university of groningen

faculty of science and engineering

Self-Supervised Learning for Joint Pushing and Grasping Policies in Highly Cluttered Environments.

Kamal Mokhtar

university of groningen / faculty of science and engineering

University of Groningen

Self-Supervised Learning for Joint Pushing and Grasping Policies in Highly Cluttered Environments.

MASTER'S THESIS

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at the company Heemskerk Innovative Technology under the supervision of

Internal Supervisor(s): Dr. Hamidreza Kasaei (Artificial Intelligence, University of Groningen)
and
External Supervisor(s): Dr. Cock Heemskerk, M.S. Luuk Doornebosch (Heemskerk Innovative Technology).

Kamal Mokhtar (s3675319)

July 28, 2022

Abstract

In recent years service robots started gaining more attention because of their capability and potential to solve several problems in our daily life. Nevertheless, many challenges remain for robots to optimally act in our people's environment. This project is oriented toward manipulating and pre-manipulating objects in the robot scene to privilege accessibility to objects of interest.

Robots often face situations where grasping a goal object is desirable but not feasible due to other present objects preventing the grasp action. We present a deep Reinforcement Learning approach to learn grasping and pushing policies for manipulating a goal object in highly cluttered environments to address this problem. In particular, a dual Reinforcement Learning model approach is proposed, which presents high resilience in handling complicated scenes, reaching an average of 98% task completion using primitive objects in a simulation environment. We also conduct qualitative testing using the service robot TIAGo in real-life. To evaluate the performance of the proposed approach, we performed two extensive sets of simulation experiments in packed objects and a pile of object scenarios with a total of 1000 test runs in simulation. Experimental results showed that the proposed method worked very well in both scenarios and outperformed the recent state-of-the-art approaches. Demo video, trained models, and source code for the results reproducibility purpose are publicly available https://github.com/Kamalnl92/Self-Supervised-Learning-for-pushing-and-grasping.

Acknowledgments

First of all, I would like to thank my family, who have always been there for me. I would like to thank Heemskerk Innovative Technology for allowing me to do my research project and prosper at the company. I would give special thanks to Luuk Doornebosch and Cock Heemskerk for their gaudiness during the graduation research project. Special thanks to Hamidreza Kasaei, University of Groningen supervisor. He has always been there for me.
Lastly, I thank the Center for Information Technology of the University of Groningen for their support and access to the Peregrine high-performance computing cluster.

Contents

Abstract **I**

Acknowledgements **II**

Page

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Project and Motivation . . . 1
- 1.3 Research Questions . . . 2
- 1.4 Thesis Outline . . . 2

2 Background Literature **3**
- 2.1 Reinforcement Learning and Robotics . . . 3
  - 2.1.1 Guided Policy Search . . . 5
  - 2.1.2 Model-free Skill Learning . . . 5
  - 2.1.3 Learning Predictive Models for Multiple Skills . . . 5
- 2.2 Tangible Examples . . . 6
- 2.3 Literature Review . . . 7
  - 2.3.1 Grasping . . . 7
  - 2.3.2 Pre-grasp manipulation . . . 8
- 2.4 Compression and Selection from the Diverse Approaches . . . 10
- 2.5 Conclusion on Paper Selection . . . 18

3 Methods **19**
- 3.1 Simulation Environment . . . 19
- 3.2 Deep Learning RL Models . . . 21
- 3.3 Push to Grasp . . . 23
  - 3.3.1 Models hierarchical framework . . . 24
  - 3.3.2 Software Architecture Push to Grasp UR5 . . . 24
- 3.4 Push to Grasp Training . . . 25
  - 3.4.1 Goal-conditioned grasping . . . 26
  - 3.4.2 Goal-conditioned pushing . . . 26
  - 3.4.3 Goal-conditioned alternating . . . 26
  - 3.4.4 Reproducibility Problems . . . 27
- 3.5 TIAGo Robot . . . 29
  - 3.5.1 Software Architecture Push to Grasp TIAGo . . . 31
- 3.6 Grasp to Grasp . . . 33
  - 3.6.1 Grasp to Grasp Methods . . . 34
  - 3.6.2 Grasp to Grasp Training . . . 37

4 Experimental Setup and Results **39**

5 Conclusion **47**
- 5.1 Summary of Main Contributions . . . 47
- 5.2 Future Work . . . 48

List of Figures

2.1 Reinforcement Learning setup. 3
2.2 Fridge filled with items. 6
2.3 Infeasible grasp action Scenario. 8
2.4 Human grasping example 9
2.5 Berscheid et al. setup. 13
2.6 Pipeline architecture of Mural et al. 14
2.7 Tang et al. architecture. 15
2.8 Breyer et al. setup. 17
3.1 CoppeliaSim simulation environment interface. 20
3.2 Gazebo simulation environment interface. 20
3.3 The architecture of the generative adversarial network (GAN). 22
3.4 Vector arithmetic for visual concepts. 22
3.5 Overview of the entire system. 23
3.6 Software architecture is developed for the push-to-grasp. 25
3.7 Grasp success rate versus the number of grasp epochs. 27
3.8 Objects pixels Q-value after applying masking 29
3.9 TIAGo robot from PAL Robotics 30
3.10 Converting the work push to grasp 30
3.11 Software architecture is developed for the push-to-grasp 32
3.12 Top image shows the RGB image of TIAGo working scene. 33
3.13 A scenario where grasping the goal object directly is impossible. 34
3.14 Setup in CoppeliaSim simulation environment. 35
3.15 Projection of the pixel-wise Q-values over the x-axis and the y-axis. 36
3.16 Grasp success rate during the 5 training stages. 37
3.17 Grasp success rate during training push-grasp stages 38
4.1 Two examples of the generated scene are packed and pile scenarios. 40
4.2 Sequence of snapshots showing the robot performing decluttering by pushing. 41
4.3 Typical scenes with different number of objects. 43
4.4 A sequence of actions of grasping the goal object. 44
4.5 Two numerical solutions 45

List of Tables

2.1 Papers and key words criteria 12
4.1 Simulation consistency results on structured scene 41
4.2 Simulation results on 100 different complicated scenes 42

1 Introduction

This section will cover an introduction about the research problem, background, motivation, and the research questions. Lastly, an overview structure of this thesis report is covered.

1.1 Background

Service robots are robots that can mimic human behavior. Unlike traditionally, where service providers always have been people, some services are handled by robots. We see tangible examples where robots are taking certain tasks, such as smart robots at hotels, retail stores, and meal delivery [PV16] [HKW18] [Fri15]. Realistically, the optimal functioning of service robots until this moment still faces many challenges. In this project, the focus is on the healthcare sector. In the next 20 years, the population aged 65 years and older in the Netherlands will nearly double [Bet19]. Service robots are gaining popularity and attention in the care field due to this aging society and the ever-increasing healthcare costs. Society 5.0 expected that humans and robots coexist in the same environment, cooperating together to improve the quality of life by providing numerous services [GEB20].

In the HiT-Care project, Heemskerk Innovative Technology (HiT) is developing a semi-autonomous service robot, which is able to perform certain generic tasks, i.e., picking up a cup in a clear scene. However, it relies on human operators using haptic telemanipulation for more complex tasks.

Manipulating objects in a cluttered environment, e.g., picking up an item from the fridge, comes with many challenges. In such a scenario, it involves several processing steps, such as object segmentation, recognizing objects, finding the object of interest, motion and manipulation planning, and the robot self-awareness of the situation (i.e., robot stuck, re-plan).

1.2 Project and Motivation

This project is focused on the pre-manipulation of objects in the scene to be able to grasp the object of interest. A study of the problem is first conducted, where problem criteria are stated to be able to find the most relevant previously done work to the problem. The criteria concerns problems such as Bin clearing, clutter set-up in the scene, novel object grasp generalization, and more which will be covered in detail in section 2. Establishing the relevance of previously done work made scoping on papers as a starting base for this project more prominent.

From previously done work, it became obvious that Reinforce learning (RL) is prevailing in such problem domains. Reinforce learning endeavors to achieve robotics complex behavior. Problems that are sophisticated to engineer (hand-craft behavior) RL provides well set of skills to learn by interacting with the environment. Problems that concern the robot to make decisions in sophisticated scenarios that might not be seen before (i.e., the objects in the scene to be pre-manipulated are set up differently than previous memories) [KBP13].

The RL model will give the robot the ability to make a decision by providing a visual indication to pre-manipulate the scene. E.x., there is a fridge shelf scene in front of the robot. The robot has a target object of interest. Nevertheless, after evaluation, there is no successful grasp that can be placed. The scene is quite occluded with other objects. The robot has to make pre-manipulation to the scene, i.e., pushing, rotating, picking, and placing obstacles somewhere else, such that it is able to safely grasp the object of interest.

1.3 Research Questions

1. What methods would be suitable for a service robot to pick up an item of interest in a cluttered scene, even when the object is not initially feasible to be grasped? What further limitation the selected method would impose?

2. What other problem aspects could be added to the already existing picking scenario? And how to solve it?

3. What other improvements could be brought to the already existing state-of-the-art systems to achieve a new milestone?

1.4 Thesis Outline

The thesis starts with the introduction section about the problem 1. Section 2 is the literature research about state-of-the-art approaches to the problem, ending with a summary of the chosen work as a start point for this project. Section 3 is about methods, firstly with an overview of the used tools and then more theoretical background about the developed approach. The approach consists of mainly three parts, the push to grasp the goal object, the grasp to grasp of the goal object, and the setup development to test on the real robot. Section 4 is about the experiment setup and result. It covers how the test is set up and discusses further the obtained results. Finlay, conclusion and future work is discussed in section 5.

2 Background Literature

In this chapter, will cover an introduction to reinforcement learning, and problems it solves. Also the prevalence of reinforcement learning in the robotics fields. Furthermore, other techniques will also be covered for possibilities for solving the research problem. Finally, educate layout of the problem and the chosen method will be introduced.

2.1 Reinforcement Learning and Robotics

Learning by interacting with our environment is a concept we think of as the nature of learning. For example, an infant playing and exploring nature has no explicit teacher. But instead, with the sensory and motor skills, they can interact and develop in the environment to reach goals and desires [SB18]. Some valuable terminology when talking about Reinforcement Learning (RL) application:

$$A_t : \mathit{Action\ at\ time\ t}$$

$$R, R_{t+1} : \mathit{Reward\ at\ time\ t,\ and\ t+1}$$

$$S, S_{t+1} : \mathit{State\ at\ time\ t,\ and\ t+1}$$

RL is a learning process to discover what to do when facing a scenario, which means learning to map situations to actions. Aiming to maximize the gained reward. See Fig. 2.1. We have an agent (i.e., a robot), the agent performs a certain action in the environment (i.e., moves an object from one place to another), the environment state (S_{t+1}) has changed, and a reward is generated. Both are fed to the agent, so the agent would know what environment state it is in and whether its prior action(s) has been beneficial. Optimally, the agent learns from the experience and optimizes its behavior.

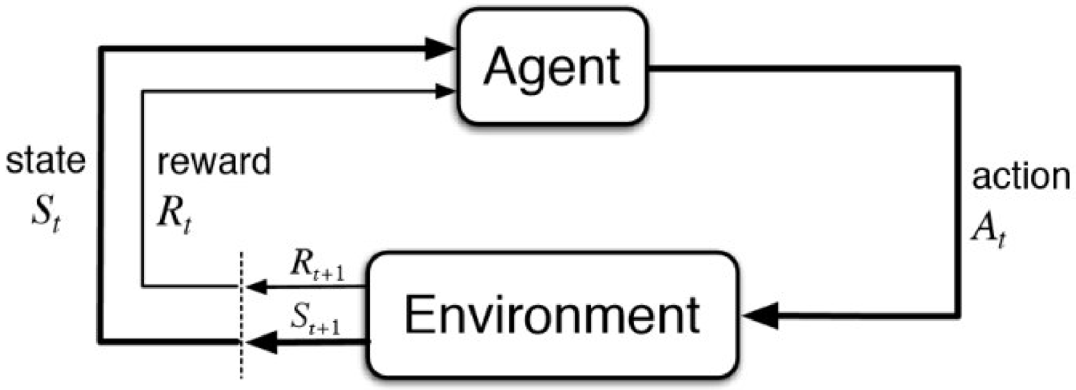


Figure 2.1: Reinforcement Learning setup diagram [Bha18].

The literature on RL on diverse robotics applications is richer than robot object manipulation. For an overview of RL use in the robotics field, I will discuss it generally. Nevertheless, object manipulation will be discussed briefly and scoped specifically to the research problem of this thesis. Relevant papers are covered in a later section 2.
In contrast, many real-world problems in robotics are best presented with high-dimensional, continuous states and actions. That introduces many challenges in the RL setting. The modern anthropomorphic robots introduce robots characterization of many degrees of freedom resulting in higher dimensionality tasks. Experimenting with real robots remains costly and hard to reproduce by others.

That is not the case with using simulation. Nonetheless, it cannot be fully replaced by simulation. Modeling errors due to the deficiency of the physics engine realizing reality and the inconsistency of the robot's motor control would accumulate, causing substantial difference [KBP13].

Not every RL method is equally suitable for the robotics domain. Several tasks scale to interesting tasks using a model-based approach. It employs policy search rather than value function-based approach's [KP09] [KS04] [CAN08]. These approaches stand in contrast to many of the mainstream data-driven RL.

The state and actions of most robots are inherently continuous, and the dimensionality can be high. Thus we face the "Curse of Dimensionality" [Rus97]. The term was coined by Bellman when he explored optimal control in continuous states spaces, facing an exponential explosion of discrete states and actions. An example of such a task ball-paddling task for a robot arm with a ping-pong racket. Legitimate representation of a robot's state would consist of its joint angles and velocities for each of its n degrees of freedom (typical 7 degrees of freedom), also the Cartesian position and velocity of the ball. Remaining, we would need the robot's actions, which would consist of the motor command, typically torque or acceleration commands [KBP13].

Simulation can alleviate many problems; earlier presented an example of the ping-pong ball-paddling task. One needs to put the ball back on the rack in each iteration, which makes real-world samples expensive in terms of time and labor. Algorithms that can learn from small examples are crucial for specific applications.

Direct transfer from simulation to an actual real robot is challenging due to errors in the model; having it trained in a suboptimal simulation environment leaves the model lacking reality. The dynamics of the robots are also not constant. They can change to many external factors such as temperature to wear. One way to overcome the issue is to have accurate models that are used as physics engines in simulation environments. Unfortunately, to the best of our knowledge, this does not exist yet at the point of writing this thesis report. Nevertheless, to mitigate this issue, methods exist to tackle the problem of the sim-to-real gap. An example of this work by Peng et al. [Pen+18]. They use dynamics randomization. Training dynamics are randomized so that the policies learned can handle the different dynamics faced, aiming to minimize the reality gap and have a close performance compared to the simulation one.

Last and not least, an often underestimated problem is the goal specification. It is achieved by designing a good reward function. In RL, the goal of the task has implicitly specified reward. It is crucial to define a good reward function in RL. Giving a reward upon achievement, i.e., agents getting a reward once it achieves the final goal, introduces the risk of the agent model being unable to converge. Having the agent receives a reward only at a far distant goal introduces the problem of reward sparsity. One way to deal with it is using Hindsight Experience Replay [And+17]. It allows sample-efficient learning from rewards that are sparse and binary—avoiding the problem of complicated reward engineering. HER is an off-policy RL algorithm that can be seen as a form of implicit curriculum. Implicit curriculum is supplying the agent with problems capable of solving and gradually increasing the difficulty of the problems. HER is inspired by human task learning. Imagine a case where a hockey player is attempting to score a hockey puck. In unsuccessful attempts, the hockey puck would be off. We humans would realize the margin error and adjust for that. Unsuccessful attempts are also beneficial for learning. Doing so with robots improves the sample efficiency and the learning speed [And+17]. Good reinforcement learning algorithms exploit the reward function unexpectedly, especially if the RL is done locally and not globally. For example, if the contact between the racket and the ball for the ball-paddling task example, many locally optimal solution policies would be to learn the be an attempt to basically keep the ball on the racket [KBP13].

Robotics learning is at the intersection of machine learning and robotics. Robotics is an interesting medium to study. It reveals an aspect of how we humans and animals learn to manipulate objects around us, for one. Nevertheless, that is not the only aspect. For the sake of the center point of this thesis on manipulation, I will bound it to that.

RL in robotics is formalized by defining a state and action space. The dynamics reveal how the actions affect the environment's state. The state space includes both the robot and the world/environment states. The state of the world is not directly observable. Instead, robots are equipped with sensors, e.g., RGB-D sensor. The sensors are used to infer and reflect the state of the world around the robot. Achieving the goal may be defined either as a target state or as a reward function to be maximized. The goal of the controller (policy in RL literature) is maximizing the reward when executed by knowing the states by direct or indirect observation of the environment and model of the system dynamics. The problem can be solved with classical methods, for instance, using optimal control. The latter uses the knowledge of the dynamics model to explore a sequence of actions, taking the agent from the start/current state to the goal state [Iba+21]. However, suppose the dynamics model is unknown, which is also relevant for the paradigm of RL. In that case, it is required to sample trajectories in order to learn how to control the robot and maximize the reward. There are two different RL methods. Model-based RL and Model-free RL. Model-based RL uses samples to learn the dynamics model of the environment, privileging optimal learning policy. In contrast to model-free methods, the dynamics are not explicitly modeled, hence the name. Requiring the model-free RL to directly learn by interaction with the environment [Iba+21].

Deep neural network policies can alleviate the need to manually design policies, leading to a moderate amount of generalization to the initial condition state as well as facilitating end-to-end training, From perception to control.

We will discuss a number of cases using RL techniques, inspired by Ibarz et al. [Iba+21]. It will also shed some light on the different directions that one might take to tackle the problems at hand.

2.1.1 Guided Policy Search

As the method name suggests, it is a neural network policy guided by another RL method, typically model-based. An example of such a method done by Levine et al. [Lev+16] the PR2 robot learns to place trapezoid blocks in a shape sorting cube. The neural network policy is referred to as a global policy. It is trained to perform tasks from raw data image sensory data, while the local policies would operate in low-dimensional (learns more efficiently) and would provide policies to insert the shape into a specific fitting hole.

2.1.2 Model-free Skill Learning

Model-free algorithms overcome some of the limitations of guided policies search. There is no need to decompose a task into multiple low-dimensional state representations, which will require substantially lower training time. Example basic Lego cube stacking task by Haarnoja et al. [Haa+18] required less than 2 hours. Compared to more complex tasks discussed previously (Levine et al. [Lev+16]) required around 20 minutes. However, as the task variability increases, one can see that the number of local policies will linearly increase as the initial states increases in number. The result of the increase in the training time will tend to be close between the two methods.

2.1.3 Learning Predictive Models for Multiple Skills

Many tasks share commons skill in common. One learned skill can be valuable in another skill which would save learning time. If robots learned from scratch for all the tasks, it would be impractical and insufficient for general purposes. One application of that is multi-task learning of vision-based manipulation skills.

2.2 Tangible Examples

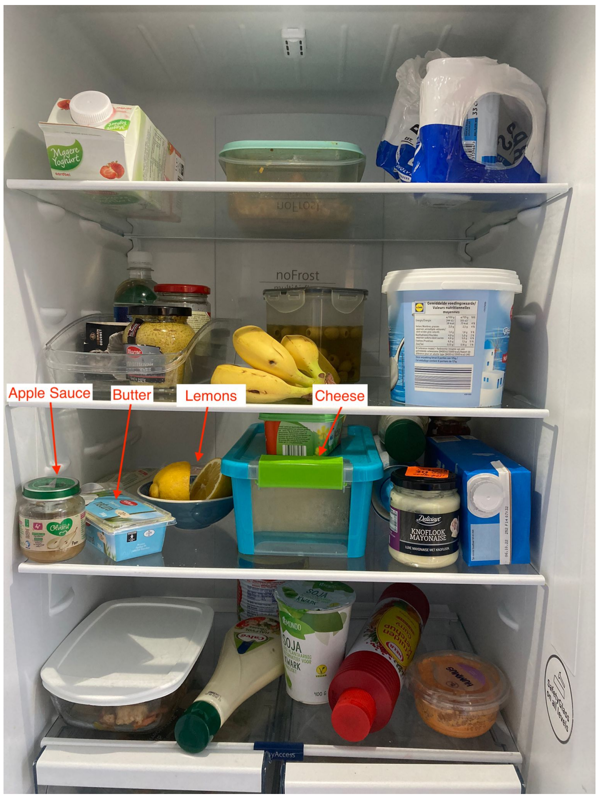


Figure 2.2: A typical case of a fridge filled with items that is easy for us humans to grasp any object within and almost impossible for robots to handle it in high resilience.

As depicted in Fig. 2.2 humans can grasp any objects in a couple of seconds and with no effort (low cognitive demand). Such daily tasks require multiple skills. First, we need to recognize and locate the object of interest. In this case, we assume it is the sliced lemons in the blue bowel. Is it accessible? If yes, we grasp it without causing chaos in the scene. Else we need to reason to remove obstacles from the scene so that our goal object is accessible. There are two options for clearing either from the right side or the left side of the goal object. The right side looks more complex as more objects on the right might fall down. A more efficient approach is to clear from the left side. As you may have noticed, the apple sauce jar is on the edge of the fridge. Grasping and removing the butter has to be elegant action. Assuming after removing the butter, the lemons are accessible to the robot arm. We might need additional shifting to create an accessible grasp. The motion plan and collision avoidance are vital aspects for the robot to perform it safely in a human environment [Wan+17].

Last and not least, the dexterity skill, in this case, is the grasping of the object(s). Not all objects are grasped easily. Some objects impose further constraints on grasping, such as plates. From a neuro-biomechanical processes perspective, Two key control processes that facilitate this resilient and smooth behaviour:

- **Predictive cognitive-motor process** that guides manipulation procedure by anticipating action outcomes.
- **Reactive sensorimotor process** that provides important error-based information for movement adoptation.

For motion plans, we humans optimize for efficiency in the form of muscle activation patterns that are advantageous [Ale97]. Unlike robots' optimization techniques, covariant gradient techniques assist in improving the trajectory quality [LL22].

2.3 Literature Review

In the previous section, we demonstrated the complexity of the problem. This section will discuss different state-of-the-art techniques used to approach this problem.

2.3.1 Grasping

Autonomous grasping is a challenging problem that has undergone enormous improvement in recent years, especially with Artificial Intelligence (AI) practical approaches. Robots' dexterity in grasping is one of the most primitive manipulations that mediate robots to operate other tasks [Dua+21]. We can see this prominently in our daily life, where we need to grasp numerous objects (i.e., tools) to operate different tasks further.

Classical approaches rely on a model-based approach, where pre-knowledge of the object model is used to estimate force closures and grasp orientation. These methods impose limitations, as it is essential to have accurate modeling of the objects, and that is not always possible [SEB12].

A mathematical model describing objects is crucial to predicting the physics of the hand and the grasped object. Various conditions arise during grasping, which is essential to be expressed by the model to achieve stable behavior resisting the different loading conditions [PT16].

The substantial increase in computing power has allowed the deep-learning methods to prevail in solving the issue. It is focused on data-driven methodology; the data are perceived in the form of images and depth information about the scene. Then in a simulation setting, the robot exploits the learned visual features to perform a grasp action. The advantage of such an approach is that it does not require explicit knowledge of the objects compared to the classic approaches, i.e., model-based [Dua+21]. State-of-the-art and leading work that uses observed geometry to evaluate antipodal grasp pose [PP15] [Pas+17] has sparked enormous interest in the deep learning approach. Inspired by the efficient framework, many studies have taken different approaches in representing the data around the deep learning model, or different models have been used, i.e., Generative Residual Convolutional Neural Network (GR-ConvNet) as model [KJS20]. Another example is research work that generates multi-view 3D object grasping, where objects are represented in the orthographic representation of the depth image from three virtual orthographic views representing the object. After one of the views is selected, it is fed to an auto-encoder that produces a heat-map for a grasp configuration selection [KK21].

2.3.2 Pre-grasp manipulation

The thesis mainly focuses on self-supervised learning for pushing and grasping objects in highly cluttered environments using deep RL. The push action is a pre-manipulation step facilitating a future antipodal grasp action depicted in Fig. 2.3.

There has been number of study's evolving around the concept of the synergy between pushing and grasping, i.e., primitive motion push to grasp [Xu+21] [Fuj+20] [BMK19] [Mur+20] [Tan+21] [YLC20] [Hua+17]

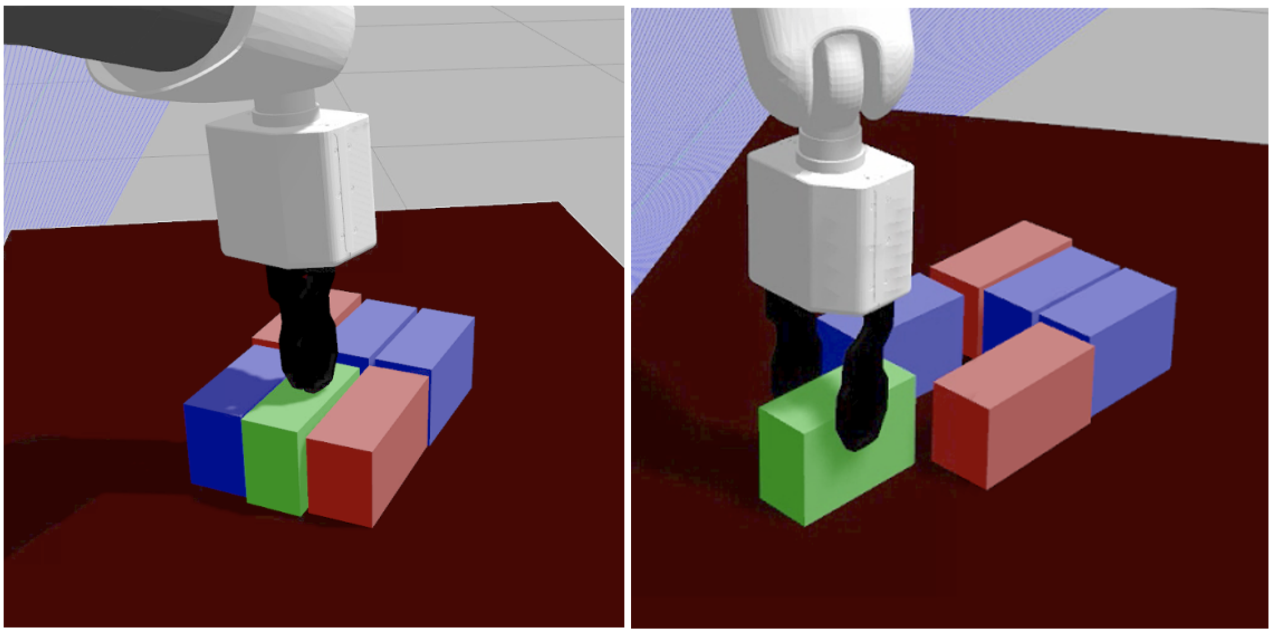
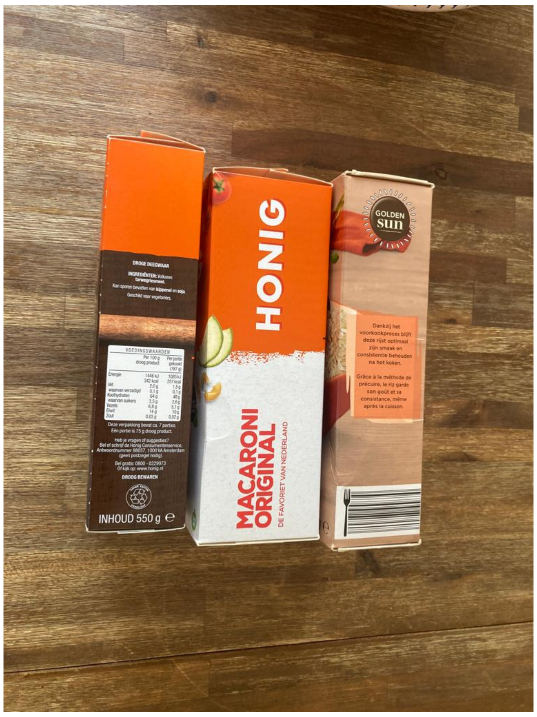


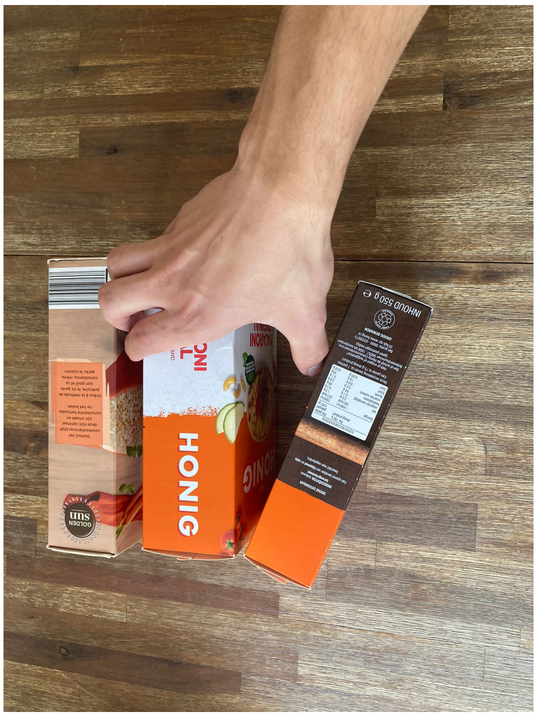
Figure 2.3: Infeasible grasp action as a result of densely packed blocks, requiring pre-manipulation push action: *(left)* in this example, the target object, shown in green, is not graspable due to other objects present; the robot pushes the target object in a way to make it graspable; *(right)* finally, the robot grasps the object successfully [MHK22].

RL is widely-used when there is a demand on a cognitive capability [MCC20], i.e., approaching a goal object by considering other objects in the environment. This field of occluded environments has not been studied much, and most studies concern bin picking (all objects are picked from one bin and placed in another) [Fuj+20]. Little work is done about having a goal object in a cluttered scene, and a robot arm is attempting to grasp it, which is a fundamental trait for a service robot. This requires efficient pre-manipulation of other objects to be able to grasp the goal object [MHK22].

Humans have a rich set of manipulation strategies desirable for service robots. These traits are appropriate for a service robot to be able to perform a task in our people's environment autonomously [CZP08], many AI applications are inspired by human behavior. In Fig. 2.4 we let a human perform the grasp action of a goal object. Thanks to evolution, it is quite easy for us humans to perform such a task. On the other hand, it is quite difficult for robots to perform such a task; regardless of the mechanical limitations that a robot arm, the intelligence that we perform in such a task. In Fig. 2.4 the goal object to be grasped is the middle object; the human performs a push action to the object with a thump while the other finger tilts the goal object making space for a grasp action. This inspires the idea of doing it sequentially by a robot arm, by first pushing, then performing a grasp action.



(a) Densely packed scenario pre pre-manipulation.



(b) Densely packed scenario post pre-manipulation.

Figure 2.4: Example of typical action by a human is grasping a goal object and pre-manipulation

Nevertheless, this is only one example of many manipulations that we humans perform. In another scenario, objects that are in a configuration that is impossible to grasp, i.e., a flat object such as a compact disc (CD), need to be slid to an edge of a table to permit a grasp action [Kap+10]. In another case, objects cannot be grasped due to the presents of other objects disabling a grasp action. The latter is the focus of this thesis; pre-grasping manipulation such as pushing is a popular approach that permits consequent object grasping. For a demonstration of this behavior, see Fig. 2.3. In the first state, the object is impossible to be grasped; the robot arm pushes the object further away from the clutter such that it permits a grasp action. There have been many recent studies concerning this problem, which we will present in the remaining of this section [MHK22].

Introducing Markov Decision Process (MDP) with a state-space, the scene, action space, transition distribution, and reward functions are elements of the RL approach. RL is bound to data consumption, and a time-dependent task introduces the problem of spars-rewards. The solution to this problem in an MDP setting is to map the current state to action and make the skill of shifting explicitly dependent on the grasping [BMK19] [MHK22].

Most of the former works are top-down grasps; however, that is not sufficient at all times. Especially if the aim is to have safe, stable grapes, it is crucial to include a grasp evaluation of objects in all different orientations. A method that uses reasoning to find an effective grasp sequence to grasp a goal object in a structured clutter uses a cascaded process. By first learning the grasp distribution for a single isolated object and then a discriminate model that captures the collision of the gripper and the present objects in the scene. It is possible to eliminate specific unattainable grasps. The Variation Autoencoder (VAE) is used to evaluate the grasp distribution of the segmented point cloud of the target object as well as the collision check [Mur+20] [MHK22].

2.4 Compression and Selection from the Diverse Approaches

This section will cover the most relevant papers for our research question. The solution has to be practical because it would provide a certain degree of handling occluded environments and be able to manipulate objects in the scene. We collected the most relevant papers and studied their different approaches. Then shaped vital problems common in most research papers and related to the practical problem. These fundamental problems are summarized in table 2.1. Each paper is discussed abstractly. At the end of this section, the paper that is chosen as a starting point for this project is substantiated and other papers for inspirations or improvements.

For clarity, the key problems will be defined first:

- Bin clearing: The robot grasps all the objects that are present in front of it and places them in another location. Clearing from one bin to another bin.
- Target driven: Instead of clearing the bin, we aim to grasp an object of interest, e.g., Grasping an object from the fridge.
- Target object invisible: The target object could be that it is initially not visible, requiring the robot to manipulate the scene until it becomes visible.
- Clutter axis: The clutter could be present in two dimensions. Objects are placed next to each other. In other cases, it could stacks, which introduces a third dimension in occlusion.
- Objects in the scene, Hight's variations: Objects in the scene have different heights and at least an $8cm$ difference.
- Structured clutter dense: Scenes where the objects are relatively structured and close to each other. It is difficult to put a metric for this, but we evaluate it visually, see Fig. 2.7. Systems that are able to solve structured cluttered dense scenes, are also able to solve structured uncluttered density scenes.
- Unstructured clutter dense: Scenes where the objects are relatively structured and close to each other. It isn't easy to put a metric for this, but we evaluate it visually, see Fig. 2.3. Systems that can solve structured cluttered dense scenes can also solve structured uncluttered density scenes.
- Walls around the clutter: If walls are present around the working scene, it imposes a limitation on the grasp freedom, i.e., a cube in an inner corner of a box is impossible to be grasped directly.
- Camera location: The camera location with respect to the robot arm/working scene. There are different locations where the camera could be located. This criterion is relevant for practical reasons and the desire to have a system that would work on a service robot looking into a shelf. Some systems are independent, having a strict camera location. Other such as those that reconstruct the 3d scene is dependent.
- Novel singular object grasp generalization: During training, not all objects are included. The robot might face objects that do not previously seen. It is helpful that the system is evaluated with objects that are not included during training. The evaluation can be tested with only one object in the scene.
- Novel cluttered objects grasp generalization: Same as the definition above, but here we have more than one novel object in the scene.

- Object weight and shape resembling kitchen objects: Systems trained and evaluated on objects close to what we would have in a kitchen or household are more relevant to our work.
- Top-down grasping: This approach of grasping the object is only grasped approaching from the top side. This could be a limitation to certain objects, such as a plate, where the necessity of a side grasp is required.
- Pushing for grasping: Is the case where there is no valid grasp in the scene due to high clutter of the object, meaning all objects are placed close to each other, see Fig. 2.3.
- Grasping for grasping: Typical handling we humans do when we would like to fetch something from the fridge. Pushing is not always valid. There are cases where we ultimately need to remove obstacles to reach the object of interest.
- Object grasped confirm: When the robot attempts to grasp an object is useful to see if they grasp action of the object actually occurred, Some of the methods use the camera. Others use the antipodal gripper distance or pressure sensor.
- Manipulation model free: Learning a policy and not learning an accurate forward model.
- Robot arm collision pre-check: Before executing a motion plan for a grasp action, it is essential to check for a possible collision.
- Gripper collision check: Not all methods check for the robot arm's whole body but only on the gripper.
- Closed loop Environment change: Suppose the robot attempts to manipulate the environment. If it changes before it executes the action, the robot has to adapt to its goal. Then the system can be called a Closed loop.
- Testing on actual robot: The physics of the simulation environment is different than in real life due to inaccuracies of the simulations. After training a system, it is robust to demonstrate that the system works in real life too.
- SimToReal gap (visual): There is a gap between the model experience in simulation and what it faces in real-life. In this case, the model deals with the visual aspect gap.
- SimToReal gap (dynamics): Same as the previous gap but dynamics related.
- Source code available: A source code provides a good start for the projects and helps validate the paper results.

Key Problem	[Zen+18]	[BMK19]	[Mur+20]	[Tan+21]	[Xu+21]	[YLC20]	[Hua+21b]	[Kal+18]	[Bre+21]
Bin Clearing	✓	✓	X	✓	X	X	X	✓	✓
Target Driven	X	X	✓	X	✓	✓	✓	X	X
Target Obj. Invisible	X	X	X	X	✓	✓	✓	X	X
Clutter Axis (if z then stacked)	3D	2D	2D	3D	3D	3D	2D	2D	3D
Objects in the scene, Hights variations	X	X	✓	✓	✓	✓	X	✓	✓
Structured Clutter Dense	✓	✓	X	✓	✓	✓	✓	✓	✓
Unstructured Clutter Dense	✓	X	X	X	✓	✓	X	✓	✓
Walls Around The Clutter	X	✓	X		X	X	X	X	X
Camera Location	nxt. WS.	w. (top WS.)	w. (front WS.)	cor. WS.	opp. to arm	opp. to arm	top WS.	nxt to arm	3D build)
Novel Singular Obj. Grasp Generalization	✓	✓	✓	✓	✓	X	X	✓	✓
Novel Cluttered Obj. Grasp Generalization	✓	X	✓	✓	✓	X	X	✓	✓
Obj. Weight&Shape Resembling Kitchen Objs.	X	X	✓	✓	X	X	X	X	✓
Top-down grasping	✓	✓	X	X	✓	✓	✓	✓	X
Pushing for Grasping	✓	✓	X	✓	✓	✓	✓	✓	X
Grasping for Grasping	X	X	✓	X	X	X	X	X	X
Grasping Model Free	✓	✓	✓	✓	✓	✓	✓	✓	X
Obj. Grasped Confirm	X	X	X	X	X	X	X	✓	X
Manipulation Model Free	✓	✓	✓	✓	✓	✓	X	✓	✓

Table 2.1: Papers and key words criteria

Continued on next page

Key Problem	[Zen+18]	[BMK19]	[Mur+20]	[Tan+21]	[Xu+21]	[YLC20]	[Hua+21b]	[Kal+18]	[Bre+21]
Robot Arm Collision Pre-check	X	X	X	X	X	X	X	X	✓
Gripper Collision Check	✓	X	✓		X	X	X	✓	✓
Closed Loop Env. Change	X	X	X	✓	X	X	X	✓	✓
Testing On Actual Robot	✓	✓	✓	✓	✓	✓	✓	✓	✓
SimToReal Gap (visual)	X	X	X	X	X	X	✓	X	X
SimToReal gap (dynamics)	X	X	X	X	X	X	X	X	X
Source Code Available	✓	✓	X	X	✓	✓	X	✓	✓

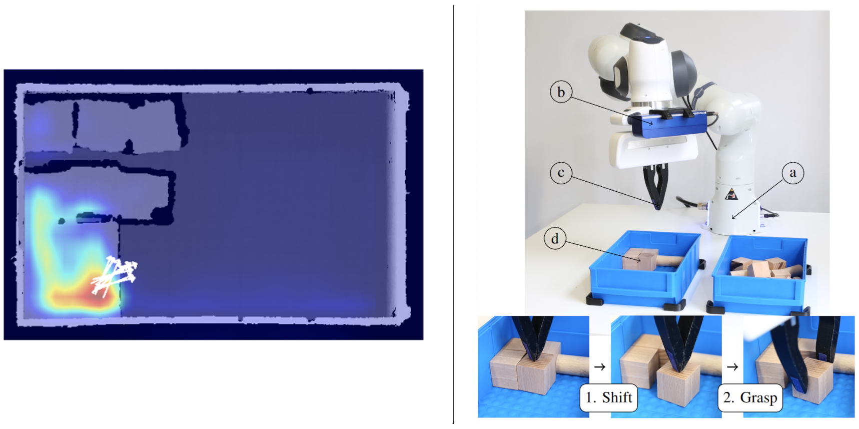


Figure 2.5: Setup of Berscheid et al. [BMK19] *right* of the image Franka rbotic arm (a), depth camera (b), custom 3-D printed arm gripper jaws (c) two industrial bins with objects (d), *left* of the image heat-map typical output of the model predicting push direction at a certain pose [BMK19].

In the remaining of this section, we will discuss the papers selected in the table 2.1, starting from the first work of Zeng et al. work [Zen+18] is one of the early works that started with the idea of pushing facilitating future grasps of objects. Two DenseNets-121, one for pushing and the other for grasping, trained similarly to generative adversarial neural networks in an RL setting. The pushing reward is calculated from successful grasp action, while the push reward if the grasp probability has increased after a push action. The orientation and the push direction are discretized to 16 predefined orientations and directions. The testing results of the completion (bin picking all objects in the scene) in the simulation environment is 82%. RL set-up as Markov decision process (MDP) is used. RL in robotics is subject to data consumption and time-dependent tasks, including sparse rewards. Thus, the process is reduced to a single-time step policy, mapping a discrete state and producing action.

The state is the orthographic depth image of the scene. Whereas the state of four parameters (x, y, a, s) element of $A = R^3 \times N$ in the planer subspace. The x, y a are the spatial coordinates extracted from the image, x, y-axis position, and a representing the rotation angle around the z-axis (Yaw). The relationship in terms of frame transformation reference between the image (state) and the robot arm needs to be known.

Berscheid et al. [BMK19], introduces a sliding window that crops the orthographic image at the given translation (x,y) and pre-set of 20 rotation angles. This way, reward estimates are calculated for each motion primitive. Consequently, max reward motion primitive can be inferred. The synergy between pushing and grasping during inference is decided with predefined threshold values. Depending on these values in a binary decision tree, it can be decided whether to push or grasp objects at a certain coordinate.

The Franka robotic arm with the 3D printed gripper is depicted in Fig. 2.5. For state observation, the camera is set on a predefined orientation above the bin, looking down vertically. This means after each action; the camera must return to observe the updated state. The objects positioned in the experiments are densely cluttered. The robot arm can't grasp the objects if it is not feasible. Most objects have quite close heights. The objects used are primitive in shape. During testing 20 objects are placed in a bin. The system is able to grasp $98.4 \pm 1.0\%$ and 0.07 ± 0.01 and grasp attempts of 122. The grasp attempts are quite high in the number of attempts. With 20 objects in the scene, it seems that the synergy between push and grasping is low. Surprisingly, testing on more complex objects such as screw drives, tape, Pliers, etc., it seems that the system is doing a decent job, reaching a value of $92 \pm 7\%$ [BMK19].

Murali et al. [Mur+20] has an interesting approach and is substantially different from the other approaches. They use a pipeline to reach a grasp action of the goal object or a grasp action to remove objects. The objects used resemble what one would find in a household, specifically in a kitchen. The pipeline first starts with RGB-D observation, which is the input to instance segmentation, and a 3D point cloud is generated around the goal object. The 3D point cloud around the goal object is cropped. The variational auto-encoder model evaluates feasible grasp actions at a different orientation. Unlike with auto-encoders, with variational auto-encoders, continuous latent space is generated. Allowing for random sampling and interpolation.

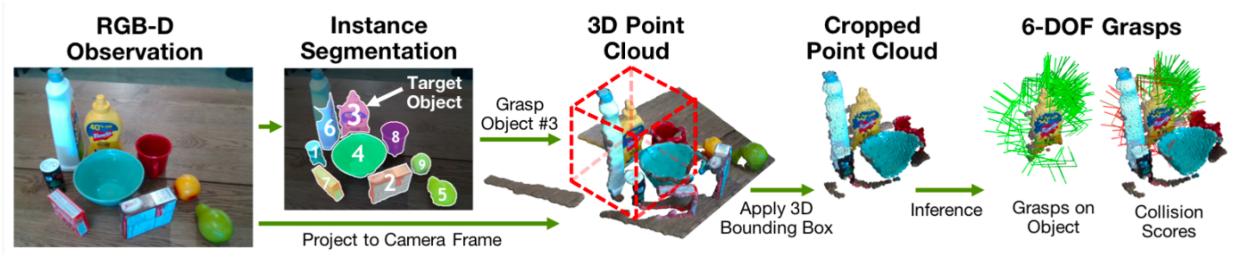


Figure 2.6: The pipeline architecture of Mural et al. [Mur+20], scene observations at the left, then instance segmentation, then cropping around the goal object (the blue bottle), the evaluation of the grasp of the goal object stand-alone and with respect to the clutter around it.

The aim is to estimate the posterior grasp distribution $P(G^*,X)$, Where X is the point cloud observation, and G^* is the space of successful grasps. The grasp learning distribution is inferred for a single isolated object $P(G^*,X_0)$. Then there is a discriminative model $P(C|X,g)$ which has been called *CollisionNet* that captures the collision between the gripper at pose g and clutter observed as X. The system is able to infer which object to remove to reach maximum grasp success. In Real robot experiments, they are able to reach an 80% success rate (grasping the goal object after pre-manipulation) on 51 trials. The competent of this work, having objects of different heights is more challenging compared to Berscheid et al. [BMK19], There is a challenging factor of having the gripper colliding with other objects and causing chaos in the scene.

Tang et al. [Tan+21] demonstrated collaborative pushing and grasping policies in a dense scene. This work on the application level is exclusive in the clutter occlusion. They have their scene stacked in the z direction. Adding dimensional increases the degree of complexity of both state and action space. The train/testing scenes consist of objects occluded and stacked (i.e., tennis ball on a plate which all placed on tea box), surrounding it all different conventional kitchen items such as food cans. To see scene images, I refer the reader to the original paper [Tan+21].

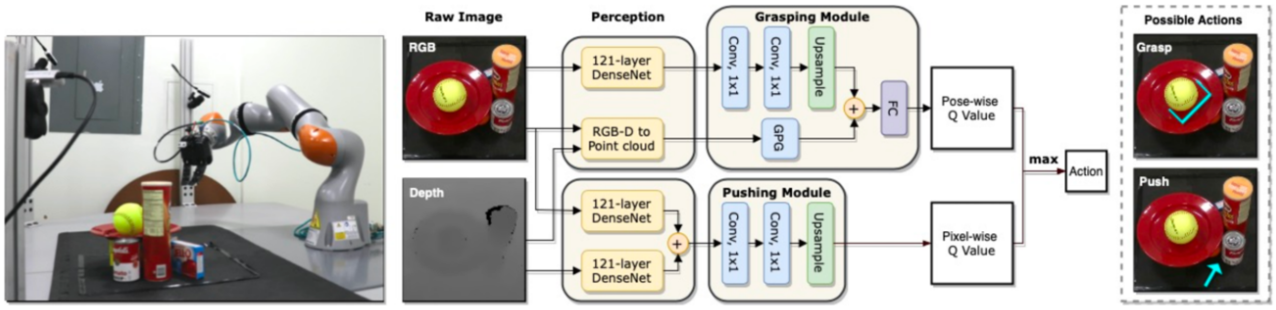


Figure 2.7: Tang et al. architecture, working scene on the left. Perception modules for both the grasp and push and final evaluation for decision making between push and grasp [Tan+21].

Furthermore, some objects are impossible to be top-down grasped (e.g., a plate), which is quite relevant to this research thesis. Two strong contributions in this work, not limited to the top-down grasping of manipulated objects and features of the stacking, resemble typical fridge shelf scenarios.

The architecture is relatively inspired by the Zeng et al. work [Zen+18]. Represented as an MDP problem, and the goal is to find an optimal policy in the RL setting. The Grasp module used taken from Pas et al. [TP18] work. It detects grasp poses on novel objects presented in the clutter from a single viewpoint cloud. The grasp module considers both the input point cloud and the geometric parameters of the robot hand. The output is a set of output parameters that are expected to be good grasps. The paper is not explicitly clear about the synergy between the grasp and push actions. The simulation testing scenes for novel objects are sparsely placed. The real-life testing of complex z direction clutter only one case demonstrated. Nevertheless, the good lesson to take from this paper is their use of advanced modules for grasping in occluded environments in place of training from complex grasping tasks from scratch.

Yang et al. [YLC20], impelled to the problem of invisible target objects. Objects in included environments not only might be initially not graspable but also not visible at all. Pushes are needed in areas where most likely the target object might be located. The problem architecture is closely related to the previously mentioned format with additional use of the critic-policy structure for the target object. The core technical contribution is two-fold, motion critic for target-oriented pushing and grasping with respect to the motion primitives and. Secondly, Bayesian-based policy for target exploration and a final classifier to coordinate push grasp action decision making.

For the task execution, top-level policies combine Q predictions and domain knowledge. Which comes from two sub-tasks:

- *Exploration*: Target is not visible, requiring the robot to break the structure to make it visible.
- *Coordination*: Though the target is clearly visible, the target might be closely surrounded by other objects, not permitting grasp action.

The explorer policy uses the height map distribution of the workspace and the history of previous actions as domain knowledge. Once the target is found, the coordination policy coordinates the pushing and the grasping with respect to the clutter around the goal object. For target-agnostic search (exploration policy) Bayesian based π_e (push actions only) is used. The product of the push maps (produced by the push model) and clutter prior are C_p as the prior probability for searching. C_p is generated from the depth heightmap be detecting varying heights. Next, the pushing failure likelihood F_p, constituting the past failing pushing experiences in a multi-model Gaussian likelihood function. Now we can formulate the exploration policy:

$$\pi_e \, : \, arg_a Max \, F_p \circ (C_p \, \circ \, A_p)$$

Where $\circ$ is entry-wise product.

The Coordination policy for pushing and grasping is donated as π_c. The binary classifier takes three elements as input. The domain knowledge, the maximum push value q_p, and the maximum grasp value q_g. The domain knowledge is the target occupancy ratio. This knowledge provides direct information to the classifier to make action decisions. With this setup, the approach learns target-oriented motion critic, which is used to explore and coordinate. The testing seems more focused on primitive objects, which reached a task success rate of 85% [YLC20].

A substantially different approach by Hung et al. [Hua+21b] is by employing a visual foresight tree to rearrange clutter surrounding the target object so that it would be grasped easily. The core component of the proposed method is the use of the Deep Interaction Push Network (DIPN) [Hua+21a] to predict future images resulting from certain pushing actions. And for the grasping action Grasp Network (GN)(similar architecture to DenseNet-121) [Hua+21a] predicts the grasp probability of the target object in the synthesized predicted images. The work is unique as it used a model-based approach learning solution.

The entire pipeline consists of the input RGP-Depth image, first fed to the grasp model. If it exceeds a simple threshold value, the goal object is grasped. If not, then it's fed to Monte Carlo Tree Search (MCTS). MCTS is a heuristic approach for decision-making. Its popular use is board games, ex. AlphaGo [1]. The synthesized images are used as states (nodes) of the MCTS, used for training. During inference, the MCTS would give a sequence of push actions to retrieve the goal object. The testing results in the simulation test are claimed to reach 100% grasp success on 10 test cases. Even though the results seem promising in real life, it isn't easy to generalize the architecture as it depends on generating the synthetic state resulting from the push actions.

Kalashinkov et al. [Kal+18] work learn vision-based dynamic manipulation skills using scalable deep RL. The implementation makes basic assumptions. The observation comes from the monocular RGB camera located over the shoulder, and actions consist of end-effector Cartesian motion and gripper opening and closing commands. The RL algorithm receives a binary reward for lifting an object successfully. No other reward shaping. Seven robots are set up to collect grasping episodes with autonomous self-supervision. Using this set makes the method feasible to deploy at a large scale, unlike most RL learning tasks in the literature. The primary challenge is to generalize effectively to previously unseen objects, as it requires a diverse set of objects during training.

Overview of the closed-loop decision-based control framework is based on a general formulation of robotic manipulation as MDP. at each time step, the policy observatories the image from the robot's camera and chooses a command to the gripper. The Q-function architecture has two inputs: the RGB image of the scene, which is passed to a typical Convolution Neural Network (CNN). In parallel status and taken action of the robot arm is processed in Fully Connected Neural network (FCN), it

[1]https://jonathan-hui.medium.com/monte-carlo-tree-search-mcts-in-alphago-zero-8a403588276a

composed of values of the gripper rotation, Cartesian vector, and the gripper open or closed. The network's output is concatenated in the middle of the CNN used for the image model. The output of the network is the Q-function. This approach is interesting as it deals with training directly with real-life objects. However, it is an expensive approach, and generalizing such an approach is not cheap compared to an approach that can be generalized using simulation.

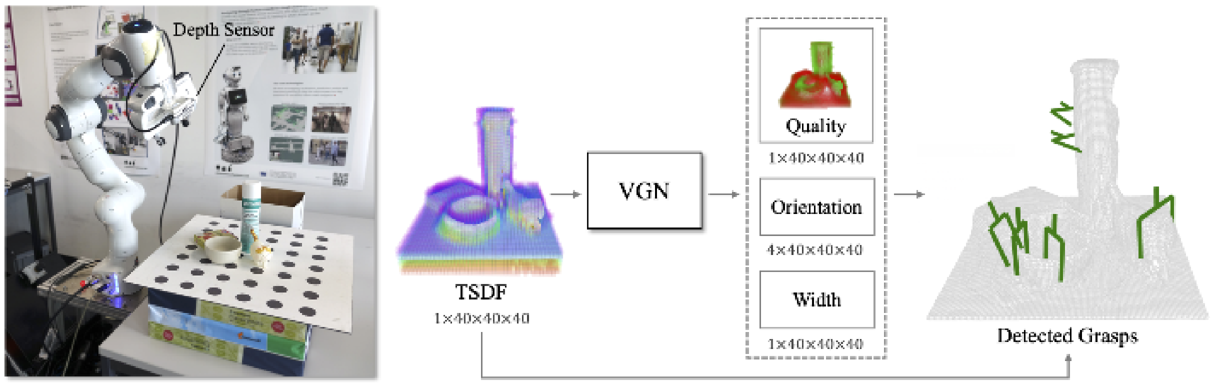


Figure 2.8: Breyer et al. [Bre+21] typical scene and system architecture. Scene converted into TSDF that is the input to the VGN. VGN outputs the grasp quality, orientation and width.

Breyer et al. [Bre+21] Utilizes Truncated Signed Distance Function (TSDF) [WAW14] the data is collected from a wrist-mounted depth camera. It scans the scene from a predefined location. The proposed Volumetric Grasping Network (VGN) takes the input of the TSDF. It produces grasp quality at different locations in the scene, associated with gripper orientation and opening width. Fig. 2.8 represents the approach and the typical scene.

The system leverages the approach by fusing multiple observations into a consistent map while smoothing out sensor noise. The VMN is a Fully Convolutional Network (FCN). It maps the TSDF output to a volume of the same spatial resolution. Each cell contains the predicted quality, orientation, and width of a grasp executed at the center of the voxel. The training is persuasive in physics simulation PyBullet[2]. The network is trained on a generated synthetic dataset of cluttered grasp trials. The system's integration allows inferring of plausible grasps for the entire workspace. The hypothesis of 3D information inclusion allows the model to capture collision between the gripper and its environment, which is a vital trait in these occluded scenes. The TSDF-based approach allowed for transformation from simulation to real without additional adjustments or further training. The real robot experiment on 68 grasps attempts showed 55 successful grasps. Most of the failures result from the lack of friction between the gripper and the manipulated objects. Relating this approach to our problem, assuming the scene is fully observable is not always the case. Such an elegant scene scanning is impossible in cases where the clutter is located on a shelf.

The synergy between pushing and grasping objects in clutter remains sample inefficient for the model to receive a reward only after the goal object is grasped. To mitigate the issue, One can relabel the goal object, also known as Hindsight Experience Replay (HER) [And+17]. It contributes to enriching the replay buffer and, in turn, faster learning. We used Xu et al. [Xu+21] as a starting point for our project. Thus, it will be explained in more detail in the next section 3. The modeling synergy of the push-grasping goal-oriented problem as Markov Decision Process (MDP) enriches the

[2]https://pybullet.org/wordpress/

concept of model decision making in some parts of it are random and others under control. For the process of pushing and grasping, there are many reasons that action previously succeeded in execution will not work in current or future attempts due to simulation physics inconsistency. These factors are challenging to all capture if the model requires total dependence on it [MHK22].

2.5 Conclusion on Paper Selection

Looking back at the first research question 1.3. We are seeking methods that are suitable for a service robot. Meaning the robot would operate in a human-centering environment, picking an item from the fridge or a shelf holding unconstructed clutter. The aim is not to take all the objects from the shelf when we are attempting to fetch an object but rather to use pre-manipulations (e.g., pushing) as efficiently as possible. When the target is invisible, it is favorable that the robot arm moves in areas where the goal object most likely would be located. Some applications limit their work to relatively same height objects as it alleviates the problem of collision checking. When the scene is so cluttered in a bin that almost only a few movements are permitted (In some cases, a sequence of movements is required to reach the goal object). Not all camera locations are realistic. The camera is usually bounded to only one location, where based on it, a plan is made event with the workspace is not fully observable.

Looking at the project period, one of the highest priorities from the criteria table 2.1 is source code availability. It is useful to have a starting point for the projects to reproduce and validate the claimed results of the paper. Most of the promising work after weighing the key problems from table 2.1 is by Xu et al. [Xu+21]. It handles a target-oriented approach, able to manipulate clutter even when objects are stacked.

Xu et al. [Xu+21] work is limited to only push as pre-manipulation action, which is not legitimate for all objects (e.g., bottles, if they are pushed, will fall on the scene and cause chaos). Furthermore, the open loop pushing action assumes the consistency of the gripper, object, and surface consistency. In reality, this is not the case. The variations in the object's heights are relatively small. It's made to avoid the problem of the robot gripper colliding with objects in the scene. Even though the workspace is limited, there are no walls around the scene of the work, which permits the robot to push obstacles outside of the scene (i.e., making use of space that might be initially not possible inside a shelf.). The novel objects that are tested are quite limited to around 8 different objects, and during training and testing, they are lay ed flat. Nevertheless, these problems can be solved by alternating the clearing strategy, which will be discussed further in this thesis.

3 Methods

This project mainly consists of three parts. The first part we called it push to grasp, the goal object needs to be cleared from occlusion by utilizing a push action(s) to be able to grasp the goal object, see section 3.3. The second part is about the use of the same principle; pre-trained models but on a different robot, see section 3.5. The third part is about developing similar system to grasp occlusion(s) in place of pushing them to clear the goal object, see section 3.6. The motivation of each part will be discussed in the related section.

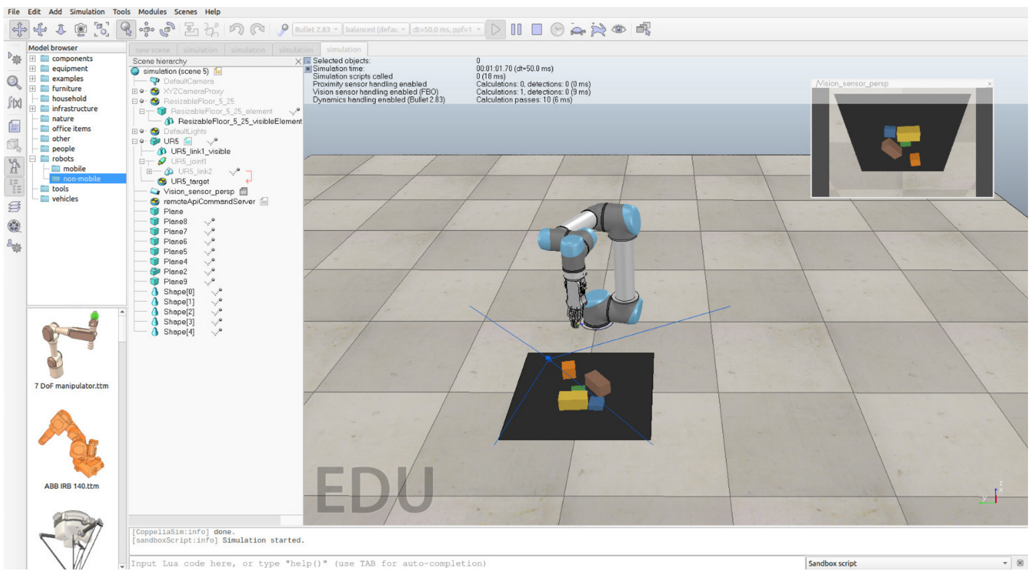
3.1 Simulation Environment

Simulation environments are useful tools to model and test systems efficiently and cheaply. Training and testing systems in real-life directly might be impossible. Maintaining the robot functioning during long hours of training would need the presence of an operator frequently to control if the operation is functioning smoothly. That is not feasible, which brings the prehensible tool of using simulation environments instead. There is a number of simulation environments, the most popularly used in robotics are V-REP/CoppeliaSim, Gazebo, MuJoCo, and Webots [Kör+21]. In this project thesis, I used both CoppeliaSim and Gazebo. CoppeliaSim Graphical User Interface (GUI) is depicted in Fig. 3.1. In the bottom left panel, there is the possibility to choose from several different robots. The panel next to it is the scene hierarchy. It displays the scene's content, scene objects, and the corresponding setup. The hierarchy represents the tree-like structure of certain objects. We can take the UR5 robot arm as an example here. It composes of different joint links that are attached coherently to make the robot arm complete. The different joint connections are where the rotation occurs, referred to as a degree of freedom. The UR5 robot arm is used in this project for the simulation. It is 6 degrees of freedom (DoF) robot arm. TIAGo service robot from Pal robotics is used in real-life. To load the robot arm in the scene, one needs a model description of the object. Robots models are saved in **"*.ttm"** files [22a].

The crucial element in simulation environments is the physics engines (Dynamics). It provides an approximation simulation of the physical properties that objects would experience in real-life, such as rigid body dynamics, collision, as well as soft body dynamics. It simulates forces of all different types. CoppeliaSim supports mainly four different physics engines. Bullet Physics, Open Dynamics Engine (ODE), Vortex Studio, and Newton Dynamics. In the push to grasp ODE is used, section 3.3. Its open source, and it simulates components such as rigid body and dynamics well. Furthermore, in grasp to grasp section 3.6 we used Newton dynamics, from experimenting it demonstrated better consistency with collision detection [22a].

Another important aspect that needs to be discussed is the connection to these simulation environments. One might have a project that needs to interact with the simulation environments. There are different methods to connect to CoppeliaSim. It is a highly customized simulation environment. The simulator can be tailored as desired. This is possible through an application programming interface (API). More thanthan six different programming languages and approaches are supported, each with its pros and cons. For an elaborate explanation, I refer the reader to [22a]. In push to grasp project used 3.3 similar to [Xu+21]. It allows customizing the simulator via a remote API client application. Allowing external applications (e.g., located on a robot, another coding project) to connect to CoppeliaSim conveniently, namely using WebSockets [22a].

Similar to the simulation environment CoppeliaSim, there is Gazebo, see Fig. 3.2. The TIAGo robot is deployed in the environment with a table in front of it. There is a set of objects present on an evaluation table. The physics properties such as color friction of all the objects in the scene are



```
Selected objects:                          0
Simulation time:                           00:01:01.70 (dt=50.0 ms)
Simulation scripts called                  0 (18 ms)
Proximity sensor handling enabled          Calculations: 0, detections: 0 (0 ms)
Vision sensor handling enabled (FBO)       Calculations: 1, detections: 0 (9 ms)
Dynamics handling enabled (Bullet 2.83)    Calculation passes: 10 (6 ms)
```

```
[CoppeliaSim:info] done.
[sandboxScript:info] Simulation started.
```

Figure 3.1: CoppeliaSim simulation environment interface, with UR5 robot arm deployed and objects in the scene.

described in a ***.rdf** file. It is crucial to have as realistic values as possible to alleviate the problem of the sim-to-real gap. Gazebo works with a number of physics engines, ODE, Simbody, Bullet, and DART. The latter is used in this project 3.5.

ROS is a Robot Operating System that holds a set of software libraries and tools assisting in building a robot. As ROS serves as the robot interface, combining it with the 3D simulator Gazebo, we establish a powerful robot simulator [22b].

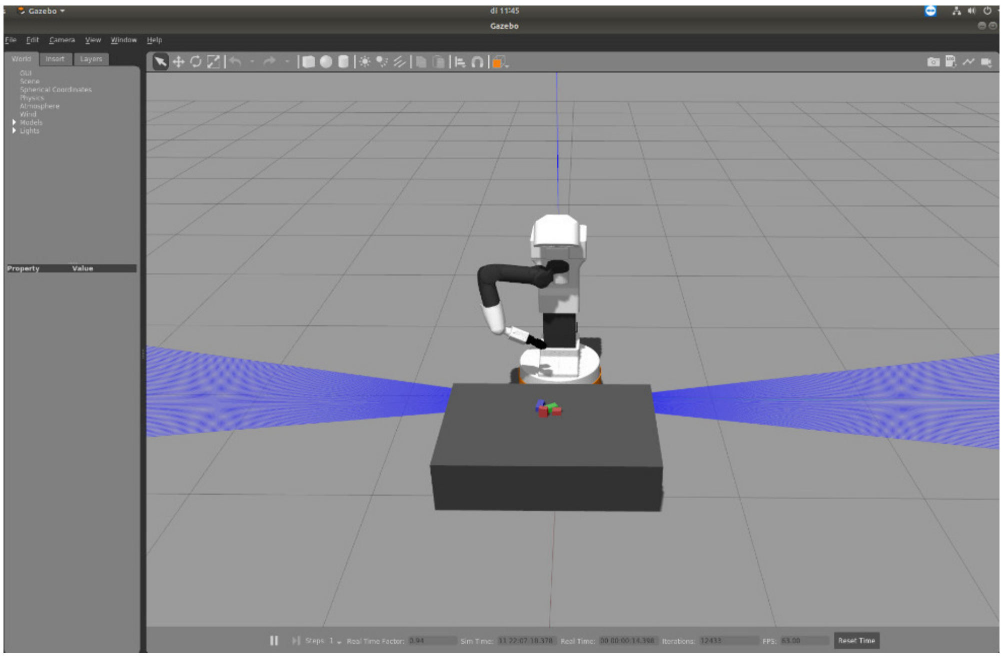


Figure 3.2: Gazebo simulation environment interface with TIAGo robot deployed and objects in the scene.

3.2 Deep Learning RL Models

The Vanilla Q-Learning algorithms consist of three parts; where first, we initialize the Q-table (normalized q-values), and the agent chooses an action from the table and executes it. The environments would respond with the changing state observed by the agent, which is used to update the first initialized Q-table using the recursive Bellman Equation. And gradually, the agents learn by repeating the process, by exploring the environment with learned actions.

$$Q(s_t, a_t) = (1-\alpha)Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a Q(S_{t+1}, a)) \quad (1)$$

- S: State/observation
- A: Action taken by the agent
- R: Reward resulted from action taken
- t: Time step
- α: Learning rate
- λ: discount factor, choice between short and long term reward [SW10].

Take the example of the game frozen lake, where agents have to navigate from one part of the map (represented in tiles) to another part of the map. The map has holes. Falling in one of these terminates the game. The map does not change between rounds. The agents would explore and exploit heuristically, reaching a sub-optimal solution to reach the route from start to finish without falling into one of the holes. The norm of such a game could consist of 9 tiles, and the agent has only four actions, going in one of the four directions (up, down, left, right). During training, the agent keeps updating the q-table. Such a game can be trained relatively in a short time (1 minute) using conventional Hardware nowadays.

However, continuous action states are normally discretized when we have a more complex environment. A basic approach of the Vanilla Q-learning reaches the limits of its capabilities in terms of time for the model to achieve a sub-optimal solution. The learning policy increases quadratically with the increase of either the action or state spaces.

Deep Networks are powerful tools for estimating a function contrary to Vanilla Q-Learning, where Q-table maps state-action to Q-value. Deep Q-Learning maps state to action and Q-value pairs. The network that estimates the action to take is called the policy network since its objective is to find the optimal policy by finding the optimal Q-function.

Interesting approach by using Generative Adversarial Network (GAN). Two neural networks contest against each other in a zero-sum game. Whichever agent wins results in a loss to the other agent. Using this approach, we can introduce two models, each dedicated to a certain task. One that would push occluded objects, another attempts to grasp the goal object. For the remaining of this section, I will briefly explain GANs in section 3.3, utilizing GANS for pushing and grasping will be discussed.

First, we introduce GAN and how they are trained. GAN is constructed by combining a generator and a discriminator; see Fig. 3.3. The generator attempts to generate close samples to real data samples, its input a point from the latent space, the latent space has no particular representation, typically 100-dimensional hypersphere, each variable drawn from a Gaussian distribution ($\mu = 0$ $sd = 1$). The generator optimally learns to map the latent space with specific output images, which is different at each training step. The discriminator focuses on distinguishing whether the inputs are generated or real data. The training is via the adversarial procedure, optimizing the discriminator and the generator alternatively. Eventually, the GAN gets balance, meaning the generator is able to produce samples that are almost similar to real data samples distribution. At the same time, the discriminator is able to achieve the highest classification results. The training labels are easily defined. We know in advance what has been fed, real data, or generated data, making it a self-supervised approach, and thus after the final classification, one can backpropagate and optimizes the networks with techniques such as gradient descent [Fen+20].

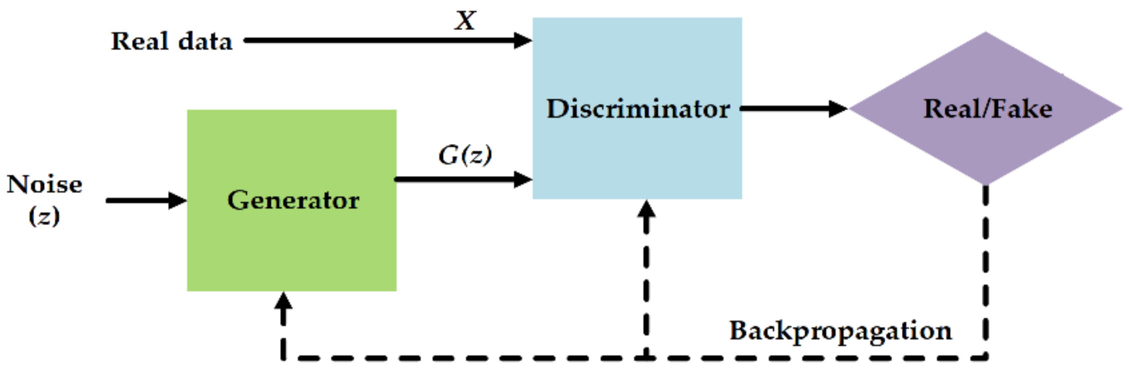


Figure 3.3: The architecture of the generative adversarial network (GAN), the generator samples form a latent space, the discriminator from samples from both the generator and a set of real data, which infers either the data is real or fake [Fen+20].

The latent space structure can be query used by the generator. One can structure the latent space, e.g., all 1s or all 0s, and facilitate it as an input query to be able to generate a specific image. A popular example illustrated in Fig. 3.4, points in the latent space is stored to be used in vector arithmetic, creating new points. An example of this man with glasses - a man without glasses + a woman without glasses gives a woman with glasses [RMC15].

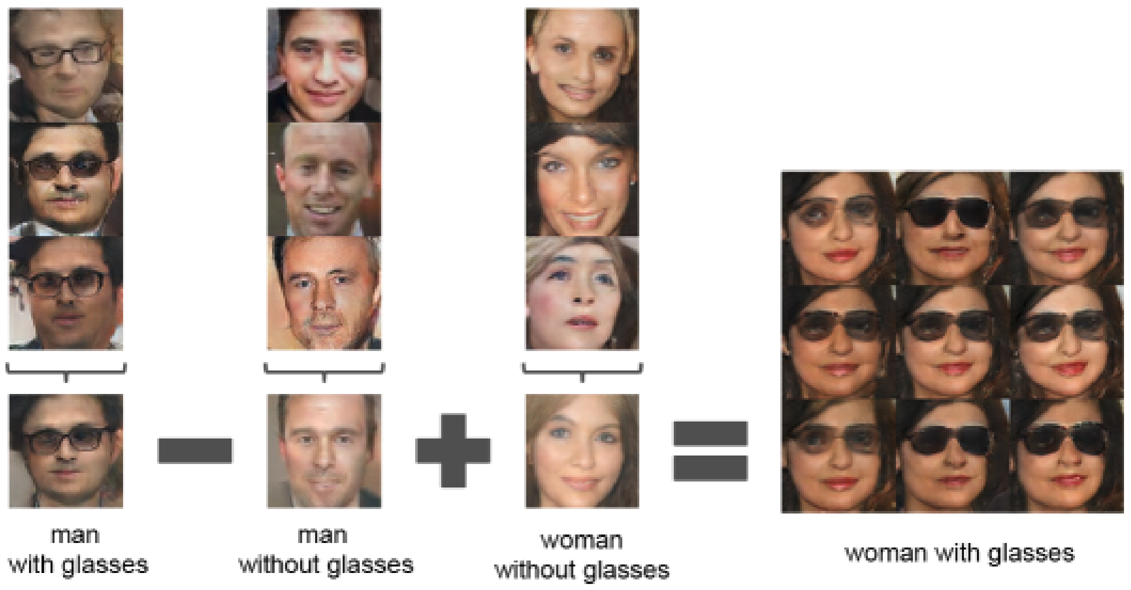


Figure 3.4: Vector arithmetic for visual concepts, latent space arithmetic on generated faces and semantic understand of images [RMC15].

3.3 Push to Grasp

As shown in Fig. 3.5, the system produces two actions, push or grasp, depending on what is feasible in the environment. Grasping the goal object is executed when the probability of grasping the goal object exceeds the threshold. Otherwise, the goal object or its surrounding objects are pushed approximately 1.3cm away from the current position to increase the probability of grasping the goal object successfully. The grasp/push points and direction are indicated on the image coordinate (output of the networks) and then transformed to the robot's reference frame to be executed.

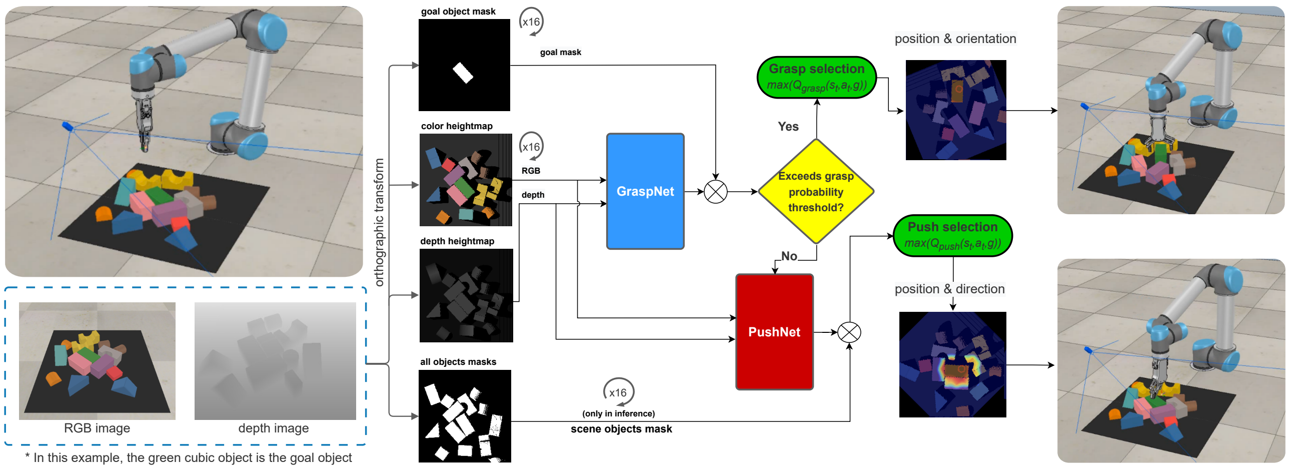


Figure 3.5: **Overview** of the entire system. RGB-D camera and UR5 robot arm in CoppeliaSim simulation environment. The camera sensory data undergoes orthographic transformation, and the goal and all objects mask are generated. The color heightmap, and depth heightmap are fed into the Grasp net and the push net, all of which experience 16 rotation steps, covering 360 degrees rotation rotating clockwise and starting at 0 degrees. The models produced pixel-wise Q values are considered whether to push or grasp and eventually execute the action. In the testing phase, the all objects mask is used.

We model the synergy between push-grasp and goal-oriented grasp as a goal-oriented MDP problem. The policy is represented as $\pi(s_t|g)$, the reward as $R(s_t, a_t, g)$, and lastly the Q-value function as $Q(s_t, a_t, g)$. Pushing and grasping in an occluded environment is formulated in a reinforcement learning setting. The camera data of the working scene is transformed into orthographic images and rotated 16 times to cover 360 degrees rotation. The rotations are approximately 22.5 degrees increase for each step. The rotation allows the models to express and learn to grasp orientation and push direction. The model maximum Q-value is selected with respect to the rotation, and then action is executed. If the grasp threshold exceeds a certain Q-value (we set it to 1.8), a grasp action to the goal object is executed. Else, push of the goal object or any other objects is performed [Xu+21].

The pushing is performed in a manner that increases the probability of successfully grasping the goal object in the future. Similar to what we would have in a Generative Adversarial Network (GAN), the grasp net ϕ_g serves as a discriminator, and the push net ϕ_p is considered a generator. It manipulates the scene to utilize an increase in the grasp Q-value, particularly the Q_g (see Fig. 3.5). The discrete mask of the goal object is used both as input in the model and as output to remove irrelevant pixel Q-values.

Unlike to [Xu+21], the mask of all objects is not clear when used. We decided during training not to restrict the model to have the Q-values only on the relevant objects. It enhances the exploration space of the model due to considering all Q-values and executing at the max value. Nevertheless, we used the objects mask at the output during inference.

3.3.1 Models hierarchical framework

The hierarchical framework consists of two networks (grasp ϕ_g and push ϕ_p) sharing the same structure similar to [Xu+21]. However, the training is different. We use three parallel 121-layer DenseNets [Hua+17], pre-trained on ImageNet [Den+09], to extract visual features. The first DenseNet tower takes the RGB data of the scene, the second one takes the normalized depth image, and the last one takes the goal object's goal mask (discrete masking). The output of these networks is then concatenated to form a single feature vector for the given observation. The obtained representation is fed into a fully convolutions network [LSD15], which contains two Kernels of 1×1 and Rectified Linear Unit (ReLU) activation function, and batch normalization [IS15]. Finally, it is bilinearly upsampled to generate a pixel-wise prediction (i.e., the output image will be the same size as the input cropped image).

3.3.2 Software Architecture Push to Grasp UR5

The software diagram of the push-to-grasp system is depicted in Fig. 3.6. There are two threads, one of which runs, receives the models' outputs, and accordingly performs actions. The other thread is related to handling the models' required inputs (fetching camera data) and feeding it to the models, also the required masking. The robot module communicates with the simulation environment CoppeliaSim using the legacy remote API. The legacy remote API is a lightweight service allowing for bidirectional data streaming. The output of the terminal is used by the evaluate module by scrapping the relevant information and calculating the evaluation metrics mentioned in section 4. Source code available at my GitHub repository[3].

[3]https://github.com/Kamalnl92/Self-Supervised-Learning-for-pushing-and-grasping

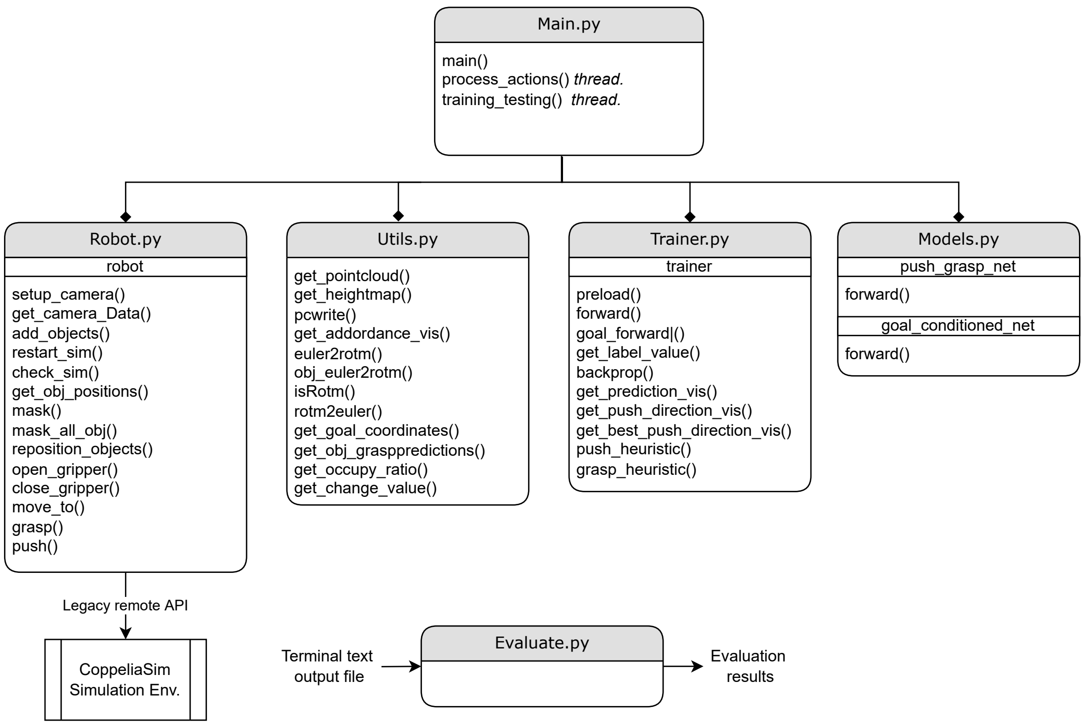


Figure 3.6: Software architecture is developed for the push-to-grasp system using the simulation environment CoppeliaSim and the UR5 robot arm.

3.4 Push to Grasp Training

We train the model in three sequential stages: the first stage is training the goal-condition grasping, the second stage is dedicated to goal-conditioned pushing, and lastly, alternating the training between the goal-condition grasp and push. The grasp reward R_g and push reward R_p, is formulated as follows:

$$R_g = \begin{cases} 1, & \text{if grasp success} \\ 0, & \text{if not} \end{cases} \tag{2}$$

$$R_p = \begin{cases} 0.5, & Q_g^{improved} > 0.1 \text{ \& scene change} \\ -0.5, & \text{no scene change} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $Q_g^{improved}$ is calculated as

$$Q_g^{improved} = Q_g^{post-push} - Q_g^{pre-push} \tag{4}$$

Furthermore, the amount of changes in the scene is calculated from the depth map surrounding the goal object. Inspired by the Bellman equation of reinforcement learning, we can formulate the state action function with respect to the goal object as $\pi(s|a,g)$. We use the epsilon-greedy action selection strategy $\varepsilon(\pi(s|a,g))$. It gives the agent the privilege to both explore and exploit options [GSL20]:

$$\nu(S_t, g) = E[R_{[t+1]} + \rho\nu(S_{[t+1]})|S_t = S, g] \tag{5}$$

Where ρ represents a discount function, an overview of the training process and the grasp success rate for the different stages of training are depicted in Fig. 3.7. In the following sub-sections, more details about the training are discussed.

3.4.1 Goal-conditioned grasping

We randomly generate sparse scenes to train the model by putting five objects in the workspace (e.g., see Fig. 3.8). In this stage, we consider the execution of each grasp as an episode and train the model in two phases. The first phase, i.e., *grasp agnostic*, is done by relabeling the goal object to increase the sample efficiency. In the second phase, i.e., *grasp explore* as the model learned to grasp objects, we assume it learned sufficient estimation of the orientations and grasp pose; hence, we omit the relabeling of the goal-object. In both phases, the ε-greedy strategy is used for balancing the trade-off between exploring and exploiting [DCR19]. Unlike what has been used by [Xu+21] which measures the Q-values to determine when to stop training, we measure the grasp success rate directly. Fig. 3.7 shows that the training could be stopped in both phases after reaching 1400 grasp epochs since the model grasp success is mostly stable. Even though the grasp success does not increase much in the grasp explore phase, the task is more challenging as only the goal object grasp is considered a success, and the environment might not have a feasible grasp due to yet untrained push model.

3.4.2 Goal-conditioned pushing

In this round of training, we fix the grasp model and only train the push model based on adversarial training, as mentioned earlier. Each episode has been defined as up to five pushes and a grasp in the end. In other words, if the Q-value of the grasp for the goal object exceeds a given threshed, the robot immediately executes the grasp action, and the episode is terminated. After a short period of 120 training grasp epochs, we observed that the goal-oriented grasp success rate increases dramatically as the push model manipulates the scene. Consequently, it increases the probability of the goal object's grasp (see Fig. 3.7 *third-row*). A positive reward is given to the push model only if it increases the future probability of grasping the goal object.

3.4.3 Goal-conditioned alternating

In the previous grasping stage, we trained the grasp model in sparse environments (i.e., around five objects per scene) to reduce the effect of occlusion. In addition, the model was trained when the push was not trained yet, which leads to the distribution mismatch problem. The grasp net is further trained after the push model is trained to overcome this issue. Models are alternately trained in an environment with ten objects present in the scene to improve the synergy between the grasp and push models.

An illustration of the training process is depicted in Fig. 3.7. Since the grasping success result is a binary result (i.e., 1 success and 0 for a failure), we use exponential smoothing with a factor of 0.9 to show trends in the data, i.e., learning progress of the grasp success. Also, the averaging mean is used to show the data variation. For the goal-condition grasping and goal-conditioned pushing, we use the mean rolling window of size 50, and the rest we set to a mean of 7.

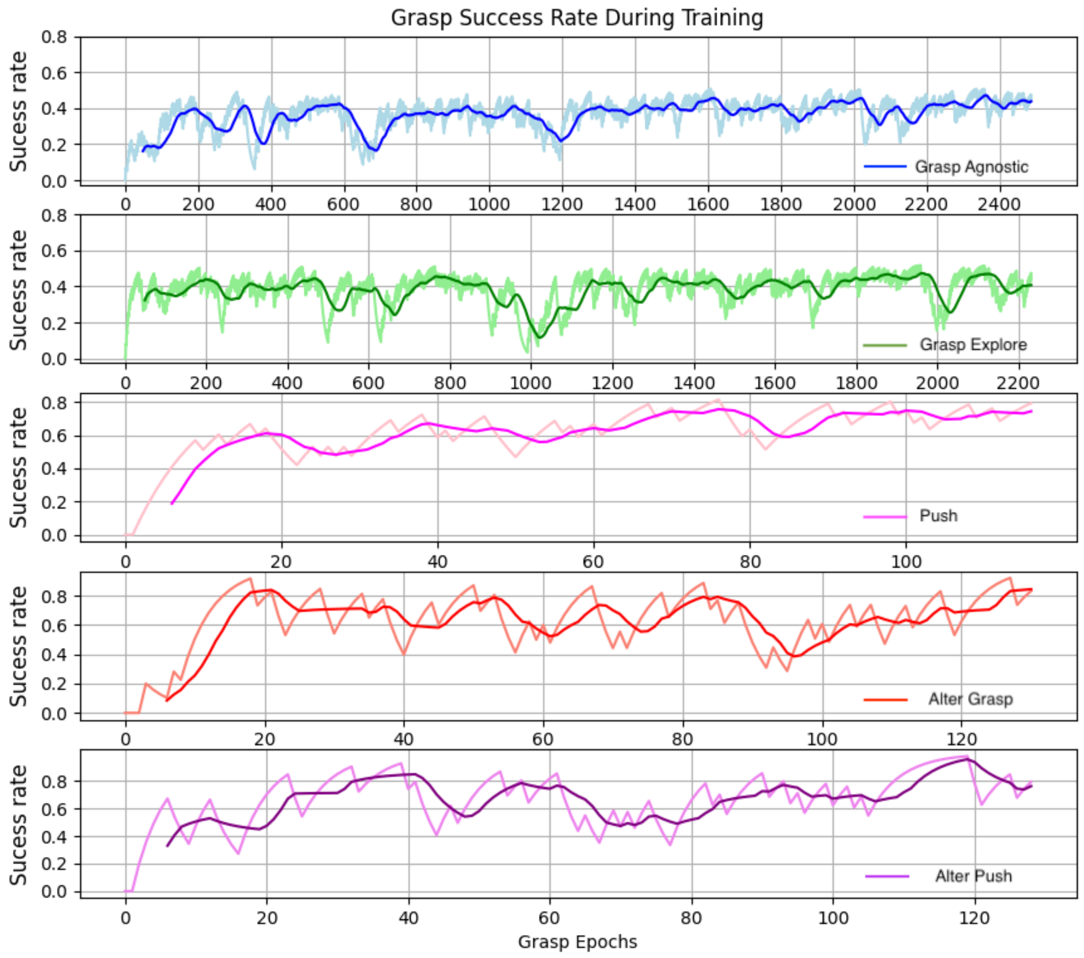


Figure 3.7: Grasp success rate versus the number of grasp epochs. The top two are related to the grasping network ϕ_g training, the middle one for the push network ϕ_p training. The last two are when the training is alternated between the grasp net and push net.

3.4.4 Reproducibility Problems

This section will focus on the problems faced when attempting to reproduce the project by [Xu+21]. There were many problems, and it took around one-third of the project period. The problems will be discussed briefly.

The models are trained on High-Performance Computing (HPC). The node used has NVIDIA V100 Graphics Processing Unit (GPU). Which has a memory of 32GB. GPUs are commonly used for model training as it allows for simultaneous calculation and results in speeding up the training time significantly[4].

The HPC runs with a Linux kernel to operate on it. One needs an Operating System to run the work. The typical approach is to build an image container. In this project, we used Ubuntu18.04, similarly

[4]https://wiki.hpc.rug.nl/peregrine/introduction/what_is_a_cluster

used by Xu et al. [Xu+21]. The container is built using Singularity[5] containerization method.

The first issue was the incorrect command given by Xu et al. [Xu+21] in the repository *Readme* file. The first part of the training was about to have to grasp a goal agnostically object. This would allow the model to learn orientations of picking objects rather than strictly focusing on the goal object. However, the flag command parser is turned on, which causes a size mismatch in the tensors in training in the first stage (goal agnostic) and the second stage goal oriented.

The second issue was using CoppeliaSim in headless mode. Some bugs appear in the headless mode in the simulation environment and not necessarily in the normal head mode. Examples of such problems are getting the camera data, depth information, and the number of objects in the working scene. It seems that these variables at some iteration point can return null values, which results in halting the training process, so the attempts to overcome such issue normally would start with figuring out the cause of the issue, then updating the simulation environment, and lastly one could add security measures in the project source code on how to handle such unexpected events (i.e., requesting the data again from the simulation server or restarting the simulation environment and continue training).

The third issue, I will not go into details about all of them, but in general, it concerns using tools such as the CUDA Tool kit[6] is an API that gives direct access to the GPU's virtual infrastructure, allowing direct access to the GPU's virtual architecture. Certain GPU are not supported or do not function properly, and sudden halting of the training occurs. Such a case demonstrates the challenge of hardware and software compatibility.

Last and not least, the trained model seemed not to perform as described by the authors' Xu. et al. [Xu+21], investigating the entire pipeline from the input information to the model output and actions execution it seemed the discrete masking proposed by [Xu+21] is not useful as it consistently results in invalid actions (see Fig. 3.8 *top-row*). Moreover, their approach requires different masking during the training and testing phases. This impacts the scalability of the system. More advanced instance segmentation methods (e.g., [Xia+20]) can be used to manage this problem. However, since the perception is out of the scope of the thesis, we simplify the problem by using RGB-colors segmentation as we know the color of the goal object in advance (i.e., green color). The discrete working mask after applying it to the pixel Q-values is illustrated in Fig. 3.8 (*lower-row*).

[5]https://sylabs.io/guides/3.5/user-guide/introduction.html

[6]https://developer.nvidia.com/cuda-toolkit

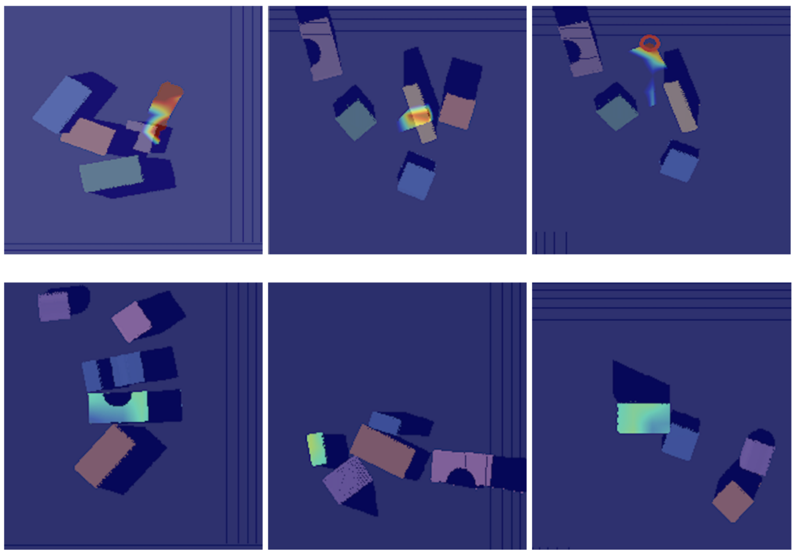


Figure 3.8: Objects pixels Q-value after applying masking: (*top-row*) masking problems in approach proposed by [Xu+21]; (*lower-row*) Accurate masking using our method.

3.5 TIAGo Robot

TIAGo robot is a service robot[7]. The motivation to use the TIAGo robot is twofold. First, to validate if the method, which is trained on a different robot, simulation environment, and used manipulation objects, is able to generalize well. The second is to have the opportunity to test it on a real-life robot. TIAGo combines perception, navigation, and manipulation. Navigation is not part of this thesis work, so we assume the robot is located in a predefined location. Its perception (RGB-D camera) is used for the sensory input information. The robot arm is a 7DoF arm. It has an antipodal gripper (PAL Gripper), as depicted in Fig. 3.9. The lifting torso is not used.

[7]https://pal-robotics.com/robots/tiago/

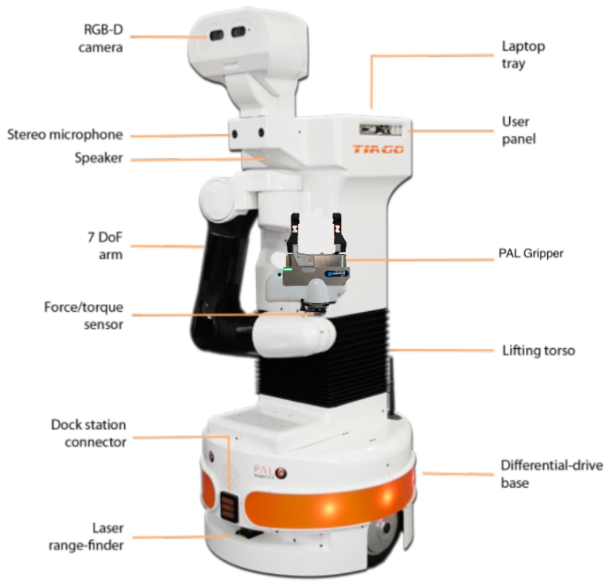


Figure 3.9: TIAGo robot from PAL Robotics, the figure describes the elements of the TIAGo robot hardware.

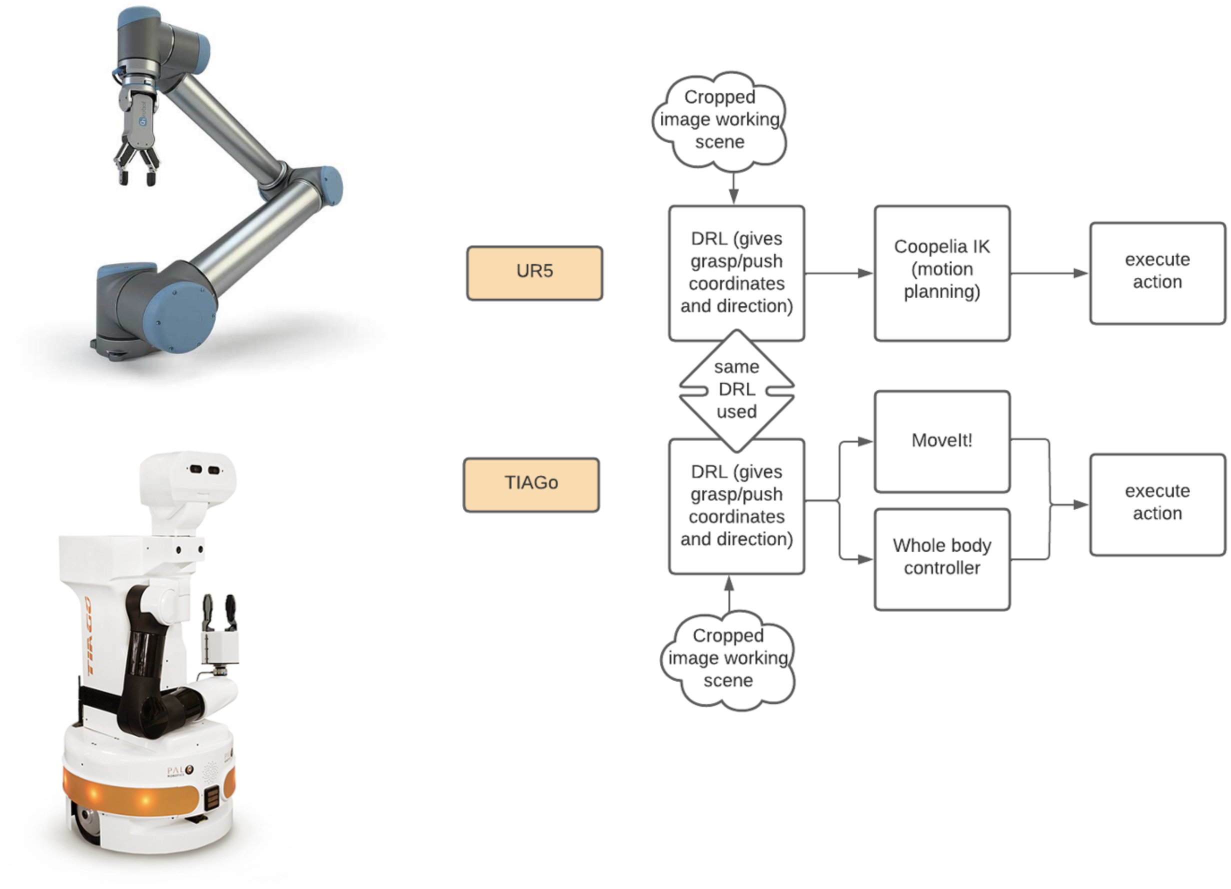

Cropped image working scene
UR5
DRL (gives grasp/push coordinates and direction)
Coopelia IK (motion planning)
execute action
same DRL used
TIAGo
DRL (gives grasp/push coordinates and direction)
MoveIt!
Whole body controller
execute action
Cropped image working scene

Figure 3.10: Converting the work push to grasp from the UR5 robot arm to the TIAGo robot.

Fig. 3.10 illustrates the transition from the UR5 robot arm and CoppeliaSim to TIAGo using Gazebo. The upper row blocks (UR5) show an abstract overview of the system. The cropped image is acquired from the sensory information, which is the input of the RL model. The RL model produces grasp coordinates and orientation on the image. It is transformed into the robot frame of reference to be executed. The coordinates are passed to the CoppeliaSim inverse kinematics (IK) motion planning to be executed.

Similar to the UR5, we can follow the same convention on the TIAGo robot. However, the tools are different. Here we use ROS (Robot Operating System) as a framework. The acquiring of the data via the camera is run in a separate script and saved the received data. Then the main project can use the acquired data for further processing. There are two motion planning methods, as depicted in Fig. 3.10. The first one is the Whole body controller (WBC), which is a stack of tasks **??**. It includes a hierarchical quadratic solver, running at 100Hz. This gives it the privilege of giving priorities to different tasks. An example of such task priorities is avoiding joint limits. Then self-collision avoidance. These are the highest priorities. Then tasks for moving the end-effector to any spatial location can be deployed.

Another approach would be using MoveIt! for the motion planning. MoveIt! is an advanced API tool that incorporates libraries for motion planning and manipulation tasks. It has a single unified interface that includes state-of-the-art IK solvers, path planning algorithms, and collision detection[8].

After some tests in Gazebo, it is concluded by visual observation that the motion plans that are produced by moveIt! are more stable, constrained, and predictable. For example, the WBC's motion plan exhibits enormous movements and reconfiguration of the robot arm. Contrary to moveIt!, it is more stable and shows more stable motion plans.

3.5.1 Software Architecture Push to Grasp TIAGo

Fig. 3.11 demonstrates the change in the software architecture. It illustrates how flexible the method is to adopt to another robot. Here we used the TIAGo robot. Most of the work was involved in the TRobot (TRobot abbreviation of TIAGo robot) module. One can easily switch between the two robots by mainly choosing the *main* for the UR5 or *Train* for the TIAGo robot. Not all methods in the TRobot are built yet. The push, grasp, and camera ROS nodes are run as separate modules, run within the Trobot functions. The data of the camera is stored in a common directory, which is read by the TRobot module. The source code for this part is confidential. Only the overall architecture is discussed.

[8]https://nu-msr.github.io/me495_site\/lecture18_moveit.html#:~:text=MoveIt!%20is%20ROS's%20most%20advanced,a%20single%2C%20unified%20ROS%20interface

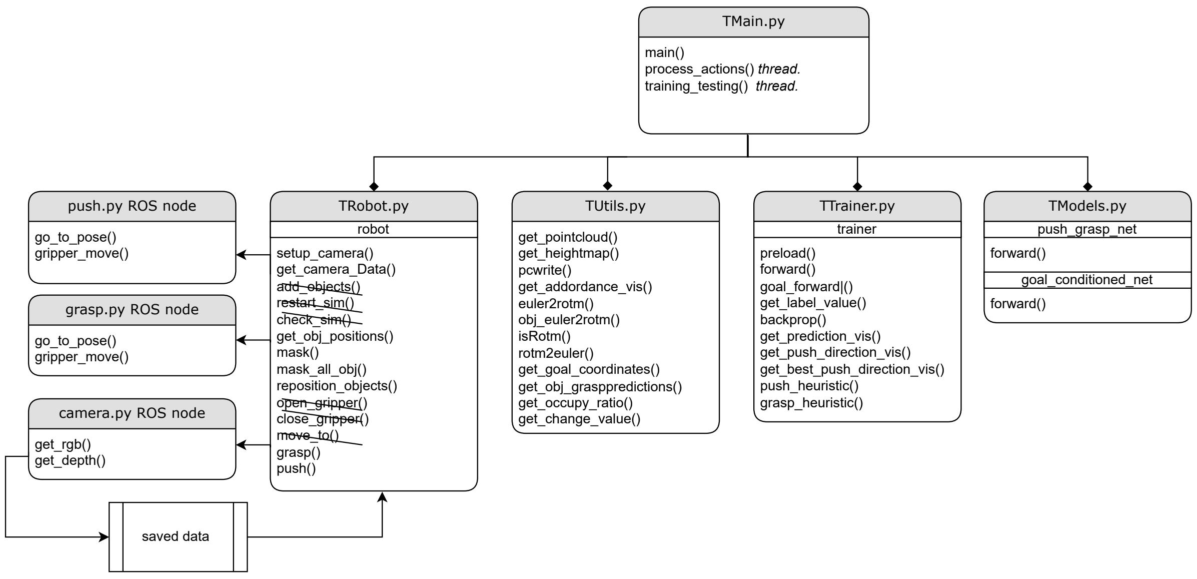


Figure 3.11: Software architecture is developed for the push-to-grasp system using the simulation environment Gazebo and the TIAGo service robot.

Next to the software architecture changes, some parameter changes are required:

- Image goal object masking, the colors, and the goal object green level are different in Gazebo compared to CoppeliaSim, similar to the rest of the objects.
- Camera position and orientation with respect to the working scene. It is part of the calculation to transform the image from a perspective image to an orthographic top-view image. To get these values, we need to use the coordinates frames (known as *tf package* in ROS), and the calculation between the camera and the working scene can be echoed instantly.
- Similar to the above motivation, the camera intrinsic parameters are required, this simply can be done by echoing the */xtion/rgb/camera_info*

Lastly, the camera depth information is not pixel-wise RGB accurate. Not all pixels of the RGB image has depth information. When we use the row depth information, we can see missing depth information, as depicted in Fig. 3.12. This can be solved using bilinear interpolation, which fills in the missing information with an interpolated value with respect to the surrounding depth values. The final result is illustrated in the bottom right in Fig. 3.12.

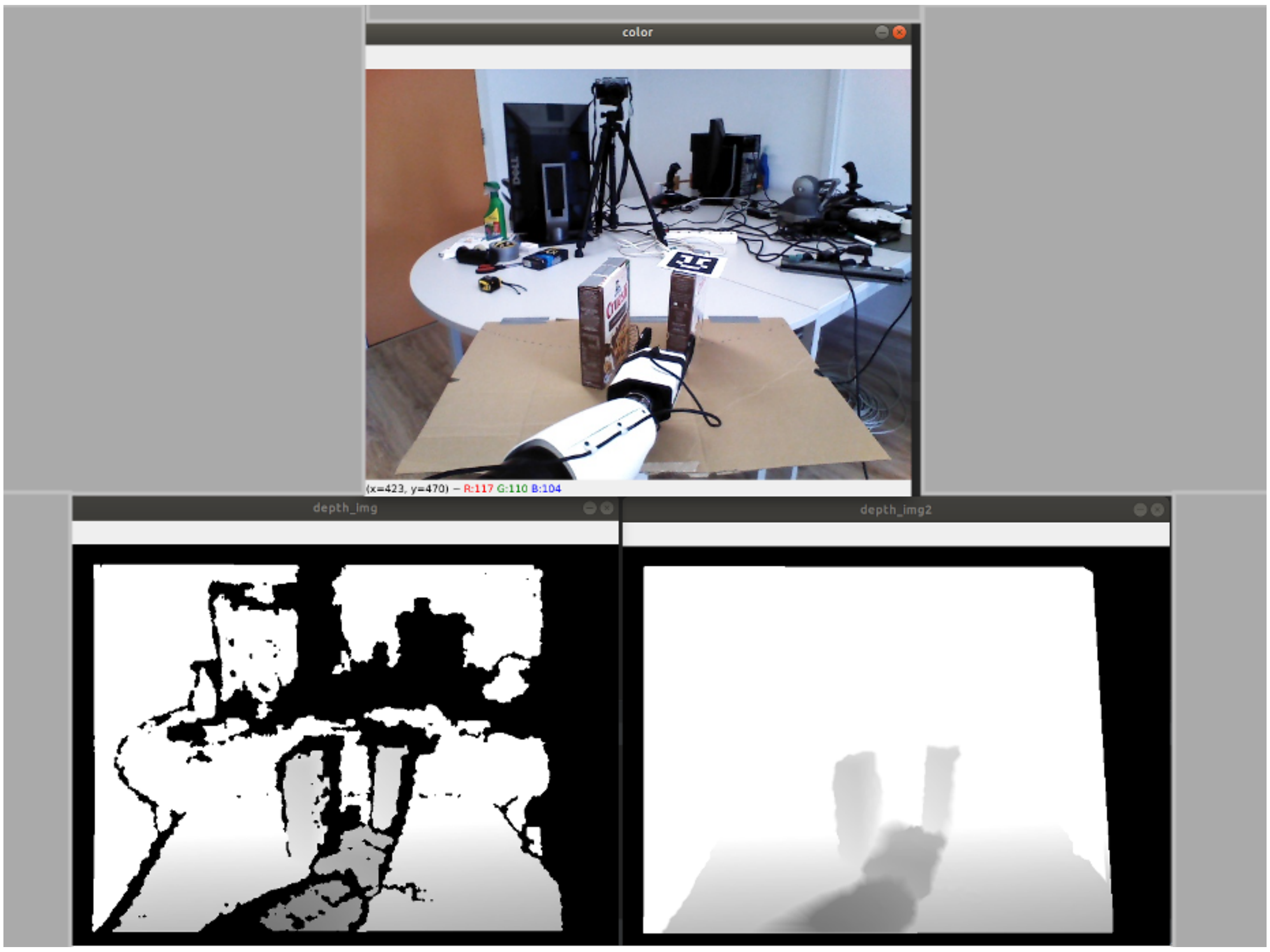


Figure 3.12: Top image shows the RGB image of TIAGo working scene. The lower left shows TIAGo depth image, which suffers from inconsistent depth information (lacking depth information). The right bottom image shows normalized depth information after applying bilinear interpolation.

3.6 Grasp to Grasp

Not all objects are legible to be pushed. All of the previous objects that are presented are primitive objects that are physically permitting push actions. Nevertheless, there is a number of objects that are not legible for a push action. An example of such objects is bottles and cereal boxes. Push actions could cause chaos in the scene. Objects would be toppled down, even ending up in orientation that is not graspable anymore, as it lays flat on the floor of the working scene. With the motivation behind this project aiming to solve the problem of retrieving a goal object from the cluttered environments where we humans live and work in it, it seems sensible to have grasp action to remove occlusion objects to be able to reach a clean grasp to the goal object. An Example of such a scenario is depicted in Fig. 3.13. We have no data to support this argument that grasping actions to remove obstacles would generalize better to the problem. However, subjective opinion seems to us that grasp removing obstacles would be more suitable. Overall, the system would make a sequence of non-goal grasp actions until the point where it permits the grasp action of the goal object itself. One might ask that there are scenarios where the system would not be able to grasp non-goal or goal objects because the objects are so densely packed, leaving no space for the antipodal gripper to access them. In this case, the push is defiantly desirable. Since we are interested in investigating the extent to which the grasp-to-grasp methodology would work in different cases and whether the architecture of push-to-grasp would be able to learn that, we decided to first start with that. In future work, one could include a higher level to leverage the push as well. So the system would consist of a push grasp of a non-goal object and finally grasping the goal object.

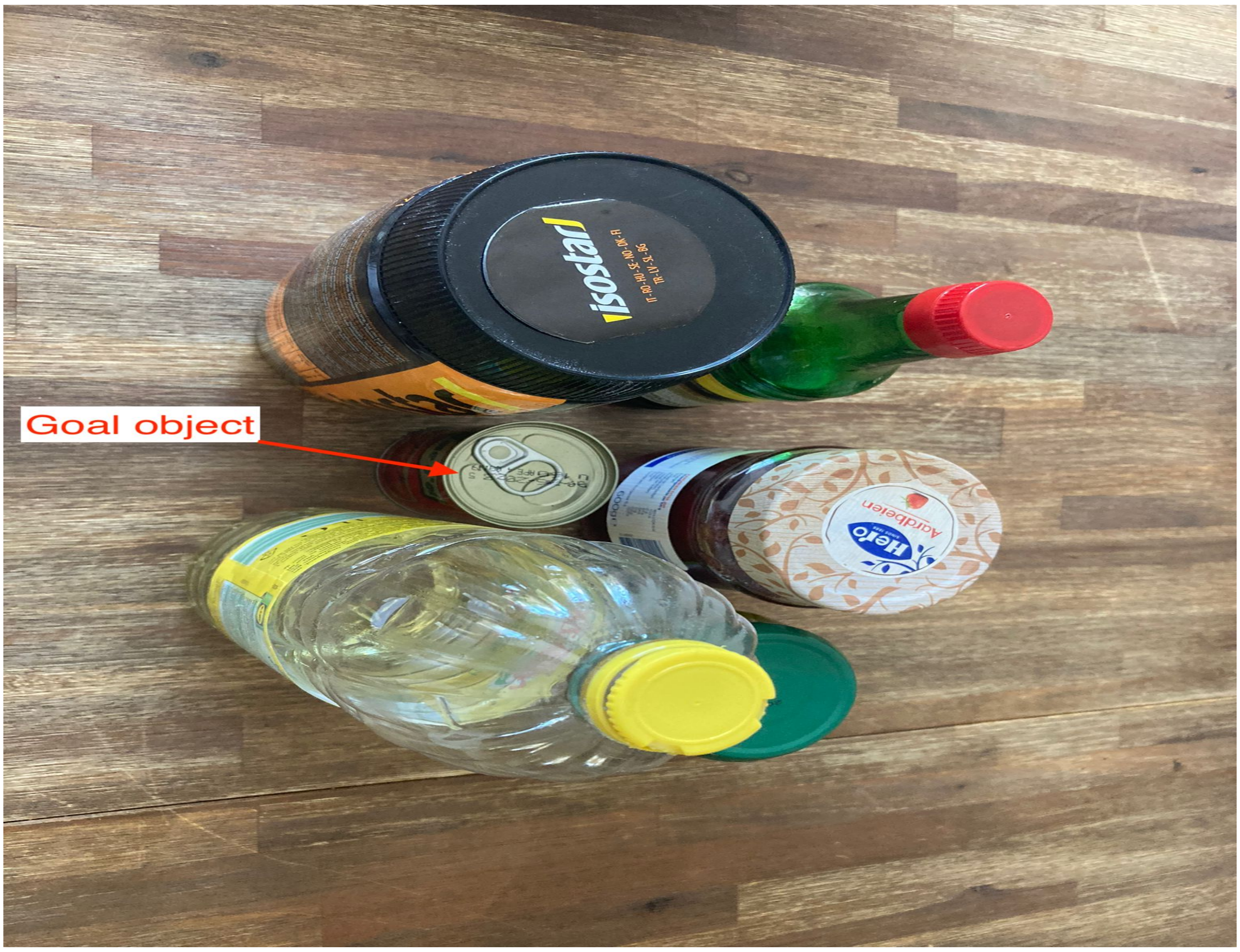


Figure 3.13: A scenario where grasping the goal object directly is impossible. An approach of grasping and removing obstacles is suitable to reach the final goal object.

3.6.1 Grasp to Grasp Methods

This chapter will cover changes in the system architecture to be able to train a model to perform grasp removal of non-goal objects. Like push to grasp, CoppeliaSim and UR5 robot arm are used to train the model. First, the setup of the system is covered. Source code available at my GitHub repository[9].

We aim to avoid chaos in the scene. Having the first push actions to reach the goal object would not work in this case. Force trying to grasp objects would lead to objects being toppled on the working floor and unable to grasp anymore. In our approach, we introduce objects that can be found in a household, with closely related sizes, height length of 17*cm* and a minimum length of 3*cm*—the width and thickness have different variations. An example of such a scenario is depicted in Fig. 3.14, the green object is the goal object. Due to the presence of the adjacent objects, the robot arm would collide

[9]https://github.com/Kamaln192/Grasp2GraspUR5

with other objects. One needs to examine the motion pattern while training. Using a non-responsive robot arm colored green would examine a motion plan if it is collision-free and act accordingly. The orientation needs to be more accurate than the previous approaches (pushing to clear the goal object from obstacles).

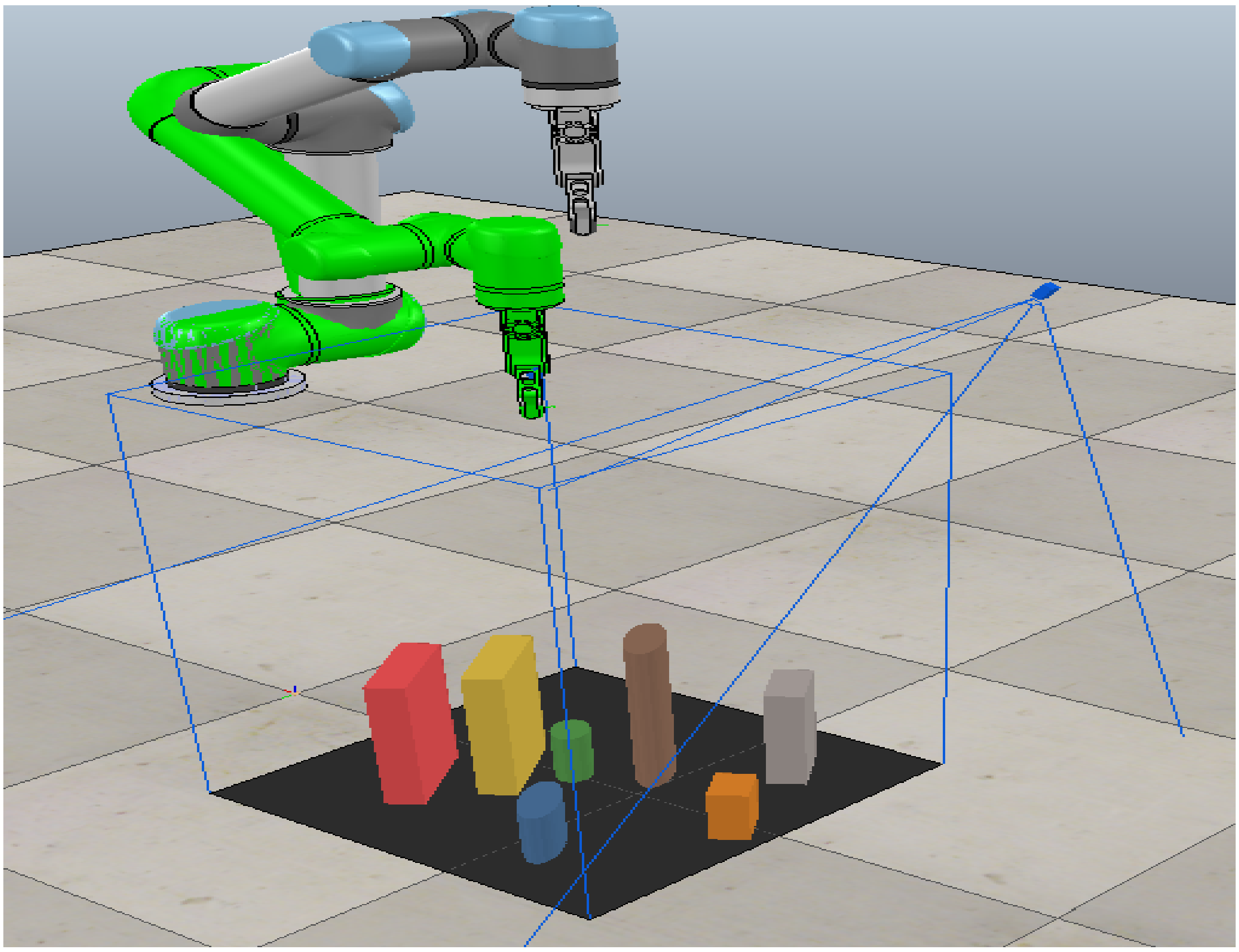


Figure 3.14: Setup in CoppeliaSim simulation environment using the strategy of grasping to remove objects. Two robot arms, one physically responsive, the other robot arm with the color green, is not physically responsive.

In the previous approach, even when the grasping is not nicely estimated but close enough, the object would comply and rotate slightly due to the up-down force of the gripper and slide into the antipodal gripper. The latter will not be possible with the grasp removing non-goal objects. There would be a risk of objects falling on the working floor, which we want to avoid.

The estimation of the grasp accuracy is quite vital. The previous approach relies on only one pixel-wise Q-value. An approach where one would consider more values of the model output would be more robust. Deep neural networks are prone to their input changes. Small changes could cause consequence changes to the output in ways that may not be predictable (e.g., adversarial attacks). There is noise in such simulation environments on purpose because in real-life sensory information also suffers from noise. A more robust approach would be to rely on collective outputs from the model. We do so by accumulating the Q-values to determine the orientation and the push direction. After determining the max heat map, we define the x-y location for the goal grasp. First, the vertical and the horizontal Q-values are projected. See Fig. 3.15. Then the median is determined, and the location is found from the crossing of the median for both the x and y-axis. Grasping a non-goal object would not work. Since the Q-values are accumulated on the axis where the object is present, taking the median, there will result in a grasp of the non-goal object at a no-where location. Thus for grasping a non-goal object, we leave it with picking the max Q-value.

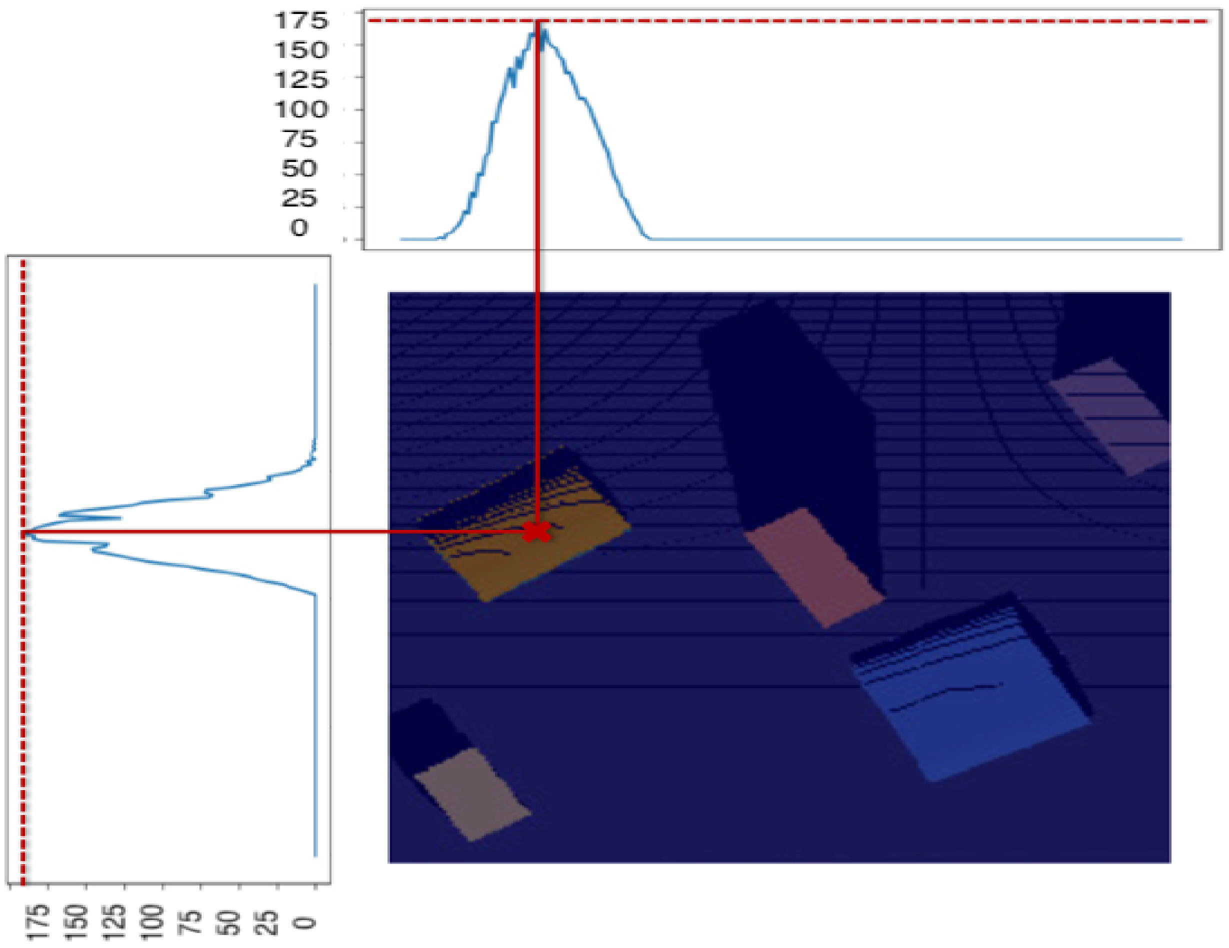


Figure 3.15: Projection of the pixel-wise Q-values over the x-axis and the y-axis. And the median value is determined for the location grasp of the goal object.

The robot arm is decided to be put in a higher configuration for two reasons. It is closer to the TIAGo robot configuration. Secondly, when an object is lifted from the clutter, it needs to be at a certain height so that it does not collide with other objects when moving in either direction.

The camera is also located a bit higher compared to the previous approach. While testing if the models are all working well, we noticed limitations for the orthographic transformation, not being able to transform correctly if the objects are higher than $17cm$. The reason is unknown to me, the resulting orthographic images show only the objects' edges, and the rest is colored black.

The previous approach using the Remote API client has no collision detection to make use of collision detection. We established the ZeroMQ node API connection [10] between the client and the server, namely the CoppeliaSim simulation environment. Once a collision detection is desired, a

[10]https://www.coppeliarobotics.com/helpFiles/

message is sent to the server to enable the collision check function. Once the movement of the non-responsive arm is completed, the client requests a collision flag to act up on it.

There are two different distinguishable cases in which the robot arm could collide with objects.

- Wrong estimation of the position and orientation of the object of interest. Antipodal gripper collision.
- The robot arm linker or gripper collides with other objects while moving in the working scene.

3.6.2 Grasp to Grasp Training

Using the same procedure of training and reward functions as for the push-to-grasp function, the model did not converge, see Fig. 3.16. The task is more complex to learn compared to push to grasp task. By checking the training log, the robot arm was quite often colliding with the goal object due to the weak estimation of the roll yaw orientation of the goal object. This was solved greatly using the projection method. The other case of colliding is with adjacent objects. In this case, the model needs to properly estimate the space around the objects to avoid the collision, which is a relatively difficult task for such models.

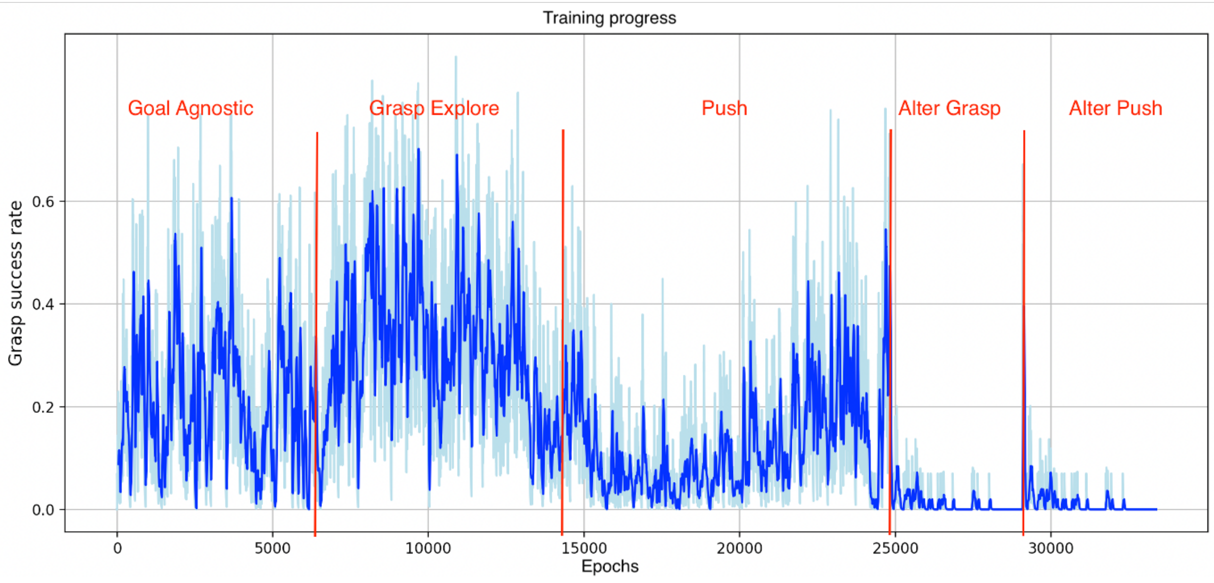


Figure 3.16: Grasp success rate during the 5 training stages.

Here we attempt to tune some of the hyper-parameters. During the first two grasp stages of training, we only spawned two objects in the scene to reduce the likelihood of having two objects too close to the goal object, resulting in a collision. However, we see that the model cannot exceed around 60% goal object grasp success, see Fig. 3.17. Stage *Push A* During the first stage of Pushing, we left the threshold value between the choice of grasp goal and grasp non-goal action at 1.5. The number of objects is 5, also for the rest of the training. In stage *Push A* we reduced it to 1.1 as we noticed the grasp of the goal object is relatively high compared to what we would like to train (grasp non-goal). In stage *Push C* it is put lower to 1.0, at *Push D*, the threshold value is put to 1.5. We see clearly that the model has not converged, and its performance is close to *Push A*.

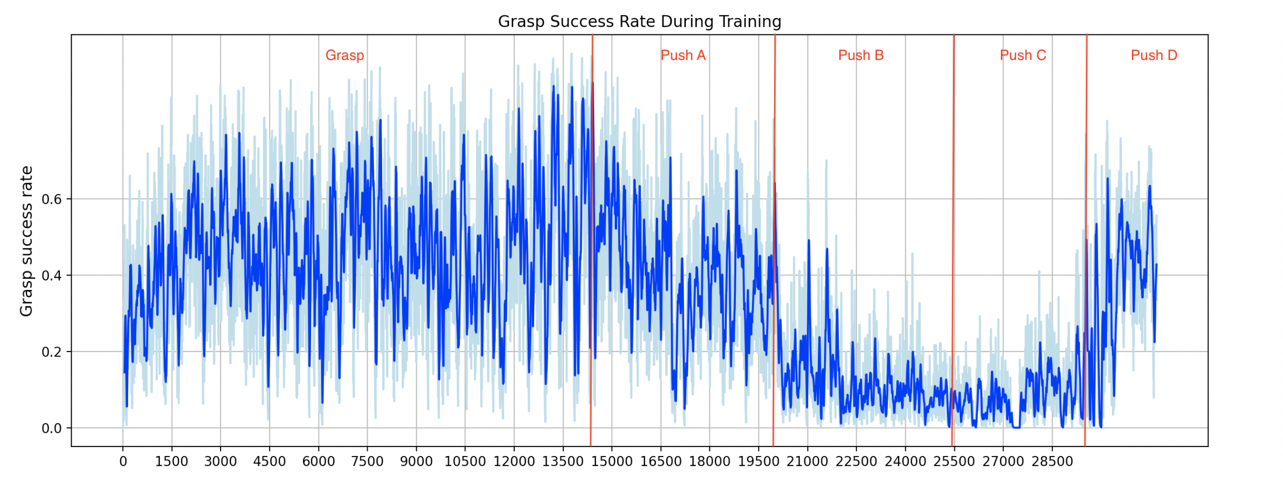


Figure 3.17: Grasp success rate during the 2 training stages. Including hyper parameters tuning in the push stage.

Changing the threshold values, the number of spawned objects and increasing training time have not improved the model learning. From the visual observation of testing the model, it was clear that the model improved its estimation of the Yaw orientation of the objects. However, it is not good enough that it learned sequence removal of objects.

4 Experimental Setup and Results

To evaluate the proposed approach push-to-grasp, we conducted extensive experiments in the CoppeliaSim simulation environment [RSF13]. Our experimental setup consists of a UR5 robot arm and RGB-D Intel RealSense SR300, as shown in Fig. 3.5. We used inverse kinematics (IK) solver for motion planning purposes [Dia10]. The networks are trained with Adam optimizer [KB14], the learning rate is fixed on 10^{-4}, and weight decay of 2^{-4} has been considered. We trained the models using NVIDIA V100 Graphics Processing Unit for faster training time. We compare our approach to Xu et al. [Xu+21] since, to the best of our knowledge, it is the current state-of-the-art approach in this domain. We train and test both approaches on the same hardware and same evaluation scenes to achieve fair compression. We use the following evaluation metrics that have been used previously by [Xu+21; Fuj+20]:

- **Completion (C)**: The mean percentage completion over n test runs. Completions are successful and equal to 1 in a test run; if the system does not exceed in failing to grasp the goal object $n = 5$ times, else it is 0. The metric measures the system's ability to complete the task, see equation 6.

- **Grasp success (GS)**: The mean percentage of successful grasp over all grasp attempts. This metric represents the accuracy of the model and its ability to estimate goal object successful grasp, see equation 7.

- **Motion number (MN)**: The mean number of push actions per completion. It reflects action efficiency, see equation 8.

$$Completion\ rate\ C\% = \frac{1}{N}\sum_{n=1}^{N} s_n,\ s = \begin{cases} 1, \text{if a} \leq 5 \\ 0, \text{if not} \end{cases} \tag{6}$$

Where N is number of tests, and a is number of grasp attempts.

$$Grasp\ scuccess\ rate\ GS\% = \frac{1}{a}\sum^{N} sg,\ sg = \begin{cases} 1, \text{if goal grasp} \\ 0, \text{if not} \end{cases} \tag{7}$$

Where sg success grasp, and a is number of grasp attempts.

$$Motion\ number\ MN = \frac{1}{N}\sum^{N} p \tag{8}$$

Where p is number of push actions.

To evaluate the proposed approach, we conducted two sets of experiments. One is a packed (structured) scenario, and the other is a pile (unstructured and complicated) scenario. In the first round of experiments, we evaluate the system's consistency by performing the same experiment 100 times in a packed scenario. An example scene of this experiment together with its its heat-maps (pixel Q-values) are depicted in Fig. 4.1 (*top-row*). The obtained results are summarized in Table 4.1. The low standard error values indicate that our approach produced a consistent sequence of actions to accomplish the task successfully. Furthermore, we observed that since the model strives to gain a reward with respect to the push reward (see Eq. 3), the Q-values were high on and around the goal object. Therefore, shifting the goal object facilitates free space around the target object to be grasped (see Fig. 4.2). By looking at representative scenarios of Fig. 4.1, we hypothesize that approximately one push to grasp the goal object is highly efficient.

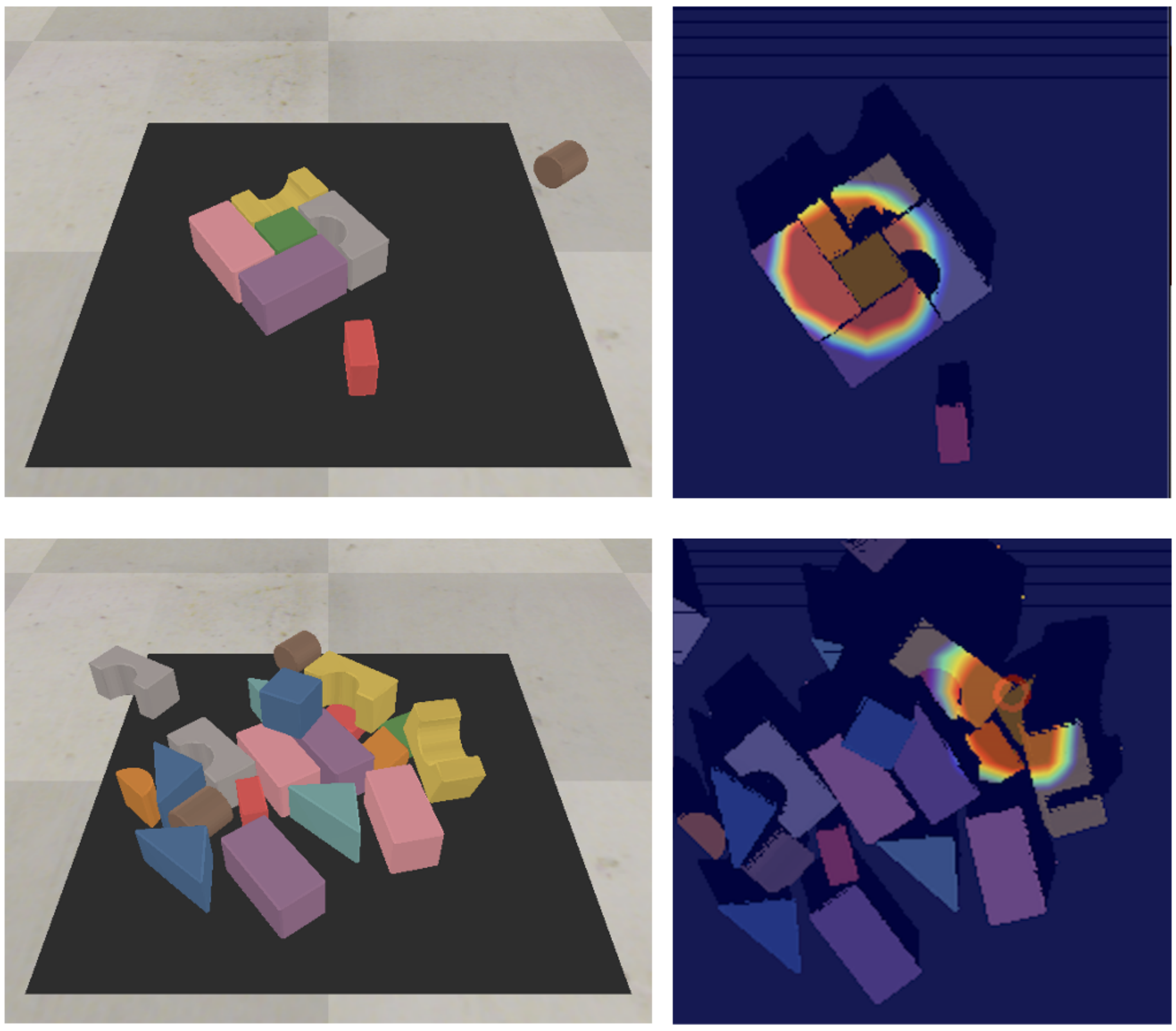


Figure 4.1: Two examples of the generated scene are packed and pile scenarios for evaluation purposes. In all experiments, the goal object is defined by green color: (*top-row*) an example of packed scene (structured) and its heat-maps (pixel Q-values); (*bottom-row*) an example of an unstructured scene (pile) and its heat-maps (pixel Q-values).

Table 4.1: Simulation consistency results on structured scene

Approach	**C%**	**GS%**	**MN**
Xu et al. [Xu+21]	83 ± 3.75	26.48 ± 2.16	3.43 ± 0.31
Ours	$\mathbf{98.98 \pm 1.01}$	$\mathbf{86.08 \pm 3.32}$	$\mathbf{1.12 \pm 0.03}$

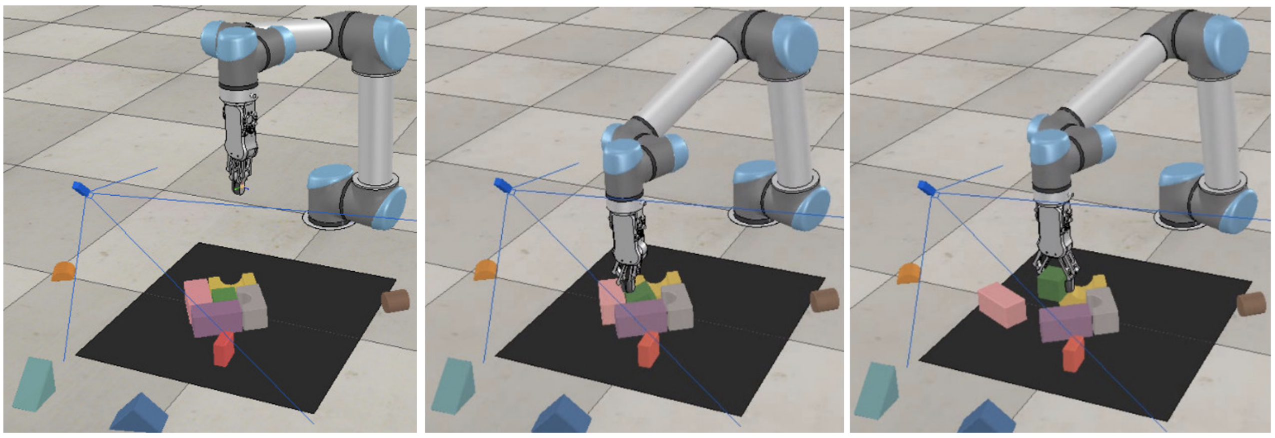


Figure 4.2: Sequence of snapshots showing the robot performing decluttering by pushing the objects surrounding the target object to make grasping the goal object feasible: (*left*) initial scene; (*middle*) pushing surrounding objects; (*right*) grasping the goal object.

By comparing the results, it is clear that our proposed approach outperforms the state-of-the-art [Xu+21] across all metrics by a large margin. In particular, our approach achieves a 100% completion rate (C), which is 10% better than Xu et al. [Xu+21]. Also, in terms of Grasp Success (GS), our proposed approach shows ($\approx$ 60%) better performance. The underlying reason was using their approach. The robot tends to perform unsuccessful grasps of the target object as it lacks the proper estimation of the space around the goal object. It led to several consecutive failures and the termination of the experiment. Our model has better accuracy in making feasible grasp estimations. It has learned to pick up the target object with a remarkably higher grasp success rate ($\approx$ 60%) difference margin compared to Xu et al. [Xu+21].

In the second set of tests, we assess the performance of the proposed approach by generating random scenes using 10, 15, and 20 objects. It is worth mentioning that having more objects in the scene makes the task more complicated. For these experiments, we drop different primitive objects in the scene sequentially from a pre-defined point in the middle and above the working scene to generate test scenes randomly. It results in more dense scenarios, and the goal object gets covered by other objects in some instances (see Fig. 4.3 *left*). We selected a goal object for each experiment out of the possible primitive set of objects. An example of such scenes is depicted on the *bottom-row* of Fig. 4.1. We generated a set of random scenes to train the model using 10 objects. We evaluate the proposed approach by generating 100 different scenes for each level of complexity (i.e., 10, 15, 20 objects per scene). The obtained results are presented in Table 4.2.

Across the completion metric and grasp success rate, we see that our system performed significantly better. However, regarding the Motion Number metric, our system scored lower.

We observed that in scenes containing 10 objects, one push action, on average, was highly efficient in completing the task.

By increasing the level of complexity (adding more objects to the scene), we expected that the model score lower across all metrics since the model has never encountered a scene containing more than 10 objects during the training phase. When we tested the agent with 15 objects per scene, we observed that the agent performed better in terms of task completion and grasp success rate. On a closer inspection, we found that even though there are more objects in the scene, in some instances, the goal object was pushed away and became isolated after other objects were spawned into the scene (see Fig. 4.3 *middle*). Such situations made grasping the goal object easier. By increasing the number of objects to 20, such cases occurred rarely, and complicated scenes were often generated, as shown in (see Fig. 4.3 *right*). Therefore, as we expected, the performance of both approaches decreased. By comparing obtained results, it is visible that the proposed approach outperformed Xu et al. [Xu+21] regarding task completion and grasp success rate. With 20 objects in the scene, our system completed the task with at least 97.22 ± 1.95 which is approximately 20% (on average) better than the result of Xu et al. [Xu+21].

Another interesting point is that, however, the proposed system performed well regarding task completion and grasp success rate metrics. It showed a higher MN than Xu et al. [Xu+21]. As this metric reflects action efficiency, one wants to aim for lower values. Since Xu et al. [Xu+21] model makes the wrong estimation of a feasible goal object grasp, we observed that it keeps attempting to grasp the goal object even when it is not feasible at all. Attempting to grasp the goal object results in moving objects in the scene, and this action is not counted in the MN metric (as MN is defined as the number of push actions per completion). Therefore, Xu et al. [Xu+21] model showed a better MN performance. Nevertheless, not in a desirable manner, it is not systematical to avoid pushing though it is essential, it loses the purpose of synergy value between the push and the grasp action. Our grasp success rate might be on the low spectrum, but this can be improved by increasing the grasp probability threshold. However, that will cause a decrease in the push efficiency as the model will make more pushes and improve clearing to the degree that it has higher grasp probability confidence. Overall, our system showed consistent performance despite the increased number of objects [11].

Table 4.2: Simulation results on 100 different complicated scenes

Approach	#objects	C%	GS%	MN
Xu et al. [Xu+21]	10	71.56 ± 4.48	20.35 ± 1.57	$\mathbf{0.64 \pm 0.25}$
Xu et al. [Xu+21]	15	71.0 ± 4.56	22.72 ± 1.75	$\mathbf{1.1 \pm 0.54}$
Xu et al. [Xu+21]	20	77.89 ± 4.27	21.59 ± 1.78	$\mathbf{1.13 \pm 0.45}$
Ours (mask)	10	$\mathbf{98.97 \pm 1.02}$	66.21 ± 3.91	1.02 ± 0.14
Ours (mask)	15	$\mathbf{100 \pm 0.0}$	$\mathbf{74.60 \pm 3.89}$	3.67 ± 0.98
Ours (mask)	20	97.22 ± 1.95	$\mathbf{69.23 \pm 4.62}$	6.09 ± 1.82
Ours (no mask)	10	$\mathbf{98.78 \pm 1.22}$	$\mathbf{68.33 \pm 4.29}$	3.19 ± 0.56
Ours (no mask)	15	$\mathbf{100 \pm 0.0}$	71.42 ± 4.28	3.27 ± 0.47
Ours (no mask)	20	$\mathbf{99.22 \pm 0.1}$	67.36 ± 4.83	7.12 ± 2.04

[11] https://studio.youtube.com/video/EUrUt9XO7sI/edit

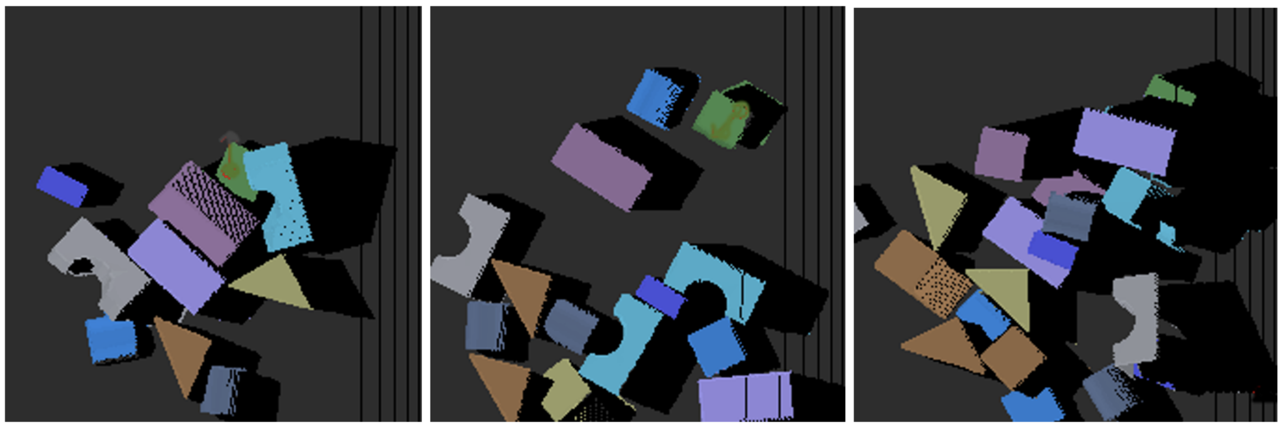


Figure 4.3: Typical scenes with different number of present objects, starting from left to right 10, 15, and 20 respectively.

We hypothesize that the goal mask input to the model is not useful. The power of such models is to learn the no-linearity of the data. Discrete inputs are not beneficial. It is clear that there is a substantial amount of pixel-wise Q-values at no objects, which could lead to necessary push/grasp actions, hence the importance of the output mask. To validate the goal mask as an input to the model, we train and test the model with a mask as an input and another without. In table 4.2, we see the values completion rate (C) and Grasp Success (GS) are insignificantly different. The motion number, on average, is higher with a 1 motion move. We think it is acceptable knowing it is a $1/3$ smaller model. A typical action sequence is depicted in Fig. 4.4.

We observed some limitations of using pushing as a pre-manipulating step for grasping during visual observation of the test scenes. The push is not always smooth. There are cases where the objects flip up due to friction. In other cases, objects keep rolling. It happens to cylinder-shaped objects. There are cases where the objects cannot be pushed at all due to surface friction with the surface or weak friction between the robot gripper and the object.

here talk about the test of the TIAGo robot arm [12] [13].

[12] https://youtu.be/07WUH11mDWU

[13] https://www.youtube.com/watch?v=Ynf2wwvjdJc&ab_channel=KamalMokhtar

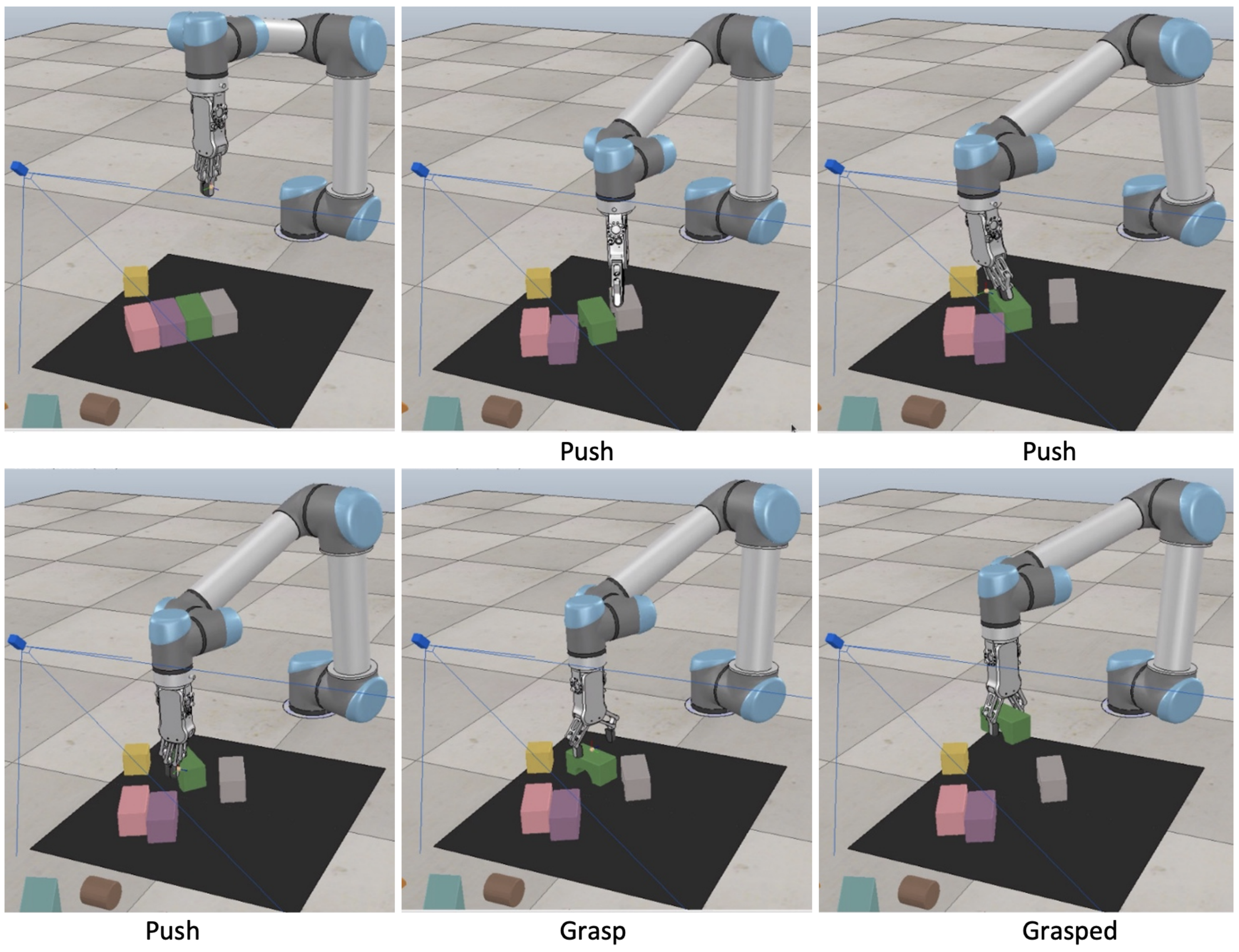
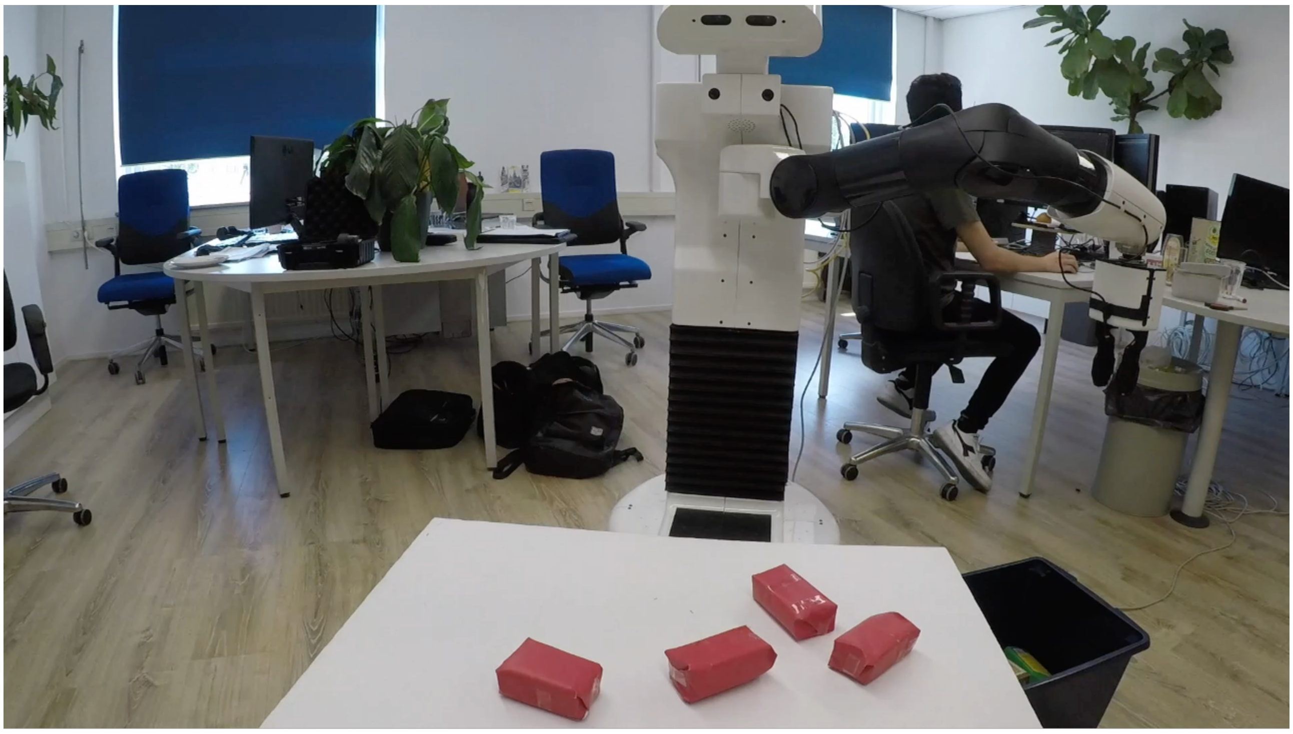


Figure 4.4: A sequence of actions of grasping the goal object from constructed clutter, using the push-to-grasp no goal object mask as an input.


(a) Front view of the working scene of the TIAGo robot.

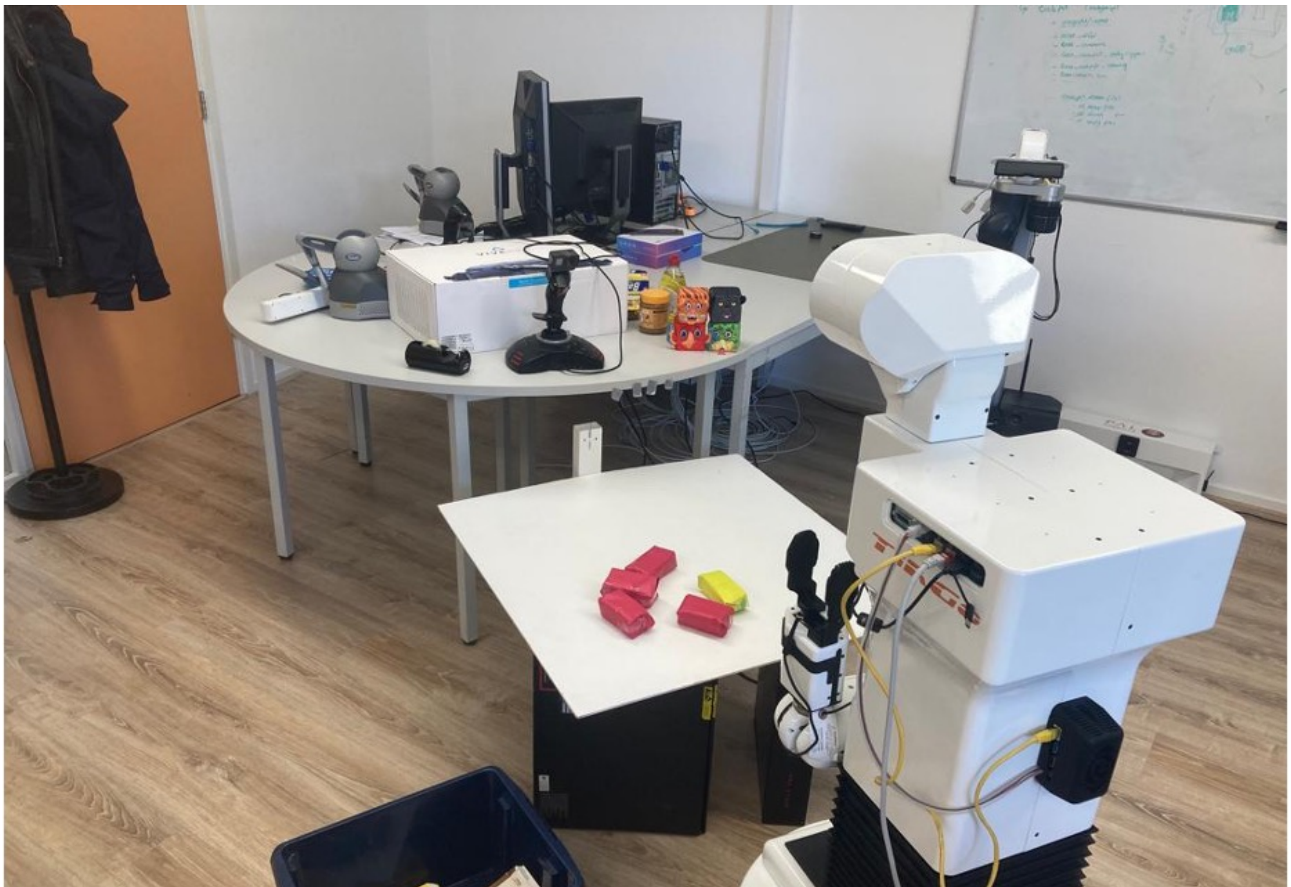

(b) Back view of the working scene of the TIAGo robot.

Figure 4.5: Front and back view of the working scene of the TIAGo robot. The objects are placed on a table with a colloidal base (boxes). To avoid collision with a hard table and damaging the robot. The object is placed in reach of the robot arm. The yellow object is the goal object.

Testing on the real robot is conducted using TIAGo robot. The objects used are sponges colored in different colors to distinguish the goal object (the yellow object is the goal object). The table is a collidable table to avoid collision with a hard service and damage to the robot.

While testing, we did qualitative tests but not quantitative tests. For qualitative test results, see the video [14].

We are able to see that the robot is able to make a decesion when to grasp or push after modifying the threshold values, so the system is well genarlaized.

[14] https://youtu.be/Ynf2wwvjdJc

5 Conclusion

This chapter will cover two main points, first a summary of the work, and answering the main questions. The second will cover aspects for future work.

5.1 Summary of Main Contributions

When starting the research graduation project, options are vast in which direction to go. This project is the first iteration of a project that explores the direction of covering two aspects, the autonomous grasping of goal objects and autonomous pre-manipulation(s) of objects to make a final feasible grasp of the goal object. So I started with doing the literature research and a summary that handles different key problems, which helps to filter the most relevant papers. This is explained in Section 2.4, which sufficiently also answers the first research question.

For the second and the third research questions, This thesis project opted to further work on the self-supervised deep RL approach enabling a robot to grasp a goal object by pushing other objects away when needed. We trained and tested the model in two challenging scenarios: packed objects and a pile of objects. We illustrated that previous work is not reproducible as claimed results. We test with objects in a packed scenario where the objects are tightly packed together and form a structured scene. And in a pile scenario, we randomly drop up to 20 objects inside the robot's workspace to make a complicated and unstructured scene. Experimental results proved that the proposed approach enabled the robot to interact robustly with the environment and successfully manipulate the goal object in both scenarios. Our system outperformed the selected state-of-the-art in terms of task completion rate and grasp success in both scenarios. In the case of the MN metric, our approach outperformed Xu et al. in the packed scenario, while Xu et al. worked better than ours in the pile scenario. We also showed that the work of Xu et al., which is the next work after Zen et al., involving the goal object mask as input is not valid approach. On the contrary, it makes it worse. The model is increased in size by $1/3$, consequently increasing the training and the inference time.

The pushing strategy has some limitations for objects that are hard to push due to friction with the surface under the object or causing objects to fall from their standing positions. In real-life settings, this limitation can cause problems as the robot is often confronted with various levels of friction. Undoubtedly, for objects that have three times the heights of the object that we used in the push testing, shifting them will cause the object to be laid on the surface. Thus, we trained and tested the model to use the approach of grasp removing the non-goal objects. It is too complex for the current architecture to learn, but curriculum learning could be a possibility.

Next to the simulation, we made some qualitative tests on a real robot, namely the TIAGo robot. The model that is trained on the CoppeliaSim and using the UR5 robot arm is deployed for the testing on the real TIAGo robot. Aspects such as the change of object shapes, and colors that are not introduced during training, we were able to conduct some tests. This shows the potential of improving the model by alleviating the mentioned aspects.

5.2 Future Work

The problem has been represented as a discrete MDP problem. This comes with its limitations. In cases where the scene changes before task completion would result in invalid action, one of the traits we humans acquire in such a task is our continuous sensory information that we perceive from the scene and the continuous adaptation based on the sensory information error input. With such skill, the robot would be able to show more resilient behavior and use full actions.

The object that has been used is relatively small in quantity, around 8 objects in total. Adding more variations, especially more complex objects, would help the models handle more complex shapes.

The approach of having the transformation from perspective image to orthographic image is only limited to objects that do not exceed the height of $17cm$. Any higher objects result in improper shapes in the transformed image. The transformation function needs to be properly checked. We were not able to find out the reason for the faulty results. As there is the orthographic image, one could add both the front image and side image and subsequently choose the perspective side with the maximum grapes or push probability. This approach will also increase the degrees of freedom to 6.

We considered pushing objects only on top of the objects, but one can also consider pushing the objects by placing the gripper adjacent to objects and performing a push/slide action. This will prevail over the problem of friction when pushing. There is the scenario that this approach will not work, i.e., having objects in the corner of the working bin.

We showed the limitation of this architecture using the method to grasp non-goal objects and remove them. The system was not able to learn sequence object removing. An approach such as of Murali et al. [Mur+20] seems quite interesting work, and we can learn something from it. It was not chosen for this project since the work cannot be validated due to the unavailable source code. Implementing such work from scratch could be challenging, given this thesis project's duration. Nevertheless, if time permits, it should be considered in the future, as it covers complex object scenes and 6-dof. All that ones need to solve such problem we have at hand.

When testing in real-life using the TIAGo robot, it was quite obvious the limitation of the color segmentation, more advanced object recognition, and segmentation should be further investigated. Also, the colloidal evaluation table has been used to avoid damaging the robot arm. In future work, collision avoidance should be developed. It is a feature that is provided by MoveIt!. Moreover, some components need verification, for example, the transformation for accurate position estimation and the acquired depth information, and for the model to generalize better, it needs to be trained with different light exposures and objects with different shapes and colors.

From a practical aspect, the python code seems fairly on a good level. However, the main.py can be improved. The code is too long and suffers from many conditions checks that are in-tangled.

This research project has covered great aspects of handling objects in an occluded environment. We developed and showed the potential and limitations of such an approach, which contributes to the research community.

References

[22a] CoppeliaSim . *CoppeliaSim Manual*. Jan. 2022. URL: https://www.coppeliarobotics.com/helpFiles/.

[22b] Gazebo . Jan. 2022. URL: https://gazebosim.org/.

[Ale97] R Alexander. "A minimum energy cost hypothesis for human arm trajectories". In: *Biological cybernetics* 76.2 (1997), pp. 97–105.

[And+17] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[Bet19] Kennisplein Zorg voor Beter. *Cijfers: vergrijzing en toenemende zorg*. July 2019. URL: https://www.zorgvoorbeter.nl/veranderingen-langdurige-zorg/cijfers-vergrijzing.

[Bha18] Shweta Bhatt. *5 Things You Need to Know about Reinforcement Learning*. 2018. URL: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html.

[BMK19] Lars Berscheid, Pascal Meißner, and Torsten Kröger. "Robot learning of shifting objects for grasping in cluttered environments". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 612–618.

[Bre+21] Michel Breyer et al. "Volumetric grasping network: Real-time 6 dof grasp detection in clutter". In: *arXiv preprint arXiv:2101.01132* (2021).

[CAN08] Adam Coates, Pieter Abbeel, and Andrew Y Ng. "Learning for control from multiple demonstrations". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 144–151.

[CZP08] Lillian Y Chang, Garth J Zeglin, and Nancy S Pollard. "Preparatory object rotation as a human-inspired grasping strategy". In: *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2008, pp. 527–534.

[DCR19] Carlo D'Eramo, Andrea Cini, and Marcello Restelli. "Exploiting Action-Value uncertainty to drive exploration in reinforcement learning". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[Den+09] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[Dia10] Rosen Diankov. "Automated construction of robotic manipulation programs". In: (2010).

[Dua+21] Haonan Duan et al. "Robotics Dexterous Grasping: The Methods Based on Point Cloud and Deep Learning". In: *Frontiers in Neurorobotics* 15 (2021), p. 73.

[Fen+20] Jie Feng et al. "Generative adversarial networks based on collaborative learning and attention mechanism for hyperspectral image classification". In: *Remote Sensing* 12.7 (2020), p. 1149.

[Fri15] Walter Frick. "When your boss wears metal pants". In: *Harvard Business Review* 93.6 (2015), pp. 84–89.

[Fuj+20] Masahiro Fujita et al. "What are the important technologies for bin picking? Technology analysis of robots in competitions based on a set of performance metrics". In: *Advanced Robotics* 34.7-8 (2020), pp. 560–574.

[GEB20] Gianpaolo Gulletta, Wolfram Erlhagen, and Estela Bicho. “Human-like arm motion generation: A Review”. In: *Robotics* 9.4 (2020), p. 102.

[GSL20] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. “{\epsilon}-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning”. In: *arXiv preprint arXiv:2007.00869* (2020).

[Haa+18] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[HKW18] Nigel Hemmington, Peter Beomcheol Kim, and Cindie Wang. “Benchmarking hotel service quality using two-dimensional importance-performance benchmark vectors (IPBV)”. In: *Journal of Service Theory and Practice* (2018).

[Hua+17] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[Hua+21a] Baichuan Huang et al. “Dipn: Deep interaction prediction network with application to clutter removal”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 4694–4701.

[Hua+21b] Baichuan Huang et al. “Visual Foresight Trees for Object Retrieval From Clutter With Nonprehensile Rearrangement”. In: *IEEE Robotics and Automation Letters* 7.1 (2021), pp. 231–238.

[Iba+21] Julian Ibarz et al. “How to train your robot with deep reinforcement learning: lessons we have learned”. In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[Kal+18] D Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation (2018)”. In: *arXiv preprint arXiv:1806.10293* (2018).

[Kap+10] Daniel Kappler et al. “Representation of pre-grasp strategies for object manipulation”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2010, pp. 617–624.

[KB14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KBP13] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[KJS20] Sulabh Kumra, Shirin Joshi, and Ferat Sahin. “Antipodal robotic grasping using generative residual convolutional neural network”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 9626–9633.

[KK21] Hamidreza Kasaei and Mohammadreza Kasaei. “MVGrasp: Real-Time Multi-View 3D Object Grasping in Highly Cluttered Environments”. In: *arXiv preprint arXiv:2103.10997* (2021).

[Kör+21] Marian Körber et al. “Comparing popular simulation environments in the scope of robotics and reinforcement learning”. In: *arXiv preprint arXiv:2103.04616* (2021).

[KP09] Jens Kober and Jan Peters. “Learning motor primitives for robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 2112–2118.

[KS04] Nate Kohl and Peter Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 3. IEEE. 2004, pp. 2619–2624.

[Lev+16] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[LL22] Shuai Liu and Pengcheng Liu. “Benchmarking and optimization of robot motion planning with motion planning pipeline”. In: *The International Journal of Advanced Manufacturing Technology* 118.3 (2022), pp. 949–961.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[MCC20] Marwan Qaid Mohammed, Kwek Lee Chung, and Chua Shing Chyi. “Review of deep reinforcement learning-based object grasping: Techniques, open challenges, and recommendations”. In: *IEEE Access* 8 (2020), pp. 178450–178481.

[MHK22] Kamal Mokhtar, Cock Heemskerk, and Hamidreza Kasaei. “Self-Supervised Learning for Joint Pushing and Grasping Policies in Highly Cluttered Environments”. In: *arXiv preprint arXiv:2203.02511* (2022).

[Mur+20] Adithyavairavan Murali et al. “6-dof grasping for target-driven object manipulation in clutter”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6232–6238.

[Pas+17] Andreas ten Pas et al. “Grasp pose detection in point clouds”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473.

[Pen+18] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[PP15] Andreas ten Pas and Robert Platt. “Using geometry to detect grasps in 3d point clouds”. In: *arXiv preprint arXiv:1501.03100* (2015).

[PT16] Domenico Prattichizzo and Jeffrey C Trinkle. “Grasping”. In: *Springer handbook of robotics*. Springer, 2016, pp. 955–988.

[PV16] Shailendra Palvia and Vijay Vemuri. “Forecasts of jobless growth: Facts and myths”. In: *Journal of Information Technology Case and Application Research* 18.1 (2016), pp. 4–10.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[RSF13] E. Rohmer, S. P. N. Singh, and M. Freese. “V-REP: A versatile and scalable robot simulation framework”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[Rus97] John Rust. "Using randomization to break the curse of dimensionality". In: *Econometrica: Journal of the Econometric Society* (1997), pp. 487–516.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SEB12] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. "An overview of 3D object grasp synthesis algorithms". In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 326–336.

[SW10] "Bellman Equation". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 97–97. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_71. URL: https://doi.org/10.1007/978-0-387-30164-8_71.

[Tan+21] Bingjie Tang et al. "Learning collaborative pushing and grasping policies in dense clutter". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6177–6184.

[TP18] Andreas Ten Pas and Robert Platt. "Using geometry to detect grasp poses in 3d point clouds". In: *Robotics Research*. Springer, 2018, pp. 307–324.

[Wan+17] Xinyu Wang et al. "Robot manipulator self-identification for surrounding obstacle detection". In: *Multimedia Tools and Applications* 76.5 (2017), pp. 6495–6520.

[WAW14] Diana Werner, Ayoub Al-Hamadi, and Philipp Werner. "Truncated signed distance function: experiments on voxel size". In: *International Conference Image Analysis and Recognition*. Springer. 2014, pp. 357–364.

[Xia+20] Yu Xiang et al. "Learning RGB-D feature embeddings for unseen object instance segmentation". In: *Conference on Robot Learning (CoRL)*. 2020.

[Xu+21] Kechun Xu et al. "Efficient learning of goal-oriented push-grasping synergy in clutter". In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6337–6344.

[YLC20] Yang Yang, Hengyue Liang, and Changhyun Choi. "A deep learning approach to grasping the invisible". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2232–2239.

[Zen+18] Andy Zeng et al. "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4238–4245.