Imperial College London
Department of Computing

Robotic Manipulation in Clutter with Object-Level Semantic Mapping

Kentaro Wada

29th March 2022

Supervised by Prof. Andrew Davison

Copyright Declaration

Abstract

To intelligently interact with environments and achieve useful tasks, robots need some level of understanding of a scene to plan sensible actions accordingly. Semantic world models have been widely used in robotic manipulation, giving geometry and semantic information of objects that are vital to generating motions to complete tasks. Using these models, typical traditional robotic systems generate motions with analysis-based motion planning, which often applies collision checks to generate a safe trajectory to execute. It is primarily crucial for robots to build such world models autonomously, ideally with flexible and low-cost sensors such as on-board cameras, and generate motions with succeeding planning pipelines.

With recent progress on deep neural networks, increasing research has worked on end-to-end approaches to manipulation. A typical end-to-end approach does not explicitly build world models, and instead generates motions from direct mapping from raw observation such as images, to introduce flexibility to handle novel objects and capability of manipulation beyond analysis-based motion planning. However, this approach struggles to deal with long-horizon tasks that include several steps of grasping and placement, for which many action steps have to be inferred by learned models to generate trajectory. This difficulty motivated us to use a hybrid approach of learned and traditional to take advantage of both, as previous studies on robotic manipulation showed long-horizon task achievements with explicit world models.

This thesis develops a robotic system that manipulates objects to change their states as requested with high-success, efficient, and safe maneuvers. In particular, we build an object-level semantic mapping pipeline that is able to build world models dealing with various objects in clutter, which is then integrated with various learned components to acquire manipulation skills. Our tight integration of explicit semantic mapping and learned motion generation enables the robot to accomplish long-horizon tasks with the extra capability of manipulation introduced by learning.

Acknowledgements

I am very thankful for the support given to me by many people over the course of my studies, without which this work would not have been possible.

I am particularly grateful to Prof. Andrew Davison for providing me with the opportunity to pursue a PhD in the Dyson Robotics Laboratory at Imperial College. As my supervisor and the director of the lab, he was incredibly supportive, patient, understanding, and trusting. He guided me to do best work possible with interesting problems helping me to draw big pictures and pursue impactful topics.

I am appreciative of Dyson Technology Ltd. for not only funding my research, but providing an opportunity for many insightful discussions with Charles Collis, Iain Haughton and the rest of the robotics research team.

I feel very fortunate being part of the Dyson Robotics Lab, where I had the opportunity to collaborate with many excellent researchers including Stephen James, Tristan Laidlow, Daniel Lenton, and Edgar Sucar. I am also thankful for the advice and discussions with all other members of the Dyson Robotics Lab, past and present: Stefan Leutenegger, Robert Deaves, Edward Johns, Michael Bloesch, Ronald Clark, Talfan Evans, Tristan Laidlow, Sajad Saeedi, Raluca Scona, Jan Czarnowski, Eric Dexheimer, Fabian Falck, Dorian Henning, Charlie Houseago, Ivan Kapelyukh, Xin Kong, Zoe Landgraf, Shikun Liu, Hidenobu Matsuki, Kirill Mazur, Seth Nabarro, Andrea Nicastro, Aalok Patwardhan, Marwan Taher, and Shuaifeng Zhi.

Iosifina Pournara gave me a lot of helps, organizing attendance to conferences and ensuring that I had access to the equipments I needed, as well as providing general support, advice and encouragement.

Finally, I would like to thank my family and friends for their unconditional care and support during the PhD. Thank you for supporting me along this long journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Robotic Manipulation and Visual Perception

To intelligently interact with environments and achieve useful tasks, robots need some level of understanding of a scene to plan sensible actions accordingly. Robotic manipulation, especially, incorporates physical interaction with objects demanding comprehensive scene understanding compared to other (usually non-physical) interactions such as navigation and communication. Warehouse robots, for example, need to identify the structure of piled boxes and determine the appropriate order and motion to unload them based on factors such as the boxes' mass, orientation constraints, and inter-box contact and support. Cleaning robots need to identify the material types of floors and objects to know how they should be cleaned by selecting suitable detergents, utensils, force, and trajectory. To tidy up a room, robots need to decide where each object will be stored, and plan an efficient storing procedure using the knowledge of objects' category and size, usage characteristics, and some canonical target locations based on the personal preferences of humans.

Robotic manipulation is the capability to interact with objects to change their position and orientation to achieve goal configurations. This interaction usually happens via the edge of a robotic arm (i.e., end-effector) conducting motions such as pushing, grasping, transporting, and placing objects from one place to another. Today, the greatest successes of manipulation have been in factories for automating manufacturing processes such as assembly, palletizing, product inspection, etc. These robots are called industrial robots, and an estimated 2.7 million industrial robots were operating around the world as of 2020 [IFR, 2020].

The Rise of Manipulation with Vision

The first manipulation robot was developed at Argonne National Laboratory in 1947, for handling radioactive materials [Goertz, 1952, Goertz, 1964]. This robot did not have any automation, and was teleoperated by a human operator. The operator's motion was replicated with a master-slave system (master: human operator, slave: robot), and the exerting forces of the robot were given to the human operator as force feedback.

In the 1950s, the first industrial robot, Unimate, was invented by George Devol [Devol, 1961]. Unimate was already in operation in 1961 on a car assembly line at General Motors, and automated the transportation of die casting from an assembly line and welding on car bodies, which was a dangerous task for human workers [CMU, 2003]. As is common in industrial robots even today, Unimate was designed to do repetitive motions in a manufacturing process given pre-defined object locations. Therefore, when this robot was situated in a different environment or set to do new task, human intervention was required to calibrate and program motions for adaptation.

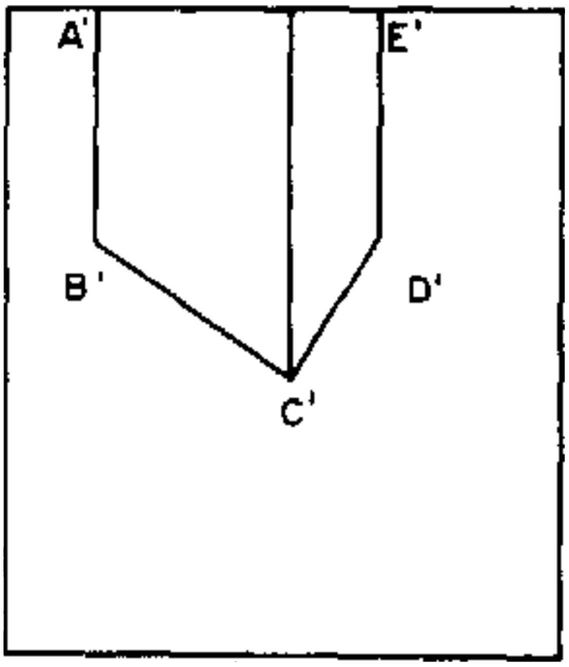
To automatically adapt to the variations in a scene, robots need the ability to perceive the world: to acquire scene representation from sensory inputs. In 1961, Ernst integrated a touch sensor with a robotic manipulator [Ernst, 1962], which allowed it to explore a region on a table to localize cubes for grasping and stacking. Since touch sensors require robots to do interaction before sensing, the operation of Ernst's robot was quite slow involving an exhaustive search of the objects.

Vision sensors (i.e., cameras) have always seemed to be a promising alternative for implementing robotic perception, considering the remarkable ability of humans to perceive the world with their eyes. Compared to the other sensing abilities of humans (e.g., touch; sound), visual sensory inputs quickly give rish information about a large region of a scene from a remote distance and without interaction.

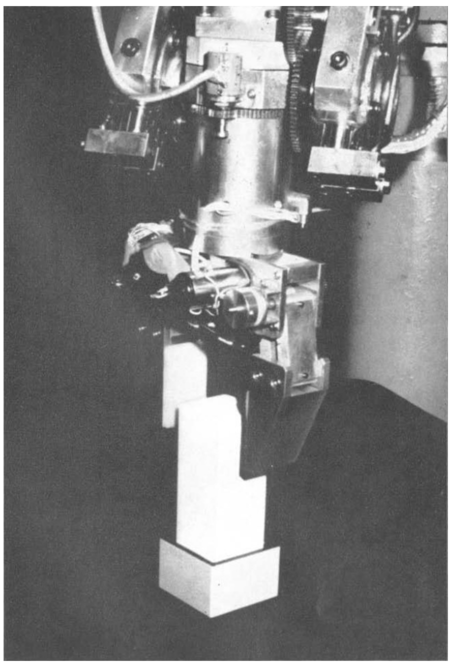
In 1963, Roberts did the pioneering work in visual perception for robots at MIT Lincoln Laboratory, identifying the 3D locations of plane-faced objects from halftone images [Roberts, 1963]. His program processed 2D images to detect edge points and lines, which described objects with a line drawing of polyhedra. This line drawing was compared to 3D models stored in the program, and the structure of a scene was recognized when the matching succeeded, identifying the location and geometry of the objects. This work had a great impact on robotics since 3D information about objects is crucial for robots to determine interaction points, and this had been missing in prior 2D image processing and pattern recognition work at that time.

Inspired by Roberts' work on 3D vision, several researches tried to integrate it actually with robotic manipulation [Feldman et al., 1969, Wichman, 1967, Nilsson, 1969]. In 1967, Wichman used a TV camera to detect two cubical blocks on a table and showed similar capabilities as in the work of Ernst with a touch sensor (block stacking) [Wichman, 1967]. After stacking the blocks, the robot also used a feedback loop to improve the alignment of the blocks. By integrating force and vision sensors, Inoue achieved more complicated manipulations such as peg-in-hole and turning a crank in 1971 [Inoue, 1971].

Through these initial experiments in vision-based robotic manipulation, it was learned that real visual data is noisy and the vision algorithms have to deal with this. Even when using simple objects with colors that are distinctive from the background: e.g., a white cuboid block on a black background shown in Figure 1.1, the detected edge points could include spurious edges and miss real edges [Shirai, 1987].



(a) Line drawing of a block.



(b) A robot inserting a block.

Figure 1.1: A robot inserting a block for assembly with 2D line drawing generated from a camera image (images are from [Shirai and Inoue, 1973]).

These results encouraged robot vision research to move towards improving algorithms to detect and estimate the pose of more complicated-shaped objects from noisy sensory inputs. In the early 1980s, it was already possible for industrial robots to handle overlapping parts in manufacturing using an edge and feature-based vision systems [Bolles, 1977, Bolles and Cain, 1982].

Although these visual perception systems for manufacturing were quite efficient, the environments where robots worked had to be heavily controlled. Typically, the objects presented to a camera had only a single category, the background was constant, and the camera was fixed; for example, clamps on a belt conveyor with a top-down camera. The detection area for objects in an image was well-defined (e.g., the area of the conveyor belt), and the pose of the objects was well-constrained (e.g., stable poses on the conveyor). To work in more general environments to accomplish general tasks, robots need further intelligence to understand scenes without these constraints.

1.2 Semantic Scene Understanding for Manipulation

In addition to 3D geometry as discussed in the previous section, semantics are another key aspect of scene understanding giving a high-level knowledge about objects. During manipulation of objects, robots not only need to know the 3D positions of an object's surface where they can interact, but also need to select sensible actions according to the categories of objects. For example, when a robot is tasked with storing a clean mug in a cupboard, it needs to know how the mug should be grasped and placed so that it can accomplish the task as humans would expect. In this mug-to-cupboard task, humans will probably grasp the mug's handle or side and place it in a stable upright orientation. Humans do this very intuitively without seemingly thinking, but there is deep reasoning behind it: the mug is clean so the rim and inside should not be touched; the mug can fall from the cupboard, so it should be placed stably. This reasoning is possible because humans know about mugs and how they should be treated, and robots also need this high-level knowledge (i.e., semantics) to achieve intelligent manipulation.

The Rise of Semantics in Robotics

Semantic scene understanding builds a world model where semantics are associated with its geometry. The use of semantic world models dates back to the days of Shakey, the first mobile robot developed at Stanford Research Institute from approximately 1966 through 1972 [Nilsson, 1984]. Combining research in robotics, computer vision, and natural language processing, Shakey could perceive scenes, understand human voice commands, and analyze these commands to execute tasks. Shakey inspired researchers to work on further development of scene understanding and navigation for mobile robots with semantics. Flakey, a successor project of

Shakey, reasoned about the beliefs, desires and intentions of users using epistemic logic [Georgeff and Lansky, 1987]. In 1985, Chatila and Laumond showed automatic decomposition of a world model into semantic chunks such as rooms, doors, corridors and obstacles [Chatila and Laumond, 1985]. The world model of rooms was generated by detecting reference objects using visual and ultrasonic sensors on a robot called HILARE. Although the level of semantics in [Chatila and Laumond, 1985] was limited to geometrically inferrable properties, its semantic decomposition of the map allowed some high-level user commands such as "Go to the next room", improving human-robot interface. Current applications of this smart navigation can be seen in human assistant and entertainment prototypes such as smart wheelchairs, offering seamless navigation towards a goal location by voice commands [Burgard et al., 1999, Simpson, 2005].

Similarly to navigation, semantics understanding is often incorporated in the task specification for manipulation: what objects robots should manipulate. There can be useful robotic tasks that only concern the geometry of objects such as moving a pile of debris from one place to another. In this case, it might be enough to have only a geometric 3D understanding of the pile, and the robot will keep scooping objects up until it finishes. The order of scooping does not matter. However, consider what happens when this task gets a bit more complicated, with the additional requirement of separating debris based on material type. This new task requires the robot to selectively manipulate debris by understanding the materials (i.e., semantic) and separating them based on that. As seen in this example, semantics added to the task specification in manipulation can significantly change the actual motions: "move everything" becomes "*separate* and move".

Semantic understanding not only enables more specific task definitions, it also allows a system to associate extra attributes to the objects detected in a scene. In the forementioned mug-to-cupboard example, the mug's attributes can be desired grasp point (the handle), placement orientation (upright), material (e.g., ceramic or plastic), mass. This extra information gives robots the possibility to generate better motions by directly using the attributes (e.g., grasping the desired point;

placing it in the desired orientation); or reasoning from the attributes (e.g., placing a ceramic mug slowly not to break it). For certain objects, the extra attributes could be a specific trajectory of motion such as valve rotation or opening a door, enabling manipulations that are difficult to accomplish otherwise.

After early attempts at navigation with semantics in the late 1980s [Georgeff and Lansky, 1987, Chatila and Laumond, 1985], there was no significant progress on the use of semantics due to little interest from industry and navigation research. It was not until the 2000s that the use of semantic world models acquired more attention from robotic researchers, when life-sized mobile manipulators and humanoid robots became available. Along with mobility, these robots had all sensors on-board, notably differentiating themselves from traditional fixed manipulators in industry. These on-board sensors allow robots to be self-contained and give flexibility to adapt to different environments without external help of sensors in the environment. This self-contained ability is crucial for intelligent robots that perform several tasks with the same hardware.

In the early 2000s, [Petersson et al., 2002] and [Taylor and Kleeman, 2003] integrated vision-based pose estimation with manipulators to show autonomous detection and grasping of an object on a table. These robots were able to distinguish the target object from other structures in a scene (e.g., table), and provide semantic attributes for grasping (i.e., grasp point); however, the demonstrated manipulations were short-range and limited to a single object, which was not very different from the use of industrial robots in manufacturing.

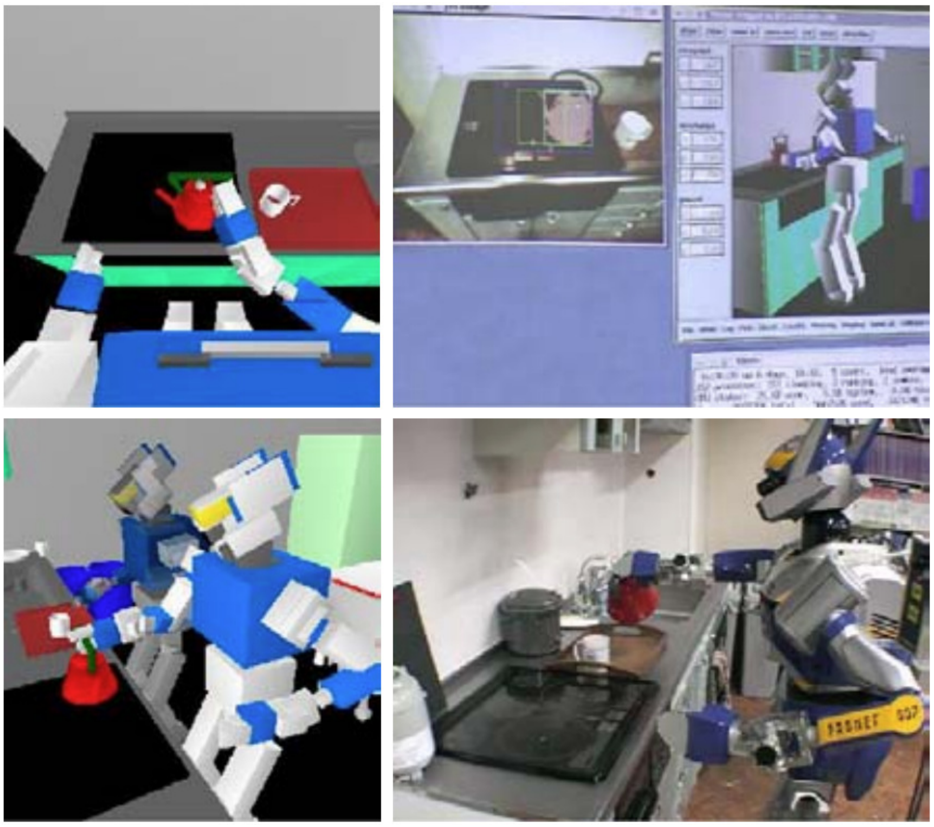
Manipulation with a 3D Semantic Map

More comprehensive semantic world models and their applications to manipulation were seen with a humanoid robot project in Japan in the 2000s. In 1997, the Humanoid Robotics Project (HRP) was launched, aiming at building humanoid robots to assist in general human activities. This project was lead by Kawasaki Industries and Hirochika Inoue, who was also leading a research group at the University of

Tokyo. At the start, several Honda P3 robots were bought from Honda Motor, which were the predecessor of ASIMO created in 2000. In the same year, HRP launched a customized version of Honda P3 as their first robot, HRP-1, which had new software system for teleoperation [Hirukawa et al., 2004]. In 2002, the second generation of HRP, HRP-2 was presented with a newly-designed hardware and software system [Hirukawa et al., 2004]. HRP-2 was expected to be a platform for robotics research even after HRP finishes, and its controller system, OpenHRP [Kanehiro et al., 2002], was designed so that researchers could customize the low-level control commands, which had not been possible in the Honda P3. Research using HRP-2 was continued at the research group at the University of Tokyo, showing various task achievements in human's daily-life activities in the 2000s.

The use of semantic world models for manipulation was heavily explored in this research group as shown in Figure 1.2. In 2004, Okada et al. presented an integrated software system for HRP-2 [Okada et al., 2004b], which had comprehensive features for humanoid robot operation such as control, recognition, dialogue, and planning. One of the significant differences from the original HRP-2 system was that it had built-in software for 3D modeling [Matsui and Inaba, 1990], with which world models were created being maintained with perception; from this manipulation motions were generated being sent to the controller. The capability of this system was presented in household environments such as a kitchen, showing the manipulation of various objects including furniture and appliances (e.g., drawers, microwave). To generate sensible manipulation motions for these various objects, object-specific desired motions were associated to each object in the world model such as the grasp point and trajectory to open a microwave [Okada et al., 2005]. After developing several methods for motion generation based on semantic world models [Okada et al., 2004a, Okada et al., 2005], they showed various task achievements (e.g., water pouring, dish washing) in a real kitchen environment [Okada et al., 2007] while using a particle filter to maintain the world model localizing objects and the robot itself with respect to the model.

After these demonstrations of humanoid robots doing various household tasks,


(a) Kettle detection and pouring.

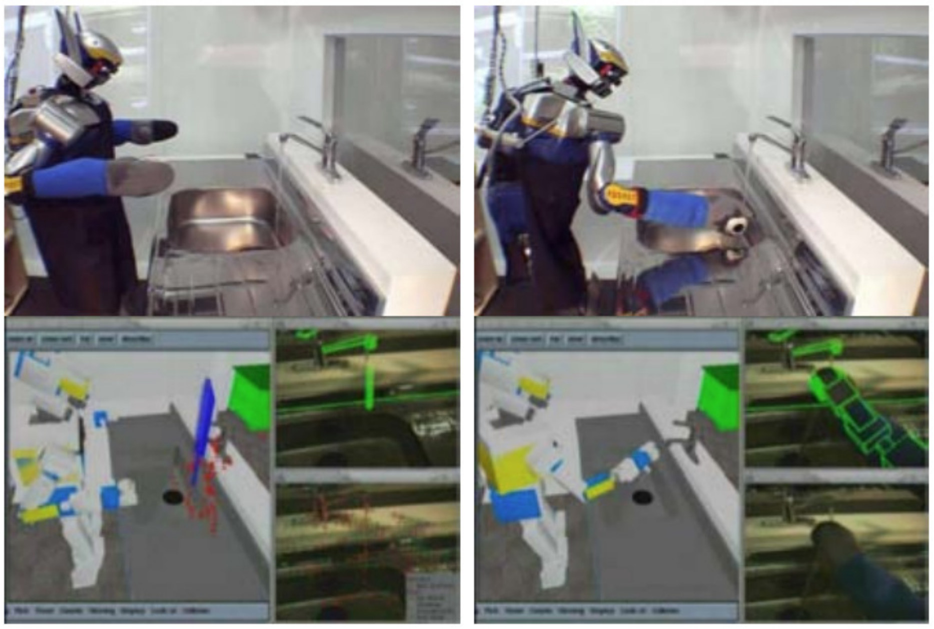

(b) Tap and water detection for dishwashing.

Figure 1.2: A humanoid robot conducting household tasks using 3D semantic map and visual perception (images are from [Okada et al., 2005, Okada et al., 2007]).

semantic world models became a standard representation for robots to generate manipulation motions. Subsequent work enriched the model to have hierarchical scene structure to work in larger-scale environments [Galindo et al., 2008, Zender et al., 2008], and expanded attribute information with an automatically built knowledge-base from the web (e.g., Wikipedia) and human observations [Tenorth et al., 2010]. Along with world representation, research on motion planning in the early 2010s was also inspired by these demonstrations. OpenRave [Diankov and Kuffner, 2008] automated trajectory generation and grasp point selection by integrating world models with collision-based motion planners and grasp synthesis. OMPL [Sucan et al., 2012] and MoveIt! [Chitta et al., 2012] abstracted the integration of motion planning with robotic systems using an emerging robotic framework at that time, ROS [Quigley et al., 2009] developed by Willow Garage.

Despite the variety of manipulation tasks demonstrated in HRP and subsequent projects, the capability of the vision system to deal with diverse objects and in cluttered environments was limited. Robots needed to have strong prior knowledge of the initial state of objects and hardly dealt with occlusions, contact and support among objects, which are common in cluttered scenes with piles of objects.

Semantic Mapping with Vision

Concurrent with the development of the use of semantic maping in manipulation, several projects improved visual capability to detect and estimate the pose of objects [Collet et al., 2009, Collet et al., 2011], demonstrating robotic grasping in a cluttered table-top environment in the late 2000s to the early 2010s.

To encourage the further development of vision-based object manipulation, a robotic competition, the Amazon Picking Challenge (APC), was first held in 2015 and continued annually until 2017. Although the task was pick-and-place of known objects similar to [Collet et al., 2009, Collet et al., 2011], this competition had several unique challenges; including object diversity and complex configuration. The object set for the task had different properties, such as texture-rich/less, convex/concaveshaped, rigid/deformable. Since previous studies on robotic manipulation were mainly applied to objects with one of these properties (e.g., texture-rich, convex-shaped, rigid objects in [Collet et al., 2009, Collet et al., 2011]), researchers had to combine different techniques to recognize all the objects that can appear during the task [Jonschkowski et al., 2016, Zeng et al., 2017].

The complicated configurations of objects in a scene was also a unique challenge in APC, introducing inter-object contacts and occlusions. Objects were presented as a random pile to the robot, and often target objects were overlapped and occluded by the other distractor objects. In such a situation, robots need deep understanding of the scene to generate appropriate motions recognizing which objects are overlapping the target in order to remove them to access the target. Several studies showed intelligent behavior in complicated piles such as occlusion removal [Schwarz et al., 2018, Wada et al., 2018]. However, their vision system was often task-specific, using a fixed top-down camera viewpoint with 2.5D scene understanding, which can fail to generalize to different tasks or environments that require three-dimensional understanding of the scene for more general manipulation such as side grasping and 6DoF motions.

As we saw in the previous section, scene representations that are generally useful for manipulation require properties of 3D and semantics. A promising representation is **object-level semantic map**, a world model composed of object models and their poses, which gives dense geometry and high-level knowledge about objects for planning manipulation. This representation includes, for example, inter-object physical relationships such as contact and support, allowing robots to reason about the causal effect of object configurations to enable high-level motion planning (e.g., removing distractors to pick a target object).

Early studies on multi-view semantic mapping emerged in the 2010s, where the geometry and semantics of objects are accumulated as 3D maps for relatively large-scale environments (c.f., table-top) using moving cameras. Stuckler and Behnke showed real-time pose tracking of object models in a scene while building a dense reconstruction of the background with surfels using an RGB-D camera [Stückler and Behnke, 2012]. Salas-Moreno et al. showed object-level mapping with joint tracking of object models and camera poses [Salas-Moreno et al., 2013]. Tateno et al. showed semantic mapping with a larger number of object models [Tateno et al., 2016]. These studies were later extended to mapping systems without explicit object models using learned object detection from images. McCormac et al. combined learned 2D object detection [He et al., 2017] and volumetric reconstruction [Newcombe et al., 2011] to map objects without pre-defined models [McCormac et al., 2018]. Rünz et al. [Runz et al., 2018] and Xu et al. [Xu et al., 2019] further extended this object mapping system to dynamic scenes, where mapped objects can be moved with human interaction. Note that these researches were generally not applied to manipulation, usually aiming at room-level scene understanding of large objects.

In this thesis, we aim to build a scene understanding system that is generally useful in many robotic manipulation tasks. As discussed above, a promising representation for this purpose is object-level semantic map, a 3D semantic world model composed of object models giving dense geometry and the high-level knowledge of objects that is necessary for advanced planning (e.g., removing distractors to pick an occluded target object). For generality in different environments, we will build a vision system that acquires this scene understanding using a single RGB-D camera mounted on a robotic arm. Using the on-board camera, the robot explores and understands a scene to discover and manipulate target objects to accomplish tasks having generality in different environments with the same hardware setup. The capability of the vision system is demonstrated in cluttered piles of objects where heavy occlusions and inter-object contacts appear.

1.3 Learning and End-to-End Manipulation

With the recent progress in deep neural networks (DNN), increasing robotic research has worked on learning-based end-to-end manipulation, where robotic actions are mapped from some inputs with learned parameters. In vision-based manipulation, the extreme of the end-to-end approach directly generates motions from images: "*pixels to torques*", without building the explicit world models we have seen in the previous section. This trend began especially after the success of DNN in 2012, called AlexNet [Krizhevsky et al., 2012], in a large-scale image classification competition, ImageNet [Deng et al., 2009]. This model had a convolutional architecture represented by convolutional neural networks (CNN) and could extract features from input images favorably to solve the classification problem. Given a large-scale dataset of more than 1M images [Deng et al., 2009], the model learned better features than hand-designed methods, and achieved more accurate classification results. As neural networks do not have strong constraints on the structure of input-output pairs, a similar approach could immediately be applied to robotic manipulation by replacing the classification output with actions such as a grasp point.

Two decades before the rise of DNN, early work in end-to-end control with visual sensory inputs can be seen in 1989 [Pomerleau, 1989], which showed autonomous road following by a land vehicle with a system called ALVINN. In this work, the direction of the vehicle was predicted by a neural network using the road images captured by a camera and a laser range finder. Unlike traditional neural network-based controllers at that time, these images were directly fed into the network without any preprocessing such as extracting a reaching point for controlling manipulator [Hunt et al., 1992, Bekey and Goldberg, 2012]. Although ALVINN [Pomerleau, 1989] showed a successful navigation with an end-to-end control, the output was limited to the 1D space of vehicle direction and was hardly applicable to manipulation, which requires three-dimensional actions as output.

For robotic manipulation, early studies of end-to-end control from images were presented in the late 2000s. In 2006, Saxena presented robotic grasping using grasp points output by a neural network [Saxena et al., 2006, Saxena et al., 2008a, Saxena et al., 2008b] as shown in Figure 1.3. As the grasp points were predicted directly from images, the system did not require any explicit models of the objects to be grasped. To generate the spatial locations of grasp points, Saxena treated this problem as pixel detection similar to other object detection problems such as face detection [Rowley et al., 1998]. The detected grasp points were mapped into 3D space with stereopsis, and a motion planner was used to generate trajectories for reaching and grasping. To train the neural network, either annotations on real images or synthetic data of object models with grasp point annotations was used.

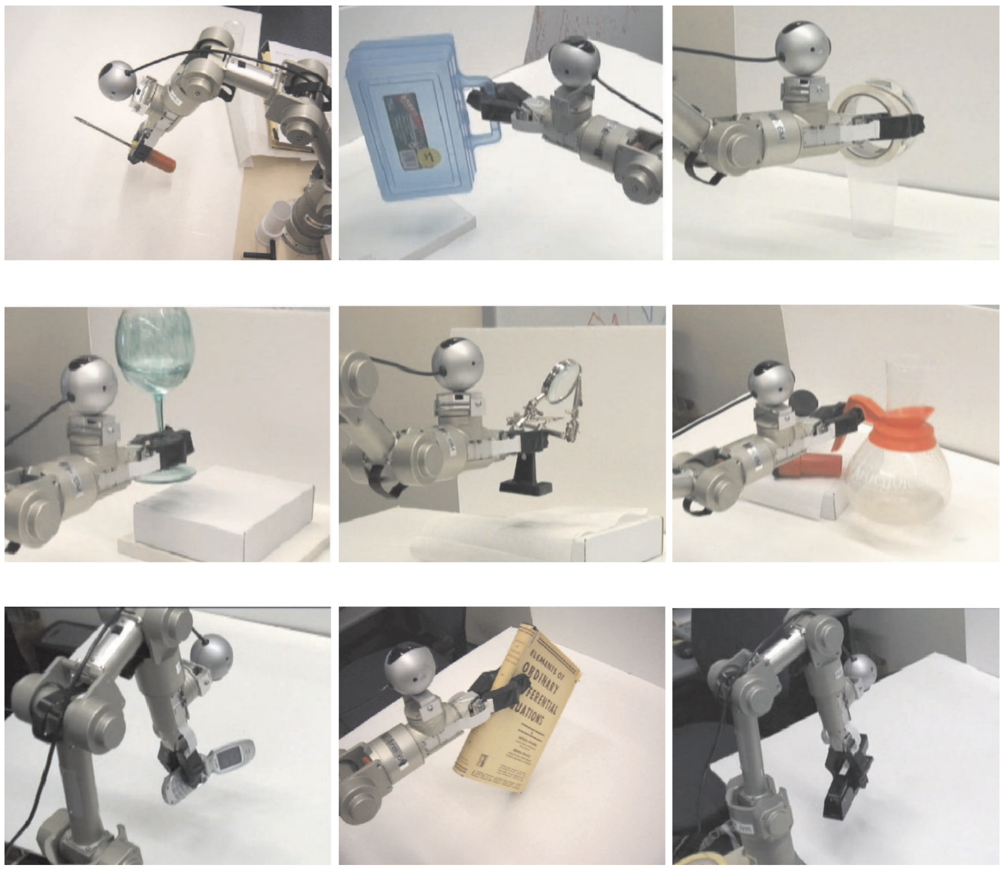
This neural network-based grasp point detection was extended using DNN, as its first application to robotic manipulation. In 2013, Lenz et al. showed an extension of this grasp detection model using deep neural networks to learn better feature extraction, enabling generalization to more diverse objects [Lenz et al., 2013]. The whole scheme of the robotic system was the same to the previous work [Saxena et al., 2008b], combining discrete detection of grasp points and continuous trajectory generation with motion planning.

Later, visual end-to-end manipulation was also applied to continuous motion generation. In 2016, Levine et al. showed direct mapping of image inputs to joint torque actions [Levine et al., 2016], achieving various contact-rich manipulation tasks such as inserting a block into a shape sorting cube, screwing a cap onto a bottle, etc. Unlike the previous work on grasp detection, this work did not have any extra components for motion generation such as inverse kinematics, trajectory planner and joint position controller (which maps a trajectory to toques). This work was one of the earliest examples of a pure end-to-end manipulation system, which does not have any non-learnable components between sensor inputs and motor commands.

Despite this successful demonstration of manipulation in various tasks, pure end-to-end robot control [Levine et al., 2016] required human demonstrations to pre-train the continuous motion trajectory. In most of their demonstrations, the actual main challenge of manipulation must be torque control in contact-rich interaction for box insertion and cup screwing. However, even learning reaching motions from the initial state, which should be easily solvable with traditional trajectory generators, required numerous time-consuming human demonstrations (~100 trials). This challenge in end-to-end manipulation also appeared in discrete grasp detection [Lenz et al., 2013] requiring human annotation of successful grasps. Subsequent studies worked on

(a) Grasp point detection on dishwasher.

(b) Dish unloading.



(c) Various object grasping trained with synthetic data.

Figure 1.3: A robot grasping novel objects via learning-based 2D grasp point detection (images are from [Saxena et al., 2006]).

reducing this burden of trajectory annotation by either self-supervision or learning-in-simulation.

With self-supervision, robots can learn motions from automatically collectable data. In 2015, Pinto and Gupta trained a discrete grasp point detection model using only the grasping experiences collected by robots themselves [Pinto and Gupta, 2016]. Similarly, using self-supervised grasping data collection, in 2016 Levine et al. showed continuous grasping motions by predicting the next end-effector transition as the network output. This work was further extended with reinforcement learning to have more long-horizon motion optimization such as pushing neighboring objects to grasp an object [Levine et al., 2018, Kalashnikov et al., 2018, Devin et al., 2018].

With learning-in-simulation, robots can learn image-to-action mappings with scripted motion demonstrations as an alternative to human trajectory demonstrations in the real world. With training purely in simulation, in 2016 Johns et al. showed grasping isolated objects on a table using depth images [Johns et al., 2016], and in 2017 [Tobin et al., 2017, James et al., 2017] showed pick-and-place of objects using RGB images only. These methods trained network models with various image augmentations in simulation, to make them robust to the differences in the visual appearance of objects between simulation and the real world.

Although these studies have made some end-to-end manipulation tasks practicable and showed a network's implicit understanding of an object's semantics, shape, and graspability, the demonstrated manipulations were still limited to quite short-term task horizons. Even in the heavily tackled robotic pick-and-place tasks [Pinto and Gupta, 2016, Levine et al., 2018], robots were often specialized only in grasping objects, placement was simplified (e.g., randomly dropping into a bin), and the vision sensors were fixed in the scene. To complete more long-horizon tasks including navigation and regrasping, robots need more comprehensive scene understanding, which have not yet been proven possible in these studies on end-to-end manipulation.

In this thesis, we aim to build a robotic system that is able to accomplish long-horizon and challenging manipulation tasks with a universal intelligence across scene and task variations. For this goal, we avoid taking a full end-to-end approach to manipulation, and instead, combine explicit scene understanding using a 3D semantic map with learning-based manipulation. As we have seen in the use of semantic world models in the Humanoid Robotics Project (§1.2), capable and intelligent object manipulation requires significant high-level knowledge about objects (e.g., grasp points, manipulation trajectories), to which a semantic map hugely contributes. In past researches, this knowledge has mainly been acquired via hand-designed labeling or analysis [Okada et al., 2007, Diankov and Kuffner, 2008], which is often suboptimal or time-consuming. Learning-based manipulation has the potential to resolve these limitations by acquiring optimal motions from data and experience. Exploiting the potential of the learning-based approach and the rich information of semantic maps, we show intelligent robotic manipulation that has long-horizon applicability.

1.4 Publications

The contributions made in this thesis have resulted in three main publications.

Paper I: Real-time Semantic Mapping with Object Models

Wada, K., Sucar, E., James, S., Lenton, D. and Davison, A. J. (2020), **MoreFusion: Multi-object Reasoning for 6D Pose Estimation from Volumetric Fusion**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [Wada et al., 2020].

Project page: `https://morefusion.wkentaro.com`

Video: `https://youtu.be/6oLUhuZL4ko`

Code: `https://github.com/wkentaro/morefusion`

This paper introduces MoreFusion, a vision system that builds object-level semantic map, a world model built from an on-board RGB-D camera. As discussed in §1.2, an object-level semantic map is a representation that can be generally useful in different manipulation tasks, giving dense geometric information and high-level knowledge about objects. To build a map that has both semantically and geometrically rich information, we represent the map as a composition of known object models, which represents the full geometry of objects being pre-built with scanning or modeling. For these object models, the vision system estimates their position and orientation in a scene (pose estimation) while a moving camera observes the scene by capturing RGB-D image sequences. Along with pose estimation, this system also does object tracking and dense mapping of background objects. Object tracking allows the system to accumulate information about the same objects and avoid duplication. Background reconstruction gives the geometry of unseen objects that do not have associated pre-built models, and can be used for motion planning such as generating collision-free trajectories.

To achieve all the requirements of the system, we combine 6D pose estimation with volumetric mapping, which has been previously separated. This integration allows us to take a novel approach to pose estimation by exploiting information from multi-view observations accumulated with volumetric mapping. Our system estimates an object's pose using the surrounding geometric reconstruction (e.g., occupied; free space) extracted from the volumetric map, and this allows the system to reason about feasible configurations of objects, avoiding intersections between objects and free spaces. This feasibility check of neighboring objects' collisions is crucial for pose estimation of partially observable objects in cluttered environments. We test this system in object piles in the real-world, and show pick-and-place of target objects under challenging conditions such as close contacts and heavy occlusion.

MoreFusion will be discribed in detail in Chapter 3.

Paper II: Fine-Grained Manipulation with a Semantic Map

Wada, K., James, S. and Davison, A. J. (2022), **SafePicking: Learning Safe Object Extraction via Object-Level Mapping**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. [Wada et al., 2022b].

Project page: `https://safepicking.wkentaro.com`

Video: `https://youtu.be/ejjqiBqRRKo`

Code: https://github.com/wkentaro/safepicking

This paper explores the integration of a semantic map with learning of short-horizon manipulation, with the specific manipulation goal of efficient and safe extraction of target objects from a pile. When robots must pick a specific object from a pile, it is often the case that the target object is overlapped and blocked by distractor objects. In this case, robots can either move the distractors away or carefully extract the target object to avoid task failure and undesirable consequences (e.g., dropping the grasped object; damaging the distractor objects). When there are many distractor objects, object extraction is the more efficient strategy. This maneuver, however, requires appropriate reasoning about how manipulation motions affect the surrounding objects' motions during and after the extraction, which is challenging even with holistic knowledge about the scene with the semantic map created by systems like MoreFusion. As discussed in §1.3, learning-based manipulation has the potential to generate optimal motions that hand-designed motion planning algorithms are unable to accomplish.

We use the capability of learning a model to generate efficient and safe object extraction from a pile. Although recent work on learning-based manipulation often works on learning from raw observations (e.g., images) [Levine et al., 2018, Kalashnikov et al., 2018], we train the model using a semantic map to maintain the integration to other capabilities (e.g., exploration of the target object with multi-view observation of a moving camera). The model receives as input the semantic map in the vector forms of object categories and poses, and is trained to generate the object extraction trajectory that least affects the other objects in the pile. We use the translation of surrounding objects as a safety metric and train the model in simulation with reinforcement learning using the safety metric as a reward. We test the system in both simulation and the real world, and show the successful safe extraction of target objects from a pile.

This system will be described in detail in Chapter 4.

Paper III: Long-Horizon Manipulation with a Semantic Map

Wada, K., James, S. and Davison, A. J. (2022), **ReorientBot: Learning Object Reorientation for Specific-Posed Placement**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. [Wada et al., 2022a].

Project page: `https://reorientbot.wkentaro.com`

Video: `https://youtu.be/ahWN84sWWJU`

Code: `https://github.com/wkentaro/reorientbot`

This paper explores the integration of a semantic map with learning of long-horizon manipulation, with the specific manipulation task of creating certain configurations of objects with specific-posed placement. When robots must place objects in a specific pose, it is often the case that they cannot immediately place the grasped object because of the constrained space at the final placement location. This requires regrasping at certain points that enable the final placement. This pipeline requires long-horizon planning of how the grasped object has to be oriented and released to enable regrasping. As discussed in §1.3, learning-based manipulation often struggles to achieve such long-horizon tasks that includes several subtasks (e.g., grasping, reorientation, regrasping, placement). This time-horizon difference is one of the significant differences from Paper II.

In this paper, the model learns only the key motions for the manipulation task, instead of learning to generate a fine-grained trajectory as in Paper II. In the task of specific-posed object placement, the key motions are two-fold: initial grasping and reorientation (placement for regrasping). With a pile of objects, initial grasping must be collision-free (i.e., valid) not only during grasping in the pile, but also during the maneuvers of reorientation. Reorientation has to expose the target grasp points that enable final placement, while keeping the validity of the initial grasp. These two key motions require a combinatoric search of possible actions: (`number of grasp points`) × (`number of reorientation poses`), a search which is infeasible to run in real time with exhaustive search. We use learned models that receive object poses as input similar to Paper II, and evaluate the possible motions to select the best. We test the system in both simulation and the real world, and show successful motion generations in long-horizon tasks of specific-posed placement.

This system will be described in detail in Chapter 5.

Additional Papers

While not described directly, the following publications were done in conjunction with this thesis:

- Sucar, E., Wada, K. and Davison, A. J. (2020), **Neural Object Descriptors for Multi-View Shape Reconstruction**. In *Proceedings of the International Conference on 3D Vision (3DV)*. [Sucar et al., 2020].
  Project page: https://edgarsucar.github.io/NodeSLAM
  Video: https://youtu.be/zPzMtXU-0JE

- James, S., Wada, K, Laidlow, T. and Davison, A. J. (2021), **Coarse-to-Fine Q-attention: Efficient Learning for Visual Robotic Manipulation via Discretisation**. *Under Review and Submitted to Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [James et al., 2021a].
  Project page: https://sites.google.com/view/c2f-q-attention
  Code: https://github.com/stepjam/ARM

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 introduces basic notation, robot hardware and planning, simulation platform, and provides a primer on 3D mapping (transformations, cameras, geometry) and deep learning.

Chapter 3 presents MoreFusion, a real-time system that build a semantic map of a scene with object models. This system combines object-level volumetric reconstruction and pose estimation, where objects are detected from 2D images, reconstructed as volume with depth images, and replaced by a pre-built object model with a confident pose estimate.

Chapter 4 introduces learning-based manipulation for fine-grained motions into a robotic system with semantic mapping. This integration is explored with the task of extracting target objects from a pile, where a learned model receives a semantic map and a heightmap as input to generate 6DoF trajectories for extraction with minimal undesirable effects on the non-target objects.

Chapter 5 explores the integration of learning-based manipulation with semantic mapping in long-horizon tasks. This integration is presented with the task of placing objects in specific poses, where a learned model evaluates numerous samples of coarse key motions to allow the system to select the best ones to find optimal motions for grasping, reorientation, regrasping and final placement.

Chapter 6 concludes the thesis with a discussion of the research presented and suggestions for future work.

CHAPTER 2

Preliminaries

Contents

2.1 Notation

This thesis makes use of the following notation:

a	This font is used for scalars.
$\mathbf{a}$	This font is used for M-dimensional column vectors, where a_i is the i^{th} element of the vector:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \end{bmatrix}, \quad \mathbf{a}^\mathsf{T} = \begin{bmatrix} a_1 & a_2 & \dots & a_M \end{bmatrix}. \tag{2.1}$$

$\mathbf{A}$	This font is for $M \times N$-dimensional matrices, where a_{ij} is the matrix element at the i^{th} row and j^{th} column:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix}. \tag{2.2}$$

$\boldsymbol{a}$	This font is used for the homogeneous coordinate vector corresponding to the coordinate vector $\mathbf{a}$:

$$\boldsymbol{a} = \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}. \tag{2.3}$$

$\mathbf{I}$	This represents the identity matrix.
$\mathbf{O}$	This represents the zero matrix.

$(\cdot)^\times$ This denotes the cross-product operator that produces a skew-symmetric matrix from a 3-dimensional vector, such that $\mathbf{a} \times \mathbf{b} = \mathbf{a}^\times \mathbf{b}$:

$$\mathbf{a}^\times = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{2.4}$$

2.2 Transformations

2.2.1 Homogeneous Transformations

We use rigid transformations between coordinate frames $\underrightarrow{\mathcal{F}}$ in three-dimensional Euclidean space $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ to represent the pose of a robot, a camera and objects in a scene. Between two arbitrary coordinate frames, as in Figure 2.1, $\underrightarrow{\mathcal{F}}_a$ and $\underrightarrow{\mathcal{F}}_b$, rigid transformations are defined as composite of a translation $\mathbf{t}_{ba}$ and rotation $\mathbf{R}_{ba}$, $\mathbf{T}_{ba}$, which is in the special Euclidean group, $SE(3)$:

$$SE(3) \triangleq \left\{ \mathbf{T}_{ba} = \left[\begin{array}{c|c} \mathbf{R}_{ba} & \mathbf{t}_{ba} \\ \hline 0\ 0\ 0 & 1 \end{array}\right] \mid \mathbf{R}_{ba} \in SO(3),\ \mathbf{t}_{ba} \in \mathbb{R}^3 \right\}, \tag{2.5}$$

and the rotation $\mathbf{R}_{ba}$ is in the group of special orthogonal matrices, $SO(3)$:

$$SO(3) \triangleq \left\{ \mathbf{R}_{ba} \in \mathbf{R}^{3\times3} \mid \mathbf{R}_{ba}^\top \mathbf{R}_{ba} = \mathbf{I},\ det(\mathbf{R}_{ba}) = 1 \right\} \tag{2.6}$$

This transformation matrix, $\mathbf{T}_{ba}$, transforms a point $\mathbf{x}_a \in \mathbb{R}^3$ in coordinate frame $\underrightarrow{\mathcal{F}}_a$ into another coordinate frame $\underrightarrow{\mathcal{F}}_b$:

$$\boldsymbol{x}_b = \mathbf{T}_{ba} \boldsymbol{x}_a, \tag{2.7}$$

where $\boldsymbol{x} := \left[\begin{smallmatrix} \mathbf{x} \\ 1 \end{smallmatrix}\right] \in \mathbb{R}^4$ represents the homogeneous point that enables the multiplication with the 4×4 transformation matrix (i.e., homogeneous transformation matrix). This operation using homogeneous point representation is equivalent to the following operation with rotation matrix and translation extracted from the transformation

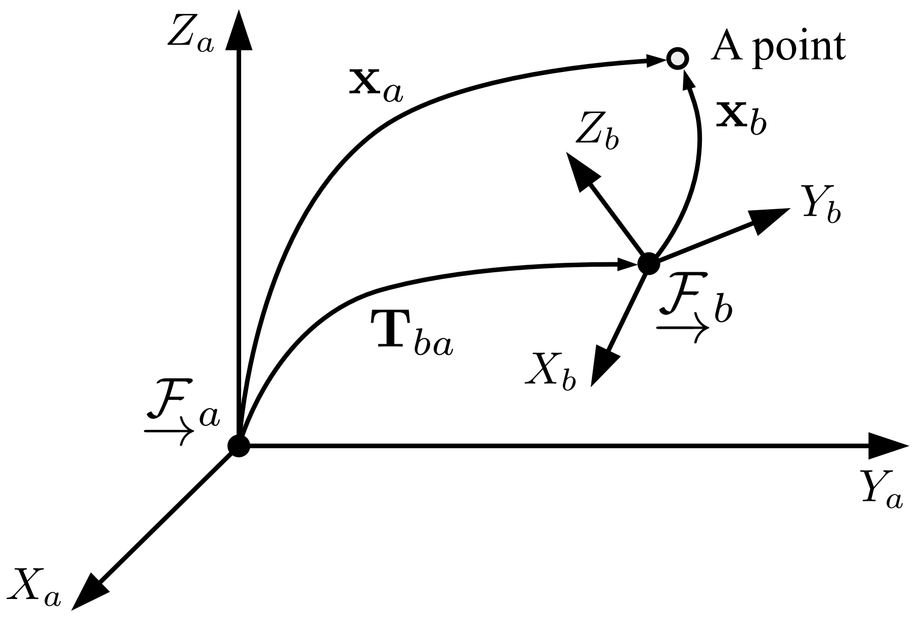


Figure 2.1: **Transformations** between 2 coordinate frame: $\underset{\rightarrow}{\mathcal{F}}_a, \underset{\rightarrow}{\mathcal{F}}_b$.

matrix:

$$\boldsymbol{x}_b := \begin{bmatrix} \mathbf{x}_b \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{ba}\mathbf{x}_a + \mathbf{t}_{ba} \\ 1 \end{bmatrix}. \tag{2.8}$$

Homogeneous operation allows us to chain transformations of points with simple matrix multiplications given another frame $\underset{\rightarrow}{\mathcal{F}}_c$:

$$\boldsymbol{x}_c = \mathbf{T}_{ca}\boldsymbol{x}_a = \mathbf{T}_{cb}\mathbf{T}_{ba}\boldsymbol{x}_a, \tag{2.9}$$

which is particularly useful when there are a number of coordinate frames in the system such as a robot, gripper, camera, and objects. The inverse of the transformation matrix, $\mathbf{T}_{ba}^{-1}$ is:

$$\mathbf{T}_{ba}^{-1} := \left[\begin{array}{c|c} \mathbf{R}_{ba} & \mathbf{t}_{ba} \\ \hline 0\ 0\ 0 & 1 \end{array}\right]^{-1} = \left[\begin{array}{c|c} \mathbf{R}_{ba}^{\intercal} & -\mathbf{R}_{ba}^{\intercal}\mathbf{t}_{ba} \\ \hline 0\ 0\ 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{R}_{ab} & \mathbf{t}_{ab} \\ \hline 0\ 0\ 0 & 1 \end{array}\right] = \mathbf{T}_{ab} \tag{2.10}$$

giving $\mathbf{T}_{ab}$, which is the inverse transformation of $\mathbf{T}_{ba}$.

2.2.2 Euler Angles

Euler angles are another common parameterization of rotation alternative to rotation matrix. While the rotation matrix representation in the previous section has 9 parameters: $\mathbf{R} \in \mathbb{R}^{3\times3}$, Euler angles describe rotation with only 3 parameters: yaw (θ_z), pitch (θ_y), and roll (θ_x).

Geometrically, Euler angles represent rotation angles along the XYZ axes respectively as in Figure 2.2. A full rotation is represented as a sequence of three individual rotations around different axes, whose order matters to represent unique rotations. A commonly used order in robotics is YPR-*modified*: yaw around the Z axis, then pitch around the *modified* Y axis, then roll around the *modified* X axis. When rotating around a *modified* axis, the new axis direction is used as the previous rotation changes the direction of the original axes. It has been shown that this YPR-*modified* is equivalent to the inverse order with *unmodified* axes: RPY-*unmodified*, which is also commonly used in robotics.

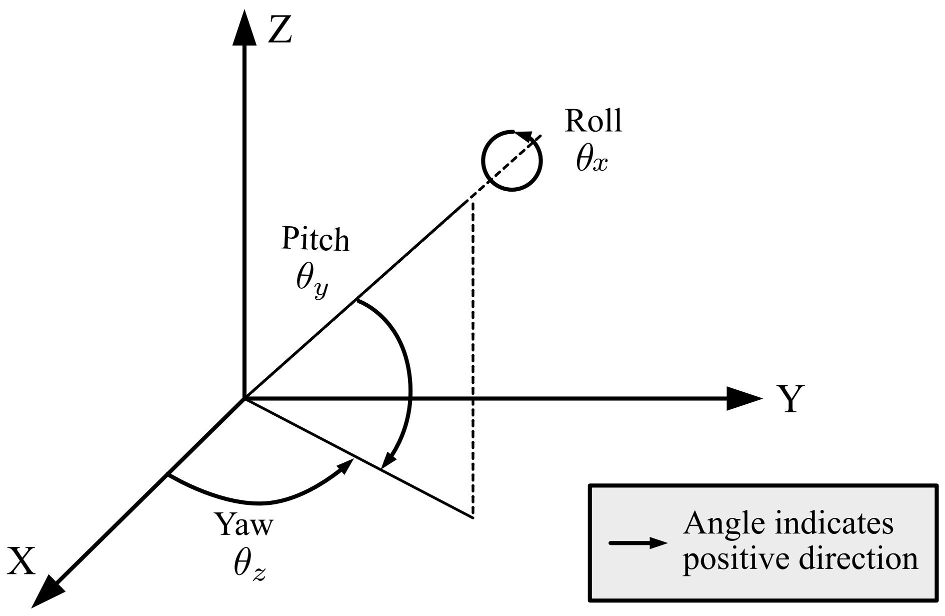


Figure 2.2: **Euler angles** define rotations with angles around the XYZ axes.

Each axis-aligned rotation of Euler angles can be represented as a rotation matrix:

$$\mathbf{R}(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.11}$$

$$\mathbf{R}(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}, \tag{2.12}$$

$$\mathbf{R}(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}. \tag{2.13}$$

The overall rotation $\mathbf{R}$ can be computed as the matrix multiplications of these rotation matrices:

$$\mathbf{R} = \mathbf{R}(\theta_z)\mathbf{R}(\theta_y)\mathbf{R}(\theta_x), \tag{2.14}$$

which represents the operations in the order of RPY-*unmodified*: right-to-left, $\mathbf{R}(\theta_x)$, then $\mathbf{R}(\theta_y)$, then $\mathbf{R}(\theta_z)$.

Euler angle gives the most compact representation of rotation with only 3 parameters that is also intuitively understandable (rotation angles around the axes); however, it has a critical disadvantage that makes it unsuitable for some applications. With Euler angles, there are cases where a unique Euler angle cannot be defined for a given 3D rotation. When pitch (θ_y) approaches $\pm 90^\circ$, a change in roll becomes a change in yaw (*degenerate case*). This means these rotations can all be represented by either the angles of yaw or roll, and there is no unique correspondence to the Euler angles representation. This non-uniqueness can be critical in some applications, such as pose prediction by a neural network, where we need another representation of rotation.

Inverse mapping from a rotation matrix to Euler angles is given as follows: Using the elements of rotation matrix $\mathbf{R} = [r_{ij}] \in \mathbb{R}^{3\times 3}$, pitch θ_y can be computed:

$$\theta_y = atan2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}). \tag{2.15}$$

Depending on the value of pitch θ_y, yaw θ_z and roll θ_x are computed as:

$$\theta_y = -90^\circ \quad \longrightarrow \quad \begin{cases} \theta_z = atan2(-r_{23}, -r_{13}) \\ \theta_x = 0 \end{cases} \tag{2.16}$$

$$\theta_y \neq \pm 90^\circ \quad \longrightarrow \quad \begin{cases} \theta_z = atan2(r_{21}, r_{11}) \\ \theta_x = atan2(r_{32}, r_{33}) \end{cases} \tag{2.17}$$

$$\theta_y = 90^\circ \quad \longrightarrow \quad \begin{cases} \theta_z = atan2(r_{23}, r_{13}) \\ \theta_x = 0 \end{cases} \tag{2.18}$$

handling the degenerate cases.

In this thesis, we use Euler angles to represent the relative robots' end-effector motions. In Chapter 4, we train a motion model that generates the 6DoF trajectory to extract objects from a pile. The waypoints of this trajectory are generated as relative motions from the previous waypoint. We use Euler angles to uniformly sample rotations along each XYZ axis with small angles, where Euler angles' degenerate cases do not happen.

2.2.3 Quaternion

Quaternion is another common parameterization of rotation. Similarly to Euler angles, a quaternion describes rotation with fewer parameters than a rotation matrix: 4 parameters q_w, q_x, q_y, q_z:

$$\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^\intercal \tag{2.19}$$

where $\mathbf{q} \in \mathbb{R}^4$ is a unit vector: $|\mathbf{q}|| = 1$.

A quaternion can be interpreted as an angle-axis representation, a rotation of $\alpha \in \mathbb{R}$ radians around the axis defined by unit vector $\mathbf{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^\intercal \propto \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^\intercal$:

$$q_w = \cos\frac{\alpha}{2} \tag{2.20}$$

$$\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^\intercal = \sin\frac{\alpha}{2} \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^\intercal . \tag{2.21}$$

Figure 2.3 depicts this interpretation of a quaternion in three-dimensional space.

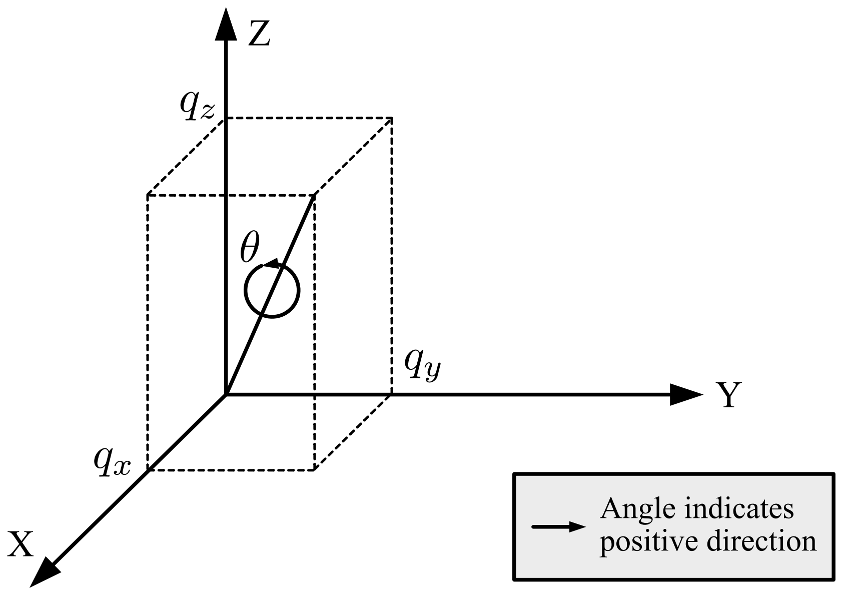


Figure 2.3: A quaternion defines rotations with angle rotation around an axis.

A rotation matrix **R** can be computed from quaternion **q** as the following:

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_wq_z) & 2(q_zq_x + q_wq_y) \\ 2(q_xq_y + q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_wq_x) \\ 2(q_zq_x - q_wq_y) & 2(q_yq_z + q_wq_x) & 1-2(q_x^2+q_y^2) \end{bmatrix}, \tag{2.22}$$

giving unique rotation matrix from a quaternion.

In the inverse mapping, a quaternion is extracted from a rotation matrix with an algorithm proposed by [Bar-Itzhack, 2000]. Given a rotation matrix $\mathbf{R} = [r_{ij}]$, this algorithm first forms a matrix **K** as follows:

$$K = \frac{1}{3}\begin{bmatrix} r_{11}-r_{22}-r_{33} & r_{21}+r_{12} & r_{31}+r_{13} & r_{23}-r_{32} \\ r_{21}+r_{12} & r_{22}-r_{11}-r_{33} & r_{32}+r_{23} & r_{31}-r_{13} \\ r_{31}+r_{13} & r_{32}+r_{23} & r_{33}-r_{11}-r_{22} & r_{12}-r_{21} \\ r_{23}-r_{32} & r_{31}-r_{13} & r_{12}-r_{21} & r_{11}+r_{22}+r_{33} \end{bmatrix}. \tag{2.23}$$

A quaternion is given by computing the eigenvector **v** that corresponds to the largest eigenvalue of this matrix **K**. Though this computation gives 2 solutions of quaternions that represents the same rotation matrix: $\mathbf{v}/||\mathbf{v}||, -\mathbf{v}/||\mathbf{v}||$, we can keep uniqueness by making the sign of v_0 be always positive:

$$\mathbf{q} = \operatorname{sign}(v_0)\frac{\mathbf{v}}{||\mathbf{v}||}. \tag{2.24}$$

This thesis thoroughly uses quaternions to represent the 6D poses of robots, objects and cameras, as compact and unique representation of rotations giving one-to-one mapping with rotation matrix.

2.3 Cameras

2.3.1 Sensor Device

The systems built in this thesis use RGB and depth images captured from an RGB-D camera. We specifically use the Realsense D435 camera [Keselman et al., 2017] in our real-world experiments, which captures depth using IR projection and stereoscopic with two IR receivers.

Table 2.2 shows the specifications of the camera provided by the manufacturer[1]. Although this camera can capture higher resolution images at higher frequency, we use 640×480 at 30 fps for both RGB and depth images to save computational resources for processing the captured images in vision pipelines. These RGB and depth images are synchronized and registered before use by the algorithms in our system. In this registration process, depth images are aligned with the RGB image captured at the same time-stamp, so that the depth value for each pixel of the RGB image is associated and can be acquired. In the experiments in this thesis, we use the camera parameters that are hard-coded on the device for this registration.

2.3.2 Pinhole Camera Model

We use the standard pinhole camera model (Figure 2.4) to model RGB-D cameras. The pinhole camera model describes the relation between 3D points in Euclidean space and their projection onto the image plane of a camera. With an ideal pinhole

[1] https://www.intelrealsense.com/depth-camera-d435/

Table 2.2: Specs of Realsense D435

RGB	Maximum resolution	1920×1080
	Field of view	$69^\circ \times 42^\circ$
	Maximum frame rate	30 fps
	Sensor technology	Rolling shutter
Depth	Maximum resolution	1280×720
	Field of view	$87^\circ \times 58^\circ$
	Maximum frame rate	90 fps
	Minimum distance	~ 28cm
	Depth accuracy	<2% at 2m

camera, the camera aperture is represented as a point and no lenses are used to focus light, which simplifies the modeling, e.g., geometric distortions, blurriness of unfocused objects.

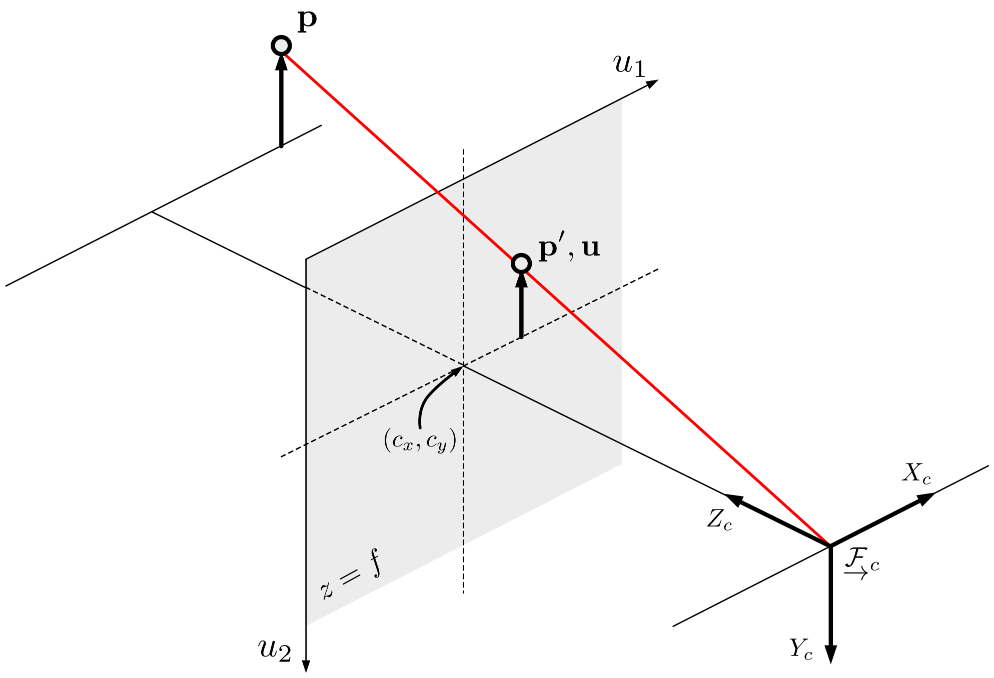


Figure 2.4: **Pinhole camera model** defines the projection of a 3D point $\mathbf{p} \in \mathbb{R}^3$ to the point on the 2D image plane $\mathbf{p'} \in \mathbb{R}^3, \mathbf{u} \in \mathbb{R}^2$.

The coordinate frame of the camera $\underrightarrow{\mathcal{F}}_c$ is defined with XYZ axes $[X_c \;\; Y_c \;\; Z_c]$. We set Z_c to face towards the image plane defining the perpendicular axis (*principal axis*) and its crossing point with the plane (*principal point*). Pixel coordinates are defined on the image plane with $[u_1 \;\; u_2]$, which are parallel to the XY coordinates of the camera frame: $[X_c \quad Y_c]$.

Let $\mathbf{p} = [x \quad y \quad z]^\intercal$ be a point in the 3D Euclidean space that is visible to the pinhole camera. This point is projected onto the image plane resulting in 2D pixel coordinates $\mathbf{u}$. To project the 3D point $\mathbf{p}$ onto the image plane as the pixel point $\mathbf{u}$, $\mathbf{p}$ is firstly projected into the image plane: $\mathbf{p}'$. Using the distance between the pinhole and principal point, the *focal length* f, $\mathbf{p}'$ can be described by $\mathbf{p} = [x \quad y \quad z]^\intercal$:

$$\mathbf{p}' = \begin{bmatrix} f\frac{x}{z} \\ f\frac{y}{z} \\ f \end{bmatrix} = f \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} = f\tilde{\mathbf{p}} \tag{2.25}$$

This projected point $\mathbf{p}'$ is then mapped to pixel coordinates:

$$\mathbf{u} = \begin{bmatrix} fk\frac{x}{z} + c_x \\ fl\frac{y}{z} + c_y \end{bmatrix} = \begin{bmatrix} f_x\frac{x}{z} + c_x \\ f_y\frac{y}{z} + c_y \end{bmatrix}. \tag{2.26}$$

Here, k, l denote the scaling between the metric ($\mathbf{p}$) and pixel ($\mathbf{u}$) spaces, accommodating the change of unit: meter to pixel, and c_x, c_y represent the offsets of the point in image coordinate space (as it is often the case, we use the origin of the image coordinate $(0, 0)$ as the left-top corner instead of the center). By definition, (c_x, c_y) also describes the pixel coordinates of the principal point.

The computation of the 2D pixel point $\mathbf{u}$ from the 3D point $\mathbf{p}$ can be also described only with matrix multiplication. Using homogeneous coordinates of $\mathbf{u}$, the mapping can be denoted:

$$\boldsymbol{u} = \begin{bmatrix} f_x\frac{x}{z} + c_x \\ f_y\frac{y}{z} + c_y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} = \mathbf{K}\tilde{\mathbf{p}}. \tag{2.27}$$

The four parameters f_x, f_y, c_x, c_y are called the *camera intrinsic parameters* and their combined matrix $\mathbf{K}$ is called *camera intrinsic matrix*.

The perspective projection defined above is not invertible as it collapses the z-dimension while converting a 3D point to 2D (i.e., all points on the same ray projected from 3D space end up at the same 2D point). However, given the depth map of the 2D image denoted $\mathbf{D}(\mathbf{u}) = z$, the collapsed z-dimension is recovered to invert the projection function:

$$\mathbf{D}(\mathbf{u})\mathbf{K}^{-1}\boldsymbol{u} = z \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \\ 1 \end{bmatrix} = z \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{p}. \tag{2.28}$$

2.4 Geometry

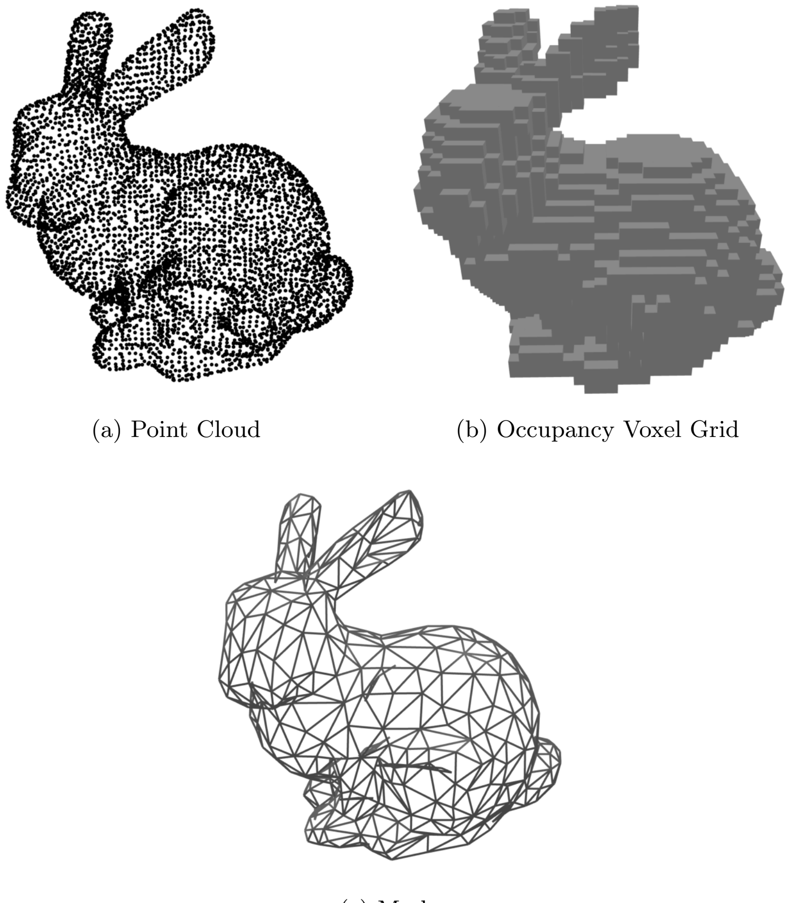
This thesis aims at building a 3D semantic maps of scenes, and the representation of 3D geometry crucially affects the performance and flexibility of map building and its utilization. In this section, we describe the 3 major representations we use: *point cloud*, *occupancy voxel grid*, and *mesh* (Figure 2.5).

A **point cloud** is a set of 3D points $\mathbf{p}_i \in P$, whose positions are represented with Cartesian coordinates:

$$\mathbf{p_i} = [x_i \quad y_i \quad z_i]^\top \tag{2.29}$$

Each point represents a point on the surface of an object, such as captured by depth sensors (e.g., depth camera; LIDAR) in real scenes or generated from 3D models in software. The point cloud example in Figure 2.5 shows the points sampled from the original bunny-shaped mesh model. Even though there are gaps between points, it represents the overall structure of the bunny model when visualized. Extra information such as color, label, or probability can be associated with each point to represent, for example, colored geometries captured with RGB-D sensors or point segmentations with semantic labels.

An **occupancy voxel grid** is a 3D tensor $\mathbf{V} = [v_{klm}]$ composed of constant-sized cubes (*voxels*). Each voxel stores a binary or real value representing the existence of objects (*occupancy*). The voxel grid example in Figure 2.5 shows a visualization of the voxels occupied by the bunny model in the grid. Compared to the point cloud representation, there are no longer holes on the surface; however, the fine-detailed surface of the original mesh is lost, while being kept to some extent by the point cloud representation. To update or query the value of each voxel, each 3D position


(a) Point Cloud (b) Occupancy Voxel Grid

(c) Mesh

Figure 2.5: Different representations of 3D geometry.

in Cartesian space $\mathbf{p}_i = [x_i \quad y_i \quad z_i]^\intercal$ is mapped into voxel coordinates (index of the matrix), keeping the spatial consistency. Using the voxel size s and grid origin coordinate $\mathbf{p}^o = [x^o \quad y^o \quad z^o]^\intercal$, the corresponding voxel coordinate $\mathbf{l}_i$ is computed:

$$\mathbf{l}_i = \begin{bmatrix} \frac{x_i - x^o}{s} \\ \frac{y_i - y^o}{s} \\ \frac{z_i - z^o}{s} \end{bmatrix} = \frac{\mathbf{p}_i - \mathbf{p}^o}{s} \in \mathbb{R}^3. \tag{2.30}$$

From these voxel coordinates of real-valued indices, integer indices are acquired as:

$$[k \quad l \quad m]^\intercal = \lfloor \mathbf{l}_i \rceil \in \mathbb{N}^3, \tag{2.31}$$

where $\lfloor x \rceil$ describes rounding of a real value into an integer. As in the to point cloud representation, each voxel can store extra information such as colors and semantics with the occupancy voxel grid representation.

A **mesh** is composed of a set of 3D points (*vertices*) and the connectivity, which defines a set of planes (*faces*) that represents the surface of the object. Let $\mathbf{p}_i \in P$ be vertices and $\mathbf{f}_j \in F$ be faces. Vertices are defined as Cartesian coordinates and faces are defined as 3 indices of vertices to define the triangles (face shapes can be any polygons, but we exclusively use triangle polygons in this thesis):

$$\mathbf{p}_i = [x_i \quad y_i \quad z_i]^\top, \tag{2.32}$$

$$\mathbf{f}_i = [k \quad l \quad m]^\top, \quad k, l, m = 1 \dots |P|. \tag{2.33}$$

The order of indices in $\mathbf{f}_i$ (clockwise, anti-clockwise) represents the front/back identity of the faces, and we follow the popular convention of 3D rendering software such as OpenGL, which treats anti-clockwise polygons as front-facing. The vertices of a mesh are similar to a point cloud representing 3D points on the surface of objects. The triangle faces fill the gaps between points that point cloud representations have. The mesh example in Figure 2.5 shows that clear definition of surfaces without seeing through the back surface (cf. point cloud) and fine-detailed surface geometry (cf. occupancy voxel grid).

As seen in the above, a point cloud and mesh are similar in representation, and we can see mesh representation as an upgrade of a point cloud by defining faces filling the gap among points. In our system, a point cloud is exclusively used for processing depth images after being captured by depth sensors. Once a mesh representation is acquired from the point cloud observation of a scene, we no longer convert the mesh model back into a point cloud.

Compared to the point cloud and mesh representations, an occupancy voxel grid has different properties. Whereas a point cloud and mesh represent only data on the surface (points and faces), an occupancy voxel grid also contains values inside and outside the objects: inside is occupied $v_{ijk} = 1$, outside is unoccupied $v_{ijk} = 0$. The actual surface of the object is defined as the boundary of these inside-outside values, typically as 0.5. This means that the surface is defined with the following *implicit equation* f(x) = 0:

$$\mathbf{V}(\mathbf{l}_i) = \mathbf{V}(\mathbf{p}_i) = 0.5 \tag{2.34}$$

$$\Rightarrow \quad \mathbf{V}(\mathbf{p}_i) - 0.5 = 0. \tag{2.35}$$

Therefore, an occupancy voxel grid is called an *implicit surface representation*, and in contrast, a point cloud and a mesh are called an *explicit surface representation*. Implicit surface representation allows operations that are difficult in explicit representation. For example, two voxel grids that have the same coordinate system can easily compute boolean operations such as conjunction and subtraction of these grids: $v^{+}_{klm} = v^{a}_{klm} \vee v^{b}_{klm}, v^{-}_{klm} = v^{a}_{klm} \wedge \neg v^{b}_{klm}$. Moreover, since an implicit representation has data points uniformly arranged in space, it can store extra information about these 3D locations such as labels of observed/unobserved areas, which is useful to differentiate confirmed-empty spaces from unobserved spaces, which are both represented as spaces with no data in point cloud and mesh representation.

Another significant advantage of voxel grids is the speed of data association. With Equation 2.30, a new 3D point $\mathbf{p}'$ can be promptly associated with a voxel $\mathbf{v}_{klm}$ to store extra data such as semantics or accumulated observations. A point cloud and mesh require nearest neighbor search to associate the new point $\mathbf{p}'$ with one of the points in the set $\mathbf{p}_i \in P$. This operation of finding the nearest points can take time, especially when the size of the point set becomes large, whereas association in voxel grids does not change in speed.

Although voxel grids have numerous advantages, they have several disadvantages. Firstly, voxel grids take large memory to store data compared to point clouds and meshes, as they also store data for non-surface volumes, which are the majority of space in most cases. This limitation can be relaxed to some extent by tree-based hierarchical representations of voxel grids that allow memory efficient storage using Octrees [Wurm et al., 2010, Riegler et al., 2017, Vespa et al., 2018], or by voxel hashing that only stores data for occupied voxels [Nießner et al., 2013, Kahler et al., 2015]. Also, as we have seen in the voxel association with rounding in Equation 2.31, a voxel grid representation requires discretization of data points, which sacrifices the accuracy of the data coordinate. One solution for this is reducing the voxel size s; however, because of memory constraints, this can only be done with some limits.

This thesis uses all of these representations in the systems of visual perception and motion generation, to exploit the advantages of each representation appropriately. Point clouds are mostly used in the early stage of the vision pipelines where depth information are processed; however, they are also used to keep the original precision of data points such as RGB-D-based pose prediction in Chapter 3. A voxel grid is used to in the middle stage of the vision pipeline after converting from a point cloud to accumulate depth observations for reconstruction and pose estimation from multi-view observations in Chapter 3. Mesh representations are used throughout this thesis, from data generation and simulation at training time to motion planning and visualization at test time.

2.5 Robots

2.5.1 Manipulator: Franka Emika Panda

In this thesis, we predominantly use the Franka Emika Panda robot (Figure 2.6) as the manipulator in the experiments. Panda is a 7DoF arm with torque sensors in all 7 joints, a payload of $3kg$, and a reach of $0.855m$. Its total weight is around $18kg$. The torque sensors installed on each joint allow adjustable stiffness and compliance of the arm with torque control. When the guiding button on the wrist is pressed, the arm will follow externally applied forces, allowing a user to teach motions to the robot intuitively. When controlled via software, if these torque sensors detect unexpected external forces, the robot can stop by itself to avoid critical damage to itself, objects, and people. Figure 2.6 shows our hardware setup, where we integrate the Panda robot with a suction gripper using a Dyson vacuum cleaner.

The Panda robot provides a control interface, the Franka Control Interface (FCI), which is composed of: *libfranka*, a C++ library that provides low-level control of the

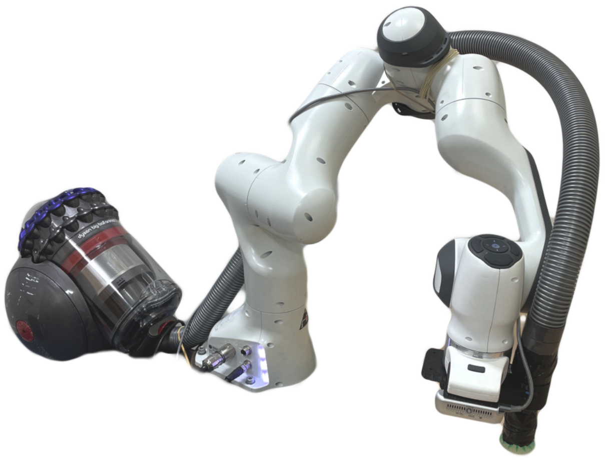


Figure 2.6: **Franka Emika Panda** robot that we use in our experiments with the custom suction gripper with a vaccum cleaner.

Panda, and *franka_ros*, a ROS [Quigley et al., 2009] integration package supporting ROS control [Chitta et al., 2017].

We use ROS control and its joint position controller to actuate the Panda robot in our experiments. This joint position controller receives a trajectory as a list of joint positions, velocities, and times, and controls the joint motors to follow the trajectory. We compute the joint position with forward/inverse kinematics and motion planning, and determine velocities and times based on the maximum velocity limits of the Panda robot. These constraints are provided officially from Franka Emika at https://frankaemika.github.io/docs/control_parameters.html.

2.5.2 Forward Kinematics

Forward kinematics are the equations of a robot to compute the coordinates of the end-effector (e.g., gripper tip) from specified joint positions:

$$\mathbf{r} = f(\boldsymbol{\theta}), \tag{2.36}$$

where $\mathbf{r} \coloneqq [x, y, z, \theta_x, \theta_y, \theta_z]^\top$ represents the end-effector pose, and $\boldsymbol{\theta}$ refers the vector of the positions of each joint; for an n-DoF robot, $\boldsymbol{\theta} \in \mathbb{R}^n$. We use forward kinematics mainly to acquire the end-effector pose after sampling of joint positions during motion planning and after reading them from the controller.

The forward kinematics for an n-DoF robot are computed as a composite of the transformations at each link. These transformations are traced from the base link of the robots and composed as matrix multiplications using homogeneous transformations (§2.2.1):

$$\mathbf{T} = \mathbf{T}_{l_0 l_n} = \mathbf{T}_{l_0 l_1} \mathbf{T}_{l_1 l_2} \dots \mathbf{T}_{l_{n-1} l_n}, \tag{2.37}$$

where l_0 represents the base link and l_n represents the end-effector. From $\mathbf{T} = [T_{ij}]$, the Euler angle elements $\theta_x, \theta_y, \theta_z$ in $\mathbf{r}$ can be acquired with the conversion equations described in §2.2, and x, y, z are equal to T_{14}, T_{24}, T_{34}.

To compute the transformation from link l_{i-1} to link l_i, let us define the rotation axis of the l_i as $\mathbf{n}_{l_i}$, the rotation angle as θ_i, and the translation as $\mathbf{t_i}$ with respect to the origin of the parent link l_{i-1}. Using these parameters, the homogeneous transformation matrix of l_i from l_{i-1} can be written as:

$$\mathbf{T}_{l_{i-1} l_i} = \begin{bmatrix} \mathbf{R}_\mathbf{n}(\theta_i) & \mathbf{t}_i \\ 0\ 0\ 0 & 1 \end{bmatrix}. \tag{2.38}$$

Here, $\mathbf{R}_\mathbf{n}(\theta_i)$ refers Rodrigues' rotation formula as follows:

$$\mathbf{R}_\mathbf{n}(\theta) = \mathbf{I} + \sin\theta \mathbf{n}^\times + (1 - \cos\theta)(\mathbf{n}^\times)^2. \tag{2.39}$$

2.5.3 Inverse Kinematics

Inverse kinematics is the inverse of forward kinematics to compute joint positions that correspond to a specified end-effector pose:

$$\boldsymbol{\theta} = f^{-1}(\mathbf{r}). \tag{2.40}$$

Unlike forward kinematics, which gives unique solutions, with n-DoF robots where $n > 6$ (6 DoF), this function can have multiple solutions for $\boldsymbol{\theta}$. We use inverse kinematics in this thesis to compute the joint positions to accomplish target grasp poses, and place poses of the robot during motion planning.

The inverse kinematics of n-DoF robots can be computed with an optimization process with Jacobian matrix $\mathbf{J}(\boldsymbol{\theta})$ of the forward kinematics function $f(\boldsymbol{\theta})$. The differentiation of the equation of forward kinematics (Equation 2.36) with respect to time gives:

$$\dot{\mathbf{r}} = \frac{\partial f}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}, \tag{2.41}$$

which represents the relation between the end-effector velocity and the joint velocities. If this Jacobian matrix is a regular matrix, this inverse relation can be acquired by simply computing inverse of the matrix, $\mathbf{J}^{-1}$:

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}(\boldsymbol{\theta})\dot{\mathbf{r}}, \tag{2.42}$$

from which θ is optimized to get $\mathbf{r}$ close to the target pose. However, generally the Jacobian matrix is not regular (especially $n > 6$), so the pseudo inverse $\mathbf{A}^{\#}$ is used instead to substitute the inverse matrix:

$$\mathbf{A}^{\#} = \begin{cases} \mathbf{A}^{-1} & (M = N = \mathrm{rank}(\mathbf{A})) \\ \mathbf{A}^{\top}(\mathbf{A}\mathbf{A}^{\top})^{-1} & (N > M = \mathrm{rank}(\mathbf{A})) \\ (\mathbf{A}^{\top}\mathbf{A})^{-1}\mathbf{A}^{\top} & (M > N = \mathrm{rank}(\mathbf{A})) \end{cases}, \tag{2.43}$$

for a matrix $\mathbf{A} = \mathbb{R}^{M\times N}$. When a robot have redundant joints $n > 6$, N is greater than M, so $\mathbf{J}^{\#} = \mathbf{J}^{\top}(\mathbf{J}\mathbf{J}^{\top})^{-1}$ is used. Using the pseudo inverse, the joint velocities can be written with modifying Equation 2.42:

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{\#}(\boldsymbol{\theta})\dot{\mathbf{r}}. \tag{2.44}$$

When a robot's joint configuration becomes close to a singularity ($\mathbf{J}(\boldsymbol{\theta})$ becomes not full-rank), pseudo inverse-based optimization becomes unstable while $|\dot{\boldsymbol{\theta}}|$ become large. Therefore, the SR-Inverse (singularity robust inverse) $\mathbf{J}^{*}$ [Nakamura and Hanafusa, 1986] is commonly used as an alternative to the pseudo inverse $\mathbf{J}^{\#}$:

$$\mathbf{J}^{*}(\boldsymbol{\theta}) = \mathbf{J}^{\top}(\mathbf{J}\mathbf{J}^{\top} + \epsilon\mathbf{I}), \tag{2.45}$$

with a damping constant ϵ to keep $\dot{\boldsymbol{\theta}}$ well-behaved near singularities.

We use SR-inverse-based optimization to solve inverse kinematics in this thesis.

2.6 Physics Simulation

Physics simulation recreates a real-world occurence by computer programs, called physics engine. A physics engine approximates certain physical processes such as collision detection, rigid body dynamics, soft body and fluid dynamics. Simulators are often classified as *real-time* or *high-precision* depending on their purpose and required precision. We adopt a real-time physics engine in this thesis for its speed. Real-time physics simulation is used throughout this thesis for data generation at training time and motion planning at test time.

We predominantly use the Bullet physics engine [Coumans et al., 2013]. Bullet features image rendering, rigid body simulations with gravity, friction, and contact forces, robot motor torques, and collision detection, and we use all of these in the systems built in this thesis.

Bullet requires convex-shaped mesh models for efficient collision detection, and if non-convex-shaped mesh models are given, it automatically treats their convex-hulls as the collision shape to compute collisions. For the general shaped object models including concavities, we use the V-HACD [Mamou and Ghorbel, 2009] algorithm to decompose them into a composite of convex shapes, *convex decomposition*. Figure 2.7 shows an example of convex decomposition using the power drill model from the YCB object dataset [Calli et al., 2015].

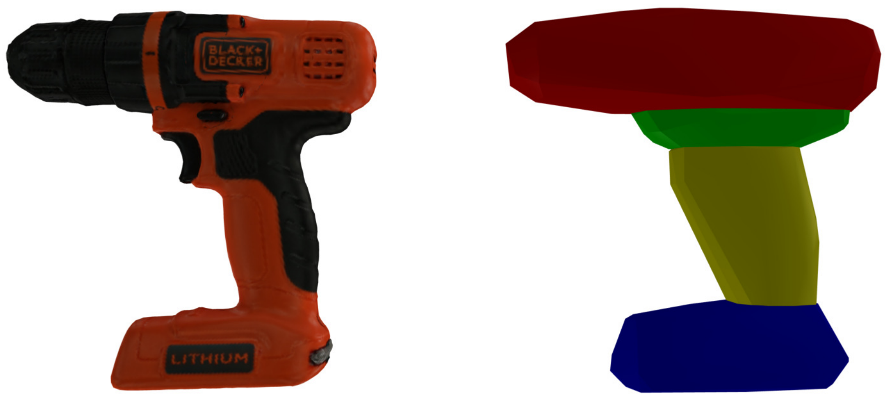


Figure 2.7: Convex decomposition of a mesh model for collision detection.

We use collision detection of physics engines to generate collision-free motions of the robot with motion planning of arm trajectory and grasp/place poses. To generate a collision-free trajectory, we use a widely-used motion planning algorithm, RRT-Connect [Kuffner and LaValle, 2000], integrating a sampling-based motion planning library OMPL [Sucan et al., 2012] with the physics engine. Figure 2.8 shows an example, in simulation, of a robot avoiding collision with a red wall to move a green cube from left to right in the workspace. In this case, the start and end joint configurations (left and right positions) are given and the motion planner generates the intermediate trajectory between them.

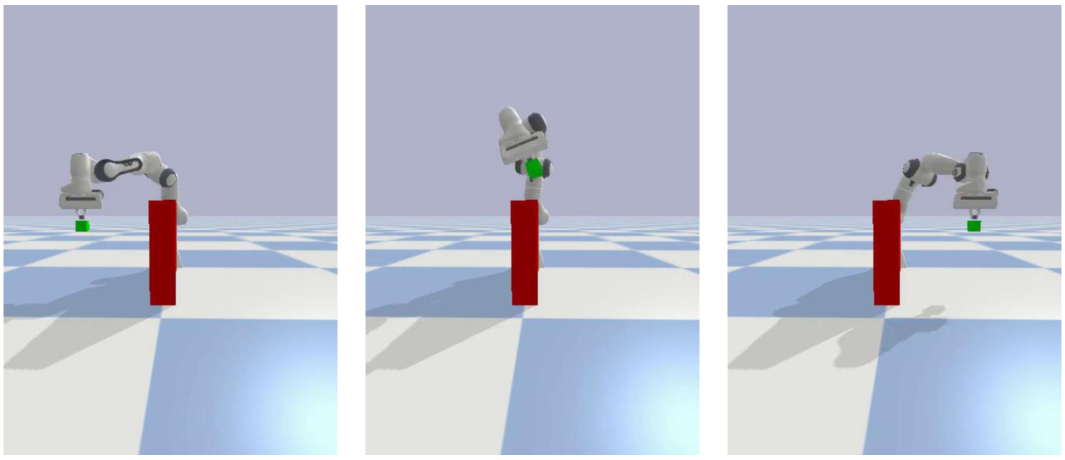


Figure 2.8: **Motion planning** of arm trajectory from one place to another without colliding the obstacle (red wall).

Rendering is also a crucial component of a physics engine to simulate vision-based robotics systems in the real world. Along with RGB and depth images, which correspond to the sensor information captured by an RGB-D camera in the real world, object ground truth masks can also be acquired in simulation. Figure 2.9 shows an example of rendering 640×480 images of RGB, depth, and object masks of objects in a pile, where object masks are overlaid representing different objects with different colors. In this thesis, we use such rendered RGB-D images and masks to train models for vision pipelines such as object detection and pose estimation, and motion planning such as grasp pose generation based on surface orientation and object visibilities.

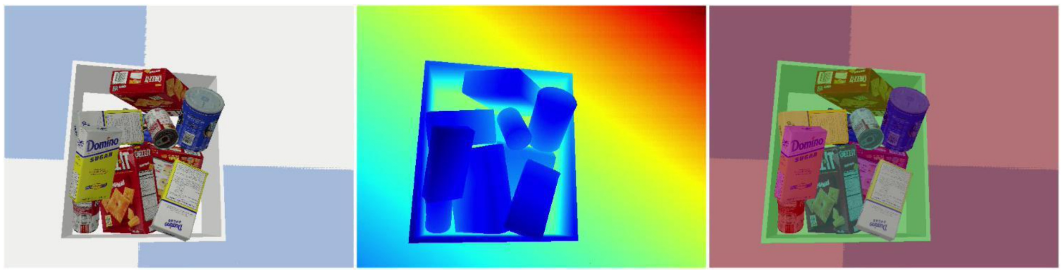


Figure 2.9: **Rendering** of RGB-D and object masks with a physics engine.

2.7 Deep Learning

Deep learning refers to machine learning methods that use artificial neural networks (ANNs). ANNs were inspired by the signal processing and distributed communication in biological neurons, where each neuron reacts to its inputs to output a new signal that is passed to the next neuron. The term "deep" in deep learning refers to the use of many layers, and often ANNs that have more than 3 layers are called deep neural networks (DNN).

2.7.1 Fully-Connected Layer

The most basic architecture of DNNs is a fully-connected layer. It computes an output $\mathbf{y}$ as the weighted sum of the input signals $\mathbf{x}$ with a shifting parameter bias $\mathbf{b}$, introducing a non-linearity with an activation function f to expand the expressiveness:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.46}$$

where $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{W} \in \mathbb{R}^{n \times m}$, $\mathbf{b} \in \mathbb{R}^n$. Common activation functions are: step functions, softmax functions, sigmoid functions, and rectified linear functions (ReLU). A simple feedforward neural network model is composed of sequential fully-connected layers:

$$\mathbf{x_{i+1}} = f_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \quad (i = 1 \dots l) \tag{2.47}$$

where l refers to the number of layers, and x_1 and x_n refer to the input and output of the network model. The layers of $1 <= i <= l - 1$ are called a *hidden layer*, and the one of $i = l$ is called a *output layer*.

The output of the hidden layer x_i is also called a *feature*. In hidden layers, ReLU is one of the most commonly used activation function f_i:

$$\texttt{ReLU}(\mathbf{z}) = \max_{\odot}(0, \mathbf{z}) = [\max(0, z_i)], \tag{2.48}$$

which computes an element-wise max, $\max_{\odot}$, comparing against 0 and zeroing the negative elements.

The output of the output layer $\mathbf{x}_l$ is used to compute an appropriate loss, with which the parameters of the network model $\mathbf{W}_i, \mathbf{b}_i$ are optimized to minimize the loss. For the output layer, an appropriate activation is selected based on the prediction target. The most basic activation is an identity function, which is often used for regression problems:

$$\texttt{Identity}(\mathbf{z}) = \mathbf{z}, \tag{2.49}$$

sigmoid function for binary classification:

$$\texttt{Sigmoid}(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}} \tag{2.50}$$

and softmax function for multi-class classification problems:

$$\texttt{Softmax}(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\sum_i e^{z_i}}. \tag{2.51}$$

A fully connected layer connects every signal of the input x_i to the output y_i (dense connection). When a deep neural network has many layers, this dense connection gives numerous combinations of output signals, achieving powerful expressiveness of the model. However, when the dimension of the input m_1 of $\mathbf{x_1} \in \mathbb{R}^{m_1}$ becomes large (e.g., images), fully connected layers have huge number of parameters of the dense matrix, $\mathbf{W}_1 \in \mathbb{R}^{m_2 \times m_1}$, causing problems with memory allocation and difficulties in parameter optimization. A naive solution would be to reduce dimensionality immediately at the initial layer, making m_2 smaller, but this approach restricts the expressiveness of the model by losing a large amount of information at the first layer.

2.7.2 Convolutional Layer

Convolutional layers [LeCun et al., 1998] mitigate the problem of fully-connected layers by introducing convolutional operations during the output computation:

$$\mathbf{y} = f(\mathbf{W} \circledast \mathbf{x} + \mathbf{b}), \tag{2.52}$$

where $\circledast$ refers to a convolutional operation. A convolutional layer treats the input $\mathbf{x}$ as a matrix (e.g., an image) instead of a vector, taking into account the spatial structure of images. It encapsulates the strong prior that natural images have self-similar structure at different pixel locations.

When an $h \times w$ image is given as the input, a fully-connected layer processes this as an hw-sized vector connecting every input of the pixels to every output for the next layer. A convolutional layer, however, treats this input as a $h \times w$ matrix and applies convolutions, computing the weighted sum locally, which restricts the region of computation with a restricted area of connection within the local window. The local window represented by a weight $\mathbf{W}$ is also called *kernel* and the window size is called *kernel size*. With a kernel size of k, the weight is 4D tensor $\mathbf{W} \in \mathbb{R}^{n \times m \times k \times k}$, mapping m-channel image-shaped features into n-channels.

Figure 2.10 depicts the convolutional operations with a 1-channel 8×8 image (e.g., a gray-scale image that contains pixel intensities) generating 1-channel output using a kernel sized 3×3. The kernel is slid across the input image by shifting the position. This sliding step is called the *stride*, and Figure 2.10 shows an example with a stride of 1. This sliding window operation is applied from top-left corner to the right-bottom corner, and makes the size of output smaller than the input image (in this case 8×8 to 6×6). To avoid this image size change, padding is often used where every corner of the image filled with zero (zero-padding). This padding operation turns the input image size from 8×8 to 10×10, so that the output size the same as the input image size, 8×8. In practice, the sliding-window operation of a convolutional layer is computed in parallel on a GPU, since each operation at different pixel locations is independent.

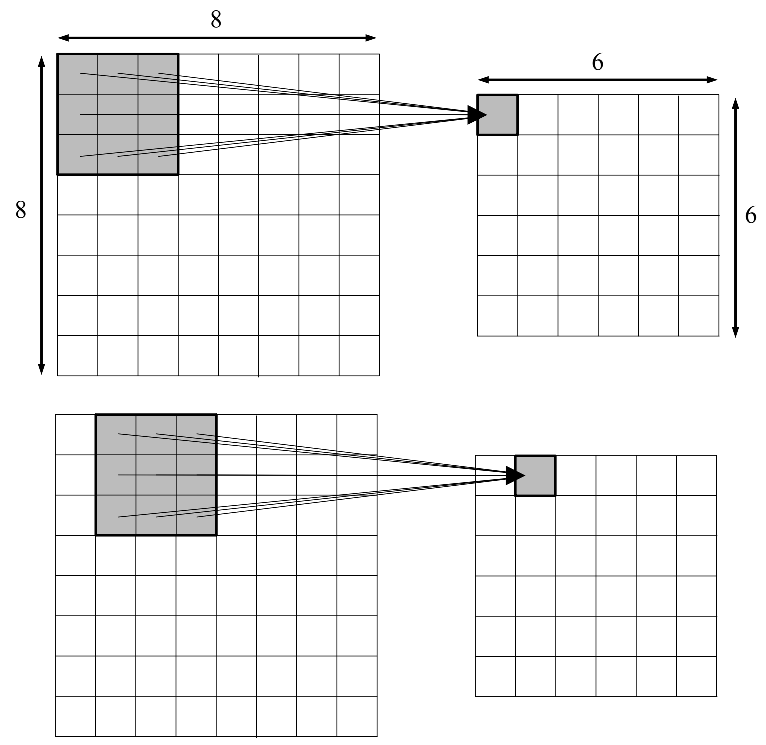


Figure 2.10: **Convolutional layer** that processes an 8×8 image to generate the 6×6 feature map with stride 1.

In the general case with m, n-channel images, the kernel computes matrix multiplication at each pixel and sums over it:

$$\mathbf{y} = f(\sum_{i}^{k} \sum_{j}^{k} \mathbf{W}_{ij} \mathbf{x}_{i'j'} + \mathbf{b}), \tag{2.53}$$

where i', j' refer to the index of the image pixel corresponding to the convolutional kernel (in the case of 3×3 kernel, this index of i', j' iterates the center of the sliding window and its 8 neighbors), and $\mathbf{y} \in \mathbb{R}^{n \times h \times w}, \mathbf{x} \in \mathbb{R}^{m \times h \times w}, \mathbf{x}_{i'j'} \in \mathbb{R}^{m}, \mathbf{W} \in \mathbb{R}^{n \times m \times k \times k}, \mathbf{W}_{ij} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^{n}$.

The convolutional layers reduce the number of parameters involved in a neural network compared to a fully-connected layer. With a $h \times w$, m-channel image given as input, a fully-connected layer treats this a vector with hwm dimension: $\mathbf{W} \in \mathbb{R}^{n \times hwm}$, which is often huge especially with high-resolution images. With a convolutional layer, however, the image structure $h \times w$ is kept while producing the feature map $\mathbf{y}$, and $\mathbf{W}$ is reused across images at different positions allowing the weight parameter size to be small: $\mathbf{W} \in \mathbb{R}^{n \times m \times k \times k}$, ($k$ is often chosen from $k = 3, 5, 7$). This small number of parameters compared to a fully-connected layer allows convolutional layers to be faster and easier to train even when many layers are stucked to construct deep architectures to increase the expressiveness with the parameter combinations.

Pooling is widely used with convolutional layers to build a neural network. A pooling layer reduces the spatial dimensions of the input to build a condensed feature map. A pooling operation is similar to convolution, but does not have any learning parameters (weights) and processes the input values with deterministic operations to compute outputs such as max for *max-pooling* and average for *average-pooling*. These pooling layers also have kernel sizes similarly to convolutional layers and apply their deterministic operations inside the kernel. The stride is selected appropriately to reduce image dimensions, for example, the size of the feature map is halved with a stride of 2. The kernel size is chosen to be larger than or equal to the stride to cover the whole image during the operation such as kernel size 2×2 with stride 2 and kernel size 4×4 with stride 3 (kernel size 2×2 with stride 3 skips some pixels).

Upsampling is the inverse layer of pooling. It uses deterministic operations such as bilinear interpolation to resize feature maps. Similarly to pooling operations, upsampling is combined with convolutional layers, and is used to restore the original resolution of an image after downsampling by pooling layers. Upsampling layers are usually used in the later stages of a neural network model. This image-form output is often used for pixel-wise regression and classification such as optical flow [Ilg et al., 2017, Sun et al., 2018] and semantic segmentation [Ronneberger et al., 2015, Zhao et al., 2017].

When building a network architecture with many layers, convolutional layers and pooling layers can be combined with fully-connected layers to predict vector-shaped outputs. After several pooling operations for downsampling and dimensional reduction, the resolution of feature map can be small enough to vectorize and process with fully-connected layers. From the vectorized features, these fully-connected layers predict outputs such as regression values or classification probabilities with appropriate activations.

We use the above-mentioned layers to build neural network models throughout the thesis. The convolutional and pooling layers in our model extract features from 2D images such as RGB and depth images for visual object recognition (object detection; pose estimation) in Chapter 3 and motion generation in Chapter 4, Chapter 5. We also use 3D convolutional layers to process voxel grids to extract 3D features to object poses in Chapter 3, and use fully-connected layers to predict the score of motions, from which appropriate ones are selected according to the scene states for motion generation in Chapter 4 and Chapter 5.

2.7.3 Gradient-based Optimization

The learnable parameters of neural network layers (weights and biases) are optimized to minimize a certain loss function. For this optimization, *gradient descent* is widely used as a first-order optimization algorithm, which iteratively steps the parameters in the negative direction of the gradient with respect to the loss function to find the minimum of the function.

With a loss function $\mathcal{L}(\theta)$ and parameters $\mathbf{W}_i, \mathbf{b}_i \in \theta$, the goal of this optimization is to update the parameters θ in the direction that reduces the loss to minimize $\mathcal{L}$:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{L}}{\partial \theta_t}, \tag{2.54}$$

where $t, t+1$ refer to training steps, and α is the *learning rate*, which defines the size of the steps.

When computing the gradient of the loss function $\frac{\partial \mathcal{L}}{\partial \theta_t}$, there are variations of how many training examples are used. Standard gradient descent uses all training examples in the dataset. Although this operation gives the true gradient, it is usually slow to converge when the dataset size becomes large, and prone to local minima where the gradient becomes 0.

Stochastic gradient descent (SGD) and *mini-batch gradient descent* are used to mitigate this problem. In SGD, a single training example is randomly selected from the dataset to compute the gradient. The stochasticity of how close the computed gradient is to the true gradient makes the optimization more robust to local minima. Despite these advantages, SGD restricts the computation of each step to a single example, not benefitting from parallel computation. Mini-batch gradient descent is in-between of the two approaches of gradient descent and SGD, and it uses several examples (a mini-batch) to compute the gradient. The size of this mini-batch (*batch size*) is selected to be small enough to have the benefit of stochasticity and large enough to maximize the benefit of parallel computation. Since the difference between mini-batch stochastic gradient and SGD is just the batch size and the batch size is often selected to be much smaller than dataset size, mini-batch gradient descent is often also considered as SGD. Later on we use the term SGD to refer to both methods.

The gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ can be interpreted as the velocity of a ball on the loss curve, and momentum is also used as a physical analogue to smooth noisy gradients and better move the parameters with the accumulation of past gradients:

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1)\frac{\partial \mathcal{L}}{\partial \theta_t} \tag{2.55}$$

$$\theta_{t+1} = \theta_t - \alpha v_{t+1}, \tag{2.56}$$

where, β_1 specifies exponential decay of momentum, which is often set to 0.9. This variant is called *Momentum SGD.*

Another approach to smooth the noisy gradient is called *RMSProp.* Whereas Momentum SGD modifies the gradient with accumulation, RMSProp adjusts the learning rate based on the gradient:

$$s_{t+1} = \beta_2 s_t + (1 - \beta_2)\frac{\partial \mathcal{L}}{\partial \theta_t}^2 \tag{2.57}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{s_{t+1} + \epsilon}}\frac{\partial \mathcal{L}}{\partial \theta_t}, \tag{2.58}$$

where β_2 refers to the exponential decay of the learning rate scalar, and ϵ is a small scalar to avoid division by zero.

Adam [Kingma and Ba, 2015] is a method that combines Momentum SGD and RMSProp to handle noisy gradients:

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \theta_t} \tag{2.59}$$

$$s_{t+1} = \beta_2 s_t + (1 - \beta_2) \frac{\partial \mathcal{L}}{\partial \theta_t}^2 \tag{2.60}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{s_{t+1} + \epsilon}} v_{t+1}, \tag{2.61}$$

taking advantages of both methods.

In this thesis, we predominantly use Adam as the optimizer for training neural network models in our system.

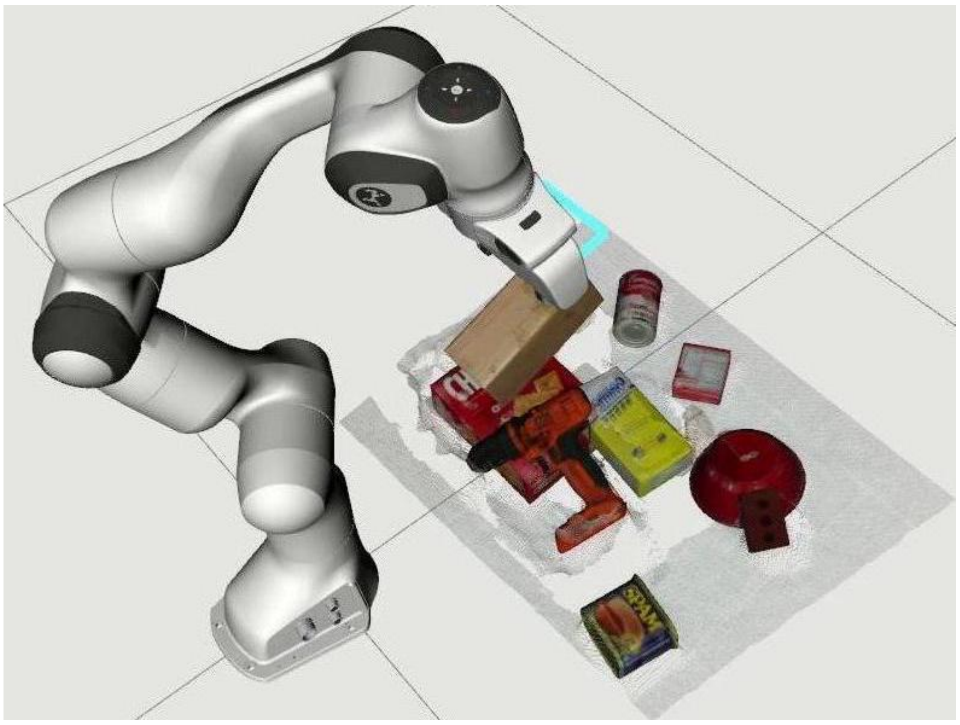
Chapter 3

Object-Level Semantic Mapping for Manipulation

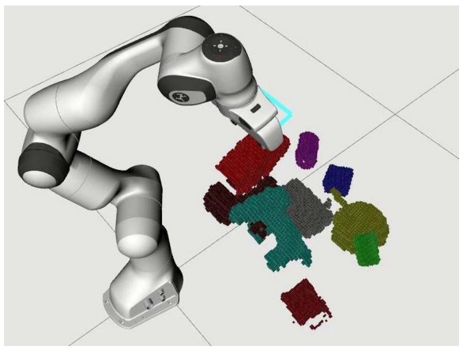
Contents

3.1 Introduction

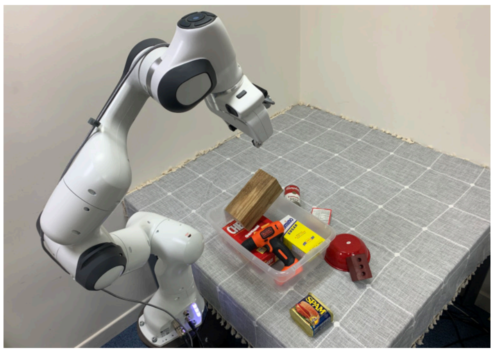
As discussed in Chapter 1, the ability to build semantic world models using on-board cameras is vital for robots to complete long-horizon manipulations and varied tasks.



(a) Pose Estimation



(b) Volumetric Fusion



(c) Real Scene

Figure 3.1: **MoreFusion**, our object-level semantic mapping system, produces accurate 6D object pose predictions by explicitly reasoning about occupied and free space via a volumetric map.

A complete semantic world model composed of object CAD models allows manipulations in complex scenes (e.g., object piles) enabling reasoning about inter-object geometric relationships (e.g., contact, support) and object-specific manipulations (e.g., desired grasp points, place poses). However, inferring object configurations in cluttered environments such as piles (Figure 3.1) is challenging even with state-of-the-art methods for object detection [Li et al., 2017, He et al., 2017] and pose estimation [Xiang et al., 2018, Wang et al., 2019] due to the limited visibility of objects that are mutually occluding and in contact. In such situations, it is crucial for the vision system to exploit multi-view observations and the geometric relationships among objects, which are often not used in previous work that focuses on single-view and per-object pose estimation.

In this chapter, we present a vision system that tackles semantic mapping of objects (object-level semantic mapping) in such challenging scenes, producing a persistent 3D multi-object representation from multi-view images. Our system, ***MoreFusion***, has four main components, as highlighted in Figure 3.2: 1) 2D object detection is fed to object-level fusion to make a volumetric occupancy map of objects. 2) A pose prediction network that uses RGB-D data and the surrounding occupancy voxel grid makes 3D object pose estimates. 3) Collision-based pose refinement jointly optimizes the poses of multiple objects with differentiable collision checking. 4) The intermediate volumetric representation of objects is replaced with information-rich CAD models. This system runs in real-time using a single RGB-D camera.

Our system takes full advantage of depth information and multiple views to estimate mutually consistent object poses. The initial rough volumetric reconstruction is utilized for neural network-based pose prediction and collision-based pose refinement by using the knowledge of these objects' surrounding geometry as occupancy information. This visual capability to infer the poses of multiple objects with occlusion and contact enables robotic planning for pick-and-place in a cluttered scene. An example of this is removing obstacle objects for picking the target red box in Figure 3.1.

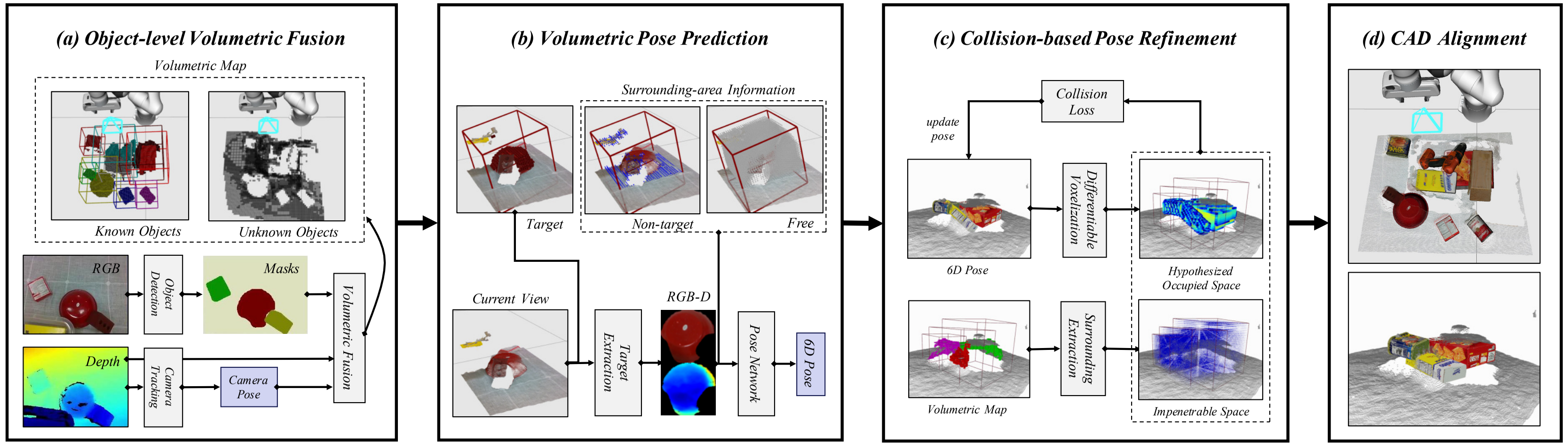


Figure 3.2: **Our 6D pose estimation system.** Object segmentation masks from RGB images are fused into a volumetric map, which denotes both occupied and free space (a). This volumetric map is used along with RGB-D data of a target object crop to make an initial 6D pose prediction (b). This pose is then refined via differentiable collision checking (c) and then used as part of a CAD alignment stage to enrich the volumetric map (d).

In summary, the following contributions are made in this chapter:

- **Pose prediction with surrounding spatial awareness**. A neural network-based model uses occupancy voxel grids as impenetrable space during object pose prediction;
- **Joint optimization of multi-object poses**, in which the scene configuration with multiple objects is evaluated and updated by gradient-based optimization with differentiable collision checking;
- **Full integration of semantic mapping of objects** in a real-time system, in which the object-level volumetric reconstruction is exploited for incremental and accurate pose estimation.

3.2 Related Work

Template-based methods are one of the main approaches to pose estimation. Traditionally, these methods involve generating templates by collecting images of the object from varying viewpoints in an offline training stage and then scanning the template across an image to find the best match using a distance measure [Huttenlocher et al., 1993, Steger, 2001]. These methods are sensitive to clutter, occlusions, and lighting conditions, leading to many false positives, which in turn require post-processing. LINEMOD [Hinterstoisser et al., 2011, Hinterstoisser et al., 2012a] is a template-based method that generates templates by combining silhouette gradient orientations from RGB images and surface normal orientations from depth images — improving the detection of texture-less objects in cluttered scenes. Extensions of this method involve the use of 3D models to generate many templates of the objects from different viewing angles [Hinterstoisser et al., 2012b], as well as an effort to increase their speed using a cascade framework [Rios-Cabrera and Tuytelaars, 2013].

Sparse feature-based methods have been a popular alternative to template-based methods for many years [Lowe, 2001, Nister and Stewenius, 2006, Philbin et al., 2007]. These methods extract scale-invariant points of interest from images, describing them with local descriptors such as SIFT [Lowe, 2004] or SURF [Bay et al., 2008], and then storing them in a database to be later matched at test time to obtain a pose estimate using a method such as RANSAC [Fischler and Bolles, 1981]. This processing pipeline can be seen in manipulation systems such as MOPED [Collet et al., 2011], where structure from motion is used to merge the information from each training image into a sparse 3D model, which is used to obtain the pose based on 3D point-to-point correspondences between a stored model and the test model. MOPED has inspired further work, such as the addition of depth information at multiple stages in the pipeline [Tang et al., 2012]. Recently, instead of using hand-crafted features, they have been learned through examples [Holzer et al., 2012, Rosten et al., 2008]. As these methods look for features in the input data, they assume that objects have sufficient texture, leading to poor performance or failure on texture-less objects.

The use of deep neural networks is now prevalent in the field of 6D pose estimation. PoseCNN [Xiang et al., 2018] was one of the first systems to train an end-to-end system to predict an initial 6D object poses directly from RGB images, which was then refined using depth-based ICP [Besl and McKay, 1992]. Recent RGB-D-based system includes PointFusion [Xu et al., 2018] and DenseFusion [Wang et al., 2019], which individually process the two sensor modalities (convolutions for RGB, PointNet [Qi et al., 2017] for point-cloud), and then fuse them to extract pixel-wise dense feature embeddings. Our work is most closely related to these RGB-D and learning-based approaches with deep neural networks. Unlike prior work that tackles pose estimation with per-object pose prediction using a point cloud, our method predicts each object's pose by utilizing a more structured volumetric representation that also incorporates the geometric information of surrounding objects, allowing inter-object consistency reasoning.

3.3 Method

Our system estimates the 6D pose of a set of known objects given RGB-D images of a cluttered scene. Each object has an associated mesh model with complete geometry and texture (CAD model), and the goal of the system is to estimate the transformation between the world model and the CAD model. We represent 6D poses as a homogeneous transformation matrix $\mathbf{T} = [\mathbf{R}|\mathbf{t}] \in SE(3)$, where $\mathbf{R} \in SO(3)$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector.

The system, summarized in Figure 3.2, can be divided into four key stages. (1) An **object-level volumetric fusion** stage that accumulates depth measurements with the object instance masks predicted by object detection along with camera tracking to produce a persistent volumetric map of known and unknown objects. (2) A **volumetric pose prediction** stage that uses the surrounding geometric information from the volumetric map along with the RGB-D and object's masks to produce an initial pose prediction for each of the objects. (3) A **collision-based pose refinement** stage that jointly optimizes the pose of multiple objects via gradient descent using differentiable collision checking between object CAD models and the occupancy voxel grids extracted from the volumetric map. (4) A **CAD alignment** stage that replaces the intermediate voxel grid reconstruction of each object with a CAD model, which contains more rich information in a compact representation (matrix of voxel grid vs. integer ID of the CAD model). In the following sections, we describe the details of each of these stages.

3.3.1 Object-level Volumetric Fusion

Building a volumetric map is the first stage of object-level semantic mapping, allowing the system to gradually increase its knowledge about the scene until it achieves confident poses of the detected objects. For volumetric fusion, we build a pipeline similar to previous work [McCormac et al., 2018, Sünderhauf et al., 2017, Xu et al., 2019], combining RGB-D camera tracking, object detection, and depth-based volumetric reconstruction.

RGB-D Camera Tracking

Given that the camera is mounted on the end of a robotic arm, we can retrieve the accurate pose of the cameras using forward kinematics (§2.5.2) and the well-calibrated parameters of the camera: intrinsic parameters for the projection (§2.3.2), and extrinsic parameters for transformation between the camera and the robot arm. However, to also allow our system to be used with a hand-held camera, we adopt the sparse SLAM framework ORB-SLAM2 [Mur-Artal and Tardós, 2017] for camera tracking. Unlike its monocular predecessor ORB-SLAM [Mur-Artal and Tardós, 2014], ORB-SLAM2 allows RGB-D input and tracks camera pose in metric space, providing the accurate scale of the map and camera positions. This is crucial for the subsequent volumetric reconstruction stage where we accumulate depth measurements into a 3D voxel grid in metric space.

Object Detection

Following prior work [McCormac et al., 2018, Xu et al., 2019], we use the state-of-the-art object detection system, Mask R-CNN [He et al., 2017] to acquire object masks from RGB images. Mask R-CNN receives an RGB image as input and processes it with convolutional layers to extract features, which are processed by another convolutional layer to predict an initial guess of object bounding boxes with object existence probabilities. These initial guesses are used as the region of interest (ROI) to extract the features only inside the region with a pooling operation called ROI-Align. These pooled features are further processed to predict object classes, refined bounding boxes, and object masks with subsequent fully-connected and convolutional layers.

We train this object detection model to detect the YCB objects [Calli et al., 2015] using existing datasets [Wada et al., 2020, Xiang et al., 2018], which are a mix of real and synthetic images. Although this model is fairly accurate, it can give false positives with a low threshold of detection confidence, so we use a relatively high confidence threshold of 75%. This high threshold favors false negatives in detecting objects; however, the multi-view prediction allows the model to run on multiple frames, and eventually finds objects with a confident prediction.

Volumetric Reconstruction

We use volumetric occupancy reconstruction to track objects and accumulate depth observations in different views. For efficient accumulation, we use the Octree-based occupancy volumetric fusion framework, OctoMap [Hornung et al., 2013]. Its Octree structure allows us to efficiently query the associated voxel $v \in \mathbf{V}$ for a query 3D point from new observation z_t, and the occupancy probability of the voxel $p(v|z_{1:t})$ at time t is updated with new observation $p(v|z_t)$ with a Bayesian update:

$$\begin{aligned} \operatorname{logit}(v|z_{1:t+1}) &= \operatorname{logit}(v|z_{1:t}) + \operatorname{logit}(v|z_t) \\ \text{where} \;\; \operatorname{logit}(p(x)) &= \log(\frac{p(x)}{1-p(x)}). \end{aligned} \tag{3.1}$$

As we collect the observations z_t from the moving camera, we fuse the new observations into the surface reconstruction, filtering sensor noise and building an accurate reconstruction of the objects.

To associate a 3D point from the observation z_t with the corresponding voxel $v^o \in \mathbf{V}^o$ of an object o, we have two tracking mechanisms for camera and object instances. Camera tracking provides the transformation from the camera to the map frame, and in our system, this is given by either the forward kinematics of the robotic arm or an external RGB-D SLAM system as mentioned before. Object instance tracking gives the corresponding voxel grid of an object instance in which the new depth observation is accumulated. Object tracking is achieved via comparisons of the 2D binary masks of the detected masks $\mathbf{D}_i$ and rendered masks $\mathbf{M}^o$ of the mapped objects. The detected masks $\mathbf{D}_i$ are obtained by object detection and the rendered masks $\mathbf{M}^o$ are given by the projection of the voxel grid of each mapped object. Comparison is accomplished by computing the intersection over the union (IoU) between these masks:

$$\mathrm{IoU}_i^o = \frac{\mathbf{M}^o \wedge \mathbf{D}_i}{\mathbf{M}^o \vee \mathbf{D}_i}. \tag{3.2}$$

When this metric (IoU_i^o) is over a threshold of 0.4, D_i is recognized as another observation of the object o and accumulated into its voxel grid $\mathbf{V}^o$. Otherwise, a new voxel grid is initialized to reconstruct a new instance of an object.

We use this object-level volumetric reconstruction as an intermediate representation, which only contains information about the visible surface of an object. In the later stages of the system pipeline, this representation will be replaced by a CAD model, which is much more compact and includes complete geometric information (e.g., the back surface).

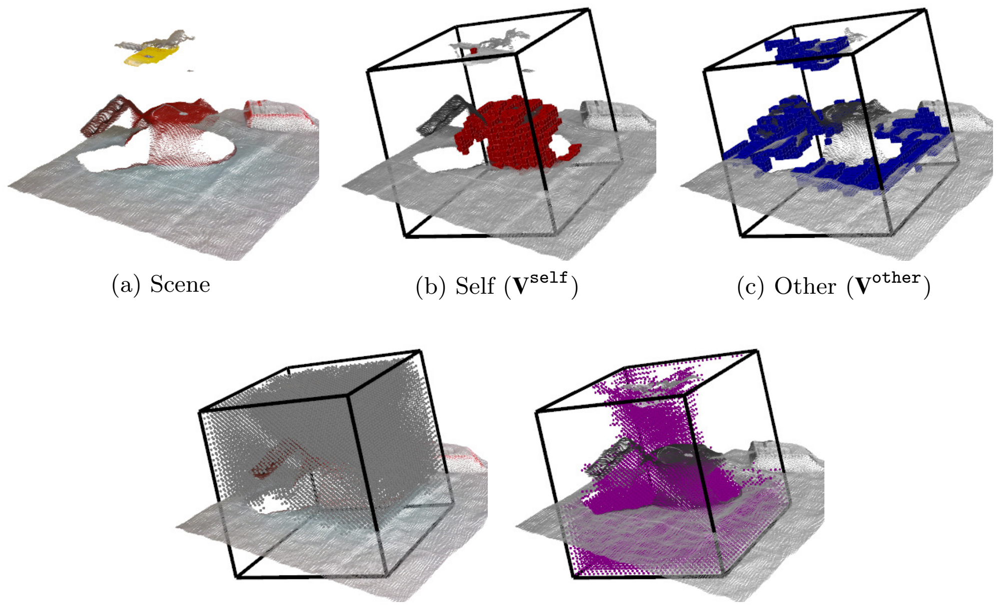
3.3.2 Volumetric Pose Prediction

Our system retrieves surrounding geometry from the volumetric map assist to the pose prediction of the objects. The surrounding geometry is extracted and represented as occupancy voxel grids, and is fed into the pose prediction model to predict a consistent pose estimate by incorporating spatial awareness of the surrounding objects. In this section, we describe how this surrounding information is represented and used in pose prediction.

Occupancy Voxel Grids as Surrounding Information

Each target object for pose prediction carries its occupancy voxel grid. The voxels that make up this grid can be in one of the following states as shown in Figure 3.3: (1) Space occupied by the object itself, $\mathbf{V}^{\texttt{self}}$ from the target object reconstruction, where depth measurements are accumulated. (2) Space occupied by other objects, $\mathbf{V}^{\texttt{other}}$ from the volumetric reconstruction of the surrounding objects. (3) Free space $\mathbf{V}^{\texttt{free}}$ where depth rays have passed through and there were no depth measurements. (4) Unknown space $\mathbf{V}^{\texttt{unknown}}$, unobserved during the volumetric mapping because of occlusions or sensor range limits.

Ideally, the bounding box of surrounding information should cover the whole volume of the target object even if it is occluded. This means the bounding box size should change depending on the target object size. Since we need to use a fixed voxel dimension for neural network-based prediction (e.g., $32 \times 32 \times 32$), we use a different voxel sizes for each object. Given the CAD model of each object, we compute the


(a) Scene (b) Self ($\mathbf{V}^{\texttt{self}}$) (c) Other ($\mathbf{V}^{\texttt{other}}$)

(d) Free ($\mathbf{V}^{\texttt{free}}$) (e) Unknown ($\mathbf{V}^{\texttt{unknown}}$)

Figure 3.3: **Surrounding spatial information.** These figures show the occupancy grid ($32 \times 32 \times 32$ voxels) of the red bowl. The free (d) and unknown (e) grids are visualized with points instead of cubes for clarity.

diagonal of its bounding box, which defines the maximum dimension of the object. The voxel size is set to be the length of the diagonal divided by the voxel dimension.

Pose Prediction Network

The initial 6D pose of each object is predicted via a deep neural network that accepts both the occupancy voxel grids (§3.3.2) and masked RGB-D images. The architecture is summarized in Figure 3.4, and can be categorized into 4 core components: (1) 2D feature extraction from RGB using a ResNet; (2) Point-wise encoding of RGB features and point cloud; (3) Voxelization of the point-wise features followed by 3D convolutions; (4) Point-wise pose prediction from both 2D and 3D features.

2D Feature Extraction from RGB Even when depth measurements are available, RGB images still carry vital sensor information for precise pose prediction. With ther color and texture detail, RGB images can be an especially strong signal for the pose prediction of highly-textured objects. Moreover, for geometrically sym-

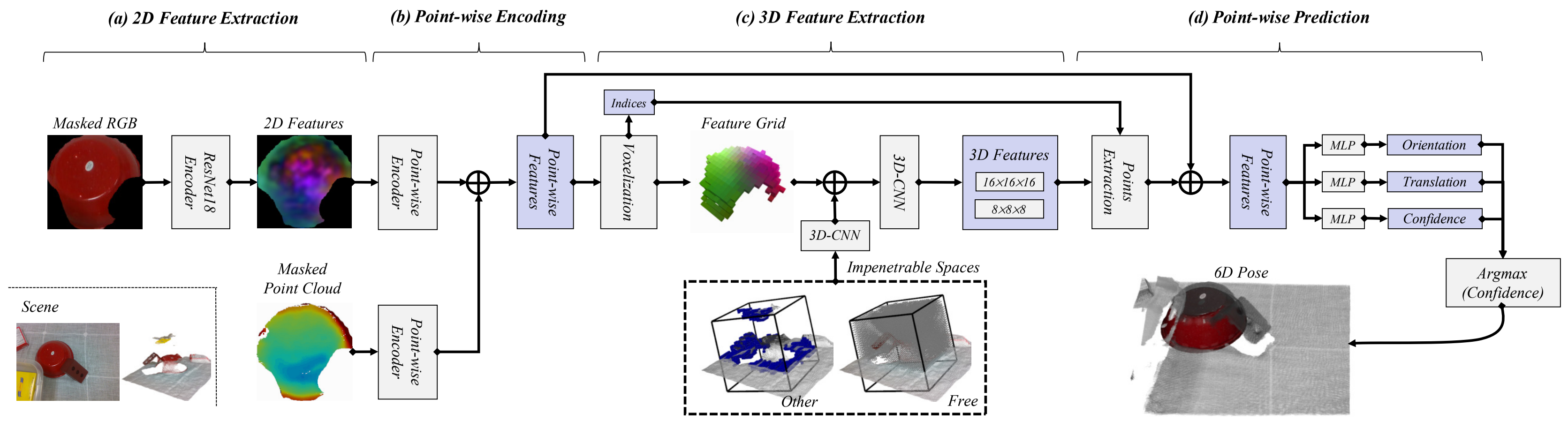


Figure 3.4: **Network architecture**for pose prediction using masked RGB-D images of the target object and surrounding information represented by occupancy grids.

metric objects, this texture information is crucial for resolving the ambiguity of the object poses, which cannot be accomplished only with depth observations.

Following [Wang et al., 2019, Xu et al., 2018], we use a ResNet18 [He et al., 2016] with added upsampling layers [Zhao et al., 2017] to extract features from masked RGB images. Though both prior methods [Wang et al., 2019, Xu et al., 2018] used cropped images of objects with a bounding box, we used masked images which makes the network invariant to changes in background appearance, and also encourages it to focus on retrieving surrounding information using the occupancy grids. For feature extraction, the original input image is first downsampled by 8 and then upsampled to the original image size while downscaling the channel size from 512 to 32. Unlike [Zhao et al., 2017], we resize the input image to the same size: 256×256, for faster batch inference.

Point-wise Encoding of RGB Features and Point Cloud Similarly to previous work on RGB-D pose estimation [Xu et al., 2018, Wang et al., 2019], both the RGB features and extracted point-cloud points (using the target object mask) are encoded via several fully connected layers (similar to the PointNet architecture [Qi et al., 2017]) to produce point-wise features, which are then concatenated.

The RGB features and point cloud increase the channel size with 2 fully connected layers for each (RGB features: $32 \to 64 \to 128$, point cloud: $3 \to 8 \to 16$). We keep the same channel size as [Wang et al., 2019] for encoding RGB but reduce the channel size for point cloud encoding. This is because, in contrast to a PointNet-based approach [Wang et al., 2019], our 3D-convolution-based approach provides the location information for each point via the volumetric structure of the feature voxel grid. High dimensional encoding would be unnecessarily redundant.

After this separated encoding of RGB and point cloud point-wise features, we concatenate them at the first $\oplus$ from the left in Figure 3.4. From each output from two fully connected layers, concatenation gives two point-wise feature vectors which have $72(= 64 + 8)$ and $144(= 128 + 16)$ channels respectively.

Voxelization and 3D Convolutional Processing From these point-wise features, we build a feature grid by averaging the features associated with the same voxels. Having the same dimensions as the occupancy grid, this feature volume can be combined with the occupancy grid extracted from volumetric fusion. The concatenated voxel grid is processed by 3D convolutional layers to extract hierarchical 3D features, reducing the voxel dimension and increasing the channel size. We process the original grid (voxel dimension: 32) with 2-stride convolutions to produce hierarchical features (voxel dimension: 16, 8).

An important design choice in this pipeline is to perform 2D feature extraction before voxelization, instead of directly applying 3D feature extraction on the voxel grid of raw RGB pixel values. Though 3D convolutions and 2D convolutions have similar behavior when processing RGB-D input, it is hard to use 3D convolutions on a high-resolution grid, unlike a 2D image, and also the voxelized grid can have more missing points than an RGB image because of sensor noise in the depth image.

Point-wise Pose Prediction from 2D-3D Features To combine the 2D and 3D features for pose prediction, we extract points from the 3D feature grid that correspond to the point-wise 2D features with trilinear interpolation. During this interpolation, the original locations of 3D points is mapped into voxel coordinates to extract features from neighboring voxels. These 3D and 2D features are concatenated as point-wise feature vectors for pose prediction, from which we predict pose $[\hat{\mathbf{R}}_i|\hat{\mathbf{t}}_i]$ and confidence c_i as in [Wang et al., 2019]. The rotation matrix $\hat{\mathbf{R}}_i$ is constructed from the quaternion predicted by the network. From the predicted confidence scores, we choose the most confident pose as the final prediction.

Training the Pose Prediction Network

Training Loss To train pixel-wise pose prediction, we use a training loss similar to DenseFusion [Wang et al., 2019], which is an extended version of the model alignment loss of PoseCNN [Xiang et al., 2018]. For each pixel-wise prediction, this loss computes the average distance between corresponding points of the object model transformed with ground truth and the predicted poses (pose loss).

Let $[\mathbf{R}|\mathbf{t}]$ be the ground truth pose, $[\hat{\mathbf{R}}_i|\hat{\mathbf{t}}_i]$ be the i-th point-wise prediction of the pose, and $\mathbf{p}_q \in P$ be a point sampled from the object model. The pose loss is formulated as:

$$\mathcal{L}_i = \frac{1}{|X|} \sum_q ||(\mathbf{R}\mathbf{p}_q + \mathbf{t}) - (\hat{\mathbf{R}}_i \mathbf{p}_q + \hat{\mathbf{t}}_i)||. \tag{3.3}$$

For symmetric objects, which have ambiguity in the correspondence with the object model, the nearest neighbor of transformed points is used as the correspondence (symmetric pose loss):

$$\mathcal{L}_i = \frac{1}{|X|} \sum_q \min_{\mathbf{p}_{q'} \in X} ||(\mathbf{R}\mathbf{p}_q + \mathbf{t}) - (\hat{\mathbf{R}}_i \mathbf{p}_{q'} + \hat{\mathbf{t}}_i)||. \tag{3.4}$$

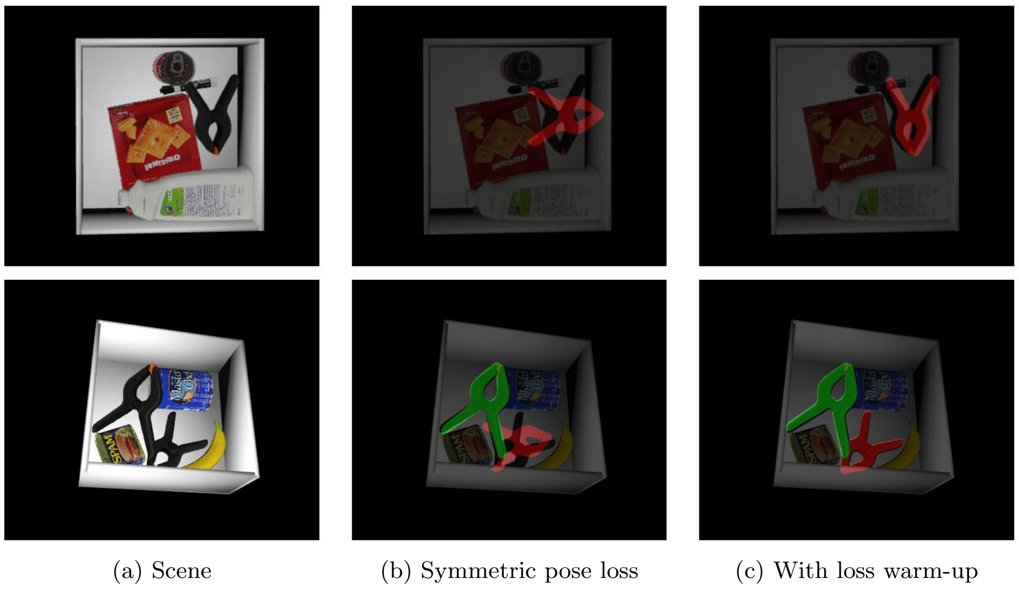
The confidence of the pose prediction is trained with these pose losses in an unsupervised way. Let N be the number of pixel-wise predictions and c_i be the i-th predicted confidence. The final training loss $\mathcal{L}$ is formulated as:

$$\mathcal{L} = \frac{1}{N} \sum_i (L_i c_i - \lambda \log(c_i)), \tag{3.5}$$

where λ is the regularization scaling (we use $\lambda = 0.015$ following [Wang et al., 2019]).

Local Minima in Symmetric Pose Loss Though the symmetric pose loss (Equation 3.4) is designed to handle symmetric objects using nearest neighbor search, we found that this loss is prone to get stuck to local minima compared to the standard pose loss (Equation 3.3), which uses 1-to-1 ground truth correspondence with the object model. Figure 3.5b shows the examples where the symmetric pose loss struggles to avoid local minima with a non-convex object.

To tackle this issue, we introduce a warm-up stage with a standard pose loss (e.g., 1 epoch) during training before switching to the symmetric pose loss. This training strategy with warm-up allows the network first to be optimized for pose prediction without the local minima problem though ignoring symmetries, and then to be optimized while considering symmetries. This gives much better results for pose estimation of symmetric objects with non-convex shapes (Figure 3.5c).



(a) Scene (b) Symmetric pose loss (c) With loss warm-up

Figure 3.5: **Avoiding local minima with loss warm-up.** Our loss warm-up (c) gives much better pose estimation for complex-shaped (e.g., non-convex) symmetric objects, for which a symmetric pose loss (b) is prone to local minima.

3.3.3 Collision-based Pose Refinement

In the previous section, we showed how we combine image-based object detection, RGB-D images, and volumetric reconstruction of the shapes of nearby objects to make per-object pose predictions with a network forward pass. This per-object pose prediction can often give good initial pose estimates, but not necessarily a mutually consistent set of estimates for objects that are in close contact with each other. In this section, we, therefore, introduce a test-time pose refinement module that can jointly optimize the poses of multiple objects.

For joint optimization, we introduce differentiable collision checking, by composing occupancy voxelization of the object CAD model, and an intersection loss between occupancy grids. As both are differentiable, this allows us to optimize object poses using gradient descent with optimized batch operation on a GPU.

An alternative approach would be sequential refinement by confirming each hypothesis one by one and feeding the confirmed hypotheses to the prediction network.

However, this approach requires a heuristic to choose the refined object for each step (e.g., closest-to-furthest from the camera for less-to-most-occluded), and one-by-one forward passes of the network, which would be generally much slower than a batch operation.

Differentiable Occupancy Voxelization

The voxelization of feature vectors mentioned in §3.3.2 uses feature vectors using points and is differentiable for the feature vector. In contrast, the occupancy voxelization needs to be differentiable for the points to optimize the transformation that produces these point locations. This means the values of each voxel in the occupancy grid must be a function of the points, which have been transformed by the estimated object pose.

Let $\mathbf{p}_q$ be a point, s be the voxel size, and $\mathbf{p}^g$ be the origin of the voxel (i.e., left bottom corner of the voxel grid). We can transform the point into a voxel coordinates with:

$$\mathbf{l}_q = \frac{\mathbf{p}_q - \mathbf{p}^g}{s} \tag{3.6}$$

For the index $\mathbf{u}_k = [k, l, m]^T \in \mathbb{N}^3$ of each voxel v_k we compute the distance δ from the point:

$$\delta_{qk} = ||\mathbf{u}_k - \mathbf{l}_q||. \tag{3.7}$$

The occupancy value of each voxel is determined to be proportional to the distance from the nearest point, resulting in the occupancy value of the voxel v_k computed as:

$$\delta_k = \min(\delta^t, \min_q(\delta_{qk})) \tag{3.8}$$

$$v_k = 1 - \frac{\delta_k}{\delta^t}, \tag{3.9}$$

where δ^t is the distance threshold.

Occupancy Voxelization for a Target Object

This differentiable occupancy voxelization gives occupancy grids from an object model and hypothesized object pose. For a target object o_m, the points sampled from its CAD model $\mathbf{p}_q$ are transformed with the hypothesized pose $[\mathbf{R}_m|\mathbf{t}_m]$: $\mathbf{p}'_q = \mathbf{R}_m\mathbf{p}_q + \mathbf{t}_m$, from which the occupancy value is computed. The point is uniformly sampled from the CAD model (including internal parts), and gives a hypothesized occupancy grid for the target object $g_m^{\texttt{target}}$.

Similarly, we perform this voxelization for the surrounding objects $\widetilde{o}_n$. Unlike the target object voxelization, the surrounding objects $\widetilde{o}_n$ are voxelized in the voxel coordinates of the target: $\mathbf{l}_q^{\widetilde{o}} = (\mathbf{p}_q^{\widetilde{o}} - \mathbf{p}_m^g)/s_m$ where $\mathbf{p}_m^g$ is the occupancy grid origin of the target object and s_m is its voxel size. This gives the hypothesized occupancy grids for surrounding objects of the target object: $\mathbf{V}_n^{\texttt{nontarget}}$.

Intersection Loss for Collision Check

Occupancy voxelization gives the hypothesized occupied space of the target $\mathbf{V}_m^{\texttt{target}}$ (the m-th object in the scene) and the surrounding objects $\mathbf{V}_n^{\texttt{nontarget}}$. The occupancy grids of surrounding objects are built in the voxel coordinates (center, voxel size) of the target object and aggregated with an element-wise max operation:

$$\mathbf{V}_m^{\texttt{nontarget}} = \max_n \mathbf{V}_n^{\texttt{nontarget}}. \tag{3.10}$$

This gives a single impenetrable occupancy grid. In addition to this impenetrable occupancy grid from the pose hypotheses of surrounding objects, we also use occupancy information from volumetric fusion: occupied space including background objects $\mathbf{V}_m^{\texttt{other}}$, and free space $\mathbf{V}_m^{\texttt{free}}$ (Figure 3.3), as additional impenetrable area: $\mathbf{V}_m^{\texttt{impen}} = \mathbf{V}_m^{\texttt{other}} \vee \mathbf{V}_m^{\texttt{free}}$. The collision penalty loss $\mathcal{L}_i^{\texttt{c+}}$ is computed as the intersection between the hypothesized occupied space of the target and the impenetrable surrounding grid:

$$\mathbf{V}_m^{\texttt{target-}} = \max_{\odot}(\mathbf{V}_m^{\texttt{nontarget}}, \mathbf{V}_m^{\texttt{impen}}) \tag{3.11}$$

$$\mathcal{L}_m^{\texttt{c+}} = (\mathbf{V}_m^{\texttt{target}} \odot \mathbf{V}_m^{\texttt{target-}}))/\sum_k \mathbf{V}_m^{\texttt{target}}, \tag{3.12}$$

where $\max_{\odot}$ is element-wise max and $\odot$ is element-wise multiplication of the voxel grid matrices.

Though this loss correctly penalizes collisions between the target and surrounding objects, optimizing for this alone is not enough, as it does not take into account the visible surface constraints on the target object $\mathbf{V}_m^{\mathtt{self}}$. The other term in the loss is the intersection between the hypothesized occupied space of the target and this grid $\mathcal{L}_m^{\mathtt{c+}}$, to encourage the surface intersection between the object pose hypothesis and volumetric reconstruction:

$$\mathcal{L}_m^{\mathtt{c-}} = (\mathbf{V}_m^{\mathtt{target}} \odot \mathbf{V}_m^{\mathtt{self}}) / \sum \mathbf{V}_m^{\mathtt{self}}. \tag{3.13}$$

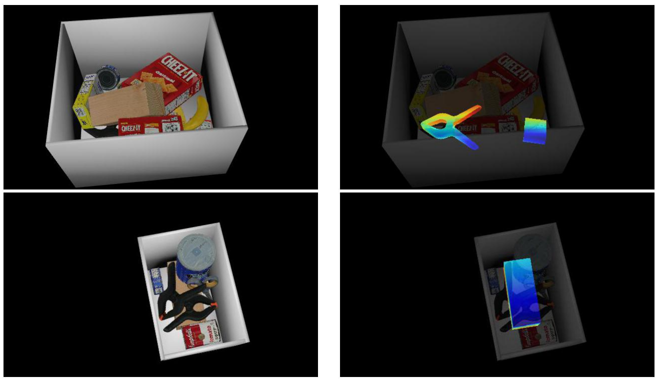
We compute these losses of collisions and surface alignment for N number of objects with a batch operation on a GPU, and sum them as the total loss $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{N} \sum_m (\mathcal{L}_m^{\mathtt{c+}} - \mathcal{L}_m^{\mathtt{c-}}). \tag{3.14}$$

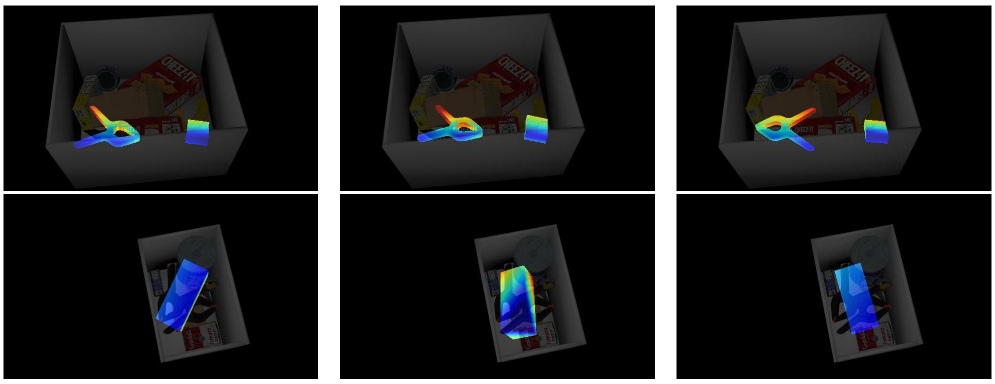
This loss is minimized with gradient descent (§2.7.3), allowing us to jointly optimize the pose hypotheses of multiple objects all together.

3.3.4 CAD Alignment

After performing pose estimation and refinement, we spawn object CAD models into the map once there is enough agreement on the poses estimated in different views. To compare object poses estimated in different camera coordinates, we first transform those poses into world coordinates using the tracked pose from the camera tracking module (§3.3.1). These transformed object poses are compared using a pose loss, which we also use for training the pose prediction network (§3.3.2). For the most recent N pose hypotheses, we compute the pose loss for each pair, which gives $N(N-1)$ pose losses: $\mathcal{L}_i$ $(1 \le i \le N(N-1))$. We count how many pose losses are under the threshold $(\mathcal{L}^t)$: $M = \mathbf{count}[[\mathcal{L}_i < \mathcal{L}^t]]$. When M reaches a threshold, we initialize the object with that agreed pose.



(a) Scene (b) Ground Truth



(c) DenseFusion* (d) MoreFusion$^{-\text{occ}}$ (e) MoreFusion

Figure 3.6: **Pose prediction with severe occlusion**. Our proposed model (MoreFusion) performs consistent pose prediction for 3 objects in clutter, while the baseline (DenseFusion*) and a variant without occupancy information (MoreFusion$^{-\text{occ}}$) fail.

3.4 Experiments

In this section, we first evaluate how well pose prediction (§3.4.1) and refinement (§3.4.2) performs on 6D pose estimation datasets. We then demonstrate the system running in a robotic pick-and-place task(§3.4.3).

Dataset

We evaluate our pose estimation components using the 21 classes of the YCB objects [Calli et al., 2015] used in the YCB-Video dataset [Xiang et al., 2018]. The YCB-Video dataset has been commonly used for the evaluation of 6D pose estimation in prior work [Xiang et al., 2018, Wang et al., 2019]. However, since all of the scenes are table-top, this dataset has limited diversity of object configurations, and few challenging cases such as occlusion and close contact among objects.

To make evaluation possible with heavy occlusion and arbitrary object orientations, we built our own synthetic dataset: Cluttered-YCB (Figure 3.6). We use the physics engine Bullet [Coumans et al., 2013], introduced in §2.6, to place object models with feasible configurations from random poses. This dataset has 1200 scenes (`train` : `val` = 5 : 1) and 15 camera frames for each.

Metric

We used the same metric as prior work [Xiang et al., 2018, Wang et al., 2019], which evaluates the average distance between corresponding points: ADD, ADD-S. These metrics are equivalent to the pose losses. ADD uses ground truth as in Equation 3.3 and ADD-S uses nearest neighbors as correspondence as in Equation 3.4, transforming the model with the ground truth and estimated pose. These distances are computed for each object pose in the dataset and plotted with error threshold on the x-axis and accuracy on the y-axis. The metric is the area under the curve (AUC), using 10cm as the maximum threshold for the x-axis.

3.4.1 Evaluation of Pose Prediction

Baseline Model

We used DenseFusion [Wang et al., 2019] as a baseline model. For a fair comparison with our proposed model, we reimplemented DenseFusion and trained it with the same settings (e.g., data augmentation, normalization, loss).

Table 3.1 shows pose prediction results on the YCB-Video dataset using the detection mask of [Xiang et al., 2018], where *DenseFusion* is the official GitHub implementation [1] and *DenseFusion** is our version, which includes the warm-up loss (§3.3.2) and centralization of the input point cloud (analogous to the voxelization step in our model). We find that the addition of these two components leads to a

[1] https://github.com/j96w/DenseFusion

big performance improvement. In the following evaluations, we use DenseFusion* as the baseline model.

Table 3.1: **Baseline model results** on the YCB-Video dataset, where DenseFusion is the official implementation and DenseFusion* is our reimplemented version. ADD and ADD-S are metrics that use point-to-point differences of an object model transformed using ground-truth and predicted poses.

Model	ADD(-S)↑	ADD-S↑
DenseFusion	83.9	90.9
DenseFusion*	89.1	93.3

Results

We compared our model (MoreFusion) with the baseline model (DenseFusion*). For a fair comparison, both models predict object poses from a single-view, meaning that MoreFusion is only allowed to use occupancy information from a single-view depth observation. We trained models using a combination of the Cluttered-YCB and the YCB-Video datasets and tested them separately with ground truth masks. The results (Table 3.2, Figure 3.6) show that MoreFusion consistently predicts better poses via its volumetric CNN and surrounding occupancy information. A larger improvement is achieved for heavily occluded objects (visibility<30%).

Table 3.2: **Pose prediction comparison**, where the models are trained with the combined dataset and tested separately.

Model	Test Dataset	ADD(-S)↑	ADD-S↑
DenseFusion*	YCB-Video	88.4	94.9
MoreFusion		**91.0**	**95.7**
DenseFusion*	Cluttered YCB	81.7	91.7
MoreFusion		**83.4**	**92.3**
DenseFusion*	Cluttered YCB (visibility$^{<0.3}$)	59.7	83.8
MoreFusion		**63.5**	**85.1**

To specifically evaluate the effect of using surrounding occupancy as an input, we tested the trained model (MoreFusion) by feeding in different levels of occupancy information: discarding the occupancy information from a single-view observation *-occ*; full reconstruction of non-target objects *+target*$^{-}$; and full reconstruction of background objects *+bg*. Table 3.3 shows that the model gives better predictions as more and more occupancy information is provided, which is what is possible in our incremental, multi-view object mapping system. This comparison also shows that even without occupancy information (MoreFusion$^{-\texttt{occ}}$), our model performs better than DenseFusion* purely because of the 3D convolutional architecture.

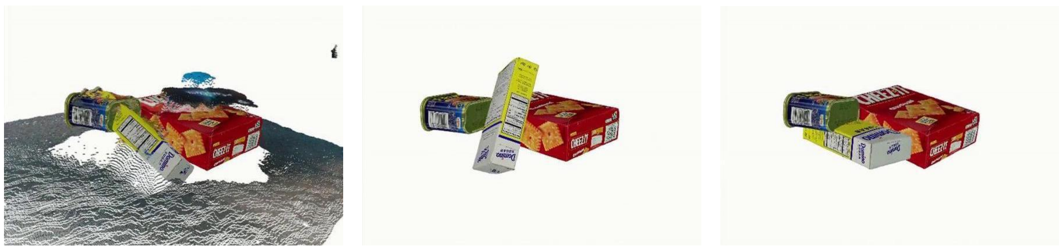
Table 3.3: **Effect of occupancy information** tested on the Cluttered-YCB dataset with the model trained in Table 3.2.

Model	ADD(-S)↑	ADD-S↑
DenseFusion*	81.7	91.7
MoreFusion$^{-\texttt{occ}}$	82.5	91.7
MoreFusion	83.4	92.3
MoreFusion$^{+\texttt{target}^-}$	84.7	93.3
MoreFusion$^{+\texttt{target}^-\texttt{+bg}}$	85.5	93.8

3.4.2 Evaluation of Pose Refinement

We evaluate our pose refinement method, *Iterative Collision Checking* (ICC), against Iterative Closest Point (ICP) [Besl and McKay, 1992]. Since ICP only uses a masked point cloud of the target object without any reasoning about surrounding objects, a comparison of ICC with ICP allows us to evaluate how well and in what cases the surrounding-object geometry used in ICC specifically helps pose refinement.

Figure 3.7 shows a typical example where the pose prediction has object-to-object intersections because of less visibility of the object (the yellow box). ICC refines object poses to better configurations than ICP by using the constraints from nearby objects and free-space reconstruction.



(a) No Refinement (b) ICP Refinement (c) ICC Refinement

Figure 3.7: **Pose refinement from intersecting object poses**, where we compare the Iterative Collision Checking (ICC) against Iterative Closest Point (ICP).

For quantitative evaluation, we used the Cluttered-YCB dataset with pose estimates refined from initial pose prediction by MoreFusion in Table 3.2. Figure 3.8 shows how the metric varies with different visibility in the dataset, and shows that the combination of the two methods (+ICC+ICP) gives consistently better poses than the others methods. With few occlusions (visibility >= 40%), ICC does not perform as well as ICP because of the discretization via voxelization (we use a 32-dimensional voxel grid). However, results are at their best with the combination of the two optimizations, where ICC resolves collisions in discretized space and then ICP aligns surfaces more precisely.

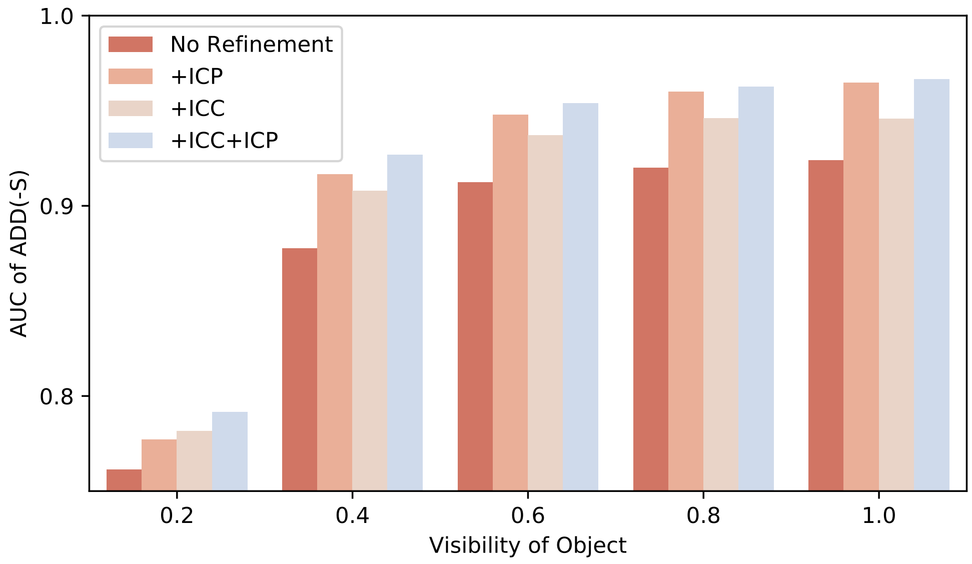


Figure 3.8: **Pose refinement results** on the Cluttered YCB-Video dataset, where the proposed Iterative Collision Check (ICC) gives the best results when combined with ICP.

3.4.3 Full System Demonstration

We demonstrate the capability of our full system, MoreFusion, with two demonstrations: scene reconstruction, in which the system detects each known object in the scene and aligns an object CAD model (shown in Figure 3.9); and secondly, a robotic pick-and-place task, where the robot is requested to pick a target object from a cluttered scene by intelligently removing distractor objects to access the target object (shown in Figure 3.10).

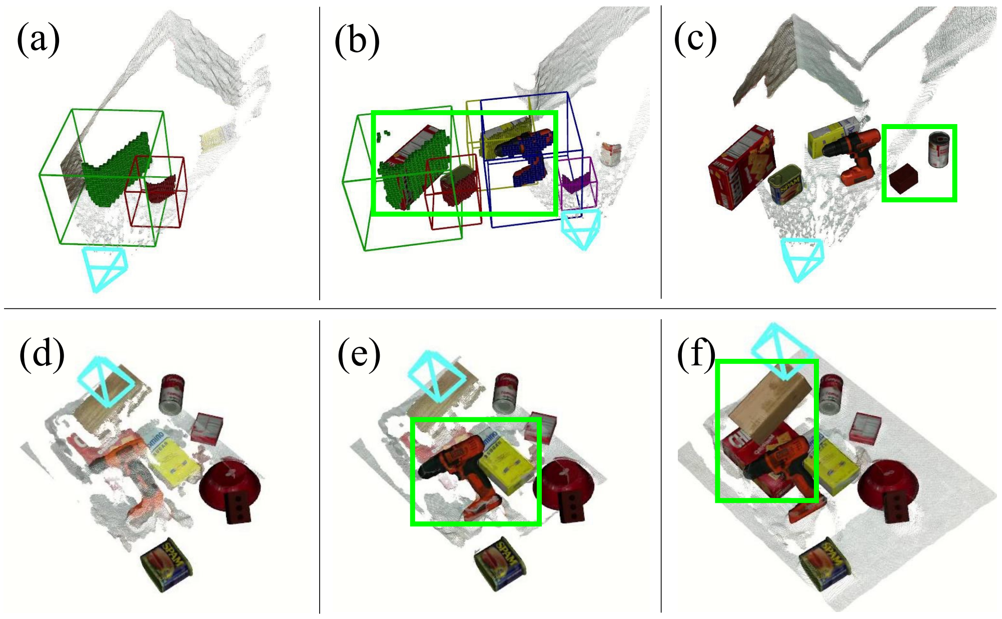


Figure 3.9: **Real-time full reconstruction.** Our system gradually increases its knowledge about the scene with volumetric fusion (a) and incremental CAD alignment (b) for the final reconstruction (c). The pose hypotheses of surrounding objects (e.g., drill, yellow box) are utilized to refine pose predictions to perform pose estimation for heavily occluded objects (e.g., red box) (d)-(f).

Object-level Scene Reconstruction

Figure 3.9 shows reconstruction results in two different scenes. This figure shows how our system successfully tracks each object in the volumetric fusion and aligns the object model incrementally (top row). In a more difficult setup (bottom row), where some objects are heavily occluded, the system first initializes less-occluded objects (drill, yellow box) and then initializes the heavily occluded object (red box) after reasoning together with the surrounding object configurations. These results demonstrate the capability of our system as a whole to gradually enrich scene understanding from volumetric fusion to full reconstruction.

Targeted Object Pick-and-Place

The ability to build an object-level full reconstruction of a scene is typically useful for the robotic pick-and-place of specified objects. These reconstructions allow the robot to reason about the picking order to successfully remove obstructing objects

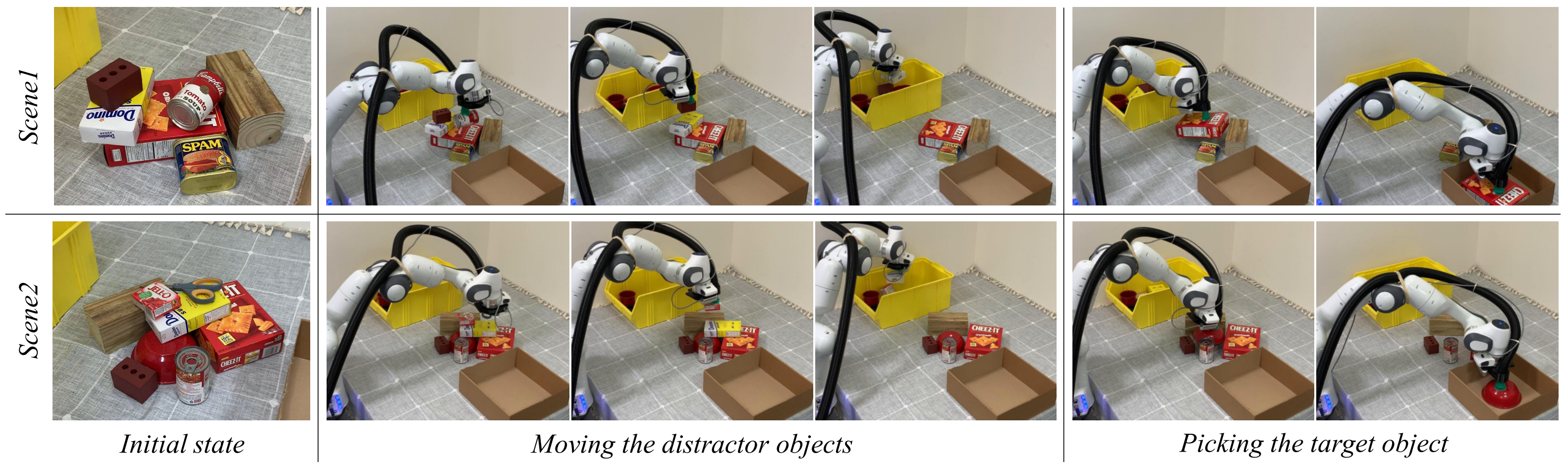


Figure 3.10: **Targeted pick-and-place demonstration**, where the robot must move obstructing objects to the yellow container, pick the target object (red box in Scene 1; red bowl in Scene 2), and then place it in the cardboard box.

before picking the target object. In Figure 3.10, we show a successful application of MoreFusion to a robot arm, which must move obstructing objects to a container, pick a target object, and then place it in the cardboard box.

3.5 Conclusion

In this chapter, we have shown consistent and accurate pose estimation of objects that may be heavily occluded by and/or in close contact with other objects in cluttered scenes. Our real-time and incremental pose estimation system, which is composed of object-level volumetric fusion, pose prediction using surrounding geometry, and collision-based multi-object pose refinement, builds an object-level map that describes the full geometry of the objects in the scene. This enables the robot to manipulate objects in complicated piles via intelligence of disassembling of occluding objects and oriented placing.

To further improve object pose estimation in cluttered scenes, an interesting future direction is to introduce physics reasoning into the pose optimization process. For a 3D object pose estimate to be feasible, the pose should not only not intersect other objects, but also be supported by other objects or surfaces, and that it provides sufficient support to other objects. A physics simulator with gravity, friction, and even deformation could be incorporated into our approach, though it is not yet fully clear how to use it in optimization efficiently. More generally, we expect that estimation of the poses of known objects will be part of a more general scene inference approach such as with optimizable shapes.

The manipulations demonstrated in this chapter were simple and exhaustive. To pick target objects in a pile occluded by other objects, the robot needed to move all the occluding objects first to reach and pick the target objects. This is because the collision-based motion planner we use in this chapter is strictly aimed at generating collision-free trajectories, and the overlapping objects have to be removed first to find solutions for the occluded target objects. Although this collision-free motion generation gives safe object rearrangement, avoiding damage and destruction, it could be possible to accomplish similarly safe target picking with fewer manipulation steps by selecting different motions. This motivates us to combine our semantic mapping system with learning-based manipulation motion generation as will be described in the following chapters: Chapter 4, Chapter 5.

Chapter 4

Fine-Grained Manipulation with a Semantic Map

Contents

4.1 Introduction

There has been continued research interest in integrating vision-based semantic mapping with manipulation for its potential to alter object states to achieve various useful tasks, such as part assembly [Stevšić et al., 2020, Zakka et al., 2020], extracting objects from clutter [Zeng et al., 2017], and arranging objects in a specific posture [Gao and Tedrake, 2019, Manuelli et al., 2019]. In Chapter 3, we demonstrated the capability of traditional manipulation pipelines, composed of perception to build an explicit scene representation of objects (semantic world model) followed by planning to search for a collision-free arm trajectory. As an alternative to this traditional pipeline, learning-based approaches have emerged recently as discussed in Chapter 1, directly inferring actions from observations (usually raw sensor information) with implicit scene understanding.

Although the traditional pipeline has been successful in structured environments, manipulation in cluttered environments is still challenging because of close contacts and occlusions among objects. In Chapter 3, we tackled in-clutter manipulation by combining vision-based semantic mapping and collision-based motion planning. However, this pipeline requires robots to move distractor objects away one by one to ensure there are collision-free manipulation trajectories, which can be inefficient when many objects are overlapping each other. In this situation, robots can struggle to safely extract occluded objects when requested to do so with a single grasp for efficiency. If robots try to extract occluded objects with a naive manipulation trajectory (e.g., moving straight upwards), this may cause destructive effects on the objects, which could be particularly critical when dealing with fragile objects.

To tackle this challenge to the traditional pipeline, in this chapter we explore the approach of integrating learning-based motion planning with vision-based semantic mapping to replace collision-based motion planning. Specifically, we learn a fine-grained 6D motion trajectory for a short-horizon task (target object extraction from a pile) with a model that predicts the residual motions of the end-effector step-by-step. This model receives as input an observation (e.g., camera image, object pose) and must predict the next best action as a transformation of the end-effector. To minimize destructive effects on non-target objects in the pile (e.g., the trajectory shown in Figure 4.1), we train the model with reinforcement learning to penalize the translations of the other objects during object extraction.

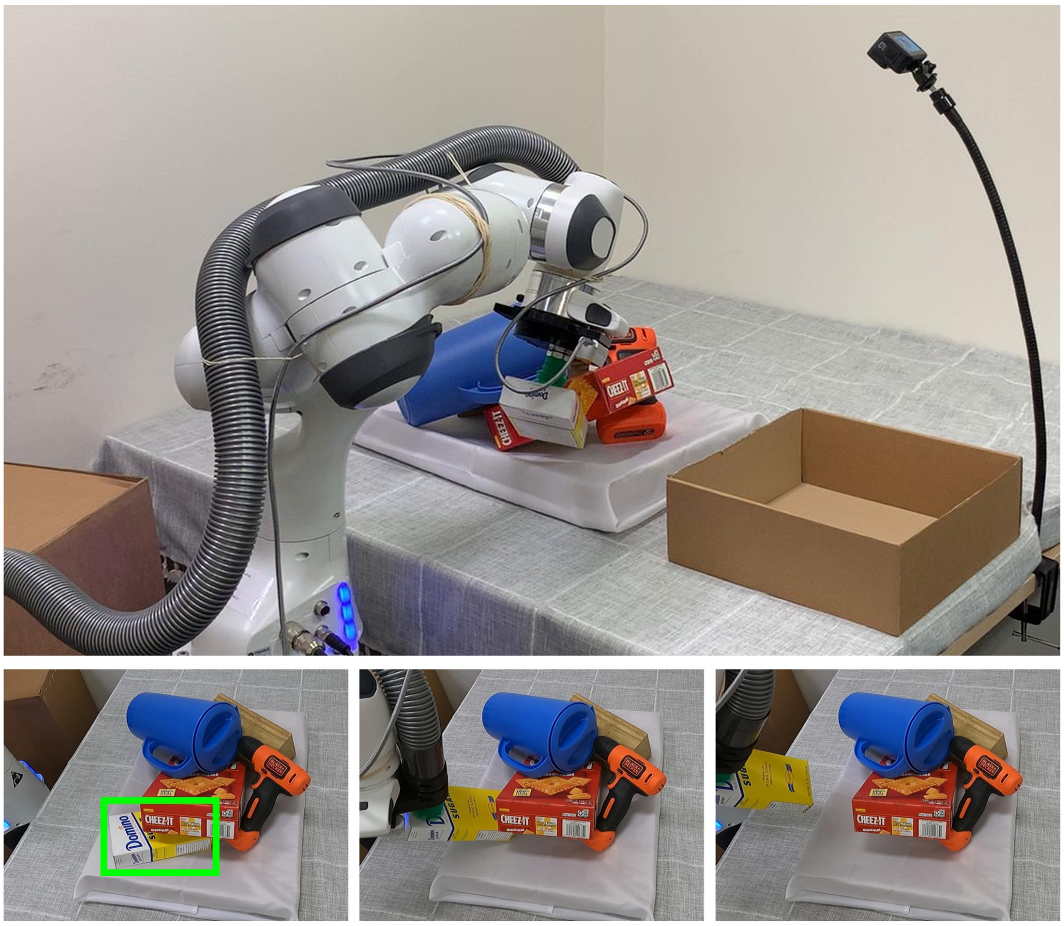


Figure 4.1: **SafePicking**, our manipulation system for object extraction, extracts target objects with minimum disturbance by generating a safe end-effector trajectory given raw observations and object poses retrieved from an object-level semantic map.

This fine-grained motion generation for a short-horizon task is in line with previous studies on learning-based object grasping [Devin et al., 2018, Kalashnikov et al., 2018, Levine et al., 2018], where a learning model predicts end-effector transformations and grasp commands. Despite the use of raw observations as input for the model (e.g., RGB-D images) being common in prior work, we argue that semantic information such as object poses can give vital cues as to how objects should be manipulated. Especially in our task of object extraction, which requires a proper understanding of the occluded parts of objects, we observe that the model generates better motions with object poses even when pose estimation errors are incorporated. On the other hand, however, when the pose estimate has errors such as misdetection and pose-difference, a model that only uses object poses (as in learning-based game agents [Brockman et al., 2016, Baker et al., 2020]) would have poor performance. To handle these estimation errors, we combine raw observations with object poses, enabling the model to gain high performance from pose information as well as robustness from the raw observations.

Our system, ***SafePicking***, shown in Figure 4.2, is composed of 1) **object-level semantic mapping** that builds a pose map of objects while exploring the target object, and 2) **learning-based motion planning** to generate an end-effector 6D trajectory from raw depth (depth images) and pose estimates (predicted object poses) using a neural network. The grasp point is determined to be the centroid of the visible surface of the object, and observations are transformed into the grasp point coordinate frame to be agnostic to the grasp point (*canonicalization*). This combination of semantic mapping and canonicalization makes our learned manipulation model general to variable object position in the workspace and enables the model to learn faster. We train the model in physics simulation.

To our best knowledge, this is the first work that tackles safe object extraction, where a robot is tasked with picking an occluded target object with a single grasp while minimizing the disturbance of the surrounding objects. In experiments, we demonstrate and evaluate our integrated system in the real world. In summary, the main contributions in this chapter are:

1. **Introducing safe object extraction** as a novel manipulation task: a robot must pick occluded target objects from a pile with a single grasp while minimally disturbing surrounding objects.
2. **Fusion of raw observations and pose estimates** in learning-based motion planning, achieving high performance and robustness to estimation errors.
3. **An integrated robotic manipulation system** with semantic mapping and learning-based motion planning, to demonstrate efficient and safe extraction of occluded target objects from a pile in the real world.

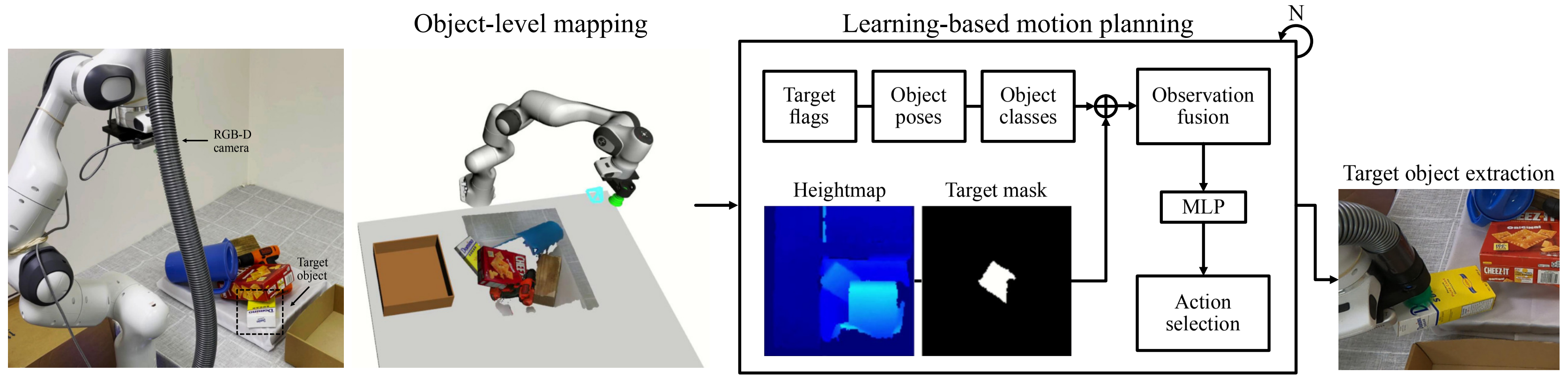


Figure 4.2: **System overview**, which consists of 1) object-level mapping with volumetric reconstruction and pose estimation of detected objects with an on-board RGB-D camera, and 2) learning-based motion planning for object extraction using as input the estimated object poses and raw depth observations in the form of a heightmap, which recursively generates an end-effector trajectory with N steps.

4.2 Related Work

The use of deep learning for robotic manipulation has become widespread (§1.3) with its progress in visual recognition with convolutional networks and observation-to-action policy learning [Levine et al., 2016, Zeng et al., 2020b, James et al., 2021b]. With convolutional networks, prior work [Levine et al., 2018, Pinto and Gupta, 2016, Zeng et al., 2018b] has demonstrated robotic grasping from a single view without explicitly modeling object geometry. For optimizing the policy for a sequential motion, other work has applied deep reinforcement learning for indiscriminate grasping [Kalashnikov et al., 2018, Zeng et al., 2018a], discriminate (i.e., targeted) grasping [Devin et al., 2018, Fang et al., 2018], and retrieval [Kurenkov et al., 2020]. Our work is along the lines of the work on discriminative object manipulation [Devin et al., 2018, Fang et al., 2018, Kurenkov et al., 2020], but we focus on how to manipulate objects *after* grasping instead of grasping itself, which has not been well explored in previous work. Moreover, we exploit object-level scene understanding for learning-based robotic manipulation by feeding in estimated poses along with raw observations given in the form of a heightmap.

4.3 Object-level Semantic Mapping

To build a map of objects and find the target objects in a scene from an RGB-D camera sequence, we use the object-level semantic mapping system, MoreFusion, introduced in Chapter 3. To find the object specified as the picking target, we query the object's geometric information from the map using the target object class, which provides its mask and estimated pose.

An alternative approach for this object exploration is using only single-view object detection. This single-view approach would work when the task workspace is small enough for a single view to capture the visible surface of all objects (as in prior work [Devin et al., 2018, Fang et al., 2018]). However, in a large workspace, a single-view can fail to capture the crucial observation of the target object because of truncation. Furthermore, when two instances of the same target object exist in a scene, and one of them is blocked by distractor objects and the other is not, with an explicit object map we could choose the less-blocked instance as the manipulation target, whereas a single-view approach may end up choosing the instance that is more challenging to manipulate. These limitations of single-view estimation motivate us to build an object-level map from multiple views.

4.4 Learning Object Extraction

To train the motion planning model, we use deep Q-learning [Mnih et al., 2015], an off-policy, model-free reinforcement learning algorithm. By collecting episodes through exhaustive exploration of the action space, this algorithm learns a policy that maximizes the cumulative reward of each episode. We use the sum of translations of the non-target objects as the negative reward (i.e., penalty) in this algorithm so that the model acquires manipulation skills that minimize the overall disturbance of a pile.

4.4.1 Grasp Point Selection

We select the centroid of the visible surface of the target object as the grasp point $\mathbf{p} = [p_x, p_y, p_z]^\intercal$. The surface geometry and mask of the target object are given from the object-level map. To determine the grasp orientation, we compute the quaternion $\mathbf{q} = [q_x, q_y, q_z, q_w]^\intercal$ that gives the minimal transformation to align the axis of the suction cup $\mathbf{n}_g$ to the normal of the grasp point $\mathbf{n}_s$ on the object surface:

$$\begin{aligned} [q_x, q_y, q_z]^\intercal &= \mathbf{n}_g \times \mathbf{n}_s && (4.1)\\ q_w &= \sqrt{\sum_i \mathbf{n}_{g,i}^2 + \sum_i \mathbf{n}_{s,i}^2} + (\mathbf{n}_g^\intercal \cdot \mathbf{n}_s). && (4.2) \end{aligned}$$

4.4.2 Fusion of Raw and Pose Observations

We take advantage of both raw and pose observations by feeding them as input to the model at both training and test time. The raw observation is a heightmap generated from depth images, giving unprocessed raw information about the visible surface of objects in the workspace. The pose observation is extracted from the object-level map giving, complete geometry and semantics though with the possibilities of misdetection and offsets on the object poses.

These two observations give different benefits. Raw observations are little processed before they are fed into the model, so are less prone to systematic errors than pose observations. In contrast, a pose observation gives complete semantics and geometry (e.g., occluded parts of the objects), which is missing in the raw observation, though is possibly subject to errors. Fusion of these two observations provides comprehensive information for the model to achieve both better performance in object extraction (less disturbance of a pile) and robustness to estimation errors (misdetection, pose-difference).

Raw Observation

For the raw observation, we build a heightmap from the depth images from the RGB-D camera. This heightmap represents the heights of the objects' surfaces from the ground plane and gives information about the visible surface of objects.

We build this heightmap by centering the XY coordinates of the grasp point p_x, p_y in the image coordinate of the heightmap, which canonicalizes the observations with respect to the grasp point, with XY bounds of the area the heightmap is representing. In our experiments, we use 0.004m as the size each cell represents and 128 as the image height and width dimensions, which gives $\pm 0.256m$ $(= 0.004 \cdot 128/2)$ XY bounds for the heightmap.

Pose Observation

As for pose observations, at training time we extract ground truth object poses and classes from the simulator we use to train the manipulation model. At test time, we extract object poses from the object-level map, which we build with MoreFusion using the on-board camera on the robot arm. To feed as the input to our model, we represent object pose with target flags (O, 1) with a binary vector, classes (O, C) with one-hot vectors, and poses (O, 7) with a set of position (O, 3) and quaternion

(O, 4), giving both semantic and geometric information for objects, where O is the number of detected objects and C is the number of object classes.

Before we feed this information into the model as input, we canonicalize the object pose as we do with the heightmap by centering the grasp point in the pose coordinate frame. We subtract the XY coordinates of the grasp point p_x, p_y from each pose of the object, which gives aligned pose observation along with the raw observation (depth information given as heightmap), allowing the model to exploit useful information from either of them as needed.

4.4.3 Manipulation Model

Action

We formulate manipulation as a residual 6D end-effector transformation, discretizing each axis of translation and rotation. We discretize the translation space in increments of 0.05 m and discretize the rotation with Euler angles (§2.2.2) in increments of $22.5 = 180/8$ degrees. Each residual has either 0, positive or negative value, to give combinations of $3^6 = 729$ possible actions.

This residual action is taken N times to generate a motion trajectory. To give information about previously taken actions to the model, we feed the previous end-effector poses as input to the model as well as the other observations of the scene (heightmap, pose). This temporal information given as input allows the model to incrementally reason about the trajectory as it is generated.

Network Architecture

Our model represents a Q-function that predicts the discounted return given the observation o_t at time t. Once trained, we evaluate this Q-function over a set of actions $a \in A$ and take the highest-valued one $\hat{a}$:

$$\hat{a} = \arg\max_a Q(a, o_t). \tag{4.3}$$

Figure 4.3 shows the network architecture. The heightmap observation is processed by a convolutional (Conv) encoder and downsized into a feature vector. This vector is concatenated with pose observations and processed by a transformer encoder [Vaswani et al., 2017]. With a transformer encoder, we can handle an arbitrary number and order of object poses as input. This encoder outputs features whose batch size is the same as the input features (the number of objects O). We compute the mean of these features to build global features following [Baker et al., 2020] and predict the Q-value with a fully-connected layer.

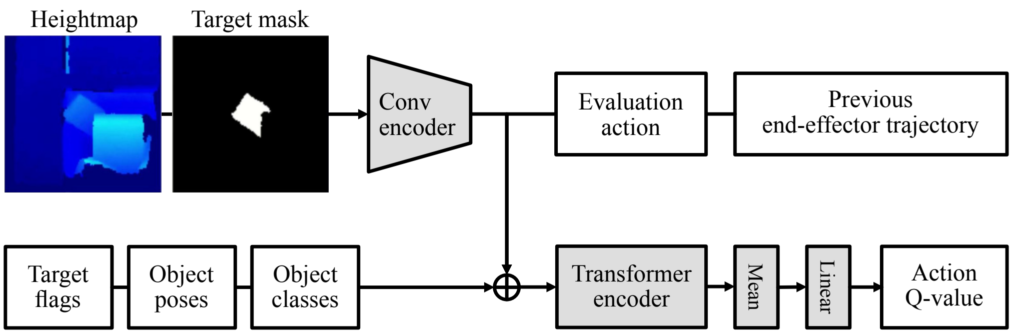


Figure 4.3: **Network architecture**, which uses heightmap and object poses to predict a Q-value for 6DoF end-effector actions. We feed end-effector residual motions as the evaluation action, from which the best-scored action is selected as the next action at test time.

Aside from the visual observations, this network receives two additional inputs: the evaluation action, and the previous end-effector trajectory. The evaluation action is fed in the form of a residual 6D pose represented as translation and quaternion vectors and is one of the candidate actions from which the best action is selected at the next step. The previous end-effector trajectory represents the history of actions the model has taken, allowing it to generate a consistent next action along with the context. The previous trajectory is input as a list of 6D poses.

4.4.4 Model Training

Reward

Disturbance of surrounding objects during manipulation can be classified in the following ways:

- Falling, which happens when overlapping objects are supported by the target, and they fall after the extraction of the target;
- Sliding, which happens when overlapping objects have to be displaced to create a space to extract the target.

Both of these effects are undesirable. The falling effect can damage the object and the sliding effect can expand the pile and workspace that can make later task continuation harder.

To cover both effects, we use the sum of the translations of non-target objects as the safety metric. When an object falls a large distance, not only does the translation of the object itself become large, but it can also hit other objects, causing a chain reaction of moving objects. By using the "sum" of the L2 norm of translations, we can encourage the model to minimize the number of objects affected as well as the translations of the individual object.

Deep Q-Learning

We compute the safety metric as the reward r_t at each time step t. Using this reward, we train the deep Q-network to minimize the following Bellman equation:

$$q_{t,a} = Q(o_t, a) \tag{4.4}$$

$$\hat{q}_{t,a} = r_t + \gamma \max_a \tilde{Q}(o_{t+1}) \tag{4.5}$$

$$\mathcal{L} = |\hat{q}_{t,a} - q_{t,a}|, \tag{4.6}$$

where γ is the time discount of the reward, Q is the live network updated every training step, and $\tilde{Q}$ is the target network, a copy of the live network updated less frequently to avoid overfitting.

Training in Simulation

We train the manipulation model in a physics-based simulation environment using the physics engine introduced in §2.6, since it is difficult to measure collisions between a grasped object and its surroundings when a robot manipulates objects in the real world. Furthermore, it is time-consuming and challenging to build different configurations of objects for each trial of the learning processes, and ensure the safety of both the robot and objects during exploration, as the robot might make a dangerous motion which breaks objects or makes object configurations not restorable.

To let the robot experience various configurations of objects, we procedurally generate object piles while simulating physics to build a set of feasible piles. For a given set of object models, we define the 3D space where those models can be spawned. Each step randomly selects one of the object models and an object position and orientation from the defined 3D space, from which we apply physics simulation until the spawned object stops moving with no collisions, making the object pile stable. This generation process produces arbitrary object orientations and overlaps that are physically feasible, from which the robot can learn a successful policy to manipulate objects by understanding the geometric relationships with the surrounding objects.

4.5 Experiments

We evaluate our method by assigning the robot to grasp and extract a target object from a pile using safety as the task metric. We train our learning models in simulation, using YCB CAD models [Calli et al., 2015], and evaluate performance in both simulation and the real world.

Training detail

We build the learning model with PyTorch [Paszke et al., 2019] and a simulation environment built with Bullet [Coumans et al., 2013] (§2.6). To train the model, we run a single process to update the model parameters using the action-state pairs collected from multiple processes that asynchronously run the learned model to act in different simulation environments. For this asynchronous data collection and training, we use an open-source framework [1].

As for the training hyperparameters, we use batch size 128, and Adam [Kingma and Ba, 2015] (§2.7.3) as the optimizer with parameters of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. From the start of training, we use Epsilon Greedy exploration, which linearly balances random action selection and policy-based action selection according to the number of iterations (in the beginning, it uses random selection more frequently), to collect data up to 5000 iterations. We use replay ratio 16 (the number of updates per data collection), and synchronize the model parameters in different processes every 100 iterations, allowing the model in the exploration processes to collect episodes with the new parameters.

Metric

To evaluate the performance of the model, we define safety metrics that represent how safely the robot extracts occluded target objects. As discussed in §4.4.4, this metric should represent the disturbance of non-target objects caused by falling or sliding. In the experiments, we use the following metrics:

- The sum of translations, which evaluates effects caused by both falling and sliding, as discussed in §4.4.4;
- The sum of max velocities, which primarily evaluates falling effects, as a larger distance fall gives higher velocity since objects are accelerated with gravity.

4.5.1 Baseline Comparison

Naive motion

In Table 4.1, we compare our method against several alternatives. The simplest motion is joint linear interpolation to the reset pose, in which the end-effector is located in free space with the suction cup faces the ground plane. When the target

[1] https://github.com/stepjam/YARR

Table 4.1: **Baseline comparison**, in which we compare our learned model with baseline methods using the safety metric. Tested in 600 unseen pile configurations in simulation.

			Safety metric	
Method	**Input**	**Noise**	**translation↓**	**velocity↓**
Naive	-	no	0.701	1.919
Heuristic	-	no	0.578	1.624
RRT-Connect	pose	no	0.520	1.643
SafePicking	pose, heightmap	no	**0.465**	**1.419**
RRT-Connect	pose	yes	0.532	1.645
SafePicking	pose, heightmap	yes	**0.465**	**1.433**

object is overlapped by distractor objects, this motion introduces many collisions among objects, causing falling and sliding, giving the lowest scores.

Heuristic motion

As a simple heuristic for extracting objects from a pile, we use a motion that extracts the grasped object with a straight motion towards the +Z direction of the world coordinate. Although this motion can cause many collisions with overlapping objects, on average it gives better results than the naive motion, which interpolates motions in joint space irrelevant to the physical prior (e.g., gravity direction) in 3D space (Table 4.1).

Collision-based motion planning

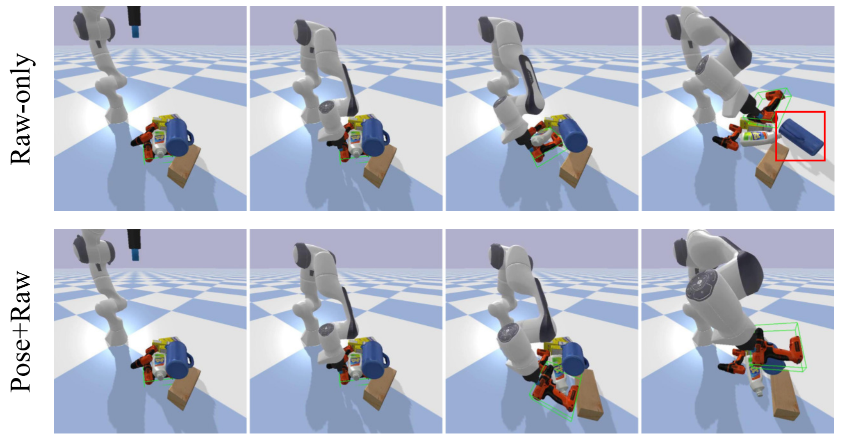
As another baseline, we use a collision-based motion planning method, RRT-Connect [Kuffner and LaValle, 2000] (§2.6). RRT-Connect samples many configurations of the arm in 3D space while checking for collisions to find the connections between them, and uses the world model of a scene to compute collisions. We use the same Bullet [Coumans et al., 2013] physics engine for collision checking, as is used for physics simulation throughout this thesis (§2.6). Although this motion planning method can produce non-destructive extraction motion when it finds a collision-free path, it struggles to find relatively safe trajectories when a complete collision-free path does not exist. In this case, the motion planner ends up giving a naive motion, which can cause significant movement of the surrounding objects. Table 4.1 shows that RRT-Connect gives comparable results to the Heuristic motions (better in translation, worse in velocity). Meanwhile, our learned model (SafePicking) gives the best results, as we would would hope given that it is trained to minimize pile disturbance with reinforcement learning instead of just minimizing collisions.

4.5.2 Model Ablations

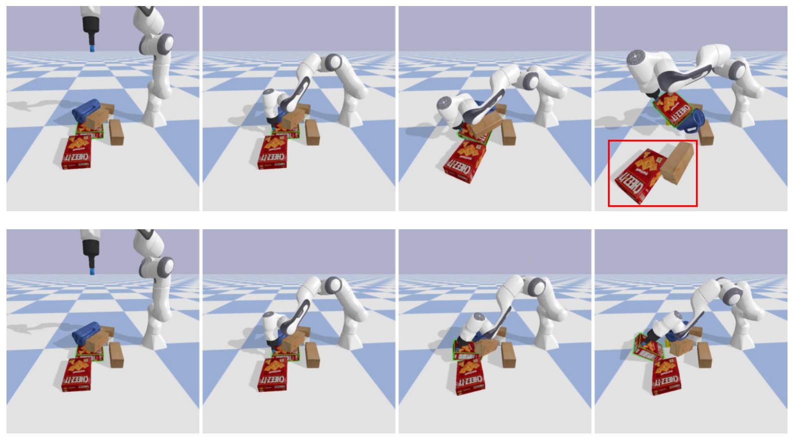
We evaluate 3 model variants with different inputs to compare the effect of different observations on the learned model (Table 4.2). We also evaluate its robustness by adding noise to the pose observation (misdetection, pose-difference) based on the visibility of objects. *Raw-only* receives as input only a heightmap generated from a depth image. *Pose-only* uses object pose as input, which gives more complete information about objects, and gives better results than *Raw-only* even with noise in the input object poses. Our full model, *Pose+Raw*, receives both heightmap and object pose as input, compensating for the error in object pose by using the visible surface information from a heightmap. The results show that the combination of pose and heightmap works best in the presence of errors. As one would expect, when given perfect pose information (no pose noise), there is no benefit to having additional heighmap information; however, in reality, predicted poses will always have some errors, and so the addition of pose noise shows the benefit of fusing poses and heighmap observations. Figure 4.4 shows qualitative results, in which the model with object poses gives better motions for object extraction with minimal disturbance of the other objects.

Table 4.2: **Ablation study**, in which we compare the variants of the learned model with/without adding noise to the pose observation. Tested in 600 unseen pile configurations in simulation.

			Safety metric	
Variant	**Input**	**Noise**	**translation↓**	**velocity↓**
Raw-only	heightmap		0.507	1.491
Pose-only	pose	no	0.477	1.430
Pose+Raw	pose, heightmap		**0.465**	**1.419**
Pose-only	pose	yes	0.487	1.449
Pose+Raw	pose, heightmap		**0.465**	**1.433**



(a) Scene 1



(b) Scene 2

Figure 4.4: **Qualitative comparison of variants of our method**, in which we compare *Raw-only*, which uses only the heightmap observation, and our full model *Pose+Raw* that uses both object poses and heightmap. We use the same and constant camera viewpoint in each scene (the column axis).

4.5.3 Real-world Experiments

We evaluate our system in the real world using the same robotic system introduced in §2.5.1: a Franka Emika Panda robot with an RGB-D camera (Realsense D435 [Keselman et al., 2017]) mounted on the wrist, and a suction gripper built using a Dyson vacuum cleaner. Similarly to Chapter 3, we integrate the robotic controller with the vision and motion planning system using the Robotic Operation System (ROS) framework [Quigley et al., 2009], which enables inter-process communication of sensory data (e.g., RGB-D images, robotic arm joint states), perception results (e.g., object detections, poses), and motion commands (e.g., target joint positions and velocities).

Evaluation Metric

For the numerical experiments in the real world, we use heightmap difference between before and after the task as the safety metric, as depicted in Figure 4.5. The use of a different metric from the evaluation in simulation is because the metrics used in simulation (total translations and velocities of the objects at every step) are challenging to measure in the real world (requiring time-consuming pose annotations). To evaluate the heightmap difference, the robot scans the object pile and builds a heightmap before and after the execution of the task (Figure 4.5a, b). The pixel regions of the target object are excluded from the comparison as the target object is intentionally moved, and its difference should not be penalized. These two heightmaps built before/after the task are compared to measure the pixel-wise height difference as shown in Figure 4.5c as a difference map (top) and a mask (bottom) with a threshold of $0.01m$. From these difference images, we compute the difference volume (with the pixel size of the heightmap of $0.004m$) and mask size to use them as the safety metrics.

Quantitative evaluation

Using the heightmap-based evaluation metric, we evaluate the variants of our learned models in the real world. For the task configuration, we use similar settings to the

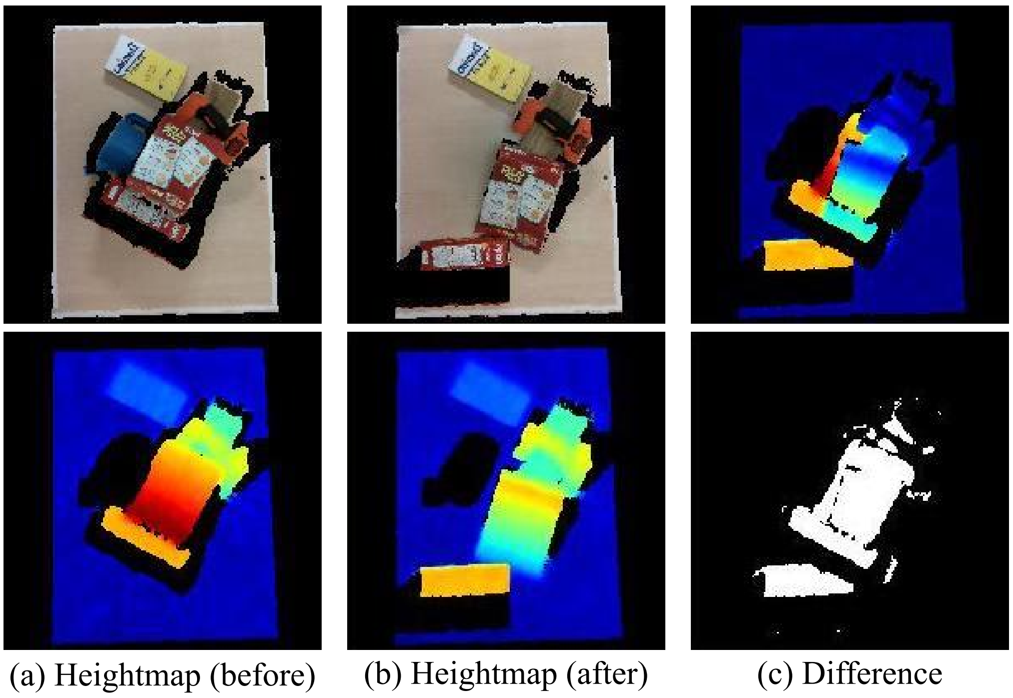


Figure 4.5: **Real-world safety evaluation** by comparing pile heightmaps built before/after manipulation. The region of the target object (blue pitcher in this case), which is filled with a black color in the heightmaps, is excluded from the comparison.

simulation, where target objects are partially occluded by the other objects in a pile and the robot is tasked with extracting the target object with a single grasp while minimizing the disturbance of the other objects. Table 4.3 shows the comparison of our learned models (Raw-only and Pose+Raw) for 20 configurations, some of which are shown in Figure 4.6. After each task, we manually reset the scene to provide the same object configuration to different methods. This comparison gives results consistent with the ablation study in simulation, showing that pose information is beneficial for motion generation and enables the model (Pose+Raw) to select safe actions compared to using only raw observations (Raw-only).

Table 4.3: **Real-world model comparison**, in which we evaluate learned models by comparing the heightmaps before/after each task. Each model is tested in the same 20 configurations with same target objects.

	Safety metric	
Variant	**Diff mask [%]↓**	**Diff volume [litter]↓**
Raw-only	7.1	3.2
Pose+Raw	**4.4**	**2.1**

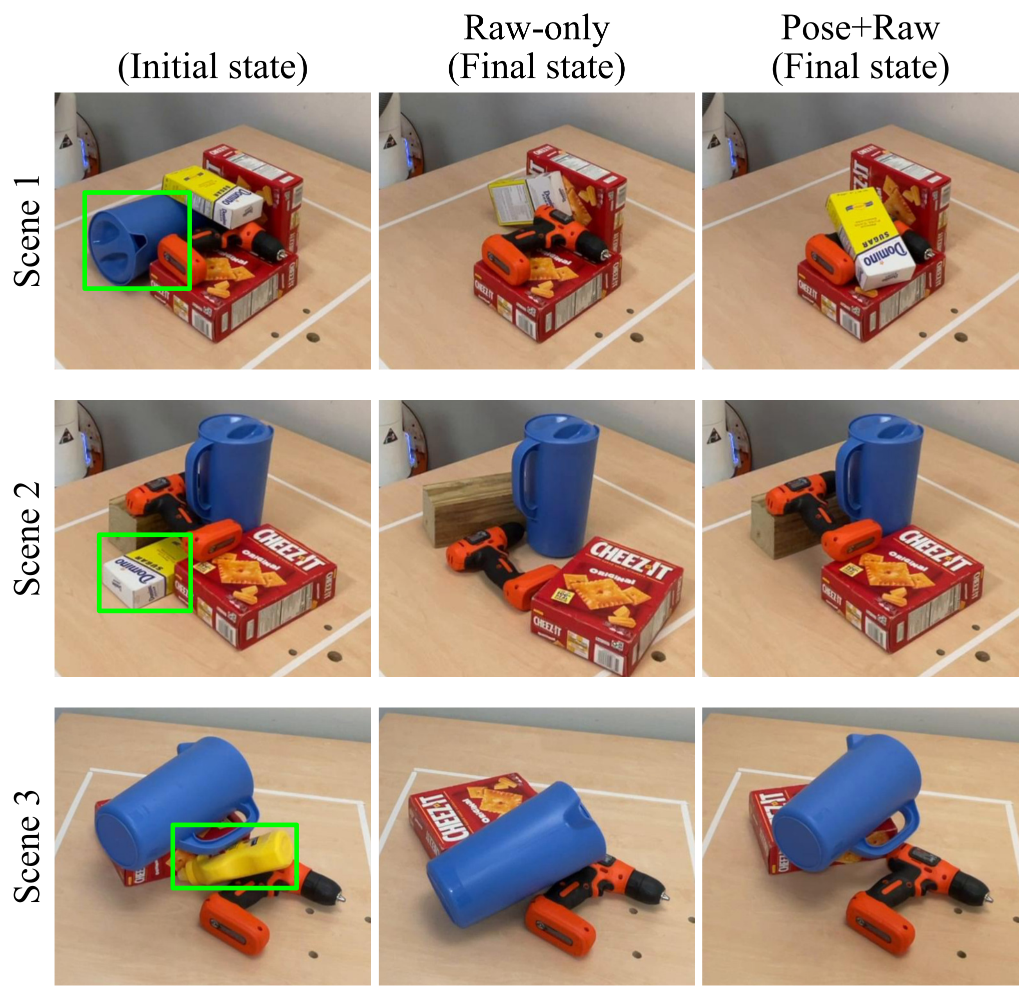
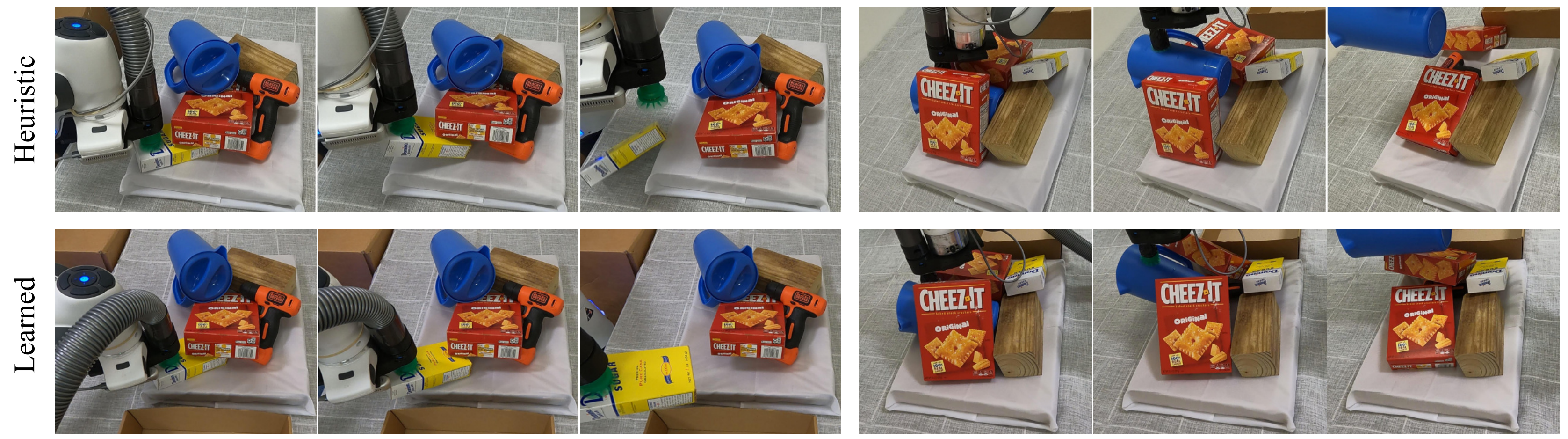


Figure 4.6: **3 pile configurations** out of the 20, we use to the learned models in the real world. We measure the performance with the heightmap-based safety metric. Consistent with the results in simulation, the model generates better motions when given a fused observation of object poses and a heightmap (*Pose+Raw*).

Qualitative evaluation

Figure 4.7 shows qualitative comparison between heuristic and learned motions. This comparison demonstrates that the learned model is more capable of safely extracting target objects while minimizing the effects on surrounding objects.

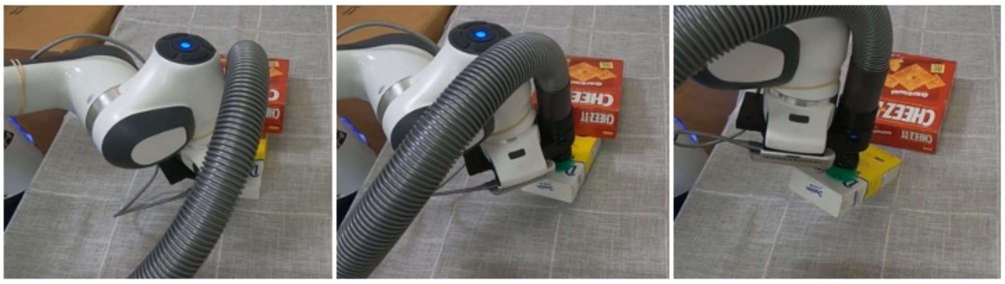
Figure 4.8 demonstrates the adaptation of our learned model to changes in a pile configuration. This result shows that the model successfully captures the geometric relationship between the target and distractor objects, changing its motion to avoid disturbance of the distractor objects.

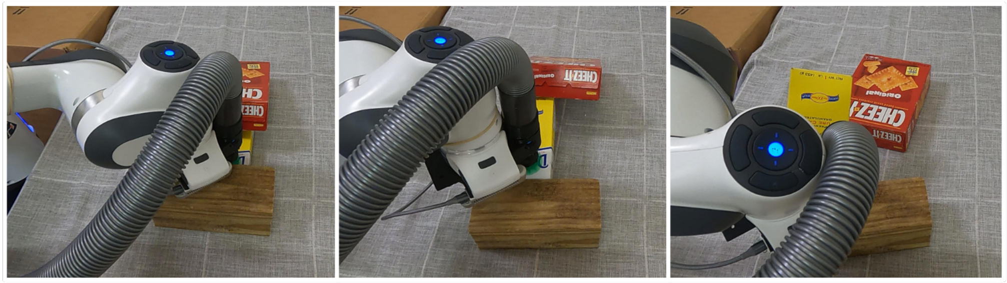


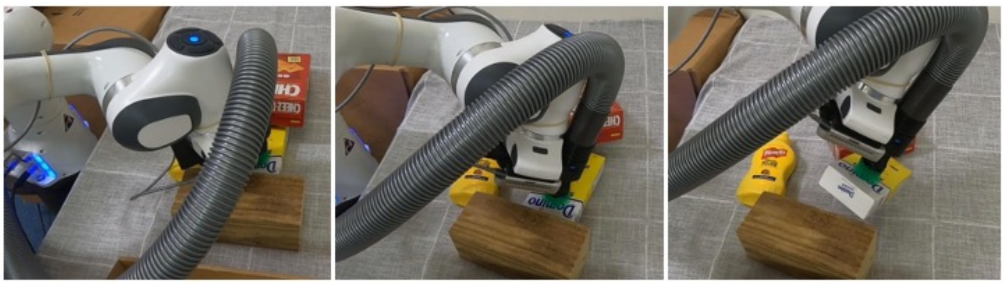
(a) Scene 1

(a) Scene 2

Figure 4.7: **Learned vs. heuristic motion.** We compare the motion from our learned planner with an upward straight motion (heuristic) to extract target objects (a: yellow box, b: blue pitcher). The learned model successfully adapts to different pile configurations, avoiding collisions (a) and the object drop from the container (b), which happen with the heuristic motion.


(a) Initial configuration


(b) +Wooden block


(c) +Mustard bottle

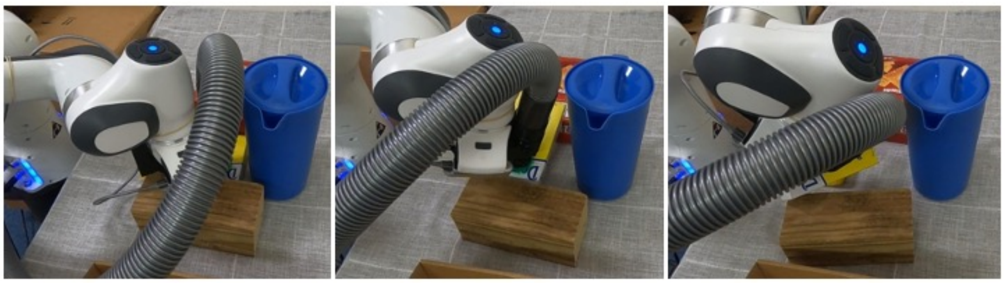

(d) +Blue pitcher

Figure 4.8: **Pile change adaptation** of the learned model, where we incrementally add distractor objects with the same target, blocking the previously selected action. The model shows successful adaptation to the change of the pile, avoiding disturbance in all cases.

4.6 Conclusion

In this chapter, we have integrated a learning-based manipulation with the object-level semantic mapping system we built in Chapter 3, focusing on generating a short-horizon yet fine-grained 6D motion trajectories to safely extract objects from a pile. The motion model is trained via reinforcement learning so that it generates a safe trajectory that enables single-grasp target picking while avoiding the disturbance to the other objects in a pile.

Our integration of learning-based manipulation is also made robust to estimation errors in the world states by combining raw sensory information with estimated state information as input. Estimation errors are unavoidable in visual perception, especially in complex scenes such as object piles, and can result in missing heavily occluded objects or estimating incorrect poses with offsets. The heightmap input gives robustness to estimation errors being directly generated from raw depth observations, and the pose input allows better motion generation via rich semantic information about the scene.

The focus in this chapter was learning fine-grained motions for short-horizon manipulation. We showed that the integration of learning-based manipulation with semantic mapping enables robots to acquire manipulation skills that cannot be achieved with a traditional motion planner or end-to-end learning-based manipulation that uses only raw observations.

Despite the successful integration of semantic mapping and learning-based manipulation for a short-horizon task, it is still unclear how useful the semantic map is for more long-horizon manipulation tasks. Long-horizon manipulation can include several steps of grasping and placement that all have to be combined in an optimal way to achieve the final task goals. In Chapter 5, we tackle this question with a manipulation task that needs long-horizon planning of motions.

Chapter 5

Long-Horizon Manipulation with a Semantic Map

Contents

5.1 Introduction

In the previous chapter, we have explored short-horizon yet fine-grained motion generation with a learned manipulation model combined with vision-based semantic mapping, in a task of object extraction of occluded target objects from a pile. However, as discussed in Chapter 1, manipulation is not limited to short-horizon tasks and can include several steps of grasping and placement requiring long-horizon planning for selecting the best motions to complete the task. In this chapter, we tackle a manipulation task of specific-posed object placement, which requires long-horizon motion planning including reorientation and regrasping to achieve desired grasps that enable the final placement.

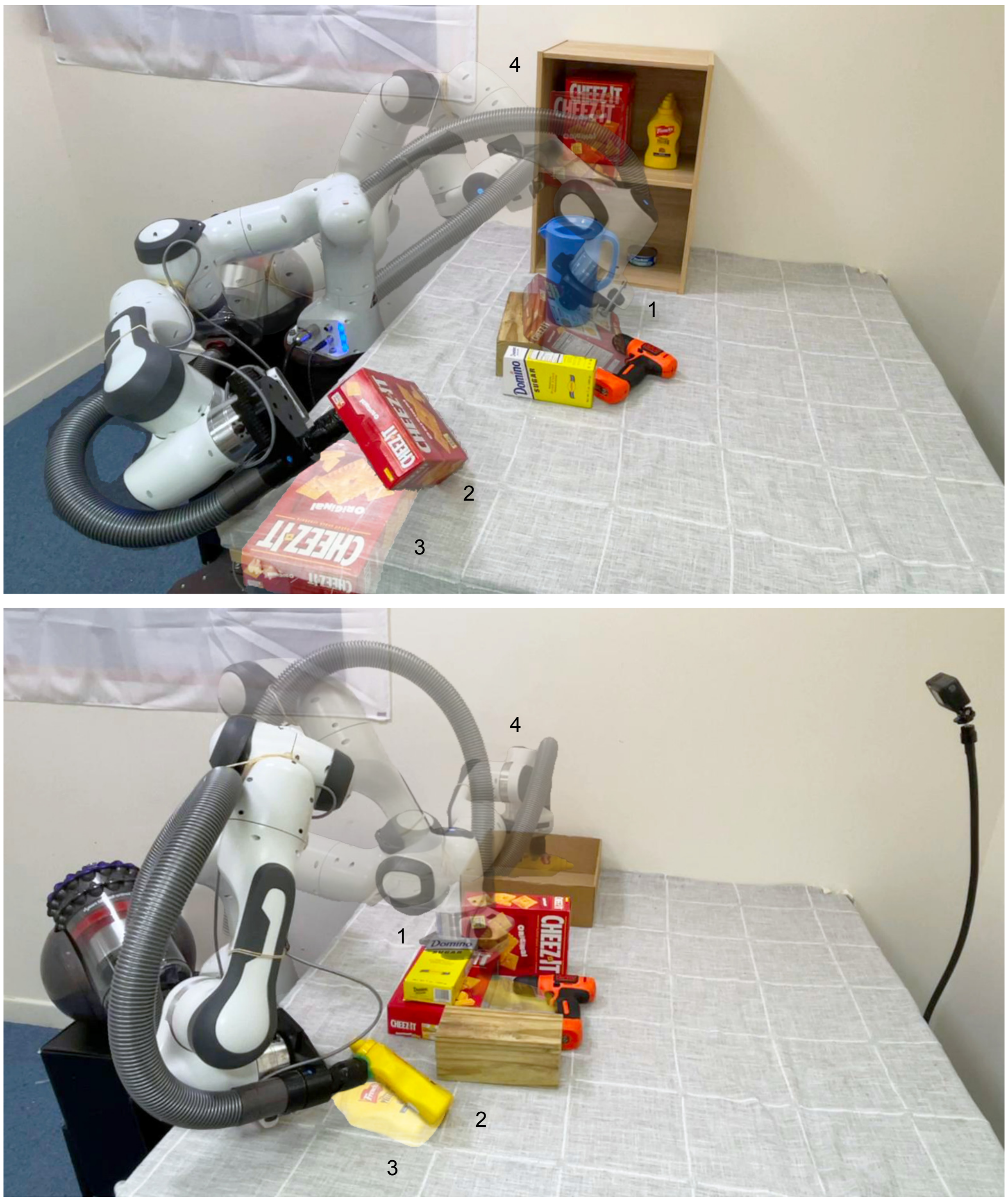


Figure 5.1: **ReorientBot**, our manipulation system for object specific-posed placement, picks, reorients, regrasps, and places objects in various target configurations. Learned components enable object reorientation with significant rotation using dynamic placement (released and stabilized with gravity), which is hardly achievable with human heuristics.

Placing objects in a specific pose is a vital capability for robots to rearrange the world into arbitrary configurations of objects. This capability enables various applications such as product display, storing, or packing, which require tidy, secure, and space-saving object arrangements. When objects must be specifically placed, reorientation is often a crucial manipulation step, to change the object pose in favor of the subsequent task steps as in Figure 5.1. Reorientation makes a specific surface of the object accessible when the goal configuration restricts the feasible grasp points and they are inaccessible in the initial state of the object. With a pile of objects, the grasp points can be initially blocked by the ground plane or the surrounding objects, forcing the robot to rotate or flip the target object for regrasping.

Traditionally, object reorientation has been accomplished with human-designed reorientation poses (e.g., 90-degree rotation), for which a motion planner generates trajectories to reorient objects [Tournassoud et al., 1987, Wan et al., 2019]. Although the motion planner can generate decent reorientation trajectories given appropriate and diverse reorientation poses, the motions are often inefficient due to the limited diversity of the pose candidates, requiring multiple steps to reorient objects when a significant orientation change is required (e.g., flipping). A single-step reorientation would be a solution to this inefficiency; however, it requires careful design of the reorientation poses, which must be both placeable and re-graspable. Because of these combinatorial and complex requirements, human heuristics (e.g., a canonical, upright orientation as the reorientation pose) do not achieve high levels of success.

To overcome the limitations of human heuristics for motion generation, an alternative is a learning-based approach to generate successful and efficient motion trajectories. Although learning-based approaches for robotic manipulation have become common [James et al., 2021b, Levine et al., 2016], especially in short-horizon manipulation tasks such as indiscriminate grasping without precise placement [Kalashnikov et al., 2018, Levine et al., 2018] and the object extraction shown in Chapter 4, it is still unclear how to best model long-horizon tasks as it becomes harder to train models as the task horizon increases. For specific-posed placement, the reorientation motion has to be selected considering the succeeding regrasping and placement.

Our method uses a sampling-based approach, where learned models evaluate the quality of candidate motion waypoints. This learned model predicts the success and efficiency of the coarse waypoints from which trajectories are generated by traditional collision-based motion planning. This waypoint evaluation (cf. trajectory evaluation by feeding a long list of waypoints) assumes that the coarse waypoints (e.g., grasp and reorientation poses for object reorientation) stand for the whole trajectory and can be used for evaluation before actually generating the whole trajectory. This early evaluation drastically reduces the planning time, allowing the model to use numerous motion candidates to find the best motion.

Using this approach, we build ***ReorientBot***, a hybrid system that integrates learned manipulation and traditional motion planning along with vision-based semantic mapping (Figure 5.2). The state of the scene is captured by the vision pipeline presented in Chapter 3, MoreFusion. Given this state information, we generate motion proposals of the trajectory's start (grasp/regrasp pose) and end (reorientation/placement pose) waypoints with prediction and filtering by learned models, which are then fed into motion planning to generate trajectories. To our best knowledge, this is the first work that shows efficient single-step object reorientation with dynamic motions, for specific-posed placement with diverse initial and goal states of objects. We demonstrate the capability of the system in the real world showing a real-time scene understanding, planning, and manipulation. To summarize, the contributions in this chapter are:

- **The first work on single-step object reorientation** with dynamic motions, enabling a robot efficiently reorient objects for object rearrangement from an arbitrary initial state to goal state;
- **Learned motion waypoint selection**, which enables hybrid, long-horizon motion planning taking advantage of the generality of traditional motion planning and the inference speed and robustness of learned models;
- **A full real-time manipulation system**, showing capable object rearrange-

ment with visual semantic mapping, motion generation with learned waypoints, and planned trajectories.

5.2 Related Work

As the crucial step in isolating objects from a scene, object picking has been widely studied since early robotic research as discussed in Chapter 1. Recent work has integrated vision-based object segmentation and grasp planning to achieve object picking in a more challenging, cluttered environment with object overlap and occlusions [Jonschkowski et al., 2016, Zeng et al., 2017, Wada et al., 2017]. To deal with unseen objects, several previous studies have trained a model generating object class agnostic grasp proposals from input images [Pinto and Gupta, 2016, Zeng et al., 2018b, Hasegawa et al., 2019]. As a different approach to grasp proposals for unseen objects, there have been several studies on training learning models that predict the next best action given as input an image of the scene [Levine et al., 2018, Kalashnikov et al., 2018]. Although these studies on learned manipulation models have shown a strong capability of picking objects in various situations (e.g., occluded, unseen), the placement motion after picking has been mostly simple (e.g., dropping in a box) without knowing how the object is grasped.

A couple of studies tackle the whole pipeline of robotic pick-and-place, including decent object placement in a specific pose. kPAM [Manuelli et al., 2019] has designed semantic keypoint detection for grasp point selection and place trajectory generation, demonstrating an intended grasp point selection and trajectory generation for placement. Shome et al. [Shome et al., 2019] have shown a tight object packing of box-shaped objects incorporating a hand-crafted reorientation motion. These studies on specific-posed placement restrict either the initial state (e.g., target grasp point is accessible), goal state (e.g., few different orientations), or shape (e.g., box) of objects. We tackle object placement with diverse initial and goal states, with various-shaped objects in the YCB object set [Calli et al., 2015].

Robotic research on object reorientation and regrasping dates back to the 1980s

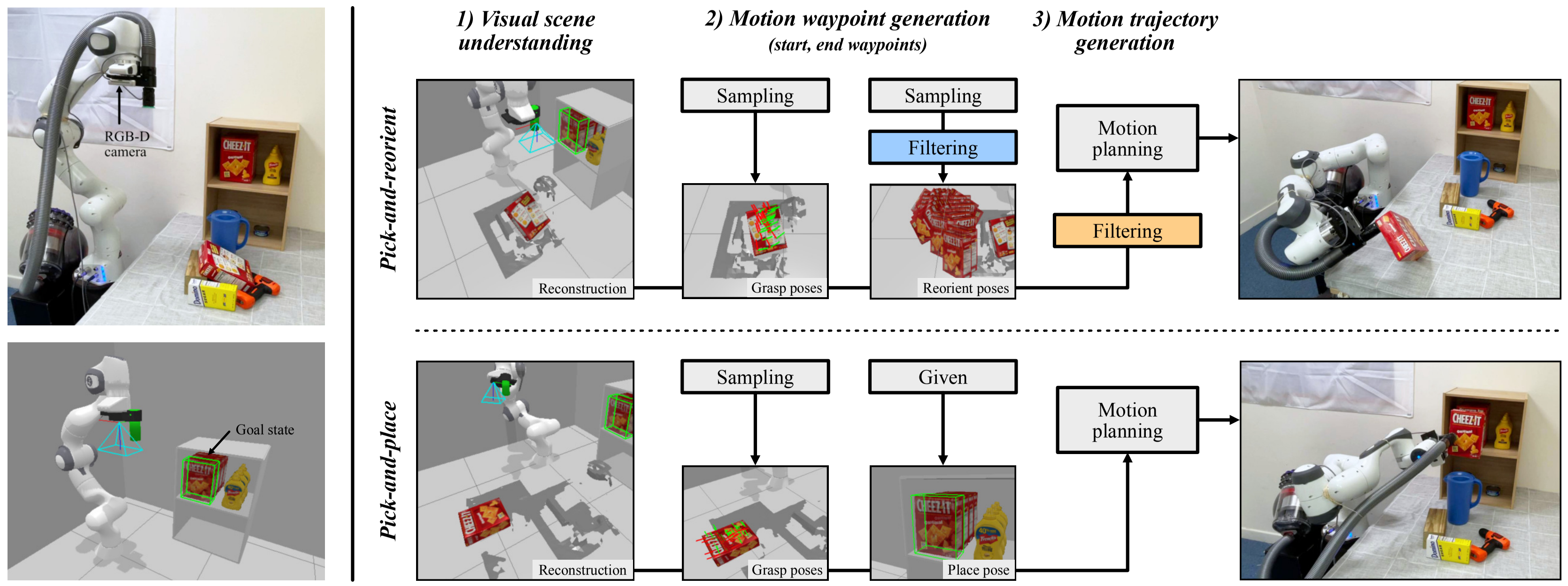


Figure 5.2: **Overview of ReorientBot**, a hybrid system of traditional motion planning and learned (🟦, 🟧) components, consisting of: 1) vision-based 6D pose estimation and volumetric reconstruction; 2) motion waypoint generation; 3) trajectory generation with motion planning.

with the seminal work by Tournassoud et al. [Tournassoud et al., 1987], and it has been tackled as one of the core skills of robotic manipulation [Cole et al., 1992, Rohrdanz and Wahl, 1997, Wan and Harada, 2016b]. Lozano-Pérez et al. [Lozano-Pérez and Kaelbling, 2014] and Wan et al. [Wan and Harada, 2016a, Wan et al., 2019] have developed a whole system of object reorientation and placement by planning multi-step reorientation with sampling stable states of the object on a plane. Although they have shown successful object reorientation given enough time of execution, they sacrifice the motion efficiency by discarding promising unstable poses that will eventually become stable in the desired orientation after being released. This restriction makes reorientation with significant rotation (e.g., flipping objects) difficult, even when achievable in a single step. In this chapter, we use unstable reorientation poses from which the object will eventually settle down to the desired state, to achieve single-step, efficient object reorientation.

In-hand manipulation has also been tackled as a solution to reorienting objects to achieve a specific orientation. Dafle et al. [Dafle et al., 2014] showed an in-hand regrasping capability with a three-fingered hand such as rolling and flipping. Andrychowicz et al. [Andrychowicz et al., 2020] and Akkaya et al. [Akkaya et al., 2019] extended this further to a five-fingered hand to show even more dexterous manipulation skills such as solving a Rubik's cube. Although promising, the robot's capability heavily depends on the specially designed robotic hand, limiting its applicable environments (the hand is often attached to a fixed base), object sizes, and poses. In this chapter, we use a suction gripper with a general-purpose robotic manipulator, both of which are widely used for robotic manipulation in industry and research communities.

5.3 Method

Given the goal state of the target object, our system runs object detection, pose estimation, motion planning, and manipulation to rearrange the target object to the goal pose. This system, shown in Figure 5.2, consists of 1) visual semantic mapping that builds a 3D representation of objects via 6D pose estimation and volumetric reconstruction with the vision pipeline introduced in Chapter 3; 2) motion waypoint generation that gives the pairs of start and end arm configurations with learning-based filtering; 3) motion trajectory generation with a collision-based motion planner while selecting the best pair of waypoints using another learned model.

This system includes reorientation and regrasping as needed, which is determined via planning the direct pick-and-place from an initial state to a goal state. If this planning fails to find a collision-free motion, the system switches to another motion planner for reorientation. We repeat this process until the motion planner finds a collision-free path for pick-and-place. We optimize the reorientation step to change the object's orientation successfully and efficiently to make the target grasp point accessible to place it in the specified goal pose. For this optimization, we sample numerous candidates of the reorientation pose, which a learned model evaluates via feasibility (the existence of a collision-free trajectory) and efficiency (the length of the trajectory) prediction.

5.3.1 Object-Level Semantic Mapping

Consider a pick-and-place task, with target objects in a pile from which the robot must grasp, reorient, regrasp, and place the objects into goal states. This pipeline requires robots to semantically understand the scene differentiating target and non-target objects. To place target objects in a specific pose, it is necessary for robots to know the objects' initial poses in a pile to compute the transformations the robots must apply to the objects to achieve the goal state via grasping and placement. For non-target objects, semantic understanding of the objects (object category and geometry) might not be as important as the target's since their geometric information is used only for collision avoidance when there are no occlusions for objects. In this chapter, we focus on the reorientation of target objects with no occlusions assuming their extraction is done beforehand by the object extraction pipeline we built in the previous chapter. We use a heightmap to represent non-target objects, which allows faster training and better generalization, being agnostic to semantics and robust to estimation errors at test time.

We run the object-level semantic mapping system, MoreFusion (Chapter 3), to find the target object using an on-board RGB-D camera with a robotic arm. MoreFusion detects objects from an RGB image with a neural network-based object detector and a volumetric 6D pose estimation for the detected objects. Using the target object class given as the task goal, the 6D pose of the target object (the initial state in a pile) is extracted from the map to be used for the motion planning in the succeeding pipeline.

The heightmap is built from depth images captured by the on-board RGB-D camera to represent the state of non-target objects. A heightmap represents the distance of the top surface from the ground at each XY location and is agnostic to semantics and semantic estimation errors as it is purely built from raw observations (depth images). As for the depth images to build the heightmap, we use the top-down view of the RGB-D camera towards the pile. We set the pile center to be the center of the heightmap, having fixed XY bounds of the workspace covered to give consistent and overall information of the scene.

5.3.2 Motion Waypoint Generation

The start and end configurations of the robot and the target object (i.e., grasp and placement poses) define the waypoints of the motion trajectory. These two waypoints are used along with the semantic map of the scene to generate collision-free trajectories. To generate these waypoints, we combine random sampling and learned filtering to select the feasible and most efficient waypoint to execute.

Sampling Waypoints for Pick-and-Reorient

The goal of the **pick-and-reorient** stage is to change the target object orientation such that the robot can grasp specific grasp points for final placement where feasible grasps can be limited (e.g., box packing; shelf storing). A grasp pose represents a starting waypoint, and a reorientation pose represents an end waypoint. We sample the grasp poses in the 3D reconstruction from visual scene understanding and sample the reorientation poses in an open, planar space near the pile.

Grasp Pose For reorientation, grasp poses are sampled on the initial state of the target object in a pile. Given the pose of the object from pose estimation, we render the object with a virtual camera in simulation to extract its mask and depth images from which we sample the grasp poses. We transform the depth image into a point cloud (§2.3.2) and compute the surface normals by applying sliding windows on the point cloud. Given the object mask aligned to the point cloud and normals, we randomly extract ~ 30 points and normal vectors on the object surface, which gives the position and orientation of the grasp pose as a quaternion $\mathbf{q} = [q_x, q_y, q_z, q_w]^\intercal$. We compute this quaternion from the normal of the suction gripper $\mathbf{n}_g$ and the surface normal vector $\mathbf{n}_s$ with Equation 4.1.

Reorientation Pose We sample reorientation poses within a workspace adjacent to the pile, though in practice the reorientation poses could be sampled on any pre-defined surface. During this sampling, we check for collisions between the CAD model of the target object and the scene using the volumetric reconstruction of non-target objects.

Since exhaustive collision checking of arbitrary positions and orientations is time-consuming, we first determine XY positions where any orientation will be collision-free. We use a cube with the dimensions of the longest axis of the object, which gives a quick collision check with the pile reconstruction. As for the workspace where reorientation poses are sampled, we use $0.5m \times 0.3m$ rectangular space discretizing it by 10 and 8 each giving $10 \times 8 = 80$ candidates. These candidate positions are evaluated with the cube to filter positions that cause collisions being too close to the pile, which is typically critical for large objects.

Given the selected XY positions, we compute the Z position and orientation using the actual CAD model of the object instead of the abstracted cube. We discretize the orientation by 8 in each axis of the Euler angles (§2.2.2), which gives $8^3 = 512$

orientations for each XY position. For each orientation, we compute the distance between the object's bottom and the ground plane and set the Z position to put the object on the plane with a small margin of $2cm$. Since we sampled positions where arbitrary orientations would be collision-free in the XY position sampling, we can reuse the same Z positions and orientations for other XY locations. This multistep sampling avoids the slow, combinatorial evaluation of reorientation poses, whose number could be (XY positions) $\times$ (orientations) $= 80 \times 512 = 40,960$.

This sampling of the reorientation poses includes unstable states on the plane, which eventually settle down to a stable state after the release. These unstable poses allow the robot to reorient the object with significant rotation in a single step, for example, grasping the backside of the object to flip to the front by leaning the object on the plane while creating a space for the suction gripper to avoid collisions with the ground plane as shown in Figure 5.1.

Learning to Select Reorientation Poses

Not all the given reorientation poses (40,960 candidates) enable the robot to regrasp the object with the intended grasp for the final placement. To filter these unuseful poses, we introduce a learned model that predicts whether the reorientation pose will enable the intended regrasping or not with the pipeline shown in Figure 5.3. This pipeline uses a learned model to evaluate reorientation pose candidates and selects the top-1000 best-scored poses to be processed in the succeeding pipeline.

The learned model (right in Figure 5.3) receives a reorientation pose and target grasp pose as input and predicts the success of regrasping after the object is released and settles down. We also feed the pile heightmap to allow the model to take the collisions into account. The model encodes the heightmap with 6 layers of 3×3 convolutional with max-pooling and ReLU activations (ConvNet). The output is concatenated with the object's initial pose, label, reorientation pose, and grasp pose and processed by 3 linear layers (MLP) to predict the grasp validity, which is trained with binary cross-entropy loss as a 0–1 probability.

To train the model, we use physics simulation and motion planning (§2.6) to evaluate reorientation poses. Given a randomly sampled reorientation pose, an object model is spawned in simulation to apply physics to infer how the object will settle down after being released with that reorientation pose. Given the stable states of reoriented objects by applying physics, motion planning is applied to test whether the target grasp pose is achievable. This planning result gives the binary label of whether the pair of reorientation pose and grasp pose is feasible (grasp validity in Figure 5.3), and is used as a training signal for the model.

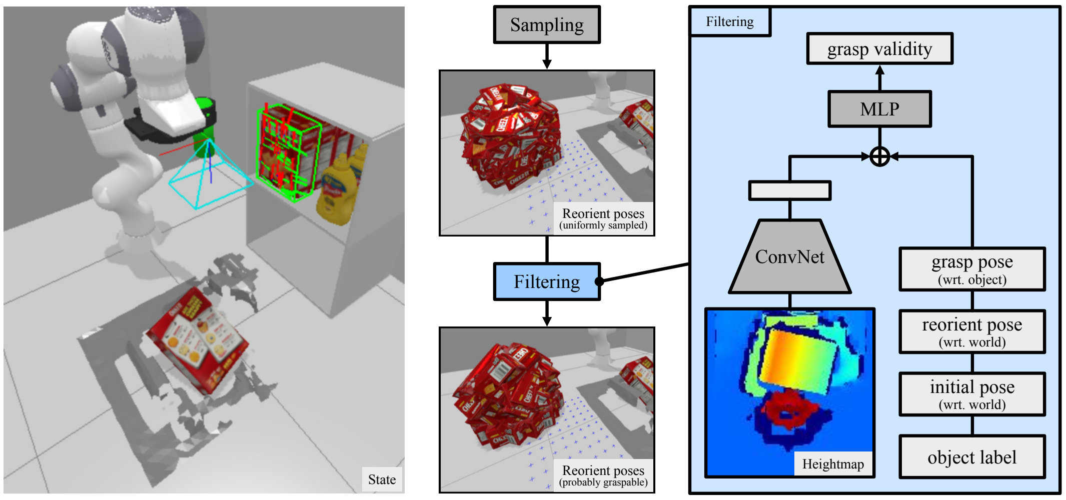

State
Sampling
Reorient poses (uniformly sampled)
Filtering
Reorient poses (probably graspable)
Filtering
grasp validity
MLP
ConvNet
Heightmap
grasp pose (wrt. object)
reorient pose (wrt. world)
initial pose (wrt. world)
object label

Figure 5.3: **Learned selection of reorientation poses**, using a learned model to select poses from the uniformly sampled reorientation poses.

Sampling Waypoints for Pick-and-Place

The goal of the **pick-and-place** stage is to place the target object in the specific pose given as a task goal. The grasp pose represents the start waypoint and the specified final pose represents the end waypoint of the pick-and-place trajectory.

Grasp Pose We sample grasp poses from the visible surface of the target object in the goal state with virtual rendering (cf. initial state for reorientation). We position a virtual camera facing the virtually placed object with a slight translation of $0.3m$ from the goal pose, whose direction is determined by the direction of the opening of the container for placement: horizontal with shelves, vertical with boxes. By using this virtual rendering, we can sample only the grasp poses visible from the opening of the container while filtering infeasible grasp poses in the back. The grasp position is sampled randomly from the visible surface, and the orientation is determined with the same process as §5.3.2, generating 30 grasp poses (left of Figure 5.3).

Place Pose For placement, we use the object's place pose given as the task goal. To this goal state, we transform the sampled grasp poses at the object's initial state to compute the end-effector pose at placement.

5.3.3 Motion Trajectory Generation

We integrate a learned model with motion planning to introduce waypoint selection for efficient planning and execution.

Learning to Select Motion Waypoints

Motion planning runs fast with a few pairs of start and end waypoints (0.1–1.0 seconds) and can generate a collision-free trajectory while evaluating and filtering unusable pairs. However, when the number of pairs becomes large (>100), the planning time becomes untenable for real-time use (10–100 seconds). For this problem, we introduce a learning-based model that predicts the validity of the pairs (i.e., the probability that the motion planner finds a collision-free path given those pairs), which we use to filter unusable waypoint pairs before feeding them into the motion planning (Figure 5.4).

With the two motion trajectories in the task (pick-and-reorient and pick-and-place), we apply learning-based waypoint selection only to the pick-and-reorient motion. This design choice is because the waypoints for pick-and-place motion are well constrained by the end waypoint (the place pose), which is unique and given as the task goal (whereas numerous possible reorient poses must be compared for pick-and-reorient), therefore it is not necessary to use the learned model for the real-time motion selection. It is also likely that placing configurations vary at test time with different object poses or placement environments (e.g., shelf storing, box packing), where it would be difficult for a learned model to adapt without retraining.

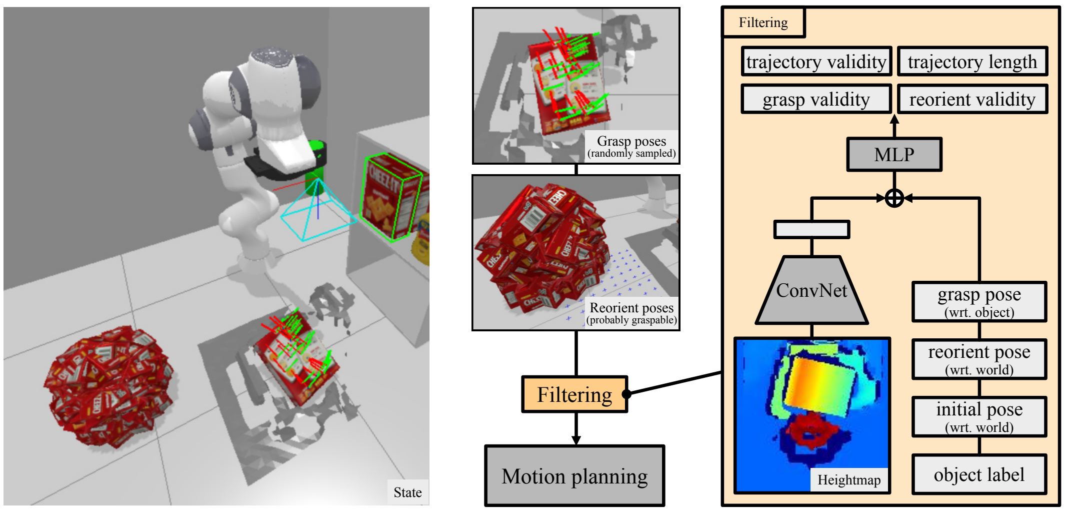


Figure 5.4: **Learned waypoint selection**, with predicting validity and efficiency to filter waypoints before motion planning.

Metrics for Selection The model predicts 3 validities: grasp pose, reorientation pose, and trajectory, each representing the existence of collision-free states between the robot and the scene. Although a single validity could cover the entirety of the grasp and reorientation pose (start and end states) and the trajectory (middle states), we separated these to help the model reason about why the whole trajectory might be invalid (e.g., which of the start/end/middle states are invalid).

After filtering by validity, several waypoints could remain as candidates with similar predicted validity scores. Therefore, we have introduced another metric: efficiency, which is often regarded as the secondary metric of robotic tasks [Batra et al., 2020]. We use joint-space trajectory length as the efficiency metric, which highly correlates with execution time.

After taking the highest-scored 10 waypoints with the trajectory validity scores, we sort them with efficiency before feeding them into the motion planner. Despite the randomness in the planning algorithm while finding collision-free trajectories, we observe a strong correlation between the given waypoints and the generated trajectory (i.e., they are consistent). With this correlation, the learned model predicts meaningful scores to select waypoints that generate the best motion trajectory.

Training the Model For the waypoint selection, we use a similar model architecture as the reorientation pose selection (§5.3.2) by only changing the outputs (right of Figure 5.4). Given the start and end waypoints (grasp pose, initial object pose, reorientation pose), this model predicts the validity and efficiency of the trajectory that will be generated by the motion planner, taking collision with objects in the scene into account using the heightmap. We train this model with binary cross-entropy loss for the validities and L1 loss for the trajectory length.

To collect training data of reorientation validities and trajectory length, we use a physics engine (§2.6) generating stable object piles by applying a physics simulation on the randomly spawned object models (train : test = 1000 : 200 configurations). We determine the target object based on its visibility (>95%). We sample grasp poses from the mesh vertices and sample reorientation poses on the pre-defined reorientation space (e.g., a space next to the pile). These sampled poses of grasping and reorientation are evaluated by collision checking and motion planning to provide the validities of the waypoints, and the validity and length of the trajectory.

Collision-free Trajectory Generation

Given the selected waypoints, we generate motion trajectories with collision-based motion planning, which uses the 3D geometry from the semantic mapping to check for collisions between the robot and the scene.

5.4 Experiments

We evaluate our method, ReorientBot, via a set of pick-and-place tasks that require appropriate object reorientation and grasp selection before being placed in a given goal pose. We train our learned models in simulation, using YCB object models [Calli et al., 2015, Xiang et al., 2018] and evaluate them in both simulation and the real world. We use a Franka Emika Panda robot, using its kinematics model (URDF) in simulation.

Implementation Detail

We use PyTorch [Paszke et al., 2019] to implement the learned models, training with Adam optimizer [Kingma and Ba, 2015] (§2.7.3) with parameters of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We stop training as the learning curve converges. For training data collection, we use a physics engine, Bullet [Coumans et al., 2013] (§2.6), to simulate the behavior of objects after being released in unstable reorientation poses for training waypoint evaluation model (§5.3.2), and stable object pile generation and trajectory evaluation for training trajectory evaluation model (§5.3.3). For the motion planning to generate collision-free trajectory from start and end waypoints, we use RRT-Connect [Kuffner and LaValle, 2000] implemented with OMPL [Sucan et al., 2012] integrating with the collision checking on the physics engine (§2.6).

5.4.1 Evaluation in Simulation

We evaluate the system in 200 unseen piles randomly generated in physics simulation (the test set used in §5.3.3). As the goal state, we randomly assign an object pose on the shelf where the same objects are tightly aligned, as shown in Figure 5.5.

To show the generality and performance variation, we use 2 types of suction grippers in this experiment:

- **I-shape** (Figure 5.5), used in previous work [Zeng et al., 2020a], which has a thin vacuum hose aligned with the gripper;
- **L-shape**, used in our real-world experiments, where the cup axis is translated from the palm for the suction hose.

As the baseline for single-step object reorientation, we designed heuristic reorientation poses that are stable on a plane and make the target grasp point accessible. For simplicity and generality among various objects, we use the upright orientation of objects and Z-axis rotation as the candidate reorientation poses. To ensure target grasp points are accessible after placement in the reorientation poses, we choose

Z-axis orientations that make the target grasp face in the -X direction (facing the robot). Figure 5.5b shows the examples of this heuristic reorientation pose for the goal state in Figure 5.5a, where the front face of the target object (cracker box) faces towards robots with some variations in Z-axis orientation.

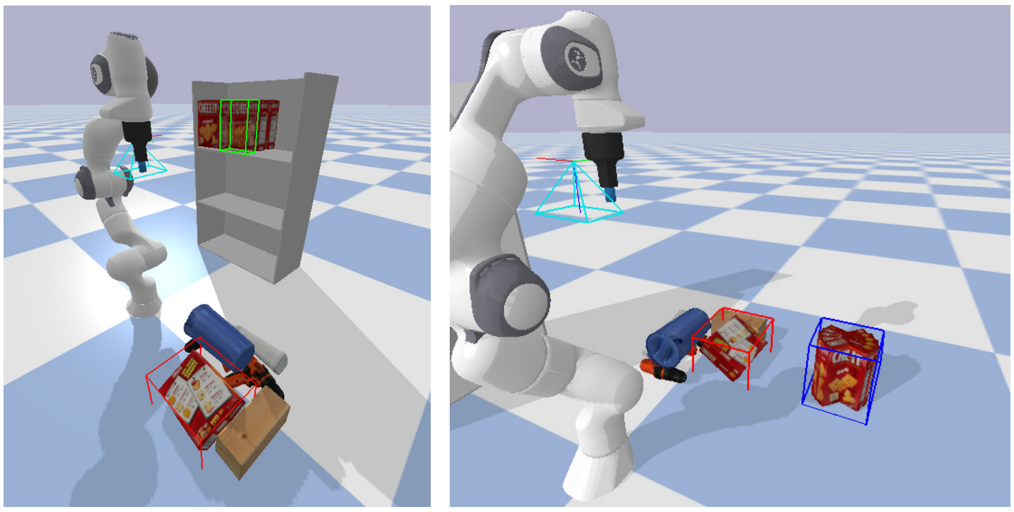

(a) Task configuration (b) Heuristic reorient poses

Figure 5.5: **Evaluation setup in simulation**, tasking the robot to rearrange the target object from the initial state to the goal state.

Task Completion

Table 5.1 shows the comparison of the success rate, whose criteria are the geometric distance between the placed state and the goal state, and with 10 seconds time limit for reorientation planning. We use the area under the curve (AUC) of the point-to-point distance with a threshold between 0 and 10cm, considering AUC>90% as success (the same metric of pose estimation in §3.4). Table 5.1 shows that, compared to the baseline, ReorientBot gives relative improvements of 23-36% in reorientation success, 17-47% in placement success, and 60-81% in overall success. The success rate with the L-shape gripper is lower than the I-shape gripper because of the more extensive gripper base and cup offset from the palm axis, which restricts collision-free arm configurations. The improvements in reorientation and placement show that the proposed learned models play a vital role for motion generation in both stages, improving the overall performance of specific-posed object placement.

Table 5.1: **Task completion**, comparing our method (ReorientBot) with the baseline (Heuristic) with goal configurations unachievable without reorientation (146 tasks).

Gripper	**Method**	**Success% (reorient)↑**	**Success% (place)↑**	**Success% (overall)↑**
I-shape	Heuristic	71.9	81.0	58.2
	ReorientBot	**97.9**	**95.1**	**93.2**
L-shape	Heuristic	74.0	58.3	43.2
	ReorientBot	**91.1**	**85.7**	**78.1**

Timing

Table 5.2 shows the comparison of the planning and execution time of object reorientation between our method and baseline. We measure the planning time with both the wall clock and execution time in the simulation. We send the motion trajectory (a list of joint positions) to the position controller with a constant speed of ~1.2 rad/s (=70 deg/s). As this execution speed can vary in the real world, we also report the trajectory length, which highly correlates to the execution time. The results in Table 5.2 show that our method has improvements of 24-30% in planning time and 20-22% in execution time compared to the baseline.

Table 5.2: **Timing**, comparing our method (ReorientBot) with the baseline (Heuristic). It reports only when both methods successfully complete task; 38/146 tasks in Table 5.1.

Gripper	**Method**	**Planning time [s]↓**	**Execution time [s]↓**	**Trajectory length [rad]↓**
I-shape	Heuristic	3.3	4.0	4.2
	ReorientBot	**2.5**	**3.2**	**3.3**
L-shape	Heuristic	3.0	3.6	3.6
	ReorientBot	**2.0**	**2.8**	**2.7**

5.4.2 Real-world Evaluation

We evaluate our system in the real world with the robotic system introduced in §2.5.1, a Franka Emika Panda robot with an RGB-D camera (Realsense D435) mounted on the arm, and a suction gripper implemented with a vacuum cleaner.

Figure 5.6 shows the sequences of the pick and place motions of the robot for the specified goal configuration on a shelf and in a box. These examples show the capability of our system to reorient objects both successfully and efficiently (with short arm trajectory), utilizing dynamic reorientation. The examples also demonstrate precise placement (e.g., inserting the yellow box into the narrow gap of the drill) and generality in various goal configurations (side and top-down placement).

5.5 Conclusion

In this chapter, we have explored learning-based manipulation in a long-horizon task, working on object placement in a specific pose, which requires several steps of grasping and placement for reorienting and regrasping the objects. Unlike the fine-grained motions in short-horizon tasks in previous chapter, long-horizon tasks require explicit segmentation of motions into small stages (e.g., grasping, reorientation, regrasping, placing) so that the model can focus on learning each segment for convergence, but also require overall motion optimization at the same time. We exploit the semantic map and known object models to segment motions specifying the goal of each segment, such as achieving a certain grasp pose, reorientation pose, and placement pose. This goal specification for each motion segment enables the system to optimize motions against the overall completion of long-horizon tasks (e.g., selecting the best reorientation pose for the succeeding regrasping and placement).

Our system integrates learned motion selection and traditional motion planning to maintain the capabilities of learning (selection of appropriate reorientation poses and efficient trajectories) and generality of planning (flexible trajectory generation based on the goal state and constraints at test time). The resulting system improves the robots' capability in object placement compared to a baseline in both efficiency and success rate and has shown capable object reorientation with significant rotation (e.g., flipping with 180° rotation) using a dynamic placement for reorientation (released and stabilized with gravity) and precise placement in various target configurations (shelf storing, box packing).

By using learned motion selection with coarse waypoints (start, end configuration of a trajectory), we have shown that learned motion models can achieve long-horizon tasks that include several manipulation steps: grasping, reorientation, regrasping,

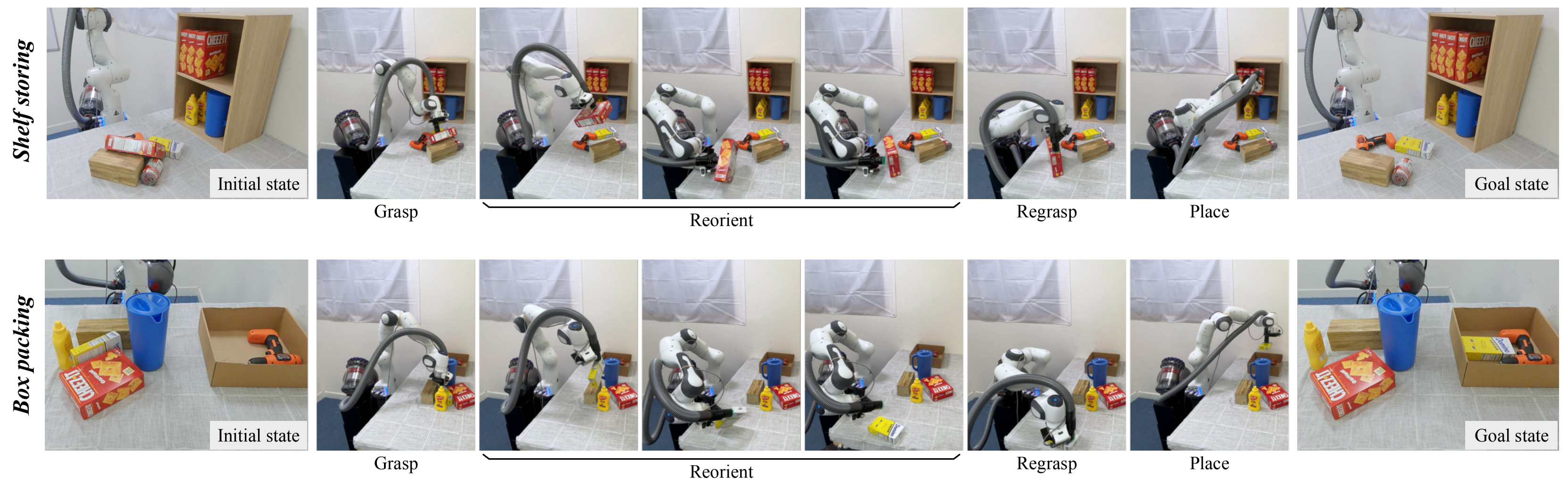


Figure 5.6: **Real-world results**, in which the robot rearranges objects from its initial state in the pile to the specified goal state. The reorient motion includes a dynamic motion to accomplish the orientation that leads to successful placement with a short arm trajectory.

and placement. The semantic information extracted from the semantic map is utilized to generate the intermediate goals in a long-horizon task, which is a different use of the semantics from previous chapter where object semantics were used as rich geometric information to reason physical support and contact among objects.

The two different applications of learning-based manipulation: fine-grained motions in short-horizon tasks, coarse motions in long-horizon tasks, indicate that these approaches can be combined appropriately to achieve even more diverse manipulation tasks. The long-horizon planning gives coarse waypoints of the motion trajectory, and fine-grained motion trajectories are generated based on the waypoints by a motion planner. In this chapter, a traditional motion planner is used to generate fine-grained trajectory from coarse waypoints, but this part can also be replaced by a learned motion planner when the learned capability is necessary as in previous chapter. Regardless of whether traditional motion planning or learned motion planning is used, the object-level semantic map is crucial for long-horizon tasks to divide a task into subtasks with intermediate goals so that overall motion can be optimized via each subtask's optimization.

Chapter 6

Conclusions and Future Work

A number of contributions have been presented in this thesis that aim to create capable robotic manipulation systems through the integration of semantic mapping and motion generation. In particular, we built a real-time semantic mapping system using an on-board RGB-D camera that is able to map objects with their CAD models. This system gives a holistic understanding of scenes and is even able to deal with complex scenes that have overlap, close contact, and support among objects (e.g., object piles). The built semantic map is used to solve various manipulation tasks such as distractor object removal, occluded object extraction, object reorientation, and specific-posed object placement. Hybrid motion generation combining learned and non-learned planning is applied with the explicit semantic map, enabling the robot to perform capable manipulations in tasks that require motion generation to have the properties of both fine-grained motions and long-horizon planning.

In Chapter 3, a semantic mapping pipeline was presented where objects in a scene were incrementally reconstructed with an RGB-D image sequence from a moving camera. The pipeline initially reconstructed objects with occupancy voxel grids by accumulating depth measurements into each object's grid and then replacing the grid with a CAD model after a confident pose estimate was acquired. The pose estimation was accomplished with neural network-based pose prediction and gradient-based pose refinement, both of which exploited the voxel grid reconstruction. Using this vision pipeline, pick-and-place of target objects in a pile was presented where a robot built a semantic map of the scene with an on-board camera and planned collision-free motion trajectory for picking the target objects. During motion planning, objects overlapping with the target were detected as distractors and removed to make the target accessible and pickable without collisions.

At this point, the system was composed in a classical way with traditional (non-learned) collision-based motion planning integrated into a vision pipeline capable of mapping various objects in a cluttered scene with a single moving camera. Although this integrated system showed successful completions of picking known target objects in a pile even with heavy and multi-layered occlusions, the manipulation capability was limited by the motion planner requiring a strictly collision-free path. For changing object states in a scene, the collision-free motion would be the safest maneuver, affecting only the state of the single object the robot is currently interacting with. However, if efficiency is also considered as a task goal, the motion planning would need to be modified to incorporate it as a new metric for achieving both safety and efficiency in task completion. In the case of picking target objects from a pile, exhaustive distractor removal would be time-consuming and inefficient, and target objects could be directly extracted with a carefully planned trajectory that minimizes the effects on other objects in the pile.

To overcome this limitation in motion planning, in Chapter 4, we introduced a learning-based motion planner that generates a trajectory while optimizing the motions for a task metric other than collision avoidance. We specifically tackled target-picking of objects with occlusions in a pile. Robots planned the safest motions to extract the target objects while minimizing effects on the distractors' translations and velocities, which could cause undesirable results such as object damage and expansion of the pile and workspace. In the new system, a learned model predicted 6DoF end-effector transformations, which generated a fine-grained trajectory with recursion, to extract objects with a short path (for efficiency) while minimizing surrounding objects' translations (for safety). Unlike common end-to-end motion models, which map raw image observations to actions, this learned model used object poses in the semantic map along with the raw sensor information from depth images. This combination of rich semantic and geometric information from object poses, and unprocessed and robust information from depth images (no estimation errors included) achieved both high success and robustness.

Although the system in Chapter 4 was capable of performing object extraction with fine-grained motions, the time horizon of the task was limited to be short: a single trajectory to extract the object. By combining with other motions, it would be possible to repeat the process of grasping, extracting, and placing for multiple objects to conduct relatively long-horizon tasks. However, in that case, each motion would need to be separately planned and optimized with the time horizon limited to each motion, and one motion would not necessarily end in the best starting state for the next motion. Long-horizon tasks include several grasping and placing sub-tasks that should be jointly optimized to achieve the best performance. A straightforward application of learning-based motion generation to long-horizon tasks would also not work due to the difficulty of training a model to convergence when the model struggles to find successful trajectories during exploration. It was not clear how long-horizon tasks could be handled by exploiting the capability of learning-based motion generation and the rich information of a semantic map.

In Chapter 5, we explored the application of semantic maps and learned motion generation to long-horizon manipulation tasks. We specifically tackled the placement of objects in specific poses, where robots may need to change the orientation of objects by regrasping so that the final placement is possible. This task requires several steps of grasping and placement: grasping from a pile, reorientation, regrasping, and placement in the target pose, all of which have to be optimized to achieve high levels of success in the overall task. To avoid the combinatorial problem of possible trajectories in fine-grained motion generation, we introduced coarse waypoints that were evaluated with learning models. With these coarse samples of motion waypoints, it was possible for learned models to evaluate their combinations in a longer horizon of tasks. These waypoints determine a rough route for the trajectory generated by the subsequent collision-based motion planner. Combined with a semantic map that allows waypoint sampling, the learned waypoint selection enables the robot to achieve capable object reorientation and regrasping for specific-pose placement.

A straightforward extension of the attempts in Chapter 4 and Chapter 5 would be possible by integrating fine-grained motion generation with long-horizon planning using learning-based methods for both. Though in Chapter 5 we demonstrated learned motions in long-horizon tasks with the fine-grained motions generated by a non-learned motion planner, the entire trajectory could also be generated using learned models. As demonstrated in Chapter 4, learned models are able to optimize motions at training time for efficiency (c.f., brute-force motion sampling) and fuse various inputs from both semantic and raw sensory observation. The two learning approaches of coarse waypoint generation and fine-grained trajectory generation based on the waypoints can be combined to take advantage of both approaches. In the context of manipulation in cluttered scenes, the capability of this new integration can be demonstrated, for example, in a single-step extraction of occluded objects for reorientation and placement, enabling robots to achieve efficient specific-posed placement even with heavily occluded objects.

For developing more capable robotic systems, an important future direction would be the integration of manipulation with navigation. Robotic navigation has been worked on for many years (§1.2) and recently it has acquired significant interest with the rise of self-driving cars. Most of the previous studies on navigation have been working on collision avoidance by setting the research challenge as to how to handle dynamic objects that appear in front of the robots, and navigation in static environments has often been regarded as an easy problem. However, when navigation is integrated with manipulation, even static-environment navigation can be challenging. With manipulation, the policy for navigation would be quite different: navigation *without* manipulation basically maximises the distance from other objects to avoid collisions, whereas navigation *with* manipulation has to take robots close to objects so they can reach them. Aiming at interaction with objects, robots not only have to reach navigation goals efficiently with a short path but also adjust the navigation goal for subsequent manipulation while maintaining a collision-free trajectory, e.g., grasping from certain directions where there are fewer objects with which to collide. When robots manipulate large objects (e.g., door, furniture) [Murooka et al., 2014, Saito et al., 2011], it would also be crucial to change the base location during manipulation for reachability and torque limits.

Manipulation with various end-effectors would also be an interesting direction to work on with advancement in both hardware and software. This thesis has focused on manipulation with suction grippers, which compensate for errors in the alignment and allowed us to simplify the grasping algorithm to focus on post-grasp manipulation. However, suction grippers often struggle to grasp objects without flat surfaces, which motivates the adoption of other types of grippers such as pinch grippers. In addition to expanding the variety of graspable objects, a different gripper structure can give robots more diverse manipulation skills, such as in-hand object manipulation with multi-fingered grippers. Multi-fingered grippers have shown the capability of changing the orientation of box-shaped objects in the real world [Andrychowicz et al., 2020] and various objects in simulation [Chen et al., 2021]. Most of the real-world experiments have been done with a gripper attached on a fixed base (without any manipulator) as the hardware is massive and heavy. It would be a vital step to make the gripper more accessible and applicable to mobile manipulators so that researchers can explore what kind of maneuvers are possible with the gripper using human-like maneuvers (humans change the way they use their hand depending on the task, e.g., grasping and pushing forms are quite different). As a variation of end-effectors, research on tool handling would also be important, where tools are automatically grasped or attached to the existing grippers. Carefully designed tools give extra capability for robots to imitate human activities (e.g., a book stand for sorting, a spatula for cooking, a towel for cleaning) and maneuvers (e.g., pushing, scooping, wiping), potentially substituting for a human-like multi-fingered gripper.

Expanding the capability of semantic mapping to a more diverse set of objects would be vital to enable robots to achieve various and challenging manipulation tasks. The vision pipeline in this thesis (Chapter 3) was limited to a certain set of objects (YCB objects [Calli et al., 2015]) limiting the variations of object shapes and properties; however, robots need to be able to handle more diverse objects to be useful in the real-world environments such as those in warehouses, retail, and households. The object set can have a wide variety of appearances, shape variations among the same object categories (e.g., diversity of mug shapes), articulated objects (e.g., doors, drawers), and deformable objects (e.g., clothes towels). Even with such challenging objects, we believe that explicit semantic understanding of a scene (cf. implicit, end-to-end) would be an important step for capable robotic manipulation, as we showed with the object set in this thesis. The semantic mapping pipeline can be naturally extended to handle more object categories with a larger model database as demonstrated in a certain scale by [Tateno et al., 2016, Avetisyan et al., 2019]. With parameterized object models (e.g., scale, height), the pipeline will be able to handle shape variations optimizing the compact parameters to adjust to the actual objects, as several works have demonstrated with a few object categories and variations [Kundu et al., 2018, Sucar et al., 2020, Runz et al., 2020]. Similar to these parametrized models, articulated objects can also be explicitly modeled with parameters such as joint angles [Li et al., 2020], and the motion generation based on the potentially noisy estimation of these would be interesting future research. Regarding completely deformable objects such as clothes, it is an open question as to how they should be modeled. Although end-to-end, implicit scene understanding could be indispensable for deformable object modeling, the vision pipeline for that can be a hybrid of explicit and implicit modeling. With a once-folded cloth, humans quickly understand the deformation with its fold and angle (explicit understanding), whereas humans probably would not try to estimate such deformation models with a crumpled cloth for manipulation (implicit understanding). The distinction between explicit and implicit scene understanding in human brains could depend on the state of the objects, and future vision pipelines might need to have both properties.

Finally, there has been a lot of research working on intelligent Embodied AI in the computer vision and machine learning communities. Embodied AI covers agents with general intelligence, including both virtual ones (e.g., in games) and physical robots, and it encourages researchers to work on more challenging machine learning problems that are not limited to static datasets but with growing datasets self-collected by agents. Building simulators that are efficient and close to reality would be a key driver to pushing forward this research in building Embodied AI and especially for robots that will be deployed in the real world. There have been several works that build simulation environments that are designed for training robot agents [Kolve et al., 2017, Savva et al., 2019, James et al., 2020, Xiang et al., 2020]; however, still very few real-world task achievements are presented and demonstrations often happen in simulation. This limited real-world deployment is presumably due to the sim-to-real gap of sensory observations and poor performance in long-horizon tasks (making it impossible to maintain the necessary levels of safety and performance in real-world applications). Semantic world models, where raw images are abstracted into a semantic representation, would be crucial for learning long-horizon tasks with high-performance and generalization. It would be an ideal future direction to keep them tightly integrated with explicit semantic mapping and learned motion generation to continue to improve the capability of robots to accomplish useful tasks in the real world.

Bibliography

[Akkaya et al., 2019] Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*. 121

[Andrychowicz et al., 2020] Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *International Journal of Robotics Research (IJRR)*, 39(1):3–20. 121, 141

[Avetisyan et al., 2019] Avetisyan, A., Dahnert, M., Dai, A., Savva, M., Chang, A. X., and Nießner, M. (2019). Scan2CAD: Learning cad model alignment in rgb-d scans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 142

[Baker et al., 2020] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2020). Emergent tool use from multi-agent autocurricula. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 96, 102

[Bar-Itzhack, 2000] Bar-Itzhack, I. Y. (2000). New method for extracting the quaternion from a rotation matrix. *Journal of guidance, control, and dynamics*, 23(6):1085–1087. 42

[Batra et al., 2020] Batra, D., Chang, A. X., Chernova, S., Davison, A. J., Deng, J., Koltun, V., Levine, S., Malik, J., Mordatch, I., Mottaghi, R., Savva, M., and Su, H. (2020). Rearrangement: A challenge for embodied AI. *arXiv preprint arXiv:2011.01975*. 128

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359. 70

[Bekey and Goldberg, 2012] Bekey, G. A. and Goldberg, K. Y. (2012). *Neural networks in robotics*, volume 202. Springer Science & Business Media. 24

[Besl and McKay, 1992] Besl, P. and McKay, N. (1992). A method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256. 70, 87

[Bolles, 1977] Bolles, R. C. (1977). Verification vision for programmable assembly. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 15

[Bolles and Cain, 1982] Bolles, R. C. and Cain, R. A. (1982). Recognizing and locating partially visible objects: The local-feature-focus method. *International Journal of Robotics Research (IJRR)*, 1:57 – 82. 15

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint:1606.01540*. 96

[Burgard et al., 1999] Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114:3–55. 17

[Calli et al., 2015] Calli, B., Singh, A., Walsman, A., Srinivasa S. and, Abbeel, P., and Dollar, A. M. (2015). The YCB object and Model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics (ICAR)*, pages 510–517. 54, 72, 84, 104, 119, 129, 141

[Chatila and Laumond, 1985] Chatila, R. and Laumond, J.-P. (1985). Position referencing and consistent world modeling for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 17, 18

[Chen et al., 2021] Chen, T., Xu, J., and Agrawal, P. (2021). A system for general in-hand object re-orientation. In *Conference on Robot Learning (CoRL)*. 141

[Chitta et al., 2017] Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Tsouroukdissian, A. R., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M., et al. (2017). ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*, 2(20):456–456. 51

[Chitta et al., 2012] Chitta, S., Sucan, I., and Cousins, S. (2012). Moveit! *IEEE Robotics and Automation Magazine*. 21

[CMU, 2003] CMU, C. M. U. (2003). Unimate. 13

[Cole et al., 1992] Cole, A. A., Hsu, P., and Sastry, S. S. (1992). Dynamic control of sliding by robot hands for regrasping. *IEEE Transactions on Robotics and Automation*. 121

[Collet et al., 2009] Collet, A., Berenson, D., Srinivasa, S. S., and Ferguson, D. (2009). Object recognition and full pose registration from a single image for robotic manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 21, 22

[Collet et al., 2011] Collet, A., Martinez, M., and Srinivasa, S. S. (2011). The moped framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research (IJRR)*, 30(10):1284 – 1306. 21, 22, 70

[Coumans et al., 2013] Coumans, E. et al. (2013). Bullet physics library. *Open source: bulletphysics. org*. 54, 85, 104, 106, 130

[Dafle et al., 2014] Dafle, N. C., Rodriguez, A., Paolini, R., Tang, B., Srinivasa, S. S., Erdmann, M., Mason, M. T., Lundberg, I., Staab, H., and Fuhlbrigge, T. (2014). Extrinsic dexterity: In-hand manipulation with external forces. In

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 121

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 24

[Devin et al., 2018] Devin, C. M., Jang, E., Levine, S., and Vanhoucke, V. (2018). Grasp2Vec: Learning object representations from self-supervised grasping. In *Conference on Robot Learning (CoRL)*. 27, 95, 98

[Devol, 1961] Devol, G. C. (1961). U.S. Patent 2,988,237. 13

[Diankov and Kuffner, 2008] Diankov, R. and Kuffner, J. (2008). Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*. 21, 28

[Ernst, 1962] Ernst, H. A. (1962). MH-1, a computer-operated mechanical hand. In *Proceedings of the Spring Joint Computer Conference*. 14

[Fang et al., 2018] Fang, K., Bai, Y., Hinterstoisser, S., Savarese, S., and Kalakrishnan, M. (2018). Multi-task domain adaptation for deep learning of instance grasping from simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 98

[Feldman et al., 1969] Feldman, J., Feldman, G. M., Falk, G., Grape, G., Pearlman, J., Sobel, I., and Tenenbaum, J. M. (1969). The stanford hand-eye project. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 14

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395. 70

[Galindo et al., 2008] Galindo, C., Fernández-Madrigal, J., González, J., and Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966. 21

[Gao and Tedrake, 2019] Gao, W. and Tedrake, R. (2019). kPAM-SC: Generalizable manipulation planning using keypoint affordance and shape completion. *arXiv preprint arXiv:1909.06980*. 94

[Georgeff and Lansky, 1987] Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 17, 18

[Goertz, 1952] Goertz, R. C. (1952). Fundamentals of general-purpose remote manipulators. *Nucleonics*, 10(11):36–42. 13

[Goertz, 1964] Goertz, R. C. (1964). Manipulator systems developed at anl. In *Proceedings of the Conference on Remote Systems Technology*. 13

[Hasegawa et al., 2019] Hasegawa, S., Wada, K., Kitagawa, S., Uchimi, Y., Okada, K., and Inaba, M. (2019). GraspFusion: Realizing complex motion by learning and fusing grasp modalities with instance segmentation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 119

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 23, 67, 72

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 77

[Hinterstoisser et al., 2012a] Hinterstoisser, S., Cagniart, C., Ilic, S., Sturm, P., Navab, N., Fua, P., and Lepetit, V. (2012a). Gradient Response Maps for Real-Time Detection of Texture-Less Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. 69

[Hinterstoisser et al., 2011] Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolidge, K., Navab, N., and Lepetit, V. (2011). Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 69

[Hinterstoisser et al., 2012b] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012b). Model-Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*. 69

[Hirukawa et al., 2004] Hirukawa, H., Kanehiro, F., Kaneko, K., Kajita, S., Fujiwara, K., Kawai, Y., Tomita, F., Hirai, S., Tanie, K., Isozumi, T., et al. (2004). Humanoid robotics platforms developed in hrp. *Robotics and Autonomous Systems*, 48(4):165–175. 19

[Holzer et al., 2012] Holzer, S., Shotton, J., and Kohli, P. (2012). Learning to efficiently detect repeatable interest points in depth data. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 70

[Hornung et al., 2013] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*. 73

[Hunt et al., 1992] Hunt, K. J., Sbarbaro, D., Żbikowski, R., and Gawthrop, P. J. (1992). Neural networks for control systems—a survey. *Automatica*, 28(6):1083–1112. 24

[Huttenlocher et al., 1993] Huttenlocher, D. P., Klanderman, G. A., and Rucklidge, W. J. (1993). Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(9):850–863. 69

[IFR, 2020] IFR, I. F. o. R. (2020). Ifr presents world robotics report 2020. 13

[Ilg et al., 2017] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. (2017). FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 60

[Inoue, 1971] Inoue, H. (1971). Computer controlled bilateral manipulator. *Jsme International Journal Series B-fluids and Thermal Engineering*, 14:199–207. 14

[James et al., 2017] James, S., Davison, A. J., and Johns, E. (2017). Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task . In *Conference on Robot Learning (CoRL)*. 27

[James et al., 2020] James, S., Ma, Z., Arrojo, D. R., and Davison, A. J. (2020). RL-Bench: The robot learning benchmark and learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026. 143

[James et al., 2021a] James, S., Wada, K., Laidlow, T., and Davison, A. J. (2021a). Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. *Under Review and Submitted to Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 32

[James et al., 2021b] James, S., Wada, K., Laidlow, T., and Davison, A. J. (2021b). Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. *arXiv preprint arXiv:2106.12534*. 98, 117

[Johns et al., 2016] Johns, E., Leutenegger, S., and Davison, A. J. (2016). Deep learning a grasp function for object manipulation under gripper pose uncertainty. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 27

[Jonschkowski et al., 2016] Jonschkowski, R., Eppner, C., Höfer, S., Martín-Martín, R., and Brock, O. (2016). Probabilistic multi-class segmentation for the amazon picking challenge. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 22, 119

[Kahler et al., 2015] Kahler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P. H. S., and Murray, D. W. (2015). Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 49

[Kalashnikov et al., 2018] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*. 27, 30, 95, 98, 117, 119

[Kanehiro et al., 2002] Kanehiro, F., Fujiwara, K., Kajita, S., Yokoi, K., Kaneko, K., Hirukawa, H., Nakamura, Y., and Yamane, K. (2002). Open architecture humanoid robotics platform. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 19

[Keselman et al., 2017] Keselman, L., Iselin Woodfill, J., Grunnet-Jepsen, A., and Bhowmik, A. (2017). Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 43, 109

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 63, 105, 130

[Kolve et al., 2017] Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A. K., and Farhadi, A. (2017). AI2-THOR: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*. 143

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*. 24

[Kuffner and LaValle, 2000] Kuffner, J. J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 55, 106, 130

[Kundu et al., 2018] Kundu, A., Li, Y., and Rehg, J. M. (2018). 3D-RCNN: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 142

[Kurenkov et al., 2020] Kurenkov, A., Taglic, J., Kulkarni, R., Dominguez-Kuhne, M., Garg, A., Martín-Martín, R., and Savarese, S. (2020). Visuomotor mechanical search: Learning to retrieve target objects in clutter. *arXiv preprint arXiv:2008.06073*. 98

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 58

[Lenz et al., 2013] Lenz, I., Lee, H., and Saxena, A. (2013). Deep learning for detecting robotic grasps. In *Proceedings of Robotics: Science and Systems (RSS)*. 25

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1). 25, 98, 117

[Levine et al., 2018] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research (IJRR)*, 37(4-5):421–436. 27, 30, 95, 98, 117, 119

[Li et al., 2020] Li, X., Wang, H., Yi, L., Guibas, L. J., Abbott, A. L., and Song, S. (2020). Category-level articulated object pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 142

[Li et al., 2017] Li, Y., Qi, H., Dai, J., Ji, X., and Wei, Y. (2017). Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 67

[Lowe, 2001] Lowe, D. (2001). Local Feature View Clustering for 3D Object Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 69

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110. 70

[Lozano-Pérez and Kaelbling, 2014] Lozano-Pérez, T. and Kaelbling, L. P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 121

[Mamou and Ghorbel, 2009] Mamou, K. and Ghorbel, F. (2009). A simple and efficient approach for 3d mesh approximate convex decomposition. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 54

[Manuelli et al., 2019] Manuelli, L., Gao, W., Florence, P., and Tedrake, R. (2019). kPAM: Keypoint affordances for category-level robotic manipulation. *Proceedings of the International Symposium on Robotics Research (ISRR)*. 94, 119

[Matsui and Inaba, 1990] Matsui, T. and Inaba, M. (1990). Euslisp: An object-based implementation of lisp. *Journal of Information Processing*, 13(3):327–338. 19

[McCormac et al., 2018] McCormac, J., Clark, R., Bloesch, M., Davison, A. J., and Leutenegger, S. (2018). Fusion++:volumetric object-level slam. In *Proceedings of the International Conference on 3D Vision (3DV)*. 23, 71, 72

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*. 99

[Mur-Artal and Tardós, 2014] Mur-Artal, R. and Tardós, J. D. (2014). ORB-SLAM: Tracking and Mapping Recognizable Features. In *Workshop on Multi View Geometry in Robotics (MVIGRO) - RSS 2014*. 72

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics (T-RO)*, 33(5):1255–1262. 72

[Murooka et al., 2014] Murooka, M., Noda, S., Nozawa, S., Kakiuchi, Y., Okada, K., and Inaba, M. (2014). Manipulation strategy decision and execution based on strategy proving operation for carrying large and heavy objects. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 141

[Nakamura and Hanafusa, 1986] Nakamura, Y. and Hanafusa, H. (1986). Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement, and Control*, 108(3):163–171. 53

[Newcombe et al., 2011] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 23

[Nießner et al., 2013] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*. 49

[Nilsson, 1969] Nilsson, N. J. (1969). A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 14

[Nilsson, 1984] Nilsson, N. J. (1984). Shakey the robot. 16

[Nister and Stewenius, 2006] Nister, D. and Stewenius, H. (2006). Scalable Recognition with a Vocabulary Tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 69

[Okada et al., 2004a] Okada, K., Haneda, A., Nakai, H., Inaba, M., and Inoue, H. (2004a). Environment manipulation planner for humanoid robots using task graph that generates action sequence. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 19

[Okada et al., 2007] Okada, K., Kojima, M., Tokutsu, S., Maki, T., Mori, Y., and Inaba, M. (2007). Multi-cue 3d object recognition in knowledge-based vision-guided humanoid robot system. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 19, 20, 28

[Okada et al., 2005] Okada, K., Ogura, T., Haneda, A., Fujimoto, J., Gravot, F., and Inaba, M. (2005). Humanoid motion generation system on hrp2-jsk for daily life environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 19, 20

[Okada et al., 2004b] Okada, K., Ogura, T., Haneda, A., Kousaka, D., Nakai, H., Inaba, M., and Inoue, H. (2004b). Integrated system software for hrp2 humanoid. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 19

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 104, 130

[Petersson et al., 2002] Petersson, L., Jensfelt, P., Tell, D., Strandberg, M., Kragic, D., and Christensen, H. I. (2002). Systems integration for real-world manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 18

[Philbin et al., 2007] Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 69

[Pinto and Gupta, 2016] Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Proceedings* *of the IEEE International Conference on Robotics and Automation (ICRA)*. 27, 98, 119

[Pomerleau, 1989] Pomerleau, D. A. (1989). ALVINN: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY. 24

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660. 70, 77

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5. 21, 51, 109

[Riegler et al., 2017] Riegler, G., Osman Ulusoy, A., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 49

[Rios-Cabrera and Tuytelaars, 2013] Rios-Cabrera, R. and Tuytelaars, T. (2013). Discriminatively trained templates for 3d object detection: A real time scalable approach. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 69

[Roberts, 1963] Roberts, L. G. (1963). *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology. 14

[Rohrdanz and Wahl, 1997] Rohrdanz, F. and Wahl, F. M. (1997). Generating and evaluating regrasp operations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 121

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 60

[Rosten et al., 2008] Rosten, E., Porter, R., and Drummond, T. (2008). Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(1):105–119. 70

[Rowley et al., 1998] Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(1):23–38. 25

[Runz et al., 2018] Runz, M., Buffier, M., and Agapito, L. (2018). MaskFusion: Real-time recognition, tracking and reconstruction of multiple moving objects. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 23

[Runz et al., 2020] Runz, M., Li, K., Tang, M., Ma, L., Kong, C., Schmidt, T., Reid, I., Agapito, L., Straub, J., Lovegrove, S., et al. (2020). FroDO: From detections to 3d objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 142

[Saito et al., 2011] Saito, M., Chen, H., Okada, K., Inaba, M., Kunze, L., and Beetz, M. (2011). Semantic object search in large-scale indoor environments. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems Workshops (IROSW)*. 141

[Salas-Moreno et al., 2013] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 23

[Savva et al., 2019] Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A Platform for Embodied AI Research. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 143

[Saxena et al., 2006] Saxena, A., Driemeyer, J., Kearns, J., and Ng, A. Y. (2006). Robotic grasping of novel objects. In *Neural Information Processing Systems (NIPS)*. 24, 26

[Saxena et al., 2008a] Saxena, A., Driemeyer, J., Kearns, J., Osondu, C., and Ng, A. Y. (2008a). Learning to grasp novel objects using vision. In *Experimental Robotics*. 24

[Saxena et al., 2008b] Saxena, A., Driemeyer, J., and Ng, A. Y. (2008b). Robotic grasping of novel objects using vision. *International Journal of Robotics Research (IJRR)*, 27(2):157–173. 24, 25

[Schwarz et al., 2018] Schwarz, M., Lenz, C., García, G. M., Koo, S., Periyasamy, A. S., Schreiber, M., and Behnke, S. (2018). Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 22

[Shirai, 1987] Shirai, Y. (1987). Three-dimensional computer vision. In *Symbolic Computation*. 15

[Shirai and Inoue, 1973] Shirai, Y. and Inoue, H. (1973). Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5:99–106. 15

[Shome et al., 2019] Shome, R., Tang, W. N., Song, C., Mitash, C., Kourtev, H., Yu, J., Boularias, A., and Bekris, K. E. (2019). Towards robust product packing with a minimalistic end-effector. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 119

[Simpson, 2005] Simpson, R. C. (2005). Smart wheelchairs: A literature review. *Journal of rehabilitation research and development*, 42 4:423–36. 17

[Steger, 2001] Steger, C. (2001). Similarity measures for occlusion, clutter, and illumination invariant object recognition. In *Joint Pattern Recognition Symposium*. 69

[Stevšić et al., 2020] Stevšić, S., Christen, S., and Hilliges, O. (2020). Learning to assemble: Estimating 6D poses for robotic object-object manipulation. *IEEE Robotics and Automation Letters*, 5(2):1159–1166. 94

[Stückler and Behnke, 2012] Stückler, J. and Behnke, S. (2012). Model learning and real-time tracking using multi-resolution surfel maps. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 23

[Sucan et al., 2012] Sucan, I. A., Moll, M., and Kavraki, L. E. (2012). The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19(4):72–82. 21, 55, 130

[Sucar et al., 2020] Sucar, E., Wada, K., and Davison, A. J. (2020). NodeSLAM: Neural object descriptors for multi-view shape reconstruction. In *Proceedings of the International Conference on 3D Vision (3DV)*. 32, 142

[Sun et al., 2018] Sun, D., Yang, X., Liu, M.-Y., and Kautz, J. (2018). PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 60

[Sünderhauf et al., 2017] Sünderhauf, N., Pham, T. T., Latif, Y., Milford, M., and Reid, I. (2017). Meaningful maps with object-oriented semantic mapping. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 71

[Tang et al., 2012] Tang, J., Miller, S., Singh, A., and Abbeel, P. (2012). A textured object recognition pipeline for color and depth image data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 70

[Tateno et al., 2016] Tateno, K., Tombari, F., and Navab, N. (2016). When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense slam. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 23, 142

[Taylor and Kleeman, 2003] Taylor, G. and Kleeman, L. (2003). Fusion of multimodal visual cues for model-based object tracking. In *Australasian Conference on Robotics and Automation (ACRA)*. 18

[Tenorth et al., 2010] Tenorth, M., Kunze, L., Jain, D., and Beetz, M. (2010). Knowrob-map-knowledge-linked semantic object maps. In *IEEE-RAS International Conference on Humanoid Robots*. 21

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 27

[Tournassoud et al., 1987] Tournassoud, P., Lozano-Pérez, T., and Mazer, E. (1987). Regrasping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 117, 121

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Neural Information Processing Systems (NIPS)*. 102

[Vespa et al., 2018] Vespa, E., Nikolov, N., Grimm, M., Nardi, L., Kelly, P. H., and Leutenegger, S. (2018). Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters*. 49

[Wada et al., 2022a] Wada, K., James, S., and Davison, A. J. (2022a). ReorientBot: Learning object reorientation for specific-posed placement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 31

[Wada et al., 2022b] Wada, K., James, S., and Davison, A. J. (2022b). SafePicking: Learning safe object extraction via object-level mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 29

[Wada et al., 2018] Wada, K., Kitagawa, S., Okada, K., and Inaba, M. (2018). Instance segmentation of visible and occluded regions for finding and picking target from a pile of objects. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 22

[Wada et al., 2017] Wada, K., Okada, K., and Inaba, M. (2017). Probabilistic 3d multilabel real-time mapping for multi-object manipulation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 119

[Wada et al., 2020] Wada, K., Sucar, E., James, S., Lenton, D., and Davison, A. J. (2020). MoreFusion: Multi-object reasoning for 6D pose estimation from volumetric fusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 28, 72

[Wan and Harada, 2016a] Wan, W. and Harada, K. (2016a). Achieving high success rate in dual-arm handover using large number of candidate grasps, handover heuristics, and hierarchical search. *Advanced Robotics*, 30(17-18):1111–1125. 121

[Wan and Harada, 2016b] Wan, W. and Harada, K. (2016b). Developing and comparing single-arm and dual-arm regrasp. *IEEE Robotics and Automation Letters*. 121

[Wan et al., 2019] Wan, W., Igawa, H., Harada, K., Onda, H., Nagata, K., and Yamanobe, N. (2019). A regrasp planning component for object reorientation. *Autonomous Robots*, 43(5):1101–1115. 117, 121

[Wang et al., 2019] Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., and Savarese, S. (2019). DenseFusion: 6D object pose estimation by iterative dense fusion. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 67, 70, 77, 78, 79, 84, 85

[Wichman, 1967] Wichman, W. M. (1967). Use of optical feedback in the computer control of an arm. Technical report, STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE. 14

[Wurm et al., 2010] Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). OctoMap: A Probabilistic, Flexible, and Compact 3D Map

Representation for Robotic Systems. In *Proceedings of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*. 49

[Xiang et al., 2020] Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al. (2020). Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 143

[Xiang et al., 2018] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Proceedings of Robotics: Science and Systems (RSS)*. 67, 70, 72, 78, 84, 85, 129

[Xu et al., 2019] Xu, B., Li, W., Tzoumanikas, D., Bloesch, M., Davison, A., and Leutenegger, S. (2019). MID-Fusion: Octree-based object-level multi-instance dynamic slam. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 23, 71, 72

[Xu et al., 2018] Xu, D., Anguelov, D., and Jain, A. (2018). PointFusion: Deep sensor fusion for 3D bounding box estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 70, 77

[Zakka et al., 2020] Zakka, K., Zeng, A., Lee, J., and Song, S. (2020). Form2fit: Learning shape priors for generalizable assembly from disassembly. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 94

[Zender et al., 2008] Zender, H., Mozos, O. M., Jensfelt, P., Kruijff, G.-J., and Burgard, W. (2008). Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502. 21

[Zeng et al., 2020a] Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., et al. (2020a). Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning (CoRL)*. 130

[Zeng et al., 2020b] Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2020b). Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics (T-RO)*. 98

[Zeng et al., 2018a] Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2018a). Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. 98

[Zeng et al., 2018b] Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al. (2018b). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 98, 119

[Zeng et al., 2017] Zeng, A., Yu, K.-T., Song, S., Suo, D., Walker, E., Rodriguez, A., and Xiao, J. (2017). Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 22, 94, 119

[Zhao et al., 2017] Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 60, 77