and grasping is a field that has been extensively studied for decades via magnitude of different approaches. This chapter outlines some of the notable methods, while mainly focusing on contributions that employ model-free reinforcement learning due to their relevance for this project.

2.1 Analytical Approaches

Analytical approaches determine grasps that satisfy target requirements through kinematic and dynamic formulations (Sahbani et al., 2012). These methods typically analyze the geometry of target object and utilised gripper in order to generate a suitable grasp pose, which can then be reached by using a separate motion planner. The approach was introduced by Nguyen (1987) through formulation of objectives for constructing stable force-closure grasps on polyhedral objects. By modelling objects as triangular mesh or 3D point cloud, force-closure grasps were later extended to remove model restrictions (Yun-Hui Liu et al., 2004). Several analytical metrics for estimating the quality of grasps were also introduced over the years to quantify good grasps (Roa and Suárez, 2015), many of which have found their applicability beyond analytical approaches.

Expert human knowledge of robot in a specific task is required to develop these algorithm, which allows them to achieve very efficient operation on a number of selected objects due to direct transfer of this knowledge. However, this also introduces a limitation because performance is restricted only to the predicted situations and scalable generalisation to novel objects is often unfeasible due to computation complexity that arises from the number of considered conditions (Sahbani et al., 2012). Moreover, geometric models of objects might not be available before interaction is required, and partial occlusion of objects in setups with passive perception similarly limits the use of geometrical analysis.

2.2 Supervised Learning

Empirical methods were introduced to overcome shortcomings and difficulties of analytical approaches by combining sampling and training to achieve learning, which in turn reduces or removes the need to manually develop a model. A common approach is to use supervised learning to detect grasp poses by training on a dataset that is labelled with indication about what regions contain grasps (Saxena et al., 2008; Lenz et al., 2015). Alternatively, a combination of analytical grasp quality metrics can be used to provide a more fine-tuned labelling of data (Mahler et al., 2017, 2018, 2019; Lundell et al., 2019). Saxena et al. (2008) applied supervised learning for detection of grasps on previously unseen objects by using handcrafted features from two or more RGB images of the scene in order to identify points in each image that correspond

to grasp locations. They then determined the 3D position of detected grasps via triangulation and used custom heuristics to estimate orientation before planning a collision-free path.

Due to the significant advancements of deep learning (DL) in recent years, there has been a trend towards applying DL for robotic grasping. Lenz et al. (2015) developed a framework that used DL to train two separate neural networks (NNs) on RGB-D data, where a small network was used to search for image patches with potential grasp candidates, and a larger network then ranked these candidates to select the most optimal grasp. With this work, Lenz et al. demonstrated the advantage of using DL instead of time-consuming design of hand-crafted features for robotic grasping. Popularity of convolution neural networks (CNNs) in computer vision applications also inspired their use for robotic grasping, which resulted in more accurate systems for predicting grasps in RGB-D images (Redmon and Angelova, 2015; Kumra and Kanan, 2017). The use of CNN also provides computationally efficiency, which allowed Morrison et al. (2018) to synthesise grasps from depth images in real-time and perform closed-loop control.

Besides 2D images, supervised DL methods have also been applied to 3D data representations. Approach by ten Pas et al. (2017) randomly samples a large number of grasp candidates uniformly from the object surface using a point cloud, without a need to segment the individual objects first. They subsequently encode a region of interest around each grasp candidate as a stacked multi-channel projected image, which is then scored by the use of CNN classifier. By selecting the grasp candidate with the highest score, they were able to demonstrate a success rate of 93% on novel objects in a dense clutter. Lundell et al. (2019) used DL on voxel grid for shape completion of partially observed objects in order to obtain multiple predictions of the full object shape. These predictions were then used to jointly evaluate analytical grasp metrics for all grasp candidates, which they sampled from a mesh constructed as mean of all shape predictions. With this work, Lundell et al. demonstrated improved success rate over methods using only a partial view or a single shape estimate.

Although supervised learning approaches can achieve high success rate, their main disadvantage is the large volume of labelled data required to effectively learn grasp generation. The process of labelling a dataset is generally automated because it is non-trivial to perform it manually due to the multitude of ways in which an object can be grasped, furthermore, human labelling introduces bias (Pinto and Gupta, 2015). However, the data collection itself is still very costly if performed on a physical setup. Levine et al. (2016) used a setup shown in Figure 2.1 with up to fourteen robots to collect 800,000 grasp attempts over the course of two months. To avoid this time-consuming process, majority of recent work relies on synthetically generated datasets. As an example, Mahler et al. (2017) achieved 99% precision by training on a dataset with 6.7 million point clouds of more than 10,000 unique 3D models, each containing grasps and corresponding analytical grasp metrics. Generalization to other gripper types is also limited and the entire dataset needs to be updated in order to support new types, which is why they created a new dataset of 2.8 million point clouds in 2018 for vacuum-based grippers. This issue was later addressed by creating a common dataset for both parallel-jaw and vacuum-based grippers

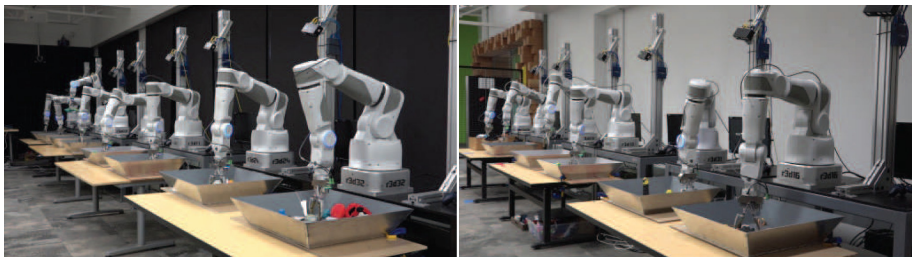


Figure 2.1: A setup with fourteen robots used by Levine et al. (2016) to collect 800,000 grasps.

by using a more complex and general analytical metric based on object's expected resistance to forces and torques (Mahler et al., 2019).

2.3 Imitation Learning

Another empirical method is based on the process of learning tasks from demonstrations, called imitation learning. Demonstrations are normally represented as trajectories that contain states or state-action pairs, which can be obtained in several different ways such as teleoperation, kinesthetic teaching, or motion capture (Osa et al., 2018). In this way, imitation learning aims to provide robots with a desired behaviour by simply showing a sequence of actions instead of manually programming them.

Behavioural cloning is the simplest form of imitation learning, in which a policy that directly maps states to actions is learned through techniques such as non-linear regression or support vector machines (Osa et al., 2018). Recently, Zhang et al. (2018) showed that DL allows behavioural cloning to be an effective way for robots to acquire complex skills. They used a virtual reality headset and hand-tracking controller to acquire teleoperated demonstrations in the form of RGB-D images, which were subsequently used to train a deep policy by the use of CNN. With this approach, Zhang et al. managed to train a simple grasping task with one object to 97% success rate while using 180 distinct demonstrations. Learning from observation is an emerging category that aims to learn policy similarly from visual demonstrations but without any labels associated with them, where the state might not be fully known (Kroemer et al., 2021).

Even though imitation learning provides a quick way of acquiring new policies, demonstrations usually do not contain all possible states that the robot might experience because collecting expert demonstrations for all scenarios can become too expensive and time-consuming (Osa et al., 2018). For this reason, the learned policy might struggle to generalize to novel objects and situations.

2.4 Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto, 2018) aims to learn an optimal policy that maximises the total reward that is accumulated during a sequential interaction with the environment. Unlike supervised and imitation learning, RL does not require any labelled datasets or demonstrations. Instead, RL agent collects information about the goal it is trying to reach through direct interaction with the environment, all while improving its own policy. This process makes RL algorithms heavily dependent on reward signals, which is why it is important to design a reward function that induces learning towards reaching the desired goal. In addition to the popular benchmarks such as board- and video games that are commonly used to develop and test new algorithms, RL has been applied to several manipulation tasks over the years. Recently, the combination of DL and RL termed deep reinforcement learning (DRL) has become a popular choice for end-to-end control in robotics research, where sequential actions are learned directly from raw input observations.

RL algorithms can be categorised depending on whether a model of the environment transition dynamics is used, i.e. model-based or model-free methods. Model-based RL algorithms have access to the model or learn it during the training, which allows the agent to predict state transitions and use such knowledge to directly learn the policy (Polydoros and Nalpantidis, 2017). If the model is correct, model-based RL allows the learning process to be much more sample efficient than any model-free method, which gives model-based RL algorithms a great potential

for applications within robotics. From this category, Probabilistic Inference for Learning Control (PILCO) (Deisenroth and Rasmussen, 2011) is model-based framework that has been applied also for manipulation tasks. With only 90 seconds of experience, Durrant-Whyte et al. (2012) applied PILCO to learn stacking task while incorporating collision avoidance into the planning. However, accurate models are rarely available and learning them can be very challenging in complex manipulation environments. Another difficulty emerges if an agent is trained and utilised in two different domains, which introduces bias to the model and ultimately leads to sub-optimal performance. For this reason, it might be significantly easier to learn a policy with model-free RL than to use model-based RL to learn transition dynamics, while achieving a similar level of performance (Kroemer et al., 2021). Therefore, the rest of this section will focus on contributions that use model-free methods to empirically learn a policy entirely from experience that is acquired via trial-and-error.

The use of model-free DRL for robotic grasping has been explored by several works in last few years. Many of these contributions typically focus on the final performance using a single object (Popov et al., 2017; Haarnoja et al., 2018a; Zhan et al., 2020) or a limited number of objects with simple geometry such as boxes, cylinders or pyramids (Tobin et al., 2017; Gualtieri et al., 2018; Gualtieri and Platt, 2018; Zeng et al., 2018; Liu et al., 2019; Joshi et al., 2020; Daniel, 2020; Iqbal et al., 2020). More recent works strive to increase this variety by training on random objects with more complex geometry (Quillen et al., 2018; Breyer et al., 2019; Wu et al., 2020; Kim et al., 2020), where the best diversity is achieved by training directly on real robots (Kalashnikov et al., 2018). Training on diverse objects allows DRL agent to learn a policy that provides the required generalisation, which is considered to be one of the most important challenges for learning-based robotic grasping (Quillen et al., 2018).

Contributions that apply DRL to robotic grasping also differ considerably in the utilised action space, where two main categories of approaches can be observed. The first category is based on pixel-wise action space (Zeng et al., 2018; Gualtieri and Platt, 2018; Liu et al., 2019; Daniel, 2020; Wu et al., 2020), in which the agent selects a pixel from the observed images in order to determine the position where an action primitive should be executed, see example in Figure 2.2. The individual pixels are usually mapped to positions in Cartesian space and the action primitive is normally the entire grasp trajectory. Grasp orientation around vertical axis is commonly discretised by extending the action space to a set of images that are uniformly rotated copies of the original image (Zeng et al., 2018; Daniel, 2020). Orientation was extended to 3D by Wu et al. (2020) via three image channels for continuous roll, pitch and yaw angles. These action primitives can also be applied for other skills such as pushing, which Zeng et al. (2018) used to allow agent to disturb objects before grasping them in order to clear space for fingers in scenes with densely packed objects. The second category of approaches in terms of action

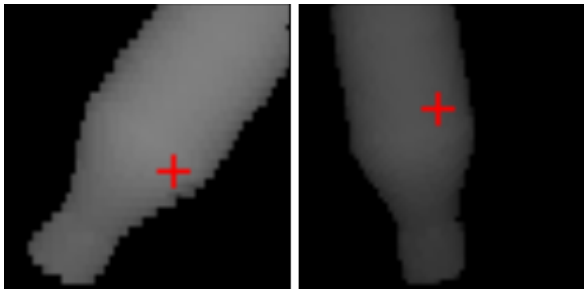


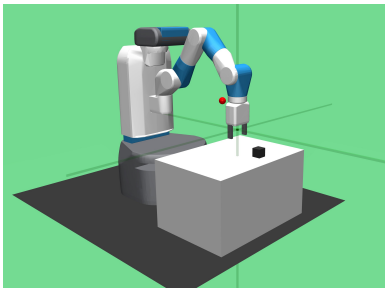
Figure 2.2: Example of pixel-wise action space for grasping by Gualtieri and Platt (2018), where the observed depth map is marked with a red cross that indicates the position for execution of the next grasp action primitive.

space are those that directly control robot motion (Quillen et al., 2018; Kalashnikov et al., 2018; Breyer et al., 2019; Joshi et al., 2020; Zhan et al., 2020; Kim et al., 2020; Iqbal et al., 2020), which is often expressed as Cartesian displacement of gripper pose in terms of relative translation (d_x, d_y, d_z) and relative vertical rotation d_ϕ. Control of the full 3D orientation in this way is uncommon, which is presumably due to the significantly increased complexity of such problem. However, there are works that directly control joints without the use of Inverse Kinematics (IK), e.g. Popov et al. (2017) uses continuous joint velocities. The gripper action also differs within this category, where some approaches automatically close the gripper after moving below certain height (Quillen et al., 2018) and others allow only closing of the gripper which subsequently terminates the episode (Kalashnikov et al., 2018; Joshi et al., 2020). A special formulation of action space was introduced by Gualtieri et al. (2018), where actions are grasp pose candidates sampled by the use their previous work ten Pas et al. (2017). Similar approach was adopted by Osa et al. (2017) for multi-finger grippers with a policy that also selected a grasp type in addition to grasp pose.

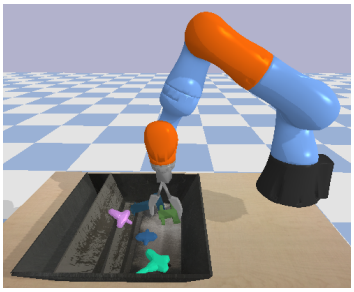
Although the observation space is for some manipulation tasks defined in form of states extracted from the simulation, e.g. gripper and object position (Popov et al., 2017; Haarnoja et al., 2018a), the vast majority of RL grasping research relies on visual image observations that are combined with CNNs. Among these are RGB image (Tobin et al., 2017; Kalashnikov et al., 2018; Quillen et al., 2018; Kim et al., 2020; Iqbal et al., 2020), depth map (Gualtieri and Platt, 2018; Breyer et al., 2019; Wu et al., 2020) or RGB-D (Zeng et al., 2018; Liu et al., 2019; Daniel, 2020). A single camera is commonly mounted statically in the environment and the preprocessing of images is usually very minimal. Zhan et al. (2020); Joshi et al. (2020) utilised two cameras simultaneously, where the first is mounted statically in the environment and the second is attached to the gripper. Despite the dominant use of 2D visual observations, the use of 3D data representation for RL grasping is currently very limited. Even though some works use point clouds as an intermediate representation (Zeng et al., 2018; Gualtieri and Platt, 2018), these are subsequently projected into one or more 2D image views that are then individually processed by a CNN. Osa et al. (2017); Gualtieri et al. (2018) also utilise point clouds but only to sample grasp candidates with non-RL methods, where RL policy is then used to select one of them. Based on this investigation, there is a general lack of methods for robotic grasping that utilise RL for end-to-end control with visual observations that are represented in 3D. The primary reason for this is presumably the popularity of existing deep learning frameworks that allow efficient use of CNN to extract features from 2D images for DRL. However, it can be argued that 2D convolutional layers do not provide the desired level of generalisation over depth information and spatial orientation compared to their well-established generalisation for horizontal and vertical position in the image (Gualtieri et al., 2018), even if applied to 2.5D data representation in form of depth map or aligned RGB-D images. Therefore, this work studies the importance of such generalisation over the full 6 DOF workspace in which robots grasp objects.

Application of RL to real world robotics problems is still limited due to several difficulties. Sample inefficiency is especially problematic in robotics because it can take few minutes to collect a single training sample of grasping on a real robot. One way to collect more data is by using multiple robots simultaneously. For example, Kalashnikov et al. (2018) used seven robots to collect 580,000 grasps over the course of several weeks, which can become very costly and unpractical. Moreover, the exploratory nature of RL is unsafe and induces jerky motion patterns, which leads to mechanical wear and potential damage of the actuators (Kroemer et al., 2021). Therefore, use of synthetic data in form of robotics simulators is common among RL research because it greatly improves the availability of training data. Figure 2.3 shows examples of simulation environments that were developed for robotic grasping. Similar to real robots, running multiple simulation workers in parallel can also accelerate the rate at which data is

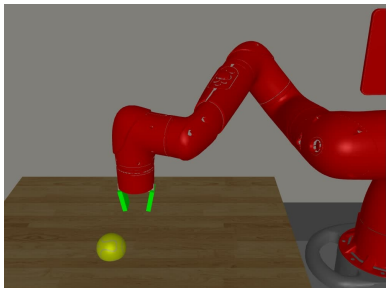
collected, e.g. Popov et al. (2017) used 16 such virtual workers. However, there is a reality gap between simulation and real robot data which must be addressed. Some of the most common sim-to-real approaches to achieve this transition will be described in subsection 2.4.1.



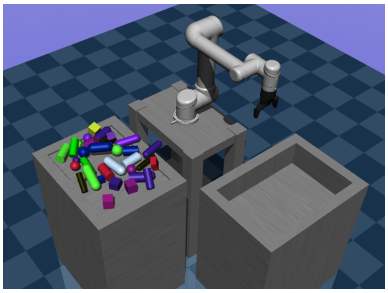
Plappert et al. (2018)



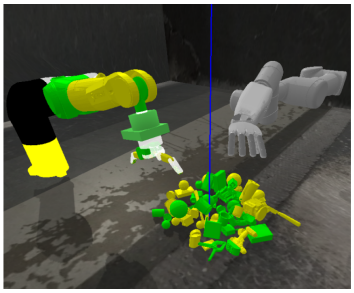
Quillen et al. (2018)



Singh et al. (2019)



Daniel (2020)



Wu et al. (2020)

Iqbal et al. (2020)

Figure 2.3: Examples of simulation environments used for robotic grasping with DRL.

Reproducibility of state-of-the-art RL methods is also not very straightforward, as robotics environments are rarely deterministic and performance of RL methods can be highly influenced by many factors such as selection of hyperparameters and scaling of the reward (Henderson et al., 2018). Random seed that is used to initialise pseudorandom generator during training can also have a significant influence on the learning curve and final performance as visualised in Figure 2.4. These issues are further magnified by robot and hardware requirements in addition to any use of proprietary software or framework, hence simulated setup with open-source software is highly preferred for RL research.

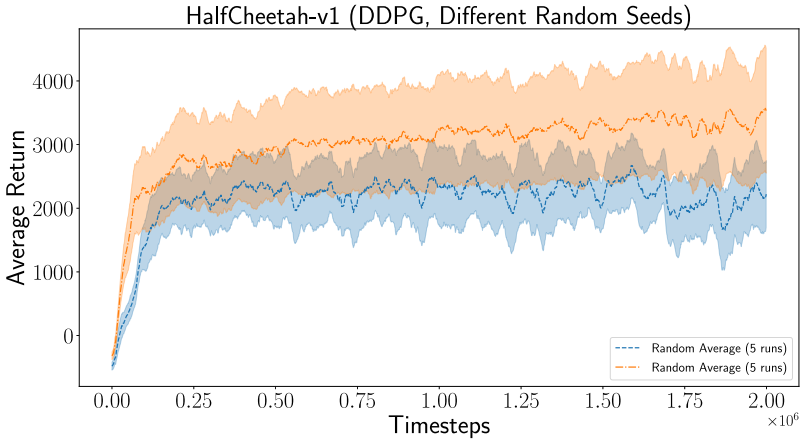


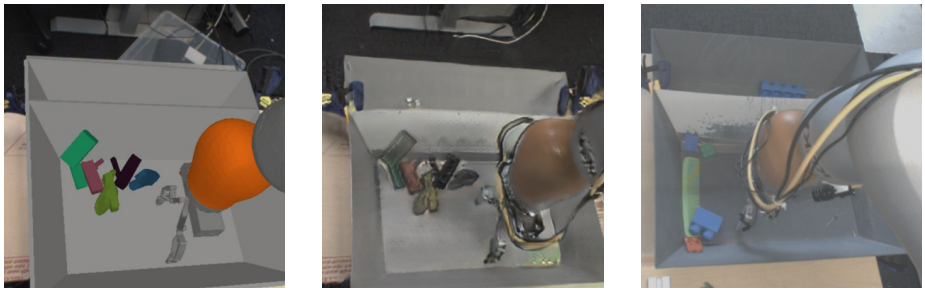
Figure 2.4: Learning curve of DDPG on a locomotion environment for two sets of five different random seeds. All runs use the same hyperparameter configuration. (Henderson et al., 2018)

2.4.1 Sim-to-Real

Due to the popularity of training DRL agents inside simulations, there are several approaches that have been applied to bridge the reality gap and achieve sim-to-real transfer without any retraining. The most straightforward approach is to reduce or completely eliminate such gap by utilising a realistic simulation software that can correctly simulate the required physical interactions and provide visualisation based on principles of physically based rendering (PBR). For example, Iqbal et al. (2020) developed a robotics simulator with a physics solver on top of a game engine that provides photorealistic rendering. However, there is a computational cost connected with such realism and compromises must be made in order to achieve a desired rate at which training data can be effectively produced. Consequently, other methods that increase the variety in the data have been utilised over the years, with aim to provide a better generalisation that would make the learned policy applicable to real world. Some of these methods are useful even for realistic virtual environments as a mean to increase diversity due to their low computational cost.

Data Augmentation The amount of available data can be increased by synthetically creating modified copies of the existing data. This approach is not only popular in supervised learning as a mean to enlarge dataset, but it has also been applied in RL (Zhang et al., 2015; Laskin et al., 2020; Zhan et al., 2020). In this context, data augmentation is commonly applied to the visual observations in form of 2D images with operations such as cropping, rotation, cut-out and adding jitter to the colour channels.

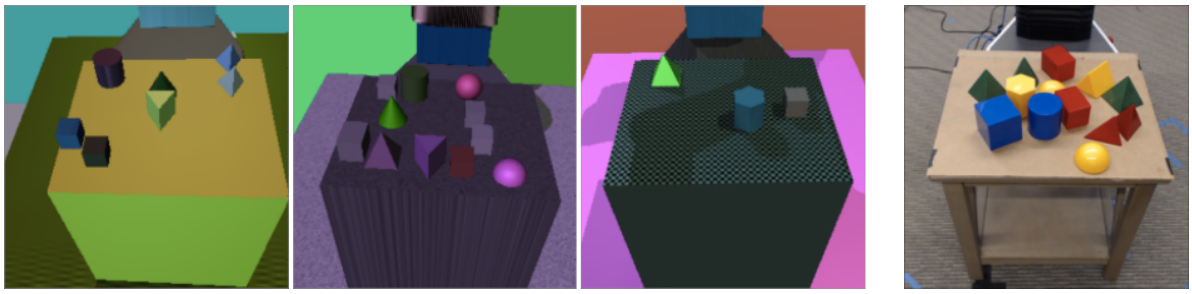
Domain Adaptation Instead of reducing the reality gap at simulation level, domain adaptation modifies observations from source domain to provide a better resemblance in the target domain. Zhang et al. (2015) applied this technique to generate synthetic images of robot arm that were similar to the training data based on real-time readings of robot's joint angle positions. In the opposite direction, Bousmalis et al. (2018) employed generative adversarial network (GAN) during training in order to adapt the synthetic images from simulation and make them closely resemble visuals of real-world domain, as illustrated in Figure 2.5.



Synthetic Adapted Real

Figure 2.5: Example of domain adaptation applied to robotic grasping. (Bousmalis et al., 2018)

Domain Randomization Another way to easily expand the variety in data is by randomly changing the simulated environment. Tobin et al. (2017) applied this method in order to randomize visual attributes shown in Figure 2.6, such as object colours, table texture, camera pose and characteristics of the illumination. Furthermore, domain randomization can be extended also to other non-visual simulation attributes such as inertial properties of robot links and hyperparameters of the utilised physics solver.



Randomized Real

Figure 2.6: Example of domain randomization for visual attributes. (Tobin et al., 2017)

2.4.2 Demonstrations

RL agents often require a large amount of random interactions with the environment before reaching the desired goal via pure exploration. For robotic grasping, a robot must first approach an object while its gripper is opened, then move into a valid grasp pose, close the gripper, and finally lift the object while keeping the gripper closed at all times. It can take millions of nearly random attempts before agent is able to perform such a sequence of actions for the first time due to the stochastic nature of the task. This is especially problematic if an agent receives only sparse rewards during the training, i.e. if reward is received only when an object is lifted above certain height. One way to mitigate this issue is by designing a dense reward that would guide the agent along the desired sequence of actions via reward engineering. However, manual engineering of reward functions for RL is challenging because human experts can introduce a significant bias towards what is perceived as the best approach, furthermore, it downgrades the end-to-end approach of DRL due to the rising need for an entirely new pipeline that extracts all objectives required to compute the engineered reward (Singh et al., 2019).

Another way to combat the issue with lengthy exploration is by introducing concepts of imitation learning into RL. Namely, demonstrations of an expert performing the task can be used at the beginning of training in order bootstrap data collection and provide an agent with knowledge about the desired goal in form of collected rewards. This process can then be followed by self-supervision through regular RL training. For example, Zhan et al. (2020) utilised a joystick to control a robot in order to collect ten demonstrations before RL training. Furthermore, these demonstrations do not need to be collected from a human expert, and a separate method can be used to collect them instead. This was shown by Kalashnikov et al. (2018) that initially used a scripted policy to provide grasps until the learned policy reached a success rate of 50%. Imperfect demonstrations are also useful in this approach because RL can improve upon them, e.g. the scripted policy of Kalashnikov et al. (2018) achieved only up to 30% success rate.

2.4.3 Inverse Reinforcement Learning

As previously mentioned, manual engineering of reward functions for specific tasks is not trivial and can introduce bias. In these situations, inverse reinforcement learning (IRL) can be applied to infer the underlying reward function that a policy is trying to optimise (Kroemer et al., 2021). This inference is based on expert demonstrations, whose trajectories contain state-action pair together with the collected reward. IRL differs from imitation learning in that it aims to extract the desired intent of the agent instead of learning how to perform the task. The reward function that is acquired by IRL can subsequently be used to learn a policy that optimises it with RL.

However, difficulties with IRL arise because there are infinitely many ways in which a reward function can be inferred for a single policy.

2.4.4 Curriculum Learning

Curriculum learning is another approach that addresses the problem with lengthy exploration in complex environments. Instead of learning the entire task from scratch, a curriculum can be designed to present the problem as a sequence of subtasks with increasing difficulty (Narvekar et al., 2020). Figure 2.7 illustrates the use of curriculum for the full game of Chess. These subtasks usually contain their own reward function, e.g. separate reward for reaching and grasping an object, which results in the notion of composite reward when solving the entire task (Popov et al., 2017). Decomposing tasks in this way is advantageous because skills with shorter horizon are simpler to learn and the agent is more likely to reach the desired state during exploration.

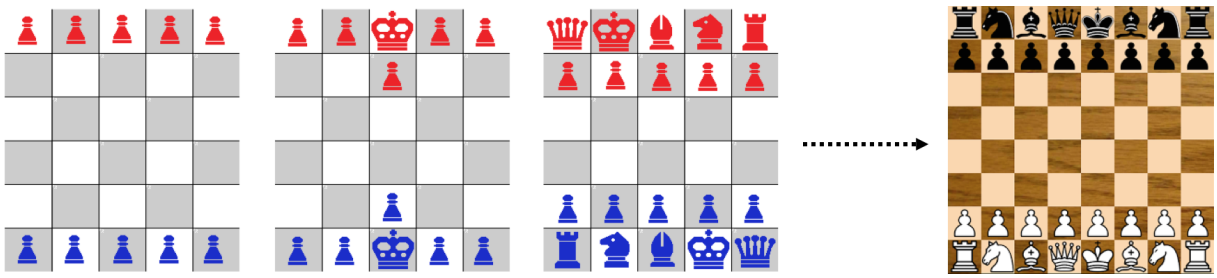


Figure 2.7: Subgames that can form a curriculum for learning Chess. (Narvekar et al., 2020)

Another way to apply curriculum learning is by first solving a simplified version of the full task, and then progressively scaling the difficulty as agent's success rate increases. Breyer et al. (2019) applied curriculum learning for several attributes to solve robotic grasping with RL. They progressively increased workspace size, maximum number of objects, initial robot height and the required lift distance. With this approach, they found that curriculum learning can significantly accelerate the training process.

2.5 Deep Learning on 3D Data

Robotic grasping requires an agent to operate in 3D workspace with 6 DOF. Projections of this space onto 2D RGB or 2.5D RGB-D images can partially represent this data and allow an agent to learn a policy directly from raw pixels. However, there are several other representations in which 3D data can be expressed. Due to the increasing availability of depth-sensing cameras and LiDARs, a number of approaches for leveraging these representations in the context of DL have been proposed in recent years. Combination of such approaches with RL could provide a number of potential benefits over current DRL research that focuses on utilising 2D or 2.5D observations to learn end-to-end robotic grasping.

3D data representations differ mostly in their structure and geometric properties, where Euclidean and non-Euclidean categories provide the main distinction (Ahmed et al., 2018). High-level overview of commonly used 3D data representations is shown in Figure 2.8.

2.5.1 Non-Euclidean 3D Representations

The first type encompasses the non-Euclidean approaches that allow data to be stored in an irregular structure. Meshes and point clouds are popular representations from this category.

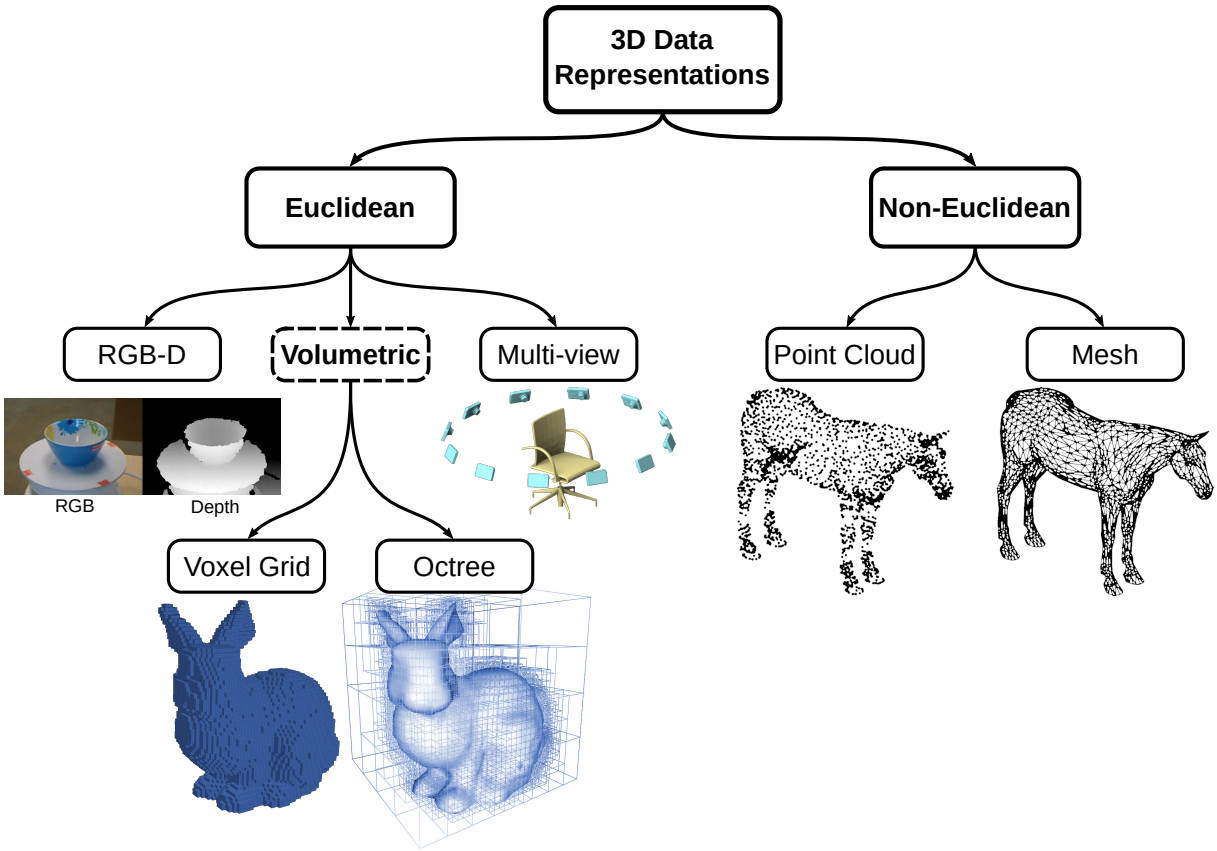


Figure 2.8: Overview of various 3D data representations. Adapted from Ahmed et al. (2018).

3D Mesh Polyhedral models are most often represented with 3D meshes that intuitively describe them in terms of vertices, edges and faces. However, 3D meshes are typically not available a priori for diverse set of real-world objects and constructing them from a single-view RGB-D observation would require extra pre-processing and result in sub-optimal mesh quality. Therefore, DL approaches for processing 3D meshes are not considered in this work.

Point Cloud Geometry of 3D objects and shapes can be approximated by representing them with a point cloud as a set of unstructured 3D points. Furthermore, additional features such as colour and intensity can be added for each point. Despite the effectiveness and availability of point clouds for capturing arbitrary 3D scenes, their irregular structure makes it difficult to process them directly with DL approaches. To enable the use of DL on point clouds, Ruizhongtai Qi et al. (2016) proposed PointNet that first preprocesses the unordered input with a spatial transformer network before using a recurrent neural network (RNN) to process the points as a sequential signal, where all extracted point features are then aggregated by max pooling. With this approach, PointNet was successfully applied for classification and segmentation. However, due to aggregation of all points together, PointNet is unable to extract detailed fine-grained patterns from local structures. This was later addressed with PointNet++ (Qi et al., 2017) by recursively applying PointNet on nested partitions of the input point cloud. Such hierarchical approach of PointNet++ improved the achievable results over PointNet, albeit with significantly worse computational performance. Approaches based on point clouds therefore pose a limitation for end-to-end robotic grasping, which requires both real-time performance and local features to describe detailed geometry for potential grasps.

2.5.2 Euclidean 3D Representations

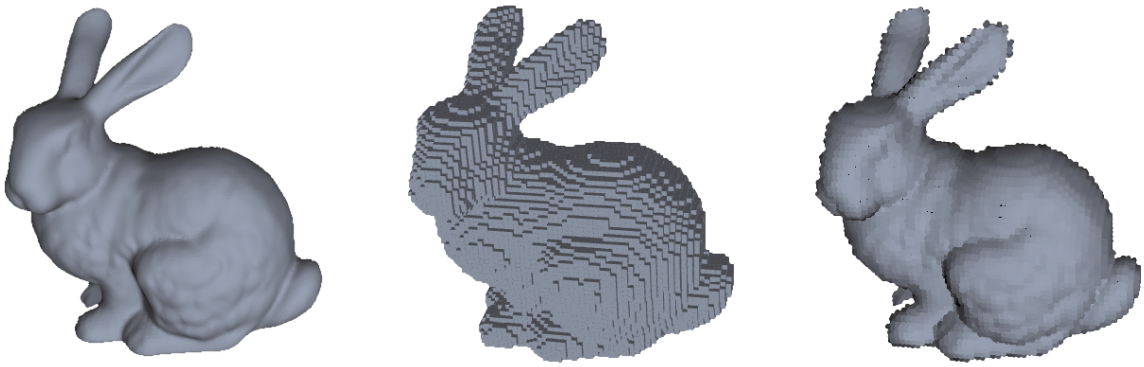
Euclidean 3D data representations have an underlying grid-like structure that allows data to be stored in a regular arrangement. Among others, this category includes RGB-D images, multi-view data and volumetric representations in form of voxel grids and octrees.

Multi-view Data The primary limitation of RGB-D images is their 2.5D nature due to the projection that produces them. This in turn limits the information they can store to a single view of the scene, hence occlusions are inevitable. A simple way to mitigate the limitation of single view is by utilising multiple vantage points simultaneously via multi-view data representation. This can be achieved when a scene is perceived by multiple RGB or depth-sensing cameras, or if virtual cameras are used to synthesise projections of the scene. The produced set of images can subsequently be used in the context of DL by individually processing them with a 2D CNN. DRL approach by Gualtieri and Platt (2018) applies multi-view data representation in form of three distinct axis-aligned views of the scene, which are combined in order to determine a 3D position via pixel-wise action space for an action primitive that places previously grasped objects. However, multi-view representations might require a large number of views in order to sufficiently describe the available 3D information, which in turn causes computational overhead and it can lead to over-fitting (Ahmed et al., 2018). Furthermore, their use of 2D CNNs still lacks the required generalisation over depth and spatial orientation (Gualtieri et al., 2018).

Voxel Grid Similar to 2D images that contain pixels in a regular grid, 3D volume can be subdivided into individual voxels. With this volumetric approach, voxel grids can be used to describe how 3D objects are distributed throughout the scene. Each voxel can either contain a simple occupancy indicator or it can have a number of features such as surface normals and colour channels. Although voxel grids are in many aspects similar to 2D images, they do not suffer from limitations induced by perspective projection and distortion. Furthermore, they are easily extendible to 3D convolutional operations due to their regularity, which gave rise to DL approaches for exploiting the full 3D geometry of objects by using CNNs. With this approach, Wu et al. (2015) and Maturana and Scherer (2015) concurrently introduced 3D ShapeNets and VoxNet designed for classification via shape analysis, where the volumetric 3D representation enabled their CNNs to generalise over shape and spatial orientation of various objects. However, voxel grids are notoriously inefficient because they fully represent both occupied and unoccupied cells. This makes their memory and computational cost grow cubically with increasing voxel resolution, which in turn restricts their use to low resolution voxels. Therefore, 3D ShapeNets and VoxNet used grid size of only 30x30x30 and 32x32x32 binary voxels, respectively.

Octree To mitigate the inefficiency of voxel grids, octrees provide a more compact volumetric representation of 3D data. Their efficiency comes from the ability to vary the size of voxels in the volume according to the object occupancy. It models the 3D representation as a hierarchical tree-like data structure, where each cell can be recursively decomposed into eight child octants. This approach provides octrees with the benefits of volumetric approach, while having a reduced memory and computational cost. This quality made octrees popular in robotics applications such as obstacle avoidance. However, the hierarchical structure of octrees increases their complexity and makes it more challenging to parallelise operations such as 3D convolutions for DL. To address this issue, Wang et al. (2017) proposed O-CNN with a novel octree data structure that is suited for DL parallelisation on GPUs, which is encoded with shuffled keys and labels for spatial and hierarchical organisation, respectively. Instead of simple occupancy, they utilise surface normals in order to preserve smoothness of the objects as visualised in Figure 2.9. To further

improve computational efficiency, their CNN approach processes only the finest leaf octants, i.e. the smallest possible octants at a certain octree depth that describe the surface of objects. O-CNN has been applied to problems such as shape classification, completion, retrieval and segmentation with competitive results (Wang et al., 2017, 2020).



Original mesh | Occupancy voxel grid | Octree with normals

Figure 2.9: Comparison between occupancy voxel grid and smoothness-preserving octree that uses normal vector instead of binary indicator. Octree is rendered with oriented disks sampled at the finest leaf octants. Smallest cell size is the same for both representations. (Wang et al., 2017)

Conclusion of Related Work

Several approaches for solving robotic grasping have been proposed over the years. Current state-of-the-art performance can be achieved by synthesising grasp poses through deep supervised learning, and then applying a separate motion planner to reach the highest-ranked grasp pose. However, supervised learning approaches require large labelled datasets that are often fine-tuned for a specific gripper. RL approaches are becoming more prominent because they can circumvent this problem via self-supervision. Furthermore, the popularised use of DRL enables end-to-end control from raw pixel observations. Applying DRL for complex robotics tasks such as grasping of diverse objects with continuous actions can however be challenging due to issues with sample inefficiency, trade-off between exploration & exploitation and numerous problems with reproducibility. The use of simulation environments provides an important stepping stone for mitigating some of these issues, where sim-to-real approaches are applied to reduce a potential reality gap.

Current DRL research for robotic grasping focuses on end-to-end approaches that utilise 2D RGB or 2.5D RGB-D image observations that are processed by CNNs to extract meaningful features. Despite the success of these methods, 2D convolutions do no provide the desired level of generalisation over depth information and spatial orientation. This work therefore investigates the potential benefits of applying DRL on visual observations with 3D data representation. Among these, octrees are selected due to their organised structure and improved efficiency over other volumetric representations.

3 Background

A theoretical overview of RL foundation based on Sutton and Barto (2018) is provided in this chapter, together with specific algorithms relevant to this project. The reader is welcome to skip to the next chapter 4 if these concepts are familiar.

3.1 Markov Decision Process

The goal of RL agent is to maximize the total reward that is accumulated during a sequential interaction with the environment. This paradigm can be expressed with a classical formulation of Markov decision process (MDP), where Figure 3.1 illustrates its basic interaction loop. In MDPs, actions of agent within the environment make it traverse different states and receive corresponding rewards. MDP is an extension of Markov chains, with an addition that agents are allowed to select the actions they execute. Both of these satisfy the Markov property, which assumes that each state is only dependent on the previous state, i.e. a memoryless property where each state contains all information that is necessary to predict the next state. Therefore, MDP formulation is commonly used within the context of RL because it captures a variety of tasks that general-purpose RL algorithms can be applied to, including robotic manipulation tasks.

It should be noted that partially observable Markov decision process (POMDP) is a more accurate characterisation of most robotics tasks because the states are commonly unobservable or only partially observable, however, the difficulty of solving POMDPs limits their usage (Kroemer et al., 2021). Therefore, this chapter presents only on MDPs where observations and states are considered to be the same.

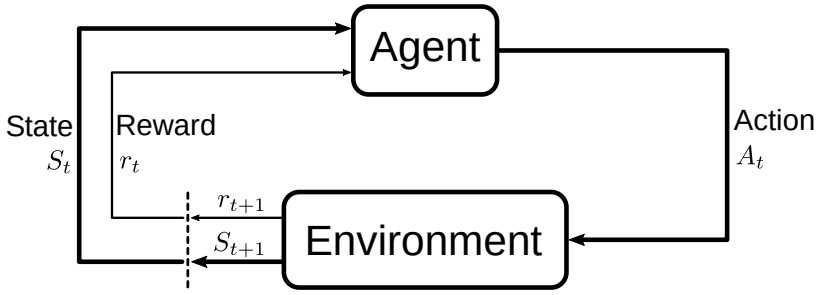


Figure 3.1: The interaction between agent and environment in MDP. (Sutton and Barto, 2018)

MDPs are typically described as a tuple $(\mathcal{S}, \mathcal{A}, p, r)$. In this work, the state space $\mathcal{S}$ and action space $\mathcal{A}$ are assumed to be continuous. The state transition probabilities are defined by function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ that represents the probability density of the next state $s' \in \mathcal{S}$ based on the current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$.

$$p(s'|s, a) = \Pr\{S_{t+1}{=}s'|S_t{=}s, A_t{=}a\} \tag{3.1}$$

The behaviour of an agent is defined by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that provides a mapping from states to actions. At each discrete time step t, the environment utilises reward function $r(s_t, a_t)$ to emit a scalar value that expresses the immediate reward $r_t \in \mathbb{R}$ for executing action a_t in state s_t. Since both immediate and future rewards must be considered in MDP setting, the return G_t that RL agent seeks to maximise is defined as a sum of discounted rewards

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i), \tag{3.2}$$

where $\gamma \in [0, 1]$ is a discount factor that determines the priority of long-term future rewards and ensures that return is finite for continuous tasks. T denotes a final time step, which either indicates the end of episode for episodic tasks or $T = \infty$ for continuous tasks. Episodic robotic grasping task with a fixed maximum number of time steps is considered in this work.

A value function can be defined to determine the expected return when following a policy π for a particular state s with value function $V^\pi(s)$. Similarly, an action-value function for taking action a in state s and then following policy π can be defined as $Q^\pi(s, a)$.

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t{=}s] \qquad = \mathbb{E}_\pi \left[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \middle| S_t{=}s \right] \tag{3.3}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t{=}s, A_t{=}a] = \mathbb{E}_\pi \left[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \middle| S_t{=}s, A_t{=}a \right] \tag{3.4}$$

The primary goal of the agent is to find the optimal policy π^* that is better than or equal to all other policies. This can be achieved by estimating the corresponding optimal action-value function $Q^*(s, a)$ for all s and a.

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) \tag{3.5}$$

Optimal action-value function satisfies Bellman equation. Intuitively, Bellman optimality equation for $Q^*(s, a)$ expresses that the value of a state is equal to the expected return for the best action a' taken in that state.

$$Q^*(s, a) = \mathbb{E} \left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| S_t{=}s, A_t{=}a \right] \tag{3.6}$$

3.2 Model-Free Reinforcement Learning

As previously mentioned, RL algorithms can be categorised into model-based and model-free methods, latter of which are considered in this work. Aside from this classification, there are two additional distinctions among RL algorithms that can be used to categorise them. One of these distinctions is related to the way in which data is collected during the training, separating algorithm into on-policy and off-policy. Last category is based on whether RL algorithm computes a value function or not, which differentiates them into value- and policy-based RL. Furthermore, RL algorithms apply various exploration strategies in order to balance their trade-off between gaining more knowledge about the environment through exploration and following the current most promising direction via exploitation.

On-policy algorithms are restricted to only use data that is collected by the specific policy that is being optimised during the training. On the contrary, off-policy algorithms can be used to train an agent on any data collected by an arbitrary policy. This distinction has significant impact

on RL robot learning with respect to sample efficiency. On-policy algorithms cannot reuse previous data during training because the policy keeps changing with each update. As opposed to this, off-policy RL algorithms can utilise each transition multiple times. For this, experience replay buffer (Mnih et al., 2015) is commonly used to store transitions when interacting with the environment, which are then used to provide samples for updating the policy. However, on-policy algorithms generally provide better convergence guarantees during learning than off-policy methods. Despite of possible instability, off-policy algorithms are typically considered to be more suitable for complex robotics tasks due to their improved sample efficiency (Quillen et al., 2018).

3.2.1 Value-Based Methods

Value-based methods aim to estimate the optimal state-value function $V^*(s)$ or more commonly the action-value function $Q^*(s, a)$. Once the optimal action-value function $Q^*(s, a)$ is found, the optimal policy π^* can be followed by selecting the optimal action a^* at each state s.

$$a^*(s) = arg \max_{a'} Q^*(s, a') \tag{3.7}$$

Such optimisation of value function is often performed off-policy and can therefore utilise experience replay. Unfortunately, these algorithms are incompatible with continuous actions, which limits their applicability for learning robotics tasks that usually require operation in continuous domain. Exception to this are tasks that allow discretisation, e.g. approaches described in section 2.4 that combine pixel-wise action space with action primitives.

Temporal difference (TD) learning is a form of value-based approach in which value function is optimised by minimising TD error δ. For action-value function, this error arises from a notion that the value of current state and selected action $Q^\pi(s_t, a_t)$ should be equal to the reward that corresponds with this state-action pair r_t, plus the discounted action-value estimate of the next state and best action $Q^\pi(s_{t+1}, a^\pi)$ that follows the policy π.

$$\delta_t = r_t + \gamma \max_{a'} Q^\pi(s_{t+1}, a') - Q^\pi(s_t, a_t) \tag{3.8}$$

Q-learning uses TD learning and in fact, the TD error for action-value function from Equation 3.8 is employed by Q-learning. With this error δ_t, estimating $Q^*(s_t, a_t)$ becomes an optimisation problem in the following form where $\alpha \in (0, 1]$ is the learning rate.

$$Q^*(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha\delta_t = Q^\pi(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q^\pi(s_{t+1}, a') - Q^\pi(s_t, a_t) \right] \tag{3.9}$$

However, classical Q-learning is inefficient for large environments because it must consider every possible state-action pair in order to determine the optimal $Q^*(s, a)$, e.g. by using a tabular approach. Therefore, a general function approximator can be used instead of large tables to solve these inefficiency. In deep Q-learning popularised by Mnih et al. (2015), NNs are used to approximate the action-value function $Q(s, a)$. Such network can be designed to process a state as the input, and output a value for each possible action. The main difference from classical Q-learning is that optimisation of $Q^*(s, a)$ is achieved by minimising TD error δ_t with respect to parameters θ of the NN. Action that provides the maximum output of the NN for a given state can then be selected for subsequent execution. In order to explore different states, a simple ϵ-greedy action selection can be employed. With this approach, the agent takes a random action with a probability equal to ϵ and action that follows the current policy π otherwise.

The benefit of utilising NNs as a function approximator for deep Q-learning is their scalability to larger environments. However, converge to optimal solution can often be difficult to achieve

due to instability. Several improvements were therefore proposed over the years to mitigate this issue. An example of improving training stability is by employing target networks (Mnih et al., 2015), where one network is used for training and a different network is used for computing TD error. This allows the target network with parameters θ' to provide a stable measure of error that does not significantly change on each update of unrelated state-action pairs, which would otherwise be common due to the large number of network parameters. These networks must then be regularly synchronised either via hard update, i.e. regular copy of parameters at fixed intervals, or by applying a soft update at each step in form of Polyak averaging with hyperparameter $\tau \in (0, 1]$.

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{3.10}$$

3.2.2 Policy-Based Methods

Instead of determining actions based on their value, policy-based methods directly optimize a stochastic policy π as a probability distribution $\pi(a|s, \theta)$ that is parameterised by θ.

$$\pi(a|s, \theta) = \Pr\{A_t{=}a|S_t{=}s, \theta_t{=}\theta\} \tag{3.11}$$

Following a policy is therefore based on sampling an action from this distribution given a state s. Typically, θ are weights and biases of NN that are optimised through gradient descent on an objective function that maximises the expected return over state-action sequences, i.e. policy gradient methods.

Policy-based algorithms are typically on-policy and therefore less sample-efficient, yet they have better convergence properties. Furthermore, these algorithms can be directly applied to continuous action spaces without any need for discretisation, which makes them appealing for many robotics problems.

3.2.3 Actor-Critic Methods

In contrast to value- and policy-based methods as the two primary categories, actor-critic methods include algorithms that utilise both a parameterised policy, i.e. actor, and a value function, critic. This is achieved by using separate networks, where the actor and critic can sometimes share some common parameters. Such combination allows actor-critic algorithms to simultaneously possess advantages of both approaches such as sample efficiency and continuous action space. Therefore, these properties have made actor-critic methods popular for robotic manipulation while achieving state of the art performance among other RL approaches in this domain.

Similar to policy-based methods, the actor network learns the probability of selecting a specific action a in a given state s as $\pi(a|s, \theta)$. The critic network estimates action-value function $Q(s, a)$ by minimising TD error δ_t via Equation 3.9, which is used to critique the actor based on how good the selected action is. This process is visualized in Figure 3.2. It is however argued that the co-dependence of each other's output distribution can result in instability during learning and make them difficult to tune (Quillen et al., 2018). Despite of this, actor-critic model-free RL algorithms are utilised in this work.

3.3 Actor-Critic Algorithms

Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Soft Actor Critic (SAC) and Truncated Quantile Critics (TQC) are examples of specific actor-critic algorithms that can be applied for robotic grasping with continuous action and observation spaces.

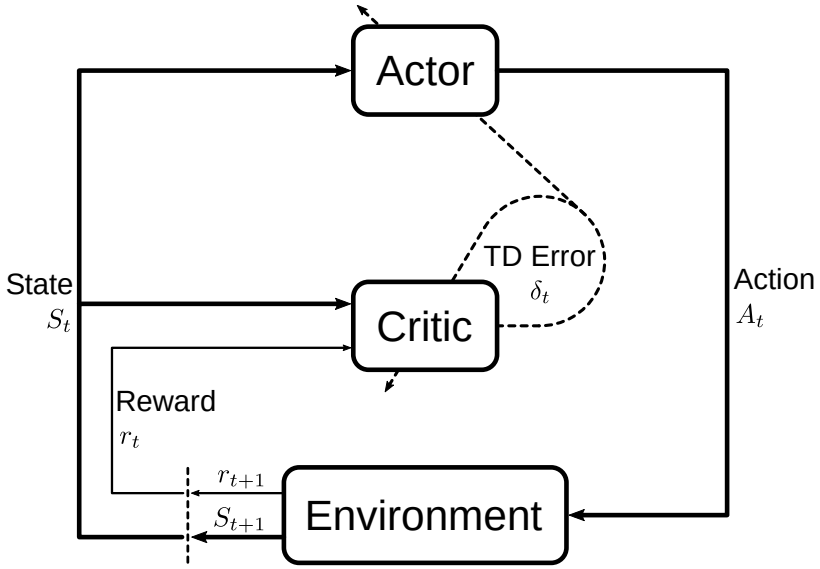


Figure 3.2: Overview of actor-critic methods. TD error δ_t is used to adjust both critic's action-value function $Q(s, a)$ and actor's policy $\pi(a|s, \theta)$. Adapted from Sutton and Barto (2018).

3.3.1 Deep Deterministic Policy Gradient

DDPG (Lillicrap et al., 2015) is an actor-critic off-policy algorithm for continuous action space that trains a deterministic policy. It makes use of experience replay buffer and target networks with soft update that were described in subsection 3.2.1. In addition to the target action-value network, DDPG also utilises a target policy network. In order to accommodate exploration, noise is added to the actions during training.

However, DDPG is notoriously unstable and sensitive to hyperparameters (Islam et al., 2017; Quillen et al., 2018), which makes it difficult to apply for complex robotic manipulation tasks. Therefore, additional algorithms and extensions to DDPG have been proposed over the years.

3.3.2 Twin Delayed Deep Deterministic Policy Gradient

TD3 (Fujimoto et al., 2018) is an extension to DDPG that aims to improve stability by mitigating overestimation of action-value function. It achieves this with three extensions. The first change is the use of two individual critics, where the smaller of the two values is used to compute TD error in Equation 3.8. The second modification is based on updating actor network less often than critic networks. The last addition includes a smoothing noise for the actor network, which makes it harder for the policy to exploit errors of the critic. Similar to DDPG, exploration during training is encouraged by addition of noise to actions. Together, these three tricks improved performance over original DDPG.

3.3.3 Soft Actor Critic

As opposed to DDPG and TD3, SAC (Haarnoja et al., 2018b) optimises a stochastic policy. It also incorporates the use of two critics as TD3, while also inheriting actor network smoothing noise due to its stochastic nature. The main addition of SAC is the use of entropy regularisation, which makes the policy optimise a trade-off between expected return and entropy, which represents the randomness of the policy. Risk of getting stuck in local minima is therefore decreased, since entropy regularisation has an inherent connection to the exploration strategy because larger entropy results in more exploration.

The objective of optimal policy π^* is therefore changed to include the entropy regularisation term $\mathcal{H}(X) = \mathbb{E}[-logP(X)]$.

$$\pi^* = arg\ \max_{\pi}\ \mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^t\Big(r(s_t, a_t) + \alpha\mathcal{H}\big(\pi(\cdot|s_t)\big)\Big)\right] \tag{3.12}$$

Entropy coefficient α, also called temperature, is a hyperparameter that determines the trade-off between expected return and entropy. This temperature can either be fixed or it can be optimised automatically during the training. With these changes, SAC is by many considered to be state of the art RL algorithm for continuous control.

3.3.4 Truncated Quantile Critics

TQC (Kuznetsov et al., 2020) is a recent extension to SAC that utilises a distributional representation of multiple critics. The action-value function $Q(s, a)$ of critics is modelled as a distribution with n-number of atoms. TQC further addresses the overestimation of $Q(s, a)$ by truncation of ~8% topmost atoms from such distributions. With these changes, authors have reported notable performance increase of TQC over SAC on complex robotics tasks.

4 Problem Formulation

This chapter systematically formulates the targetted task of robotic grasping as an MDP, while describing the applied reward function alongside the utilised observation and action spaces. The corresponding implementation of such formulation with detailed specifications is covered in the next chapter 5.

4.1 Task Definition

In this work, the agent is assumed to be a high-level controller that provides sequential decision making in form of gripper poses and actions. Therefore, the environment is considered to not only include all objects and the physical interactions between them but also the robot with its actuators and low-level controllers. Episodic formulation of the grasping task is studied, where a new set of objects is introduced into the scene at the beginning of each episode when the environment is reset. During each episode, the aim of the agent is to grasp and lift an object certain height above the ground plane that the objects rest on, which also terminates the current episode. Furthermore, an episode is also terminated after 100 time steps and whenever the agent pushes all objects outside the union of the perceived and reachable workspace. Placing of objects after their picking is not investigated in this work.

Due to the benefits of employing robotics simulators to train RL agents, e.g. safe and inexpensive data collection, robotics simulator will be used in this work to train the agent. Once the agent is trained in a virtual environment, the learned policy will subsequently be evaluated in a real-world setup via sim-to-real transfer. The conceptual setup of this work that should be similar in both domains is illustrated in Figure 4.1.

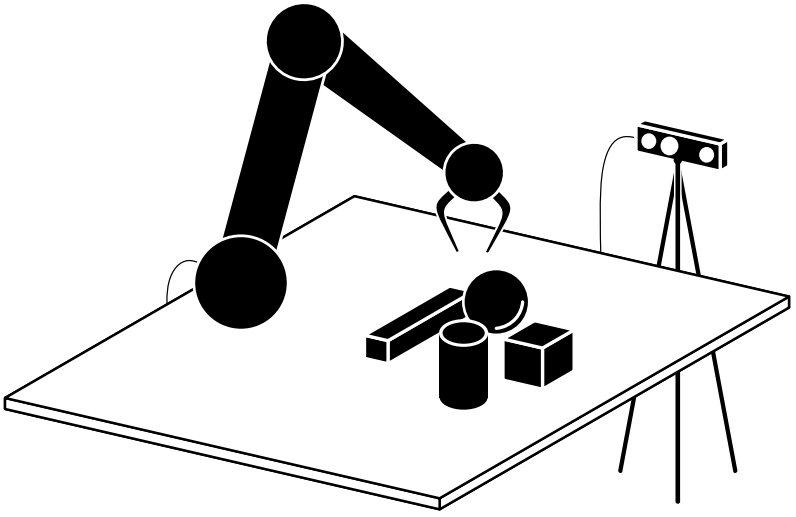


Figure 4.1: Conceptual setup for the task of robotic grasping that needs to be constructed both inside a robotics simulator for training and in real-world domain for subsequent evaluation.

4.2 Observation Space

The observation space for the grasping task used in this work comprises of visual and proprioceptive observations. Furthermore, a number of sequential observations is stacked together for each transition in order to provide the agent with temporal information about environment states.

4.2.1 Octree

The visual observations utilised in this work are represented in form of 3D octrees. As already mentioned, the visual perception originates from a statically mounted RGB-D camera, which is assumed to provide a new RGB image and depth map of the scene at each time step. Before constructing an octree, the depth map is first used to create a point cloud of the scene as an intermediate representation. This point cloud is colourised with a corresponding RGB image that is registered to the optical frame of the camera's depth sensor. Therefore, the resulting point cloud is in form of an unstructured list of (x, y, z, r, g, b) tuples that represent individual points.

Hereafter, three assumptions about the use of volumetric 3D data representation for end-to-end robotic manipulation are set forth. First, aspect ratio of 1:1:1 is considered to provide generalisation over all possible directions of movement, i.e. traversing a fixed distance along any of the primary axes should result in a movement over the same number of cells. Second assumption considers the volume that each cell occupies, which shall remain fixed over the entire duration of training and evaluation. This is considered to be beneficial because fixed scale of cells provides a consistency over distances between any two cells. Lastly, each cell should correspond to a specific position of space that remains fixed with respect to the robot pose, regardless of the camera pose. This assumption is considered to be necessary as it allows NNs to create relations among individual cells and their significance in space.

Due on these assumptions, the approach that is commonly used in classification and segmentation tasks, i.e. rescale a point cloud to fit inside a fixed volume (Wang et al., 2017), cannot be applied in this work. However, it is assumed that the relative pose of camera with respect to robot is known, e.g. through calibration process, therefore, the previously obtained point cloud is transformed into the robot coordinate frame in order to achieve invariance to camera pose. Furthermore, such point cloud is subsequently cropped in order to occupy a fixed volume in space with aspect ratio of 1:1:1. This volume is considered to be the observed workspace and it is subsequently used to construct the octree observations as illustrated in Figure 4.2.

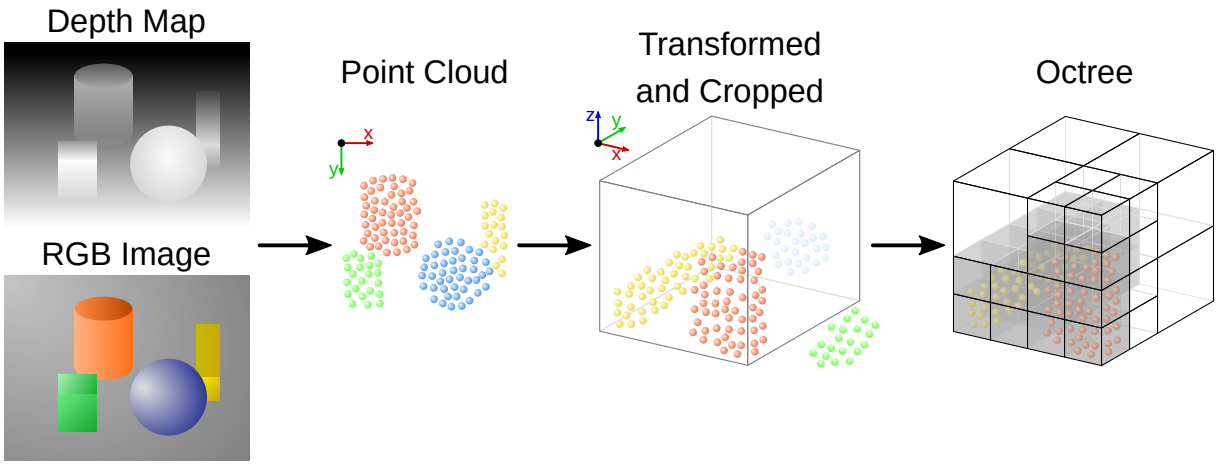


Figure 4.2: Process of constructing an octree from depth map and RGB image via an intermediate point cloud, which is transformed into the robot coordinate frame and cropped to a fixed volume.

The octree structure by Wang et al. (2017) allows arbitrary data to be stored at the finest leaf octants. Three distinct features are utilised in this work, namely the average unit normal vector $\overline{n}$, the average distance between the centre of the cell and points that formed it $\overline{d}$ and the average colour $\overline{rgb}$. As illustrated in Figure 4.3, all of these features are computed independently for each octant based on the points from the point cloud that produced it. Normals $n_i = (n_{x_i}, n_{y_i}, n_{z_i})$ are selected because they provide smoothness-preserving description of the object surfaces, as previously shown in Figure 2.9. Since point cloud acquired from RGB-D camera does not usually contain normals, they must be estimated from a local neighbourhood prior to constructing an octree. The average distance to the points $\overline{d}$ allows the perceived surface to be offset in the direction of normals, which allows octrees with lower resolution to be used while still preserving smooth transitions between the cells. Colour feature $rgb_i = (r_i, g_i, b_i)$ are expected to provide an agent with additional input that could allow semantic analysis in addition to shape analysis, which might be especially beneficial for distinguishing dissimilar objects that are in contact. Besides $\overline{n}$ being normalised as a unit vector, $\overline{d}$ and all channels of $\overline{rgb}$ are normalised to be in a range $[0, 1]$.

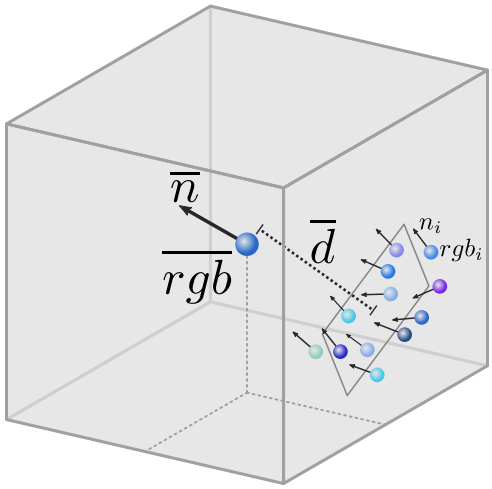


Figure 4.3: Representation of features for a single finest leaf octant from an octree. All points from the source point cloud are used to determine the average unit normal vector $\overline{n}$, average distance to the centre of the cell $\overline{d}$ and the average colour $\overline{rgb}$.

4.2.2 Proprioceptive Observations

In addition to the visual observations acquired by an RGB-D camera, it is considered to be beneficial to also include proprioceptive observations. Gripper pose and gripper state are used in this work because these observations are independent of the utilised robot. Although both of these could be determined solely from the visual observations, occlusion can introduce significant uncertainness. Furthermore, these readings are easily obtainable from any robot. The state of the gripper g_s is represented as $\{closed : -1, opened : 1\}$. The position of the gripper is encoded as (x, y, z) vector represented with respect to robot's base frame. Gripper orientation is also with respect to robot's base frame, and represented as the first two columns of the rotation matrix $[(R_{11}, R_{21}, R_{31}), (R_{12}, R_{22}, R_{32})]$ because they provide continuous description of 3D orientation without ambiguities, contrary to quaternions of Euler angles (Zhou et al., 2020).

4.2.3 Observation Stacking

A single set of visual and proprioceptive observations does not fully describe the state of the environment. In order to better satisfy Markov assumption, dynamics of the system must also be observed, including all data based on the temporal information. Mnih et al. (2015) addressed this in a simple way by stacking last n historical observations together and combining them into a single observation that fully describes the state.

Despite the increase in the amount of similar data that needs to be processed, this work applies a similar observation stacking method due to the simplicity of such solution. More specifically, three sequential octrees and proprioceptive observations are stacked together, i.e. $n = 3$. At the beginning of each episode when three observations are not available yet, the first observations is utilised multiple times to form the stacked observation.

4.3 Action Space

In this work, the action space for end-to-end robotic grasping comprises of continuous actions in Cartesian space. By utilising actions in Cartesian space instead of joint space, the action space is invariant to the specific kinematic configuration of a robot. Furthermore, Cartesian actions provide better safety guarantees, where traditional IK and motion planning approaches can be employed to reliably provide commands for low-level joint controllers while avoiding self-collisions.

The utilised actions are illustrated in Figure 4.4. For gripper pose, the actions comprise of translational displacement (d_x, d_y, d_z) and relative rotation around z-axis d_ϕ that are both expressed with respect to robot base coordinate frame. These actions are normalised in the range $[-1, 1]$ and subsequently rescaled to metric and angular units before applying them. The gripper action g is also in a continuous range $[-1, 1]$, where positive values open the gripper and negative values prompt closing of the gripper. Therefore, RL agent is allowed to take any combination of continuous actions by selecting the corresponding values for a tuple $(d_x, d_y, d_z, d_\phi, g)$.

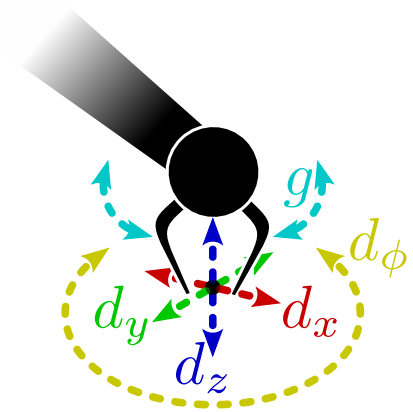


Figure 4.4: Action space of the grasping task, where (d_x, d_y, d_z) indicates a translational displacement, d_ϕ is a relative yaw rotation, and the gripper closing and opening is denoted by g.

4.4 Reward Function

Although it would be desirable to provide the agent only with a very sparse reward after successfully grasping and lifting an object, such approach would prolong the training due the sparsity of achieving a success through random exploration. Therefore, this work makes use of a composite reward function that combines together sparse rewards from four distinct stages, i.e. reaching, touching, grasping and lifting. These stages follow a hierarchical flow, where the agent must first approach an object, then touch, grasp and finally lift it. During each episode, the agent is allowed to obtain a reward from each of these stages only once in order to discourage any rewarding behaviour that would not lead to the desired goal of the final stage, such as repeatedly pushing an object in order to continually accumulate reward for touching.

The proportion and scale of each component from the reward function can be treated as a tunable environment hyperparameter because it directly influences the policy that the agent aims to optimise. Generally, reward at the last stage should be much higher than the reward given at first stage, which is only meant to guide the training of the agent. Therefore, an exponential function r_{exp}^{i-1} is used to determine the individual reward for each stage i. The base $r_{exp} \in [1, \infty)$ can be tuned, where r_{exp} = 7 was empirically found to provide satisfactory results for the implemented grasping task, with theoretical maximum achievable reward of r_{max} = 400.

In addition to positive reward for accomplishing the task, the agent is also given negative reward of -1 for each time step during which the robot is in collision with the ground plane in order to discourage the number of undesired collisions. Furthermore, a small reward of -0.005 is subtracted at each time step until termination in order to encourage the agent to accomplish the task as fast as possible. All rewards are summarised in Table 4.1.

Composite	**Reaching**	r_{exp}^0 = 1	*(once per episode)*
	Touching	r_{exp}^1 = 7	
	Grasping	r_{exp}^2 = 49	
	Lifting	r_{exp}^3 = 343	
	Collision	-1	*(each time step)*
	Act Quickly	-0.005	

Table 4.1: Overview of the reward function that is utilised in this work for the grasping task, where r_{exp} = 7 was tuned and each episode has at most 100 time steps.

5 Implementation

A concrete implementation of applying DRL for robotic grasping with octree-based observations is presented in this chapter. First, design and creation of a simulated RL environment is described, which is then followed by specifics of the utilised RL framework and architecture of CNN for extracting features from octrees of the scene. The full implementation is open-source and available on GitHubhttps://github.com/andrejorsula/drl_grasping.

5.1 Simulation Environment

As presented in section 2.4, simulations are often used for RL training in order to significantly increase the rate at which data can be collected in a safe manner. In order to implement a virtual setup for training of robotic grasping based on the design from chapter 4, a simulation must be capable of accurately modelling the physical interactions between a robot and the manipulated objects. Furthermore, it must feature a high fidelity rendering of the scene to provide the required visual observations from viewpoint of a virtual RGB-D camera. Therefore, selection of a robotics simulator is of great importance because it directly influences the robustness of sim-to-real transfer and determines the additional steps that must be taken to achieve such transfer.

5.1.1 Selection of Robotics Simulator

There is a variety of simulation tools that could be applied for training RL agents for robotics, some of which are based on video game engines due to their mature state. Generally, a trade-off between accuracy, stability and performance that must be considered. Although most everyday objects have certain properties of soft bodies, rigid-body dynamics usually provide a satisfactory degree of realism for generic robotic grasping without suffering much performance loss. Therefore, a considered simulator shall have an appropriate physics engine for handling environments with a number of rigid bodies, with support for actuated joints that can be used to connect links of a robot. Similarly, PBR rendering capabilities are highly preferred because of the utilised visual observations. Some of the popular simulators for robotics RL research are therefore described with aim to select one that will be used to implement the environment.

MuJoCo (Todorov et al., 2012) MuJoCo is a physics engine that can accurately model physical interactions. It has been a popular choice for robotics research for years, including RL applications. Unfortunately, MuJoCo is a proprietary software, which has resulted in the decline of its use over the recent years in favour of open-source alternatives. Furthermore, it has limited rendering capabilities.

PyBullet[1] PyBullet simulator is built on top of Bullet physics engine, with an experimental support for PhysX[2] back-end. PyBullet is gaining popularity for robotics RL research due to its open-source nature and active development. It provides fast and reliable physical simulations, albeit the available rendering is not photorealistic.

Gazebo Classic (Koenig and Howard, 2004) Gazebo is one of the oldest open-source robotics simulators and it has a large active user-base because it is the primary simulator for the community of Robot Operating System (ROS) (Quigley et al., 2009). Instead of developing everything from scratch, Gazebo is built on top of already existing physics and rendering engines. By default, it utilises ODE[3] physics engine but others such as DART (Lee et al., 2018) and even Bullet are also supported. For rendering, it makes use of OGRE[4] 1 that unfortunately has limited rendering capabilities.

Ignition Gazebo[5] Due to the limitations and outdated architecture, Gazebo Classic is planned to be deprecated in favour of Ignition Gazebo, i.e. the next generation of Gazebo. Although it is in its early development, Ignition Gazebo supports DART physics engine and has an upcoming support for Bullet. In addition to OGRE 1, PBR rendering is enabled by using OGRE 2, and there is also a partial support for ray tracing with OptiX[6]. Both physics and rendering engines can be loaded during runtime due to the utilised plugin-based architecture. Although little RL robotics research has been conducted with the use of Ignition Gazebo so far, Ferigo et al. (2020) introduced Gym-Ignition as a framework that simplifies its for RL research.

Isaac[7] Isaac Sim is a new and promising robotics simulator that is being developed by Nvidia. It utilises PhysX physics engine and has support for state-of-the-art PBR rendering. Isaac Gym is extension of Isaac for RL. One of its significant advantages is that physics computations, rendering as well as the process of determining rewards can be offloaded to GPU and enable running large number of environments in parallel. Unfortunately, the proprietary nature of Isaac might limit its use and possible customisation. Furthermore, Isaac Gym is still available only as an early access as of May 2021 with limited functionalities.

From the considered robotics simulators, Ignition Gazebo is selected in this work due to the following reasons. Compared to MuJoCo that requires a license, it is open-source, which significantly encourages reproducibility. Although Isaac might be a very promising choice for robotics RL research in the future, it is still under development and its proprietary nature could make it difficult to extend for the needs of this work. PyBullet is currently considered to be a one of the best open-source options due to its maturity and a large amount of RL research that has already been conducted with it. However, it lacks PBR rendering capabilities that are already part of Ignition Gazebo. Furthermore, the plugin-based architecture of Ignition Gazebo simplifies addition of new physics engine, where Bullet support is already pending. Its ability to switch between various physics engines during run-time could eventually provide Ignition Gazebo with one of the best physics-based domain randomisation, as it would not only allow randomising the physics parameters but also of the entire physics implementation. The major

[1] https://pybullet.org
[2] https://developer.nvidia.com/physx-sdk
[3] https://ode.org
[4] https://ogre3d.org
[5] https://ignitionrobotics.org
[6] https://developer.nvidia.com/optix
[7] https://developer.nvidia.com/isaac-sim, https://developer.nvidia.com/isaac-gym

disadvantage of the selected Ignition Gazebo robotics simulator is its relatively early stage and a very limited amount of RL research conducted with it. Despite of this, the full availability of its source code makes it possible to extend where needed. Gazebo Classic was excluded from this considerations due to its planned deprecation.

Therefore, Ignition Gazebo is used to create an environment for robotic grasping with RL. For the physics engine, the default option of DART is kept unchanged. For rendering engine, OGRE 2 is selected due to its PBR capabilities. Gym-Ignition (Ferigo et al., 2020) is utilised because it simplifies interaction with Ignition Gazebo with focus on RL research. Furthermore, Gym-Ignition facilitates the process of exposing OpenAI Gym interface for the environments, which provides a standardised form that makes environments compatible with most RL frameworks that contain implementations of algorithms.

5.1.2 Environment for Robotic Grasping

In order to create a new RL environment for robotic grasping, several different aspects must be considered and implemented into a single integrated system. This among others includes an accurate model of a robot that needs to be combined with appropriate motion planning and low-level controllers, as well as perception in form of RGB-D frames that can be used for visual observations. Furthermore, a set of 3D object models with appropriate appearance, mechanical and inertial properties is required for training and subsequent evaluation.

Robot Models

Support for two articulated robotic manipulators with different kinematic chains and grippers is implemented in order to demonstrate flexibility of the developed environment and applied DRL. This is often neglected from the current robotics RL research, hence it is unknown how well a system reacts to a change of robot, both with respect to the use of same hyperparameters during training and the learned policy itself. These two robots are 6 DOF Universal Robots UR5[8] with OnRobot RG2[9] sweeping-parallel gripper and 7 DOF Franka Emika Panda[10] with its default parallel gripper, both of which are shown in Figure 5.1.

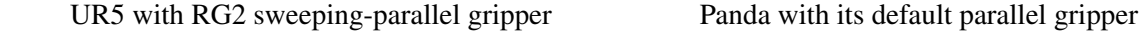


Figure 5.1: Robot models used inside the simulation environment for robotic grasping.

Manufacturers of these robots and grippers provide associated 3D CAD models of individual links together with a model of the kinematic chain. However, inertial properties of links and dynamic properties of joints are usually not provided. Therefore, these need to be estimated. In order to do so, inertial properties for both robots and grippers are estimated based on the combination of their documented weight and 3D mesh model, while assuming a uniform density across the bodies. Empirically, it was found that redistributing a portion of hand's mass to each finger provides more stable grasp, which is assumed to be due to the internal mechanical coupling of fingers to the body of hand that is otherwise not accounted for solely from the 3D mesh. For dynamic properties of joints, friction and damping were manually tuned for each joint with aim to achieve a stable manipulation across a variety of control frequencies. Although these

[8] https://universal-robots.com/products/ur5-robot
[9] https://onrobot.com/products/rg2-gripper
[10] https://franka.de/panda

estimated values are not based on the real robots, no negative effects for sim-to-real transfer are expected because the action space of DRL agent is in Cartesian space.

With this information, description that uses Simulation Description Format (SDF) compatible with Ignition Gazebo was created for both robots (Orsula, 2021). A simplification for the sweeping-parallel RG2 gripper was made in order to provide better a stability. It was modelled by using a single actuated revolute joint per finger, whereas the full model would use three additional passive joints on each finger. Parallel gripper for Panda is modelled with two prismatic joints, i.e. one for each finger.

Motion Planning To control the motion of both robots, a joint trajectory controller described in appendix A was implemented for Ignition Gazebo. It follows trajectories that are generated in Cartesian space by the use of MoveIt 2[11] motion planning framework. In this framework, the default configuration of TRAC-IK (Beeson and Ames, 2015) and RRTConnect (Kuffner and LaValle, 2000) were used for solving kinematics and motion planning, respectively. An advantage of utilising MoveIt 2 is that a single interface can be used to control both simulated and real robots during sim-to-real transfer, however, this feature is not applied in this work.

RGB-D Perception

In order to acquire visual observations from the environment, a virtual RGB-D camera is utilised. Using OGRE 2 rendering engine, it provides aligned RGB image and depth map simultaneously. For both, the resolution is set to 256×256 px with a field of view of 52°. Framerate of the camera is set to 10 Hz. Gaussian noise $\mathcal{N}(0, 0.001)$ is added to both RGB and depth data in order to slightly increase the realism of observations. The effect of this when compared to real observation is shown in Figure 5.2 on point cloud. Since the pose of camera is known with respect to the robot, the acquired point cloud is then transformed into the robot base coordinate frame.

Figure 5.2: Effect of adding Gaussian noise to RGB-D data in order to improve resemblance to real-world perception.

Middleware

ROS 2 is used in this project as a middleware that facilitates communication among the primary nodes of the system, e.g. RGB-D data stream, requests from motion planner and the simulation environment itself. Whenever data between ROS 2 and the transport layer of Ignition Gazebo is required, a bridge between them is used to convert the messages. The selection of ROS 2 was made because it significantly simplifies the initial research-based development and enables use of helpful libraries and tools for robotics. However, this choice brings a disadvantage for RL because the underlying socket-based transport reduces determinism of the simulation, which prevents exact reproducibility of results even for the same random seed.

Dataset

Dataset of scanned objects by Google Research (2020) was selected for training and evaluation of robotic grasping inside the simulation environment. It is available as a collection from Ignition

[11] https://moveit.ros.org

Fuel, which is a web-based application that allows hosting and sharing of simulation assets. The selected collection contains a thousand of common household objects that are 3D scanned. Their realistic appearance and diverse geometry make them ideal for training of robotic grasping with aim to achieve generalisation. From the dataset, 100 objects shown in Figure 5.3 were selected and split into training and testing subsets with a ratio of 80/20.

Figure 5.3: Training (left) and testing (right) datasets of diverse scanned objects that are utilised in the simulated RL grasping environment. Collection is provided by Google Research (2020).

All of these objects contain only their corresponding mesh geometry and material texture but lack all other properties. Inertial properties were therefore estimated from their geometry in a procedure that is similar to the aforementioned robot models. Mass of each object used during such estimation was randomly selected alongside other properties of the model, which is detailed below in subsection 5.1.3. This also include the scale of their geometry, as many of these objects would be too large to fit inside the utilised grippers.

The 3D scanned objects contain meshes with a very high resolution, which makes them unsuitable for computing physical interactions due to the enormous computational cost it would bring. Therefore, a low resolution copy of each mesh is created for use as a collision geometry, alongside the original mesh that is kept for visual appearance. Such copy is automatically generated for each model by simplifying the original mesh geometry though decimation procedure based quadric error metrics by Garland and Heckbert (1997). The algorithm was configured to reduce the geometry to 2.5% of the original faces, which was afterward clipped to the range between [8, 400] faces in order to avoid outliers.

Performance of Simulation

Having a performant simulation accelerates the data collection, which can in turn enable faster iteration for RL research due to reduced training duration. Besides reducing computational load by decimating geometry of objects, few more tricks are applied in this work.

Disabling of Collision for Robot Links During early trials, it was found that a collision never occurs between robot links and the environment. This is primarily because MoveIt 2 is used to plan collision-free trajectories. Furthermore the action space is restricted only to the yaw rotation, which further reduces possible collisions. Therefore, collision geometry of robot links is disabled during the training with aim to bring a slight performance gain. The collision geometry of gripper, i.e. hand and fingers, is kept enabled for both robots as these are required for interaction with the objects.

Larger Simulation Step Size As previously mentioned, dynamic properties of robot joints were manually tuned in order to obtain stable manipulation across a variety of control frequencies. The primary purpose of this tuning is to allow the use of larger simulation step size, which

determines the rate at which simulation progresses. This in turn affects the accuracy of physics as well as the frequency of low-level controller. A step size of 4 ms is used for the grasping environment because it was found to have a balanced trade-off between physics stability and performance.

With performance in mind, the control rate of RL agent is set to a lower frequency of 2.5 Hz. This is because the agent only provides high-level control, whereas the motion planner and low-level joint controllers take care of interactions that require faster reaction times.

5.1.3 Domain Randomisation

Even though the simulation environment uses objects with realistic appearance and PBR-capable rendering engine, domain randomisation can still provide advantages for sim-to-real transfer as described in subsection 2.4.1. Therefore, domain randomisation is applied for several properties at each reset of the environment, i.e. before the beginning of each episode. Unless otherwise state, uniform distribution is used for sampling of random variables.

Random Objects At each reset, a number of random objects from the utilised dataset is spawned. Each object is first randomly and uniformly scaled, such that its longest side is between 12.5 and 17.5 cm. Hereafter, object's inertial properties are recomputed to account for the new scale, while also randomising its mass to be in range [0.05, 0.5] kg. Lastly the coefficient of friction for the object is randomised in range [0.75, 1.5]. In this way, visual, inertial and mechanical properties of each object are random for every episode.

Random Pose of Objects Besides randomising the type and attributes of each object, the pose at which they spawn is also randomised. It is randomly sampled for each object from a predefined volume in 3D space. In case two objects are overlapping, one of them is spawned again with a new unique pose.

Random Ground Plane Material Textures To further randomize visuals of the environment at each reset, a random material texture is given to the ground plane on top of which objects are spawned. Similar to the objects, 100 different PBR materials are used with a split of 80/20 for training and testing, respectively. PBR materials are used, therefore, each of them uses four different texture maps, i.e. albedo, normal, specular and roughness.

Random Camera Pose In order to further increase variety in observations and provide invariance to camera pose, it is randomised at each reset. The pose of the camera is randomly sampled from an arc around the centre of workspace, except for $\pm 22.5°$ behind the robot in order to avoid complete occlusion of the scene. Thereafter, a random height for the camera in a range [0.1, 0.7] m is selected. The camera is then oriented towards the workspace centre and placed 1 m away from it. This step is expected to provide significant benefits for sim-to-real transfer by allowing camera to be positioned in a location that is suitable for the real-world setup, instead of trying to reproduce simulation setup as closely as possible.

Random Initial Joint Configuration Finally, the initial joint configuration of the utilised robot is randomised. At the beginning of each episode, Gaussian noise $\mathcal{N}(0, 6°)$ is added to each joint in the default configuration.

Examples of fully randomised episodes are shown in Figure 5.4. The aim of this variety in observations is to enable sim-to-real transfer that would allow agent to achieve similar degree of success rate in real-world domain after training only in the simulation

Figure 5.4: Examples of domain randomisation applied to the implemented simulated environment for robotic grasping.

5.1.4 Demonstrations and Curriculum

As mentioned in section 2.4, demonstrations and curriculum learning can mitigate issues with lengthy exploration. Both of these concepts are therefore investigated in this work and implemented in the following way.

Demonstrations For demonstrations, approach by Kalashnikov et al. (2018) with the use of a scripted policy is applied. Since off-policy RL algorithms with experience replay buffer are employed, the demonstrations can be simply loaded into such buffer at the beginning of training. More specifically, 5000 transitions are loaded into the replay buffer. The scripted policy is described in appendix B.

Curriculum Similar to demonstrations, the use of curriculum can improve learning for tasks in complex environment. This work utilises a curriculum that progressively increases the number of spawned objects and area on top of which these objects are spawned based on current success rate determined by moving average with $n = 100$. The spawn area increases linearly from 2.4×2.4 cm at 0% success to 24×24 cm at success rate of 60%. Similarly, training begins with a single object, with an addition of extra object every 20% until reaching a maximum of four objects at 60% success rate.

5.2 Deep Reinforcement Learning

The implementation of DRL in this work focuses on integration of octree-based feature extraction for solving vision-based robotic grasping. A framework for DRL is first selected, which is then followed by description of the architecture of the feature extractor and network for actor and critics. Lastly, the applied process of hyperparameter optimisation is presented.

5.2.1 Framework for Reinforcement Learning

It can be very time-consuming and error-prone to implement DRL algorithms from scratch due to several issues that could arise. Therefore, a framework with pre-existing implementations of the utilised actor-critic algorithms from section 3.3, i.e. TD3, SAC and TQC, is utilised. After a brief investigation of the available frameworks for model-free RL, Stable Baselines3 by Raffin et al. (2019) was selected due to its reliable implementation of the utilised algorithms, open-source nature and active development. Underneath, PyTorch (Paszke et al., 2019) is utilised as a machine learning backend that enables training of NNs via its automatic differentiation engine.

In order to enable octree-based feature extraction, the implementation of algorithms was extended with few modifications. These primarily consisted of support for octrees inside replay

buffer, formation of octree batches and integration of the octree-based feature extractor with NNs of actor and critics. All other configuration of the algorithms was performed through their hyperparameters.

5.2.2 Feature Extraction

With visual features, the first part of the network can often be considered as a feature extractor that transforms raw data into more abstract features. This fact is often employed in network architectures for actor-critic DRL methods, where a feature extractor CNN network is shared between the actor and critics. This work therefore utilises the same approach, where a common feature extractor transforms raw input into features that are then provided as input for actor and critic networks. To extract features from octrees, O-CNN implementation by (Wang et al., 2017) is used as a base for the employed feature extractor.

Construction of Octree

First, an octree is constructed from the aforementioned transformed point cloud of the scene during each step. For this, a volume of 24×24×24 cm is defined to be the observable workspace as a space that is coincidental with the workspace of the robot. Therefore, each point cloud is cropped to occupy only this volume in order to preserve assumption about volumetric 3D data representations from subsection 4.2.1.

Maximum depth of the octree was selected as $d_{max} = 4$ in order to provide metric resolution of each finest leaf octant of 1.5×1.5×1.5 cm. This depth was found to provide enough detail for grasping of objects from the utilised dataset, while not slowing down the training due enormous number of cells. Every octree therefore contains a theoretical maximum of 4096 cells, however, an average of 13% of these cells are occupied at any given time in the created simulation environment. This is primarily because only a single view of the scene is used, where each occlusion prohibits the formation of new cells in the occluded regions behind the visible surfaces. Therefore, it is expected that for each additional depth, the workspace volume can be increased eightfold for the utilised dataset while the actual number of occupied cells would be increased at a much slower rate.

As it was previously described in subsection 4.2.1, each occupied finest leaf octant contains the average unit normal vector $\overline{n}$, the average distance between the centre of the cell and points that formed it $\overline{d}$ and the average colour $\overline{rgb}$. All of these features are extracted directly from the point cloud that is used to create the octree, where each octet considers only the points that belong to its volume. Since the cropped point cloud does not contain normals, these are estimated for each point from their nearest neighbourhood, where maximum of 10 closest neighbours at a maximum distance of 5 cm are considered. Position of the camera is then used to orient all normals correctly. Once these are found, an octree is created from the point cloud by hierarchical subdivision of the cells. An example of created octree is visualised in Figure 5.5.

Figure 5.5: Octree that created from a corresponding point cloud of the scene. In this visualisation, each finest leaf octet is represented as a square that is oriented by $\overline{n}$ and offset from the centre of the octet by $\overline{d}$. Furthermore, each octet is colourised with its corresponding $\overline{rgb}$ feature.

Network Architecture of Feature Extractor

After octrees are created, NN can use them to extract abstract visual features about the environment. Due to the popularity of CNN architectures for image-based DL, 3D octree-based CNN is employed in this work with. In order to incorporate proprioceptive observations from subsection 4.2.2, these are processed only slightly and concatenated with the features extracted from octrees. The developed architecture of the network is represented in Figure 5.6.

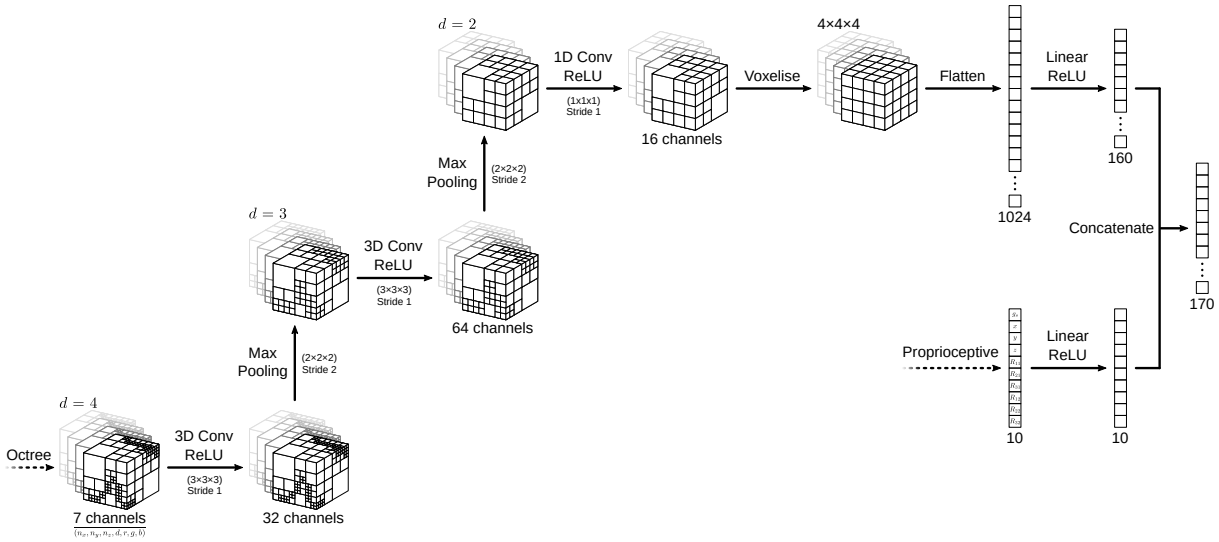


Figure 5.6: Architecture of the octree-based CNN feature extractor. Auxiliary features from proprioceptive observations are concatenated to features extracted from octrees in order to provide a single output feature vector. This network is duplicated for each of the three observation stacks and their output is concatenated into a feature vector with length of 510.

The network begins with processing octrees at the maximum depth $d = d_{max} = 4$. Each octree contains seven channels, which encompass the aforementioned features. From this depth, the octree is processed through a series of 3D convolutions, ReLU (Rectified Linear Unit) activation function and maximum pooling. After each pooling operation, the depth of the octree is decremented such that next convolutional layer computes features that are at a larger scale. This series of modules is applied twice, such that the depth of the octree is reduced to $d = 2$. While doing so, the dimensionality of channels is increased to provide a wider feature space. However, it is necessary to reduce the number of channels before the next step. For this, 1D convolution is applied in order to compress the feature space by combining together features from the different channels for each cell. Once the dimensionality is reduced, the octree is voxelised in order to acquire a structure that has a static size regardless on the input, which enables use of more traditional DL layers. It is achieved by padding the octree at $d = 2$ with zeros wherever a cell is not already occupied. Once voxelised, the feature space is flattened into a feature vector that is then processed by a single fully connected layer followed by ReLU activation in order to provide the final set of features from octree observations.

The proprioceptive observations are also processed by the same feature extractor. However, only a single linear layer with ReLU activation and the same dimensionality is used because these features are already at a higher level compared to the raw octrees. Hereafter, the features extracted from octree are combined with proprioceptive features into a single feature vector. The number of utilised channels and the dimensionality of feature vectors is presented in Figure 5.6, which results in total of 226,494 learnable parameters.

In order to enable observation stacking described in subsection 4.2.3, the feature extractor is duplicated for each of the three stacks. Once all observation stacks are processed individually, their output is concatenated into a single feature vector that can be used by actor and critic networks. A separate network for each stack is utilised instead of a common network because it allows agent to extract different set of features from historical and current observations. The disadvantage of this approach is increased number of parameters that must be learned, which could potentially slow down the training process. Such effect is therefore investigated during experimental evaluation.

5.2.3 Actor-Critic Network Architecture

Once the feature vector is extracted, it can be used as an input for approximator of the utilised algorithm, e.g. another network or a set of networks. Only actor-critic RL algorithms TD3, SAC and TQC are employed in this work, therefore, a single high-level network architecture can be used for all of them. The implementation of these networks for specific algorithm would differ in the output they provide. For example, SAC and TQC require actor to provide a stochastic policy as opposed to TD3, where TQC also utilises a distributional representation of critic's output.

Figure 5.7 illustrates the utilised network architecture that combines a shared feature extractor with actor and critic networks. Identical architecture that consists of two fully connected layers with ReLU activations is employed for both actor and critics, albeit with a separate set of parameters. With the utilised number of nodes, there are 524,288 learnable parameters for each actor and critic network.

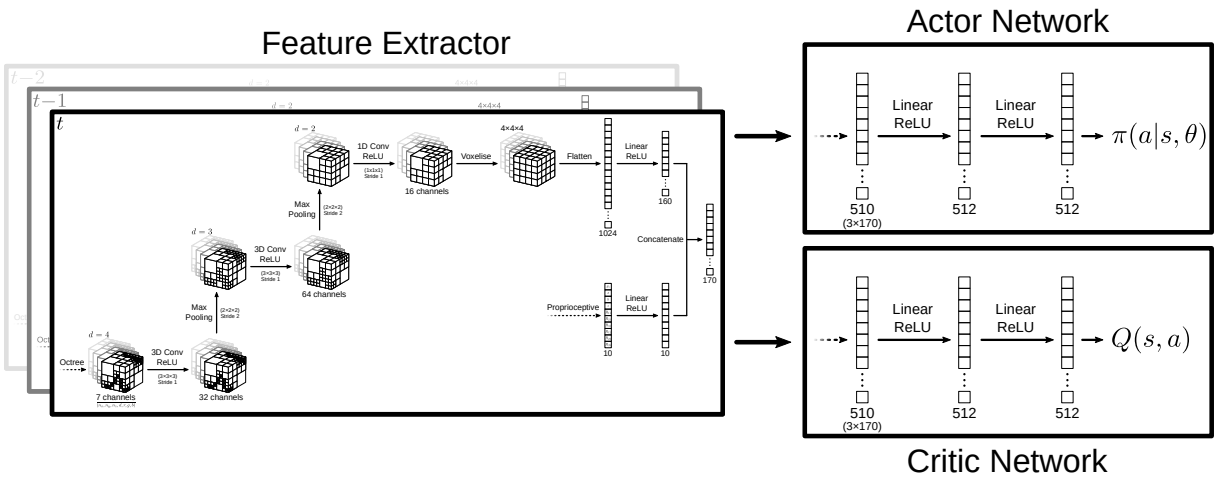


Figure 5.7: Network architecture that is used in this work for all utilised actor-critic algorithms. The feature extractor from Figure 5.6 is duplicated for each stack in order to process two historical observations in addition to the current one.

5.2.4 Hyperparameter Optimisation

Selection of hyperparameters can significantly affect the learning curve as well as the final performance of a learned policy. This brittleness of DRL to hyperparameters therefore means that their optimisation is of great importance and needs to be performed for each environment. In this work, both automatic optimisation and manual fine-tuning is performed with aim to obtain a set of hyperparameters that would allow robust learning of policy for the created environment, observations and utilised RL algorithms.

First, an automatic hyperparameter optimisation is applied by the use of Optuna, which is a hyperparameter optimisation framework developed by Akiba et al. (2019). Optuna and other similar frameworks address the problem of selecting a viable combination of hyperparameters for DL by performing a number of different trials that are used to iteratively search the hyperparameter space and find a combination that provides the best results according to some metric. In terms of RL, this metric is a reward that an agent is able to accumulate over the course of some evaluation period. Optuna generally consists of two parts, which are the sampler and the pruner. Sampler selects a set of hyperparameters from the hyperparameter search-space for the next trial. Such selection can either be completely random, e.g. at the beginning of an experiment, or by applying algorithms that perform statistical analysis from all previous trials. Pruner in this context is a strategy that allows early stopping of non-promising trials with aim to limit the amount of wasted resources. Pruning requires that evaluation episodes of each trial are run at regular intervals, where each new trial is compared to the performance of all previous trials and pruned if the accumulated reward is comparably too low.

For the grasping environment, Optuna is first applied to optimise hyperparameters in order to get a baseline that provides a reliable performance. This optimisation was performed using SAC, where the search space consisted of most hyperparameters including the size of the feature extractor and actor-critic networks. Size of the replay buffer, batch size and initial entropy were not optimised automatically. Replay buffer and batch size were selected to be adequately large for the utilised system in terms of maximum RAM and VRAM usage, respectively. Initial entropy is kept consistent because it directly influences the performance during the early stages of trials and large initial entropy could result in unwanted pruning of trials. Total of 70 trials with a maximum trial duration of 100,000 time steps were used. A set of 20 evaluation episodes was performed every 25,000 time steps, which could trigger pruning. At the end, the best performing set of hyperparameters was used for subsequent manual tuning.

Manual tuning is applied because the automatic optimisation with Optuna requires a lot of compute time for complex environments such as the one created in this work. This is also the reason why only 70 trials with a maximum trial duration of 100,000 time steps were used, which already took approximately three weeks of compute time. Manual tuning of targetted hyperparameters was therefore performed with several more trials, which were manually initiated and stopped. Focus of this process was mostly on hyperparameters of the implemented environment, e.g. reward scale, and on the octree-based feature extractor such as the maximum depth and network size. The resulting hyperparameters for all utilised actor-critic algorithms can be seen in appendix C.

6 Experimental Evaluation

This chapter presents results of experiments that were conducted in order to evaluate the use of DRL for robotic grasping with octree-based observations. The created simulation environment is analysed with respect to the feasibility of sim-to-real transfer in order to validate the applicability of all results for use in real-world domain. Furthermore, various configurations and ablations are studied to provide comparative investigation of different approaches and their advantages for learning robotic manipulation with DRL.

6.1 Experimental Setup

All experiments utilise the simulation environment for the training of all RL agents. Generalisation to novel objects is evaluated for all agents in the same simulation but on a testing dataset, where one of the trained agents is in addition evaluated also on a real robot.

6.1.1 Simulation

Unless otherwise stated, the training in simulation is identical to the setup described in section 5.1 with full-scale domain randomisation. UR5 robot with RG2 gripper is utilised as the primary robot for all experiments because it is the robot that is also tested in real-world domain. However, one of the agents is trained using the Panda robot. Panda is also used to evaluate possible generalisation to new robots with a transfer of an agent trained on UR5, and vice versa. The same random seed is used to train all agents.

When evaluating the trained agents, testing datasets for both object models and PBR textures are used. The environment is configured to present the agent with the full task, i.e. largest possible workspace and maximum number of objects. Each episode can last at most 100 time steps and agent succeeds only if an object is lifted 12.5 cm above the ground. The random seed is changed for all evaluated agents to a new common value that is different from a seed used during training. In order to encourage reproducibility, this simulation setup is available as a pre-built Docker image[1].

6.1.2 Real

Real world setup shown in Figure 6.1 is used to evaluate sim-to-real transfer. This setup consists of a UR5 robot with RG2 gripper and Intel RealSense D435 RGB-D camera[2] that is mounted on a tripod in front of the robot. Pose of the camera with respect to the robot is calibrated with a procedure described in appendix D. Similarly, appendix E presents the configuration and post-processing of the camera output.

[1] https://hub.docker.com/r/andrejorsula/drl_grasping
[2] https://intelrealsense.com/depth-camera-d435

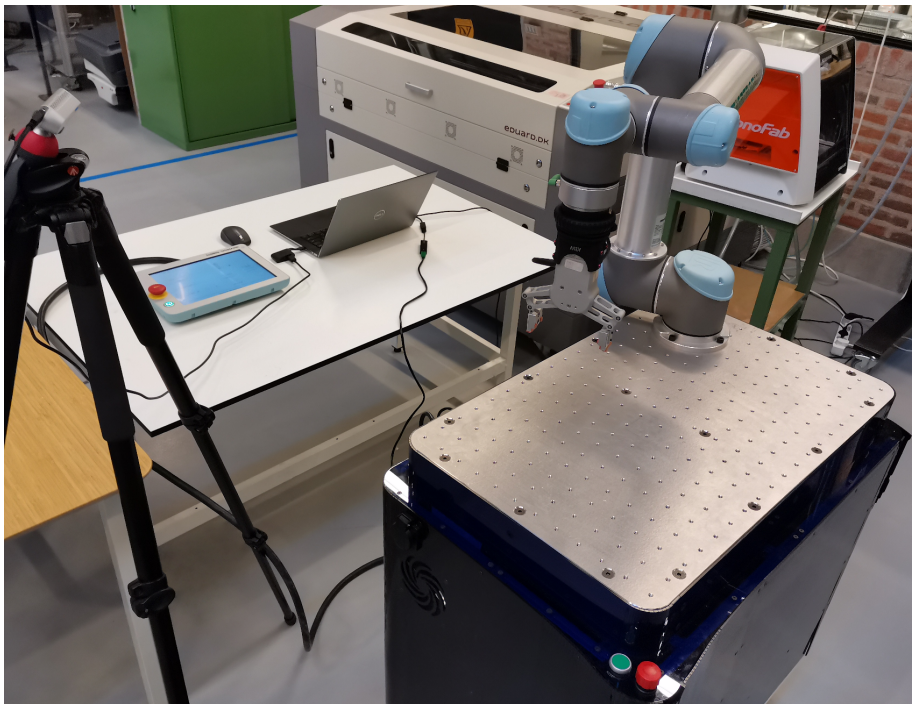


Figure 6.1: UR5 robot with RG2 gripper and RealSense D435 camera in a setup that is used to evaluate sim-to-real transfer.

Figure 6.2 shows 18 different objects that were used in real world during the testing. Mostly compliant objects were selected in order to reduce the risk of damage to the gripper due to the unpredictability of end-to-end RL policy trained in a different domain. The same workspace volume and number of objects are used as in the simulation. Similarly, the goal of the agent is to lift an object within 100 time steps.

Figure 6.2: A set of 18 objects that were used during the evaluation of sim-to-real transfer.

6.2 Results

Results of the following experiments are presented in this section. First, actor-critic algorithms are compared on the created simulation environment in order to select the best performing one for the task of robotic grasping. Hereafter, octree-based 3D observations are compared to traditional

2D and 2.5D image observations, and studied with respect to camera pose invariance. Similarly, invariance to the utilised robot is evaluated for both training process and transfer of already learned policy. Lastly, results of sim-to-real transfer are presented.

All agents were trained over the duration of 500,000 time steps, which is assumed to provide a comparative analysis among the different experiments from this work. It is however expected, that the final performance for many of these agents can be improved with a longer training duration. During evaluation of agents on novel scenes inside simulation, 200 episodes are used for each agent.

6.2.1 Comparison of Actor-Critic Algorithms

TD3, SAC and TQC were trained using the same grasping environment with a network architecture presented in subsection 5.2.3 and hyperparameters from appendix C. The success rate during training and the final success rate on novel objects and textures is presented in Figure 6.3 for all three algorithms.

Figure 6.3: Success rate of TD3, SAC and TQC algorithms on the created grasping environment.

Based on these results, TQC is utilised for all subsequent experiments.

6.2.2 Comparison of 2D/2.5D/3D Observations

3D octree observations are now compared to more traditional 2D RGB and 2.5D RGB-D observations. Besides the success rate, this comparison also includes computational complexity in terms of memory usage and processing time. For octrees, feature extractor from subsection 5.2.2 is used. For RGB and RGB-D images, an analogous CNN feature extractor described in appendix F is employed instead. In order to make the comparison fair, the exact same architecture is used with approximately the same number of learnable parameters as listed in Table 6.1. TQC hyperparameters from appendix C are used for all agents. In order to utilise the same replay buffer size for all three agents, the resolution of RGB images and depth maps had to be reduced to 128×128 px for all observation types, including octrees. Note that the change of image resolution does not impact the number of learnable parameters for octrees.

	2D RGB Image	**2.5D RGB-D Image**	**3D Octree**
Learnable Parameters	229,248	229,680	226,494

Table 6.1: Number of learnable parameters per each observation stack for the utilised RGB, RGB-D and octree-based feature extractors.

The three different feature extractors are first trained in the fully randomised environment. However, as results in Figure 6.4 indicate, 2D and 2.5D observations are unable to provide invariance to camera pose.

Figure 6.4: Success rate of RGB, RGB-D and octree-based feature extractors on environment with randomised camera pose.

Figure 6.5: Success rate of RGB, RGB-D and octree-based feature extractors on environment with a fixed camera pose.

Therefore, this experiment is repeated for an environment where the camera pose is static and remains unchanged throughout the entire training and subsequent evaluation on novel scenes. Figure 6.4 provides results for such environment with reduced domain randomisation.

Lastly, comparison of memory usage and computational time is presented in Table 6.2.

	2D RGB Image	**2.5D RGB-D Image**	**3D Octree**
Size *(per sample)*	X	XX	XXX (average) XXX (maximum)
Pre-processing *(average, per sample)*	—	—	X
Forward *(average, per sample)*	X	X	X
Forward *(average, batch of 32)*	X	X	X
Update *(average, batch of 32)*	X	X	X

Table 6.2: Comparison of computational complexity for RGB, RGB-D and octree-based observations with their corresponding feature extractors. Pre-processing is performed during data collection and consists of point cloud processing, estimation of normals and creation of octree.

6.2.3 Invariance to Camera Pose

Based on the results of the previous experiment, agent with octree observations and fully randomised camera pose is evaluated with respect to its learned generalisation to different camera poses on novel scenes. A total of X various poses with different azimuth and height are evaluated, with corresponding results shown in Figure 6.6.

Figure 6.6: Success rate to novel scenes for different camera poses.

6.2.4 Invariance to Robot

In addition to training agents with octree observations on UR5 robot with RG2 gripper, an agent is also trained on Panda robot in order to study the robustness of state-of-the-art actor-critic algorithm with octree observations to different kinematic chains and gripper designs. Comparison between of success rate between UR5 and Panda can be seen in Figure 6.7.

Figure 6.7: Success rate of UR5 and Panda robots using the same environment, algorithm and hyperparameters.

Furthermore, feasibility of transferring a policy trained on one robot to another is investigated. Such transfer is evaluated on novel scenes with a policy trained on both UR5 and Panda. Results for this experiment can be found in Table 6.3.

		During Evaluation	
		UR5	**Panda**
During Training	**UR5**	UR5/UR5	UR5/Panda
	Panda	Panda/UR5	Panda/Panda

Table 6.3: Comparison of success rate on novel scenes for policies trained one robot and evaluated on another. UR5 robot with RG2 gripper and Panda robot with its default gripper were evaluated.

6.2.5 Sim-to-Real Transfer

Finally, an agent trained inside simulation is evaluated in real-world domain to study the feasibility of sim-to-real transfer for environment with extensive domain randomisation and octree-based observations. Setup described in subsection 6.1.2 is used, where objects are randomly replaced after each success or 100 time steps. With this setup, 41 out of 60 episodes were successful, which results in a success rate of 68.3%. Figure 6.8 shows few examples of successful grasps and a recording is available on YouTube[3].

Figure 6.8: Examples of successful grasps accomplished by a policy that was transferred from simulation to real-world domain.

6.3 Ablation Studies

Besides results presented in the previous section, various ablations of the full approach are studied in order to determine contributions of its components. It is believed that these results are applicable also for other robotics tasks that utilise visual observations. Figure 6.9 presents these ablations with respect to their learning curve and attainable success rate.

Figure 6.9: Success rate of various ablations of the full method with octree observations.

Demonstrations Preloading experience replay buffer with demonstrations ...

Curriculum Similar to demonstrations, curriculum is meant to ...

Colour Features By removing colour features from the octree ...

Proprioceptive Observations Addition of proprioceptive observations has ...

Sharing of Feature Extractor between Actor and Critics Using a shared feature extractor for actor and critic ...

Separate Feature Extractors for Stacked Observations If a shared feature extractor is used for all stacked observations, ...

[3] https://youtube.com/watch?v=btxqzFOgCyQ

7 Discussion

Similarly, only a single random seed was used for all the agents due to time-consuming training procedure, however, use of several different seeds would provide a more.

8 Conclusion

9 Future Work

Bibliography

Eman Ahmed, Alexandre Saint, Abdelrahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada, and Björn Ottersten. 2018. *Deep Learning Advances on Different 3D Data Representations: A Survey*.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Patrick Beeson and Barrett Ames. 2015. TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 928–935. https://doi.org/10.1109/HUMANOIDS.2015.7363472

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor Sampedro, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. 2018. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. https://arxiv.org/abs/1709.07857

Michel Breyer, Fadri Furrer, Tonci Novkovic, Roland Siegwart, and Juan Nieto. 2019. Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters* PP (Jan. 2019), 1–1. https://doi.org/10.1109/LRA.2019.2896467

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv:1606.01540 [cs]* (June 2016). http://arxiv.org/abs/1606.01540 arXiv: 1606.01540.

Paul Daniel. 2020. *Deep Reinforcement Learning for robotic pick and place applications using purely visual observations*. Master's thesis. Technical University of Berlin, Germany.

Marc Peter Deisenroth and Carl Edward Rasmussen. 2011. PILCO: a model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML'11)*. Omnipress, Madison, WI, USA, 465–472.

Hugh Durrant-Whyte, Nicholas Roy, and Pieter Abbeel. 2012. Learning to Control a Low-Cost Manipulator Using Data-Efficient Reinforcement Learning. In *Robotics: Science and Systems VII*. MIT Press, 57–64. http://ieeexplore.ieee.org/document/6301026 Conference Name: Robotics: Science and Systems VII.

Diego Ferigo, Silvio Traversaro, Giorgio Metta, and Daniele Pucci. 2020. Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning. In *2020 IEEE/SICE International*

Symposium on System Integration (SII). 885–890. https://doi.org/10.1109/SII46433.2020.9025951 ISSN: 2474-2325.

Thomas G. Fischer. 2018. *Reinforcement learning in financial markets - a survey*. Working Paper 12/2018. FAU Discussion Papers in Economics. https://www.econstor.eu/handle/10419/183139

Scott Fujimoto, Herke Hoof, and Dave Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. (Feb. 2018).

Michael Garland and Paul Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics* 1997 (July 1997). https://doi.org/10.1145/258734.258849

Google Research. 2020. *Google Scanned Objects*. Open Robotics. https://app.ignitionrobotics.org/GoogleResearch/fuel/collections/Google%20Scanned%20Objects (Ignition Fuel).

Marcus Gualtieri, Andreas ten Pas, and Robert Platt. 2018. Pick and Place Without Geometric Object Models. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 7433–7440. https://doi.org/10.1109/ICRA.2018.8460553 ISSN: 2577-087X.

Marcus Gualtieri and Robert Platt. 2018. *Learning 6-DoF Grasping and Pick-Place Using Attention Focus*.

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. 2018a. Composable Deep Reinforcement Learning for Robotic Manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6244–6251. https://doi.org/10.1109/ICRA.2018.8460756 ISSN: 2577-087X.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. 2018b. *Soft Actor-Critic Algorithms and Applications*.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and D. Meger. 2018. Deep Reinforcement Learning that Matters. In *AAAI*.

Shariq Iqbal, Jonathan Tremblay, Andy Campbell, Kirby Leung, Thang To, Jia Cheng, Erik Leitch, Duncan McKay, and Stan Birchfield. 2020. Toward Sim-to-Real Directional Semantic Grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 7247–7253. https://doi.org/10.1109/ICRA40945.2020.9197310 ISSN: 2577-087X.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. 2017. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. (Aug. 2017).

Shirin Joshi, Sulabh Kumra, and Ferat Sahin. 2020. Robotic Grasping using Deep Reinforcement Learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 1461–1466. https://doi.org/10.1109/CASE48305.2020.9216986 ISSN: 2161-8089.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. 2018. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*.

Taewon Kim, Yeseong Park, Youngbin Park, and Il Hong Suh. 2020. *Acceleration of Actor-Critic Deep Reinforcement Learning for Visual Grasping in Clutter by State Representation Learning Based on Disentanglement of a Raw Input Image*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (Dec. 2014).

N. Koenig and A. Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Vol. 3. 2149–2154 vol.3. https://doi.org/10.1109/IROS.2004.1389727

Oliver Kroemer, S. Niekum, and G. Konidaris. 2021. A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms. *J. Mach. Learn. Res.* (2021).

J.J. Kuffner and S.M. LaValle. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, Vol. 2. 995–1001 vol.2. https://doi.org/10.1109/ROBOT.2000.844730 ISSN: 1050-4729.

S. Kumra and C. Kanan. 2017. Robotic grasp detection using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 769–776. https://doi.org/10.1109/IROS.2017.8202237 ISSN: 2153-0866.

Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. 2020. Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics. In *International Conference on Machine Learning*. PMLR, 5556–5566. http://proceedings.mlr.press/v119/kuznetsov20a.html ISSN: 2640-3498.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. 2020. Reinforcement Learning with Augmented Data. *arXiv:2004.14990 [cs, stat]* (Nov. 2020). http://arxiv.org/abs/2004.14990 arXiv: 2004.14990.

Jeongseok Lee, Michael Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha Srinivasa, Mike Stilman, and Karen Liu. 2018. DART: Dynamic Animation and Robotics Toolkit. *The Journal of Open Source Software* 3 (Feb. 2018), 500. https://doi.org/10.21105/joss.00500

Ian Lenz, Honglak Lee, and Ashutosh Saxena. 2015. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research* 34, 4-5 (April 2015), 705–724. https://doi.org/10.1177/0278364914549607 Publisher: SAGE Publications Ltd STM.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. 2016. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *The International Journal of Robotics Research* 37 (March 2016). https://doi.org/10.1177/0278364917710318

Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* (Sept. 2015).

Huaping Liu, Yuan Yuan, Yuhong Deng, Xiaofeng Guo, Yixuan Wei, Kai Lu, Bin Fang, Di Guo, and Fuchun Sun. 2019. Active Affordance Exploration for Robot Grasping. In *Intelligent Robotics and Applications (Lecture Notes in Computer Science)*, Haibin Yu, Jinguo Liu, Lianqing Liu, Zhaojie Ju, Yuwang Liu, and Dalin Zhou (Eds.). Springer International Publishing, Cham, 426–438. https://doi.org/10.1007/978-3-030-27541-9_35

Jens Lundell, Francesco Verdoja, and Ville Kyrki. 2019. Robust Grasp Planning Over Uncertain Shape Completions. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Nov. 2019), 1526–1532. https://doi.org/10.1109/IROS40897.2019.8967816 arXiv: 1903.00645.

Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. 2017. Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. *arXiv:1703.09312 [cs]* (Aug. 2017). http://arxiv.org/abs/1703.09312 arXiv: 1703.09312.

Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. 2018. Dex-Net 3.0: Computing Robust Robot Vacuum Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning. *arXiv:1709.06670 [cs]* (April 2018). http://arxiv.org/abs/1709.06670 arXiv: 1709.06670.

Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Kenneth Goldberg. 2019. Learning ambidextrous robot grasping policies. *Science Robotics* 4 (Jan. 2019), eaau4984. https://doi.org/10.1126/scirobotics.aau4984

Daniel Maturana and Sebastian Scherer. 2015. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 922–928. https://doi.org/10.1109/IROS.2015.7353481

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. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. https://doi.org/10.1038/nature14236 Number: 7540 Publisher: Nature Publishing Group.

Douglas Morrison, Peter Corke, and Jürgen Leitner. 2018. Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach. *arXiv:1804.05172 [cs]* (May 2018). http://arxiv.org/abs/1804.05172 arXiv: 1804.05172.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew Taylor, and Peter Stone. 2020. *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey*.

V. Nguyen. 1987. Constructing stable grasps in 3D. In *1987 IEEE International Conference on Robotics and Automation Proceedings*, Vol. 4. 234–239. https://doi.org/10.1109/ROBOT.1987.1088008

Rui Nian, Jinfeng Liu, and Biao Huang. 2020. A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering* 139 (Aug. 2020), 106886. https://doi.org/10.1016/j.compchemeng.2020.106886

Andrej Orsula. 2021. *Manipulators*. Open Robotics. https://app.ignitionrobotics.org/AndrejOrsula/fuel/collections/manipulators (Ignition Fuel).

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Bagnell, Pieter Abbeel, and Jan Peters. 2018. An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics* 7 (Nov. 2018), 1–179. https://doi.org/10.1561/2300000053

Takayuki Osa, Jan Peters, and Gerhard Neumann. 2017. Experiments with Hierarchical Reinforcement Learning of Multiple Grasping Policies. In *2016 International Symposium on Experimental Robotics (Springer Proceedings in Advanced Robotics)*, Dana Kulić, Yoshihiko Nakamura, Oussama Khatib, and Gentiane Venture (Eds.). Springer International Publishing, Cham, 160–172. https://doi.org/10.1007/978-3-319-50115-4_15

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and Soumith Chintala. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*.

Lerrel Pinto and Abhinav Gupta. 2015. Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours. (Sept. 2015).

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. 2018. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. (Feb. 2018).

Athanasios S. Polydoros and Lazaros Nalpantidis. 2017. Survey of Model-Based Reinforcement Learning: Applications on Robotics. *Journal of Intelligent & Robotic Systems* 86, 2 (May 2017), 153–173. https://doi.org/10.1007/s10846-017-0468-y

Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. 2017. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. *arXiv:1704.03073 [cs]* (April 2017). http://arxiv.org/abs/1704.03073 arXiv: 1704.03073.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5105–5114.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. 2009. ROS: an open-source Robot Operating System, Vol. 3.

Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. 2018. Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6284–6291. https://doi.org/10.1109/ICRA.2018.8461039 ISSN: 2577-087X.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. 2019. Stable Baselines3. https://github.com/DLR-RM/stable-baselines3.

J. Redmon and A. Angelova. 2015. Real-time grasp detection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 1316–1322. https://doi.org/10.1109/ICRA.2015.7139361 ISSN: 1050-4729.

Máximo A. Roa and Raúl Suárez. 2015. Grasp quality measures: review and performance. *Autonomous Robots* 38, 1 (Jan. 2015), 65–88. https://doi.org/10.1007/s10514-014-9402-3

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas Guibas. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. (Dec. 2016).

A. Sahbani, S. El-Khoury, and P. Bidaud. 2012. An overview of 3D object grasp synthesis algorithms. *Robotics and Autonomous Systems* 60, 3 (March 2012), 326–336. https://doi.org/10.1016/j.robot.2011.07.016

Ashutosh Saxena, Justin Driemeyer, and Andrew Ng. 2008. Robotic Grasping of Novel Objects using Vision. *I. J. Robotic Res.* 27 (Feb. 2008), 157–173. https://doi.org/10.1177/0278364907087172

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. 2020. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (Dec. 2020), 604–609. https://doi.org/10.1038/s41586-020-03051-4 Number: 7839 Publisher: Nature Publishing Group.

Chathurangi Shyalika, Thushari Silva, and Asoka Karunananda. 2020. Reinforcement Learning in Dynamic Task Scheduling: A Review. *SN Computer Science* 1, 6 (Sept. 2020), 306. https://doi.org/10.1007/s42979-020-00326-5

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (Oct. 2017), 354–359. https://doi.org/10.1038/nature24270 Number: 7676 Publisher: Nature Publishing Group.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. 2019. *End-to-End Robotic Reinforcement Learning without Reward Engineering*.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. 2017. Grasp Pose Detection in Point Clouds. *The International Journal of Robotics Research* 36, 13-14 (Dec. 2017), 1455–1473. https://doi.org/10.1177/0278364917735594 Publisher: SAGE Publications Ltd STM.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 23–30. https://doi.org/10.1109/IROS.2017.8202133 ISSN: 2153-0866.

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

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (Nov. 2019), 350–354. https://doi.org/10.1038/s41586-019-1724-z Number: 7782 Publisher: Nature Publishing Group.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chunyu Sun, and Xin Tong. 2017. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics* 36 (July 2017), 1–11. https://doi.org/10.1145/3072959.3073608

Peng-Shuai Wang, Yang Liu, and Xin Tong. 2020. *Deep Octree-based CNNs with Output-Guided Skip Connections for 3D Shape and Scene Completion*.

Bohan Wu, Iretiayo Akinola, Abhi Gupta, Feng Xu, Jacob Varley, David Watkins-Valls, and Peter K. Allen. 2020. Generative Attention Learning: a "GenerAL" framework for high-performance multi-fingered grasping in clutter. *Autonomous Robots* 44, 6 (July 2020), 971–990. https://doi.org/10.1007/s10514-020-09907-y

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1912–1920. https://doi.org/10.1109/CVPR.2015.7298801 ISSN: 1063-6919.

Yun-Hui Liu, Miu-Ling Lam, and D. Ding. 2004. A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain. *IEEE Transactions on Robotics* 20, 5 (Oct. 2004), 805–816. https://doi.org/10.1109/TRO.2004.829500 Conference Name: IEEE Transactions on Robotics.

Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. 2018. Learning Synergies Between Pushing and Grasping with Self-Supervised Deep Reinforcement Learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4238–4245. https://doi.org/10.1109/IROS.2018.8593986 ISSN: 2153-0866.

Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. 2020. *A Framework for Efficient Robotic Manipulation*.

Fangyi Zhang, Juxi Leitner, Michael Milford, Ben Upcroft, and Peter Corke. 2015. Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control. (Nov. 2015).

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. 2018. Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 5628–5635. https://doi.org/10.1109/ICRA.2018.8461249 ISSN: 2577-087X.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. 2020. On the Continuity of Rotation Representations in Neural Networks. *arXiv:1812.07035 [cs, stat]* (June 2020). http://arxiv.org/abs/1812.07035 arXiv: 1812.07035.

Appendices

A Joint Trajectory Controller

B Scripted Policy

C Hyperparameters

Hyperparameters used for the created environment for robotic grasping are listed below for TD3, SAC and TQC actor-critic algorithms.

Hyperparameter	**TD3**	**SAC**	**TQC**
Optimisation Algorithm	Adam (Kingma and Ba, 2014)		
Learning Rate Schedule	Linear, $1.5 \cdot 10^{-4} \rightarrow 0$		
Mini-batch Size	32		
Update Frequency	After Every Episode		
Gradient Steps per Update	100		
Replay Buffer Size	40000		
Discount Factor γ	0.999		
Target Update Rate τ	$5 \cdot 10^{-5}$		
Number of Critics	2		
Activation Function	ReLU		
Exploratory Action Noise	$\mathcal{N}(0, 0.025)$		
Target Policy Noise	$\mathcal{N}(0, 0.25)$	—	—
Initial Entropy Coefficient	—	0.1	
Entropy Target	—	$-dim(\mathcal{A}) = -5$	
Number of Atoms	—	—	25
Number of Truncated Atoms	—	—	3

D Calibration

Figure D.1: Setup used for calibration, where ArUco markers are used as an intermediate frame between the robot and camera.

E Camera Configuration and Post-Processing

F Feature Extraction from RGB and RGB-D Observations

This is a network architecture for the feature extractor that was used for RGB and RGB-D observations for experiment 6.2.2. It is analogous to octree-based feature extractor from subsection 5.2.2. For RGB-D observations, the input image contains an additional channel for depth information. All input channels are normalised to range $0, 1$, where maximum depth of 2 m is used.

TODO: Figure