Learning Task-Specific Grasps

by

Matthew Stephen Corsaro

B. S., Rensselaer Polytechnic Institute, 2015

M. S., Northeastern University, 2017

A dissertation submitted in partial fulfillment of the

requirements for the Degree of Doctor of Philosophy

in the Department of Computer Science at Brown University

Providence, Rhode Island

May 28, 2023

This dissertation by Matthew Stephen Corsaro is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ______________________________
George Konidaris, Director

Recommended to the Graduate Council

Date ______________ ______________________________
Stefanie Tellex, Reader

Date ______________ ______________________________
Srinath Sridhar, Reader

Approved by the Graduate Council

Date ______________ ______________________________
Thomas A. Lewis
Dean of the Graduate School

Abstract

Abstract of "Learning Task-Specific Grasps" by Matthew Stephen Corsaro, Ph.D., Brown University, May 28, 2023.

Grasping is one of the most important open problems in robotics; the very point of a robot is to exert force on the world to achieve a goal, and most such exertions require the robot to execute a grasp first. For a home robot to be effective, it must load a dishwasher with breakable plates; for a repair robot to be effective, it must operate tools; for a caretaker robot to be effective, it must perform chores for those with illnesses. All of these activities require manipulating objects, which in turn requires grasping them effectively. Additionally, to be useful, the robot must be able to perform these tasks on objects it has never seen before, in applications where manipulation failures can be very costly. Deploying a robot to such an environment, where exact operating conditions are unknown and vary between instances, is therefore challenging because systems and algorithms developed in a lab may perform poorly when introduced to a novel environment. A robot must quickly learn to manipulate new objects it encounters using limited prior knowledge.

In this dissertation, I examine robot grasping in three contexts. First, I propose a general grasp detection system that enables a multi-finger gripper to use multiple types of grasps to pick objects of varying sizes from dense clutter. For example, precision grasps are necessary for precisely picking small objects from the surface of a table using fingertips, while power grasps stably hold large objects by enveloping them with the gripper's fingers. Given a visual representation of the scene, the system proposes a set of potential candidate grasp poses. These poses are evaluated using a neural network model that takes as input point clouds centered at a grasp pose and returns the probabilities that a grasp of each type would succeed at the given pose. This system is trained using a dataset generated in simulation and evaluated on a real robot. Explicitly modeling grasp type boosted the system's object removal rate by 8.5% over the highest performing baseline.

Next, I propose a framework for specializing a generic grasp detector to a task-oriented grasp detector. A generic grasp detector detects a stable grasp, which is sufficient for picking up an object but may not be sufficient for manipulating it. For example, a stable grasp very close to the fulcrum of a door handle will make it hard to turn, while grasping far from the fulcrum will make it easier. A task-oriented grasp detector is a classifier that predicts which grasp poses serve as initial states that enable a given manipulation controller to complete a task. As these classifiers are instance dependent, they cannot be trained in simulation and transferred to the real world. Instead, they must be trained directly in the task for which they are required. To this end, I introduce the Augmented Task-Oriented Grasp Detection Network (ATOG), which learns to predict which grasp poses allow a robot to successfully manipulate an object from a single-digit-sized training set. ATOG achieves this via a deep architecture built on an existing network that has been pre-trained to predict general grasp stability. Given a partial point cloud containing the local geometry around a grasp pose and the pose's relation to the object, ATOG predicts whether the grasp will enable the robot to successfully execute a motor skill. I evaluate ATOG in four simulated domains; it outperforms the nearest baseline by up to 6.5%.

Finally, I propose a learning algorithm that learns a task-oriented grasp detector for a given task while simultaneously learning the manipulation policy that the grasp must enable. Learning a policy to control a robot to perform a specific task is difficult because of the large action space and potentially sparse reward signal. This learning process can be simplified by bootstrapping the policy with a grasp controller and learning after a grasp has been executed. For instance, a robot would execute a grasp on the back side of the handle of a hammer, then learn a control policy that raised the hammer over a nail and struck the head down onto the nail. Though bootstrapping with a grasp controller simplifies the policy learning process, a task-oriented grasp classifier still must learn which grasp poses enable the policy to succeed. This joint learning problem is challenging due to the entanglement between the task-oriented grasp detector and the manipulation policy, which changes over time as it is learned; selecting different grasps changes the initial states of the manipulation policy, while a grasp pose that one policy fails the task from could enable an updated policy to complete the task. My proposed Grasp-Aware Reinforcement-Learning Agent (GARLA) overcomes a key obstacle to robot learning with grasping, enabling a robot to quickly learn both how to manipulate an object and where to grasp the object to begin the manipulation. With GARLA, a robot could be deployed to a novel environment and learn to manipulate novel objects within a small number of attempts.

Acknowledgements

Working with George Konidaris has been the most fun and rewarding academic experience in all my years of schooling. His ability to create a long-term vision, his determination to see it through, and his technical prowess to solve difficult problems are inspiring. I am excited to see how he and the rest of the lab continue to make steady progress towards accomplishing these goals. George had insightful suggestions when I hit a wall, and his kindness and tact were always appreciated. I am eternally grateful for his wisdom, support, and encouragement.

Stefanie Tellex's infectious enthusiasm for making robots do something new was a continuous source of motivation for me. Her passion for robotics, desire to empower all people with collaborative robots, and selfless community outreach efforts left a lasting impression on me. Stefanie, along with George and Michael Littman, fostered a wonderful and welcoming lab environment that I looked forward to returning to every day and I will miss dearly. This environment was able to flourish because of the similar culture found throughout the Computer Science Department. I thank the faculty, Lauren and the Astaff, and John and the Tstaff for their support over the years. The robotics lab would not function without Suzanne, I am forever grateful for her hard work.

I am extremely grateful for the guidance provided by my thesis and research comps committee members, Srinath Sridhar and Daniel Ritchie. Though I was only able to work with Srinath in the latter years of my Ph.D., our discussions about research and the job search process were enlightening; his questions and insights during my thesis proposal helped to prepare me for the defense. Daniel's expertise in 3D learning and suggestions for point-cloud-based neural network architectures were of paramount importance for the success of the MMGPD system I describe in Chapter 3. I am grateful for his advice during the research comp process, as well as his support during my thesis proposal and defense.

This dissertation would not have been possible without the hard work of my collaborators. I would like to thank Akhil, Ben, Sree, and Omer for their assistance with the GARLA system introduced in Chapter 5. Akhil and Omer's reinforcement-learning expertise and Ben and Sree's assistance with the robot learning experiments were crucial.

My favorite thing about working at Brown was having the privilege to work alongside so many kind, bright, and talented people. From discussing problems to new papers, lunch plans, video games, or graduate school life, spending time with everyone in the robotics lab was a joy. Thank you to Ben, Ben, Barrett, Kavosh, Nakul, Max, Cam, Lawson, Dave, Yuu, Steve, Andrew, Akhil, Thao, Eric, David, Michael, Jason, Kaiyu, Ifrah, Lucas, Sam, Sam, Saket, Raph, Tuluhan, Anita, Shane, Benedict, Skye, Tasha, Jess, Nihal, Bingjie, Nishanth, Seiji, Kyra, Eren, Sree, Vanya, Arthur, Eddie, Atsu, Sidd, Dilip, Jonathan, Josh, Willie, Waleed, Chan, and everyone else who made the lab feel like home.

I also owe a great deal of thanks to my past advisors and collaborators for helping me gain the necessary experience to begin this journey. I am grateful to Qian, Yaro, Paul, Alperen, Leif, and everyone else in Rob Howe's lab who helped introduce me to robotics during my 2014 internship. I appreciate Rob's kindness in taking in an unaffiliated student for the summer, for his guidance, and for submitting letters of recommendation during my master's and Ph.D. application processes. My interest in robot grasping and manipulation was further ingrained during my master's with Rob Platt. I am grateful to Rob for allowing me to work closely with Marcus and Andreas on the Grasp Pose Detection project, a framework that much of the work in this dissertation builds off of. I learned so much about deep learning, robot debugging, and the academic process from Rob, Marcus, and Andreas, as well as Uli, Colin, Sammie, Yuchen, and Chris. I am also grateful for the mentorship of

Neil, Brad, and Radu during my various summer internships that were invaluable opportunities to dive into different problems in robotics.

Finally, a most heartfelt thank you to my family and friends for your consistent support over the years. I could not have done it nor dreamed of making it this far without you. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robots have great potential to change society. Increasingly intelligent automation will enable robots to perform mundane or dangerous industrial tasks, and increase the rate at which items are shipped from warehouses. More importantly, robots can help people. Care robots will assist the elderly or those with disabilities to complete routine tasks, enabling them to be more independent of others. In order to complete the challenging manipulation tasks posed by industry and best assist humans in varying home contexts, robots must quickly learn to operate in new environments, developing robust policies to interact with different objects. For instance, home assistive robots must open cabinets, put away groceries or clothes, prepare food, and load a dishwasher with breakable plates and glasses. These tasks are difficult because they require precise multi-step manipulation policies where the robot must stably grasp a variety of objects to avoid dropping them or allowing them to slip while manipulating them; dropping a salt shaker into a pot of boiling water and being unable to retrieve it would ruin dinner.

Learning to stably grasp objects and learning to manipulate after grasping are both difficult but important sub-problems that are often studied independently. Grasping in and of itself is a challenging and important open problem in robotics. It is the most essential skill for pick and place tasks and a prerequisite for many other manipulation tasks, such as tool use [42]. Grasping is often

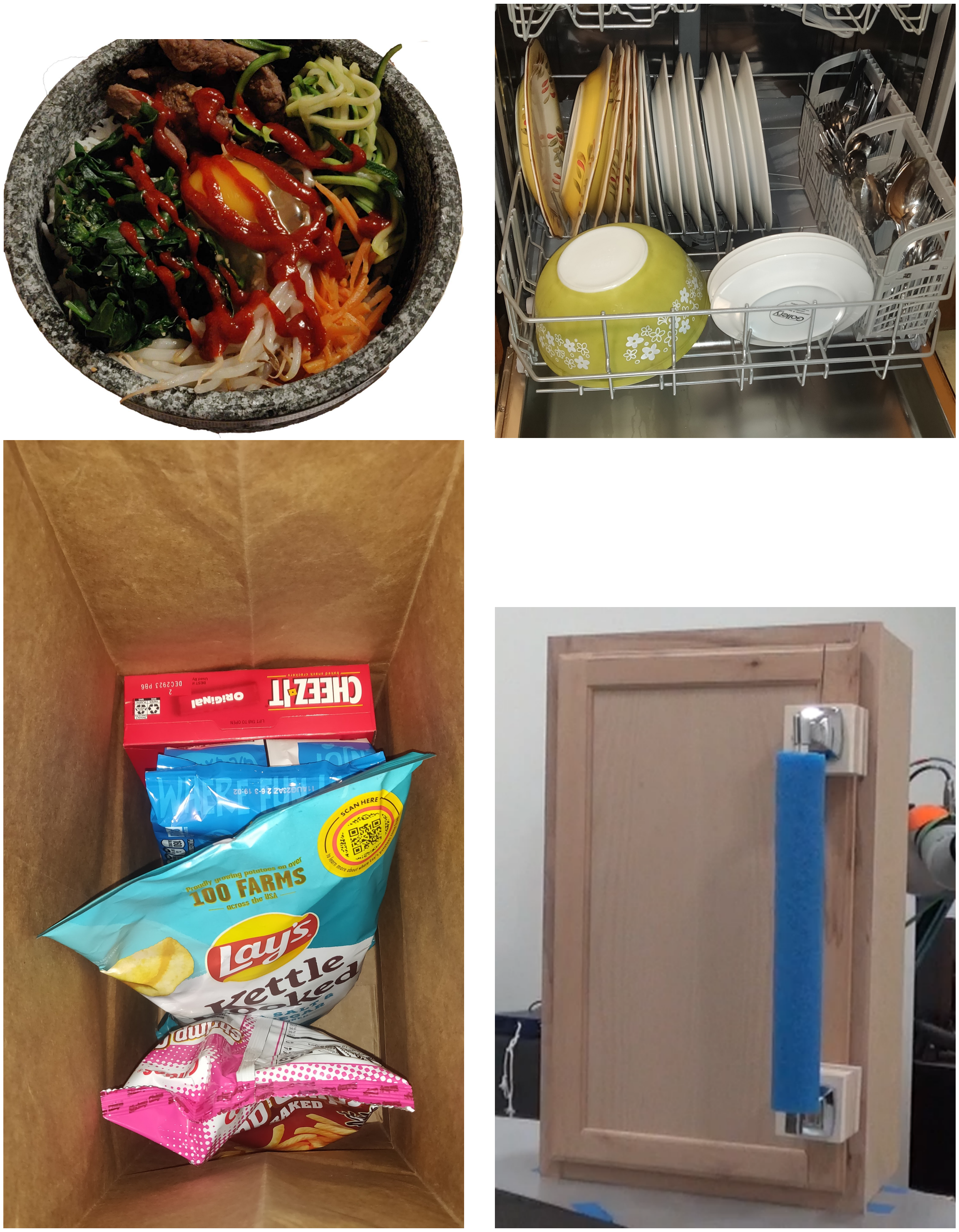


Figure 1.1: Several difficult household manipulation tasks assistive home robots must complete. They must prepare complex meals, load and unload a dishwasher with delicate plates and glasses, unpack a dense bag of groceries, and manipulate articulated objects such as cabinet doors.

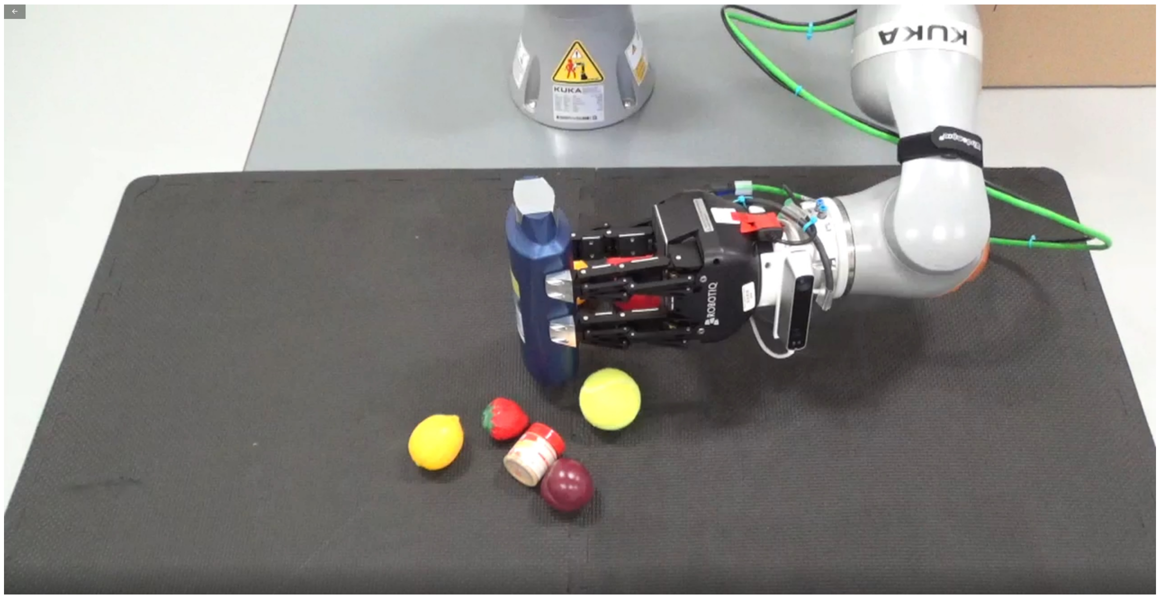


Figure 1.2: A robot grasping an object from a cluttered scene in order to place it in a box.

studied in the context of picking. It is treated as a computer vision problem, often called grasp detection, where algorithms return grasp poses and the robot executes a grasp at the desired pose with a simple policy [95]. As many robots are equipped with localized vision sensors, the they can use motion planning to move their end effectors directly to a grasp pose with a Cartesian pose command [35].

Grasp detection can be extended past the simple pick and place domain to perform task-oriented grasping. Task-oriented grasps are grasps that enable a robot to perform some downstream task besides pick and place after grasping an object. Task-oriented grasps are executed with a simple policy that moves the end effector to the Cartesian pose specified by the grasp; the robot uses a more complex policy or controller to complete the task. For instance, a robot must grasp a door handle to open the door, grasp a spatula to flip a pancake, or grasp a pitcher to pour liquid. After executing a grasp to complete these tasks, a more complex manipulation controller is required to perform the opening, flipping, or pouring actions. Importantly, stable grasps that enable a robot to pick and place an object may not enable it to complete a given task; a task-oriented grasp must stably hold the object, but must also afford the robot the ability to complete the task. The grasp cannot cause the robot to obstruct a segment of the object necessary to complete the task, and must allow the robot to properly manipulate the object after grasping as required. For instance, when tasked with pouring liquid from a bottle, a grasp on the top of the bottle could enable a two-armed robot to stably lift it, but would obstruct the robot from unscrewing the cap with its other gripper once lifted. The sub-task of opening the bottle, however, would require a grasp on the cap. Task-oriented grasps for pouring from the bottle must again leave the top unobstructed. Stable grasps that enable the robot to complete one task may prove unsuitable for other tasks. In the case of a robot that must hammer a nail, appropriate task-oriented grasps would be located along the handle of the hammer so its head would be free to hit the nail. These grasps must enable the robot to strike the face of the hammer downwards, further constraining the set of potential task-oriented grasps. If a multi-finger robot was tasked with using a more complex tool such as a drill, not only would it have to grasp the back side of the handle to properly position the tip, it would also require a free finger to operate the trigger. When selecting a task-oriented grasp, the robot must consider the stability of a given grasp as well as whether the grasp enables it to complete the given task.

After executing a task-oriented grasp, the robot must learn a policy to complete the task. Unlike the simple controllers that could enable robots to grasp objects, these manipulations often cannot be performed by specifying one Cartesian goal for the robot's end effector. Instead, they require the robot to bring the object to a goal state while respecting the constraints that govern the object. When pouring liquid, a policy that tips the container upside-down before transporting it to the goal would cause the liquid to spill. If a robot had to shut off a sink, rotating a faucet handle in the wrong direction would cause the flow of water to increase rather than stop. Rotating the handle too much with significant force would cause it to break. Learning to manipulate the world through interaction is more difficult than learning to select a grasp to attempt. While a robot could select and execute a grasp on an object before manipulating it, a successful manipulation could consist of multiple steps that change based on the state of the robot and its environment. For instance, a robot must push a lever down when grasping from the top, or pull it when grasping from the bottom. To open a door, the robot must learn to turn the handle to unlatch the door before pulling or pushing it open. In order to feasibly learn to manipulate the world, robots must leverage existing strategies to grasp objects before learning to use them to complete tasks.

In this dissertation, I first detail a system that enables robots to learn where and with which grasp type they should stably grasp an object. Next, I extend this system to the task-oriented case, where the robot must learn where to stably grasp objects to perform a downstream task. Finally, I build towards integrating this task-oriented grasp classifier with a reinforcement-learning agent to simultaneously learn which grasps enable task success while learning how to manipulate the object to complete the task. My thesis statement is:

With a classifier that learns from limited data and weighting that considers uncertainty, we can train a task-oriented grasp classifier jointly with a manipulation policy to efficiently learn to manipulate objects.

It is vital for robots to learn to perform complex manipulation tasks quickly in the real world. In particular, robots must learn which grasps enable them to execute a given manipulation controller within a small number of attempts. In order to do this, structure should be inserted into a robot's actions. Rather than learn a complex manipulation policy that grasps and manipulates an object, a robot must learn distinct abstract skills, such as grasping. This dissertation is a study of empowering robots to quickly and effectively learn to manipulate objects through grasping.

1.1 Contributions

When a robot is introduced to a new environment, it must quickly learn to manipulate the objects around it. In this dissertation, I propose a method for simultaneously learning task-oriented grasp detection and durative-contact manipulation. This system utilizes existing motion planning techniques to execute a grasp, rendering grasping a vision-based detection problem. After detecting a suitable grasp pose and moving the end effector to it, the system executes a manipulation controller to complete the task. The main challenges such a system faces are learning to classify grasp poses from few labeled examples and jointly learning grasp pose classifiers and a manipulation policy. By augmenting a grasp-detection network to learn to detect task-oriented grasps from few labeled examples and proper weighting based on changes in the policy, I can train a task-oriented grasp classifier jointly with a manipulation policy to efficiently learn to manipulate objects in the real world.

In this dissertation, I make three major contributions. First, in Chapter 3, I introduce the Multi-Modal Grasp Pose Detector that allows a robot arm equipped with a multi-finger gripper to clear a pile of clutter using multiple types of grasps for different objects. I empower a robot to utilize different types of grasps, such as precision grasps for small objects and power grasps for heavy objects, with a new deep neural network architecture that predicts whether each possible type of grasp would succeed at a given grasp pose. Through extensive real-robot experiments, I demonstrate that a robot equipped with multiple types of grasps clears clutter from a table more effectively than one that uses few grasp types. Then, in Chapter 4, I describe an Augmented Task-Oriented Grasp Detection Network that is efficiently trained from limited labeled data to detect grasps for more complex manipulation tasks given a manipulation policy. This classifier leverages a pre-trained generic grasp classifier to predict whether a given pose would enable a manipulation policy to succeed. As this small classifier is trained only with the grasp pose and a predicted grasp quality score, it can be trained efficiently from few examples. Finally, in Chapter 5, I define an algorithm for jointly training a task-oriented grasp classifier and a manipulation policy. This is a difficult non-stationary learning problem because the labels used to train a grasp classifier can change as the simultaneously learned policy evolves. By weighting examples based on predictions from the policy, my Grasp-Aware Reinforcement-Learning Agent selects grasps to learn a more optimal policy. A robot equipped with these tools will enter a new environment and quickly learn to perform complex manipulation tasks.

Chapter 2

Background

Intelligent robots process information obtained from their surroundings and perform actions that change their environments to achieve a goal. In order to precisely manipulate the world around them, it is imperative that robots effectively grasp tools, handles, actuators, household items, and other manipulands (objects that are the targets of manipulation). Grasping is the most important skill an intelligent robot possesses, as most healthcare, household, warehouse, and factory tasks require manipulation.

Given an object to grasp, a robot must determine which areas on the object afford it the ability to grasp. The robot must select a pose in one of these areas that it can move its gripper to. A pose $p = (o, d)$ consists of an orientation o and a position d. A position $d \in \mathbb{R}^3$ is the vector from the origin of a defined frame of reference to a point in three dimensions, where the magnitude of this vector is the Euclidean distance between the two points. An orientation $o \in \mathbb{SO}(3)$ can be represented by a 3×3 special orthogonal rotation matrix, which is a matrix with a determinant of one where all columns are orthogonal to each other, all columns have norms of one, and the transpose of the matrix is equal to the inverse of the matrix [8]. This matrix represents the three-dimensional rotation that transforms the reference frame's orientation to the orientation o. The pose $p \in \mathbb{SE}(3)$ is represented as a transformation matrix from the special Euclidean group in three-dimensions.

This pose could be defined in the robot's end effector's frame of reference, which may originate at a fixed point at the center of a robot's gripper's palm with an orientation where one axis of rotation is orthogonal to the plane of the palm, another is parallel to the gripper's closing direction, and the third is the cross product of the two. An arbitrary position d and orientation o represented in the robot's end effector's reference frame describe exactly how the gripper should translate and rotate to reach the corresponding pose p. Grasp pose detection, a problem studied throughout this dissertation, involves detecting poses at which the robot can place its gripper and close its fingers to successfully grasp an object.

In this chapter, I define three problems addressed in this dissertation: grasp detection, task-oriented grasp detection, and manipulation policy learning. Grasp detection is the process of detecting poses at which a robot can achieve a stable grasp on an object to lift it, which is vital for pick and place tasks. Task-oriented grasp detection involves detecting grasps that enable a robot not only to stably grasp a manipuland, but also perform some downstream task with it after achieving a grasp. However, learning a policy for manipulating these objects, such as a hammer, after grasping them to perform a task is also a difficult problem. After formally defining each problem, I present a survey of existing works that provide possible solutions for that problem. Detailed analyses comparing my systems to these related works are provided in Sections 3.2, 4.2, and 5.2.

2.1 Grasp Pose Detection

2.1.1 Problem Definition

Grasp pose detection algorithms return a Cartesian pose $g \in \mathbb{SE}(3)$ consisting of a position and an orientation in 3-dimensional space. The input to a grasp pose detection system is often some sort of localized visual representation of a scene, V. This visual input could, for instance, be a depth image, which is a matrix where each value represents the distance from the sensor to an obstacle for the corresponding pixel, or point cloud, which is a set of 3D points, as shown in Figure 2.1. V is

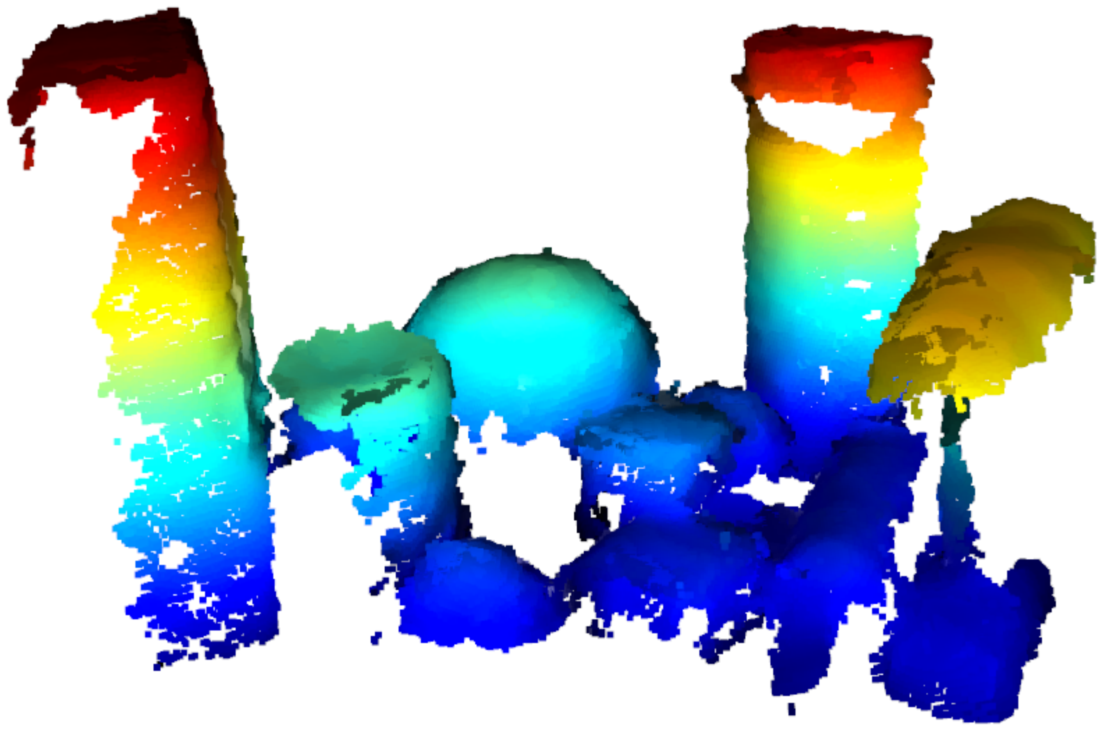


Figure 2.1: An example of a point cloud that may be passed as input to a grasp pose detector.

localized when the extrinsic transformation from the robot to the sensor with which V is captured is known; this transformation is calculated through a calibration procedure.

A grasp pose detector uses the information encoded in V to select the grasp pose g. Formally, this is expressed as

$$\mathbf{GD} : V \rightarrow g. \tag{2.1}$$

This pose g corresponds to a grasp for the robot to execute by moving its joints such that its end effector is positioned at g. Some grasp pose detectors can instead output a discrete set of grasp poses G with corresponding scores S_g, or a distribution of grasp poses $\mathbf{G}$, and some other algorithm could select a grasp g from these sets or distributions. A set of joint states that bring the end effector to the desired g are computed with inverse kinematics, and the robot plans a path that safely brings it to these states while avoiding collisions with obstacles in the environment. With its end effector at the desired grasp pose g, the robot closes its gripper to form a grasp on the object. The robot then plans a path to some user-specified goal location; often, this consists of a point directly above the grasp pose to lift the object, then a point above a box or receptacle where the arm can drop the object.

As the goal for grasp pose detection is to pick an object and drop it in some goal area, the selected pose g must enable the robot to execute a stable grasp on an object. Therefore, when grasping an arbitrary object seen in V, the grasp pose detector must analyze the geometry in V that is local to a given grasp pose.

2.1.2 Related Work

Many of the early approaches to grasp detection approach the problem through planning and by analyzing forces and moments exerted on the planar projections of graspable objects. Nguyen [70] generates 3-degree-of-freedom (DoF) grasp poses by identifying sets of potential contacts on objects whereby grasps could be formed. The system determines whether a set of contacts form force closure on an object, which is achieved when an arbitrary force and moment can be exerted on the grasped object and the grasp will not fail without some external work applied to the object. Ferrari and Canny [22] introduce a geometric grasp quality measure, which maximizes the wrench exerted on an object by a gripper's fingers while minimizing the sum of applied forces to keep the gripper's power requirements low. They present a simple planning algorithm and provide examples for calculating the metric with a parallel-jaw and a 3-jaw gripper.

Another class of older grasp detection systems rely on complete 3D models of objects to compute grasps. Once these algorithms identify and localize an object, they use an analytical grasp solver to plan a grasp on the corresponding CAD model [40]. GraspIt is a popular simulation tool used to plan and optimize grasps [60]. GraspIt contains models of many standard robot hands, and includes a tool to generate hand models from the standard URDF format. Potential grasp candidates can be evaluated by importing the corresponding full object model and the robot hand model, moving the hand to the candidate's grasp pose, and simulating closure. GraspIt reports a grasp quality metric, such as the epsilon grasp quality, ϵ_{GWS}, introduced by Ferrari and Canny [22]. Weisz and Allen [101] revisit grasp quality metrics using GraspIt and 3D object models, noting that a grasp is not best defined by its estimated quality metric, but by an estimate of the stability of a grasp quality metric. If an object moves slightly or the robot does not execute a grasp exactly at the desired pose, a grasp estimated to succeed could fail. The metrics evaluated on its neighbors must be considered to ensure a grasp is stable. Algorithms that rely on simulation-in-the-loop are not robust enough to plan grasps on novel objects for which no CAD model exists in their database. Though some recent work has combined a neural-network-based object reconstruction pipeline with an analytical grasp solver, reconstruction can fail on novel object classes and when objects are cluttered closely together [99]. Instead of using deep neural networks to explicitly complete objects before planning grasps, neural networks can be used to plan grasps given some partial representation of graspable objects.

With the rise of deep learning, deep neural networks have been employed to perform data-driven grasp detection, where a system leverages a large body of labeled data to choose how to make a new grasp. This body of work is summarized well in several surveys [9, 68]. Redmon and Angelova [77] developed one of the first data-driven grasp detection systems. Their most basic network performs regression to predict a single grasp, represented by a position and orientation in the X-Y plane and a gripper width and height. The input to this convolutional neural network is an RGB image, and they assume a method for mapping from pixels to points in 3D space is available. They introduce a modified network architecture that jointly predicts a single optimal grasp as well as the class of object on which the grasp would be performed. The final network augmentation, MultiGrasp, instead assumes that after discretizing the image into cells, an optimal grasp could be found in each cell. The network predicts the grasp in each cell along with the probability that the selected cell contains a globally optimal grasp.

With recent advances in deep learning, data-driven grasp detection pipelines become successful. One common framework involves training a deep neural network end-to-end to predict a single optimal grasp given a representation of the scene. Works in this line often train their system using real robot data. Levine et al. [45] first train a network to predict whether a task-space action would lead to a successful grasp directly from monocular color images. The more recent QT-Opt system utilizes self-supervision and reinforcement learning to train a network to learn a Q-function whose state representation is a monocular image of a bin of objects [32]. One major issue with these frameworks is the amount of real robot data they require. Their dataset was collected over a period of four months by seven robot arms running self-supervised experiments concurrently; it took over 800 robot hours. Additionally, the input to their system is a color image without depth information captured from a fixed viewpoint. The system would likely perform poorly if the environment was modified. Furthermore, an end-to-end system that learns to grasp an arbitrary object in the scene would not be useful as a general grasping skill. If a particular item was desired, a network modification would be necessary to force the system to focus on that object.

Another common data-driven grasping framework returns a set of ranked possible grasps. Rather than detect a grasp end-to-end, a number of grasp candidates are first proposed. Each candidate is evaluated in the system's next phase. Additional pruning could be integrated into either half of this framework. For instance, if a specific object in a scene was desired, grasp candidates could be generated exclusively on that object once it was detected and segmented. Dex-Net and Grasp Pose Detection (GPD) are two of the most influential systems that employ this paradigm [55, 95].

Mahler et al. [55] introduce Dex-Net 2.0 and train a grasp quality convolutional neural network to score planar grasps to be performed by a two-fingered gripper. The input to this network is an RGBD image with overlaid finger contact points. The grasps used to train this GQ-CNN are generated using wrench space analysis on a dataset of 3D object models primarily collected from the 3DNet dataset. These grasps are labeled with the epsilon grasp quality metric. Improvements to this system include Dex-Net 3.0 [56], which adds an additional network for evaluating proposed suction grasps, and Dex-Net 4.0 [57], which introduces a POMDP framework. However, this system is limited because it can only score planar two-fingered grasps and suction grasps. Since it operates on depth images, it cannot make predictions based on input from an arbitrary number of views. Additionally, the candidates it proposes are 3-degree-of-freedom (DoF) grasps, not full 6-DoF poses.

Gualtieri et al. [28] describe an algorithm that first generates 6-DoF candidate grasp poses. These candidates are represented as a series of images, where average heightmap projections that contain observed and occluded points, as well as estimated surface normals, are generated from multiple captured point clouds. For each of the three pairs of axes of a given grasp pose pose, a set of images is created by projecting the local points onto the plane formed by the axis pair. One grayscale image is an averaged heightmap of the observed points, another is an averaged heightmap of the occluded points, and a third three-color channel image represents the average surface normals; this is a total of 5 channels per axis pair with a single grasp represented by a 15-channel image. These images are fed into a convolutional neural network that predicts the probability that the grasp will succeed. During training, these images are generated using the real point clouds from the BigBIRD data set. Each training image is labeled with whether or not the corresponding grasp forms a "softened" frictionless antipodal grasp, which is determined analytically using BigBIRD's 3D object models. Once a set of candidate grasps have been classified, an optimal grasp is selected using a geometric heuristic. After examining the utility of pretraining on simulated data and using prior knowledge about object category, the algorithm is validated on a Baxter robot by attempting to grasp items from a pile of ten densely cluttered objects. This system is limited in that the analytic metrics used to label grasps can only be applied to two-finger, parallel-jaw grippers. Furthermore, the heightmaps that the neural network learns from are an inefficient and lossy representation.

Since the release of these seminal works, several improvements have been made to the proposal-evaluation grasp detection pipeline. Liang et al. [47] introduce the PointNet deep neural network architecture to the GPD paradigm. Rather than project point clouds onto a set of images on which a classifier is trained as GPD does, PointNetGPD learns to predict whether a grasp would succeed from the point cloud directly. This more efficient learning framework increases test-set classification accuracy and the grasp success rate when grasping singulated or cluttered objects on a real robot.

While GPD projects a partial point cloud into a set of heightmaps to use as input to its deep neural network, complete object geometry is useful for analyzing grasps so all potential contact points can be analyzed. GPD estimates complete object geometry with a heuristic that fills in regions occluded from the sensors' views. With the two separate views employed, it can often be assumed that the obstructed regions behind an object contain the back sides of these objects. The three occluded heightmap channels are a rudimentary representation of completed objects. Instead, several works have explored methods to use completed object models to perform grasp pose detection more accurately. Varley et al. [99] present a method that first reconstructs an object using a deep network that operates on occupancy grids; a partial point cloud of an object is converted to an occupancy grid that is fed into the network, and the network outputs a completed occupancy grid. The system produces a mesh from this occupancy grid that is then passed to GraspIt, where a grasp is planned analytically. However, this system does not take uncertainty and model errors into consideration; if an object is completed incorrectly, grasps planned with incorrect geometry would likely fail. Lundell et al. [54] address this uncertainty and also plan grasps on completed object models using GraspIt. Their system handles uncertainty through dropout layers in their completion network architecture and Monte Carlo sampling, enabling the network to output a set of potential completed object shapes. When selecting a grasp, the system computes grasp quality metrics across all possible object models, selecting a grasp that is likely to succeed but unlikely to fail completely if the geometry estimate in one model is wrong.

Other works have integrated completion models with data-driven grasp detection frameworks to increase prediction accuracy and grasp success rate. Namely, der Merwe et al. [16] propose a deep neural network that jointly reconstructs objects and estimates grasp quality. An object reconstruction sub-network is first trained to reconstruct an object from a partial point cloud. Given a partial point cloud and a query point, this network outputs a signed distance function (SDF) value representing the distance to the predicted surface of the object. This SDF representation enables the system to reconstruct objects without converting the point cloud to a voxel grid or mesh, and enables the network to learn geometric gradients that make grasp learning more efficient. Once this branch of the network is trained to reconstruct objects, the section of this sub-network that processes the point cloud is re-used to predict grasp success with a new dataset. Given a point cloud, a possible grasp configuration, and the size of the point cloud, this branch of the network outputs the probability that the grasp would succeed. Like PointNetGPD, this framework utilizes a deep neural network architecture that operates directly on point clouds. A grasp is selected by performing optimization back through the network to find a grasp that is stable while enforcing collision constraints with the known and predicted object geometry. Yang et al. [108] train one network to propose a grasp pose from partial object geometry and another to reconstruct a complete point cloud from this partial geometry. To execute a grasp, they refine the predicted grasp by projecting it onto the predicted reconstructed point cloud. However, the grasp proposal network does not benefit from the object reconstruction network architecture.

Some authors have examined ways to improve upon GPD's simple grasp proposal heuristic. This heuristic, which proposes grasps by sampling points from the cloud and aligning the gripper with the estimated normal and curvature directions, produces a variety of potential grasp poses, but relies on the grasp classification network to predict which of the many potential candidates would succeed. One of the first systems to train a deep neural network to predict and refine grasp candidate poses was presented by Mousavian et al. [63]. Their framework learns a latent representation of a grasp space using a variational auto-encoder. When executing a grasp, this grasp sampling network proposes grasps by selecting random values in the latent space and reconstructing a grasp given a point cloud. These sampled grasps are evaluated with an evaluation network and refined using the calculated gradients from this network. Sampling grasps in this latent space rather than Euclidean space, then refining them with an evaluation network, enables a robot to grasp objects more effectively than GPD. Sundermeyer et al. [89] follow up on this work by instead representing parallel-jaw grasps with a lower-dimensional contact point representation that is continuous, bound to the object geometry, and unambiguous. Their network directly outputs a distribution of possible grasp poses, and the system selects a grasp based on kinematic feasibility and the confidence values associated with each generated grasp. ten Pas et al. [96] describe a method for efficiently generating reliable grasp poses that outperforms Graspnet [63]. Their grasp proposal deep neural network predicts a set of orientations that would lead to a successful grasp centered at a given sampled point. This network outperforms Graspnet in both precision and detections per second, and when paired with a grasp classification network, the system outperforms GPD in precision-at-high-recall.

While the aforementioned enhancements to the GPD pipeline help to improve the grasp success rate for grasp pose detectors, there is a fundamental issue with the problem definition. These detectors detect one type of grasp specifically for two-finger parallel-jaw grippers, where two plates are moved toward each other linearly to perform a grasp. Some works study grasping specifically for multi-finger hands, which are more complex than parallel-jaw grippers. However, these grippers offer advantages when grasping; they enable the gripper to make contact in more areas to achieve a more stable grasp, as well as execute different types of grasps, such as power grasps that firmly grasp large objects or precision grasps that enable a robot to manipulate small objects.

Multi-Finger Grasp Detection

Varley et al. [98] propose one of the first data-driven grasp detectors for multi-fingered grasping. Their system uses a convolutional neural network to predict the optimal position to place the fingertips of a gripper. Planning contact points is not appropriate for all multi-finger grippers, particularly underactuated grippers.

Kappler et al. [34] published another early multi-finger grasp detection work. Their neural network predicts the success of a given grasp, and takes as input a collection of heightmaps. Their experiments show that a network trained with physics-metric based labels outperforms a network trained with epsilon-based labels on a held-out test set. However, they do not evaluate their system on a real robot. Schmidt et al. [82] train a convolutional neural network to predict a grasp pose for a multi-fingered hand directly given a depth image. They define a penalty metric for a grasp candidate, and score each generated candidate prior to labeling each grasp in their training set. Each training example is labeled with the single lowest-cost grasp. Predicting a single grasp given a depth image could be problematic if executing the predicted grasp proved to be infeasible because of occluded obstacles.

Ku et al. [43] detect grasp points for the index finger and thumb of the Robonaut 5-fingered gripper by locating features in a neural network that are used for object classification, and mapping them back to depth data. They generate a dataset containing 144 point clouds, RGB images, and hand and finger poses by manually placing a robot hand around a set of six cylindrical and six rectangular objects. Hierarchical convolutional neural network features are discovered by tracing filter activations throughout an Alexnet-based network that had been trained for the Imagenet classification task. Features of interest are tracked backwards through the network to the RGB images, and then mapped out to the point cloud. Thumb and index finger poses are then mapped from the feature positions within the point cloud. This system only uses two of the available five fingers, and introduces human bias since the examples are all manually generated hand poses. Zhou and Hauser [110] train a neural network to predict the probability of grasp success for an underactuated, three-fingered gripper. Their deep neural network predicts the probability of a successful grasp given a depth image and a grasp pose. Their optimization-based grasp detection system is evaluated in simulation, but not on a real robot. Lu et al. [53] also predict grasp success for a three-fingered gripper using a convolutional neural network. They verified their system by running experiments on physical robots with two and three-fingered grippers. They posit that one of the difficulties restricting the success of neural network approaches to multi-fingered grasping is reliance on grasp candidate generation procedures. They avoid this issue by performing probabilistic inference over grasp parameters with the trained network to maximize the probability of grasp success. They present grasping results with a three-fingered gripper for only six objects, and perform only five grasps on each of these objects.

Shao et al. [83] propose a network that takes the point cloud of an object along with a representation of the gripper geometry that predicts a set of contact points for each fingertip to make contact. This design enables a network trained to detect grasps for one specific gripper or set of grippers to be transferred to a new gripper without retraining the network. However, since this network only outputs fingertip configurations, it only enables a fully actuated gripper to execute precision grasps. Similarly, the network defined by Liu et al. [50] that enables multi-finger grippers to grasp objects is only capable of detecting precision grasps. This network takes in a set of depth images and performs regression to predict a grasp configuration for a given high-DoF gripper. It learns to execute precision grasps using a differentiable grasp metric that also teaches it to avoid grasps in collision.

Song et al. [87] propose a multi-level network to detect grasps for multi-finger grippers. This network detects objects and their graspable parts, filters a set of top-down grasp poses, selects an optimal grasp to execute, and predicts the pre-grasp configuration of the gripper that best executes this grasp. Liu et al. [49] propose a network that predicts high-DoF grasp configurations directly from an occupancy grid of an object. Their consistency loss term ensures that the network predicts a stable grasp from a pool of candidate grasps, while the collision loss term ensures that the grasp is not in collision with the object. However, by regressing to a single grasp, the system could fail by returning an unreachable or infeasible grasp. Wu et al. [103] propose a reinforcement-learning system where the actions consist of executing a multi-finger grasp or zooming in on the input image to narrow down the area of attention for potential grasps. This attention mechanism increases the grasp success rate, but predicting a single grasp could again result in failure if the selected grasp was infeasible.

Each of the works detailed so far in this section enable a multi-finger robot gripper to effectively grasp objects. However, none of them take advantage of one of a multi-finger gripper's most unique abilities and explicitly model grasp type. Grasp types are described in detail in Section 3.3.2. Grasp types are useful when a robot is tasked with grasping a variety of objects. For instance, small objects close to the surface of a tabletop require a precision grasp using the gripper's fingertips, while heavy objects that may slip out of the fingertips require a power grasp where the fingers and palm cage the object with high stability. Deng et al. [15] propose a grasp pose detection system for a multi-finger gripper that detects up to six types of grasps. It does this by predicting which type of grasp would best grasp each area of an object, then selecting the optimal grasp location and executing a grasp of the corresponding type. However, the network that predicts the optimal grasp type is trained using human-labeled data. Rather than rely on data containing human biases, robots should learn when to apply different grasp types through interaction. Santina et al. [80] use transfer learning to train a network to select appropriate grasp primitives for grasping an object. This system is trained using first-person videos of humans executing grasps on a variety of objects. Similarly, human biases are embedded in this system.

Lu and Hermans [52] train two classifiers to predict power and precision success probabilities from a shared embedding for a 4-finger Allegro Hand. This embedding is generated by reducing the size and dimensions of a voxel grid of the object using principal component analysis. With this low-dimensional representation, a simple Bayesian network is trained and used to find an optimal grasp pose. Each classifier is evaluated separately on singulated objects on which power grasps are always preferred. The system never chooses when to utilize each grasp type in the evaluation.

The prior works described throughout this section effectively enable robots to grasp objects using multi-finger grippers. However, most do not explicitly model grasp type, and assume these complex grippers are capable of executing one type of grasp. Several works instead investigate how complex grippers can leverage their grasp types to grasp and lift objects more effectively. Some of these works are trained to predict optimal grasp type only from human-labeled data. Others train a system to evaluate grasps of different types from a shared embedding space, but do not leverage this system to effectively grasp objects. In order to operate most effectively in various home environments and grasp both small and large objects from clutter, robots must make effective use of all of their grasp types. To this end, in Chapter 3, I present a system that enables a robot to grasp objects of varying sizes by jointly detecting an optimal grasp pose and grasp type.

2.2 Task-Oriented Grasping

2.2.1 Problem Definition

Task-oriented grasp pose detection is a more general version of the grasp pose detection problem. While grasps selected by a grasp pose detector must enable a robot to pick and place an object, grasps selected by a task-oriented grasp detector must enable a robot to complete a given task. Not only must a task-oriented grasp detector detect stable grasps, but the detected grasps must also enable a robot to perform a downstream task with the grasped object. Task-oriented grasps on a tool afford a robot the ability to use the tool, while a task-oriented grasp on a door handle enable the robot to open the door. Objects could have multiple types of task-oriented grasps; a mug, for instance, would have task-oriented grasps on its handle that enable a robot to pour liquid from it, while grasps on the body that leave the handle clear enable a robot to hang the mug on a peg.

In the case of general grasp pose detection, a grasp is often considered successful if it achieves force closure on an object, or if the robot can lift the object without dropping it, sometimes shaking it, sometimes placing it in a different location. A task-oriented grasp is successful if the robot successfully grasps the manipuland and then successfully executes a manipulation control policy. This policy could be provided to the robot or learned through a data-driven method, but manipulation control learning is considered outside the scope of task-oriented grasping.

A task-oriented grasp pose detection algorithm **TOGPD** returns a 6-DoF pose $g \in \mathbb{SE}(3)$ at which a grasp would enable the robot to complete a task T given a manipulation controller $\pi_{\mathbf{M}}$. The input to a grasp pose detector can be some representation of the scene V, often obtained through visual sensors. Formally, this is expressed as

$$\mathbf{TOGPD} : V, T, \pi_M \rightarrow g. \tag{2.2}$$

In data-driven task-oriented grasp detection, a labeled dataset of grasp poses G_l is used to train a classifier sub-system. This dataset could be generated from a robot executing the manipulation policy π_M [109], data generated off-policy [33], or human demonstrations of task-oriented grasps [39].

As with grasp pose detection, many task-oriented grasp detectors are implemented as two-stage systems where a set of candidate grasp poses is generated and evaluated, with the grasp corresponding to the highest evaluation score being executed [65, 51, 17]. Here, a grasp pose generation algorithm **GEN** generates a set of grasp pose candidates $G \subseteq \mathbb{SE}(3)$ given a representation of the scene: $\mathbf{GEN} : V \to G$. Then, a grasp evaluator **EVAL** assigns a score s to each grasp $g \in G$ based on the task and object representation:

$$\mathbf{EVAL} : g \in G, V, T \to s \in \mathbb{R}. \tag{2.3}$$

The robot executes the grasp pose $g_{max} = \operatorname{argmax}_g \; S$, where S is a vector of evaluator scores corresponding to each grasp pose in G:

$$\mathbf{EVAL} : G, V, T \to S \subseteq \mathbb{R}. \tag{2.4}$$

2.2.2 Related Work

As with early task-agnostic grasp detection works [60], where grasp quality is optimized using simulation, some early task-oriented grasp detection works rely on simulation to predict how grasps would perform in the real world. Dang and Allen [14] define semantic affordance maps, which consist of an approach direction, a function that extracts features from a depth image, and a set of manually defined example semantic grasps from the given approach direction. These maps are knowledge bases built in simulation that map encoded depth images to approach directions and potential semantic grasps. At execution time, a retrieved semantic grasp is refined using a grasp planner within a simulation environment.

Many early task-oriented grasping systems rely on human ground-truth labels, which are encoded as human demonstrations of manipulation tasks, human-annotated data, or expert-written heuristics. Faria et al. [19] propose a probabilistic framework that generates models of manipulation tasks and objects using multi-modal data collected from instrumented human manipulation demonstrations. Faria et al. [20] follow up on this work by proposing a task-oriented grasp generation algorithm based on this representation. After detecting objects and assigning them primitive models, a suitable grasp known to work with that primitive type is selected to be executed using Bayesian inference with a Bayesian network trained with the human grasp dataset. Balasubramanian et al. [5] present an early use of kinesthetic teaching, where a human moves a robots arm into a position they deem suitable for grasping an object to perform a task.

Song et al. [86] introduce task constraints to the grasp generation pipeline. They define four feature subsets used in a Bayesian Network: the task, object descriptors, action features that describe the grasp, and constraint functions defined by humans to ensure grasp stability. A mixed Bayesian network, consisting of Gaussian Mixture Models, learns to predict whether a task is feasible given an object, action, or constraints. Example grasps are generated in simulation, with a human assigning ground-truth task labels used to train the model. Bekiroglu et al. [6] define a probabilistic framework that evaluates grasp stability and task appropriateness. Like Song et al. [86], they train a Bayesian network, using pre-defined object, action, and haptic features, as well as binary task variables assigned by a human expert. The network structure is learned, unlike the human-defined network presented by Song et al. [86]. Like Song et al. [86], the Bayesian network is used to predict whether a grasp would succeed for a given task.

Prats et al. [73] classify hand preshapes and use planning. This system requires a model of the articulated object to plan an optimal grasp. Heuristics select a graspable sub-part and action to be performed before selecting a grasp, defined as a hand preshape and target frame, with an additional set of heuristics based on the task. Li et al. [46] use shape matching and a human motion database to select task-appropriate grasps. Given a query object and a database of hand poses, feature representations are generated and compared to find a set of potential grasp candidates. An alignment step generates and prunes a set of transforms that align the pose candidates with the object geometry. The resulting set of aligned grasp candidates are clustered, and pruned based on the wrenches that must be exerted on the object to complete the task.

Haschke et al. [30] also consider task wrenches and propose a task-oriented grasp quality metric. By minimizing the forces required to resist external disturbances while maximizing a given task wrench, a robot can find optimal stable grasps that afford it the ability to complete given tasks. Pardi et al. [72] define a system that considers the collision-free post-grasp motion space when selecting a stable grasp for pick and place and insertion tasks. A given task is represented as a trajectory of object poses that approach a goal configuration. After generating a set of candidate grasp poses, a task trajectory is generated for each pose using inverse kinematics and the desired object trajectory. The grasp candidate that minimizes collisions with a defined set of obstacles is executed.

Affordances became a popular way to represent object parts that afford robots the ability to complete a task with the advent of deep learning, as training accurate affordance detectors became tractable. Chen et al. [11] define affordance coordinate frames, 3D keypoints with an axis that generalize between object instances. A Faster R-CNN module detects objects in RGB-D images, and three sub-networks then assign a keypoint, axis, and part association to each object. These affordance coordinate frames are used as grasp poses and manipulation waypoints in a set of robot experiments. Monica and Aleotti [62] present a system that detects and segments objects from a point cloud, then detects a grasp on the desired segment using an off-the-shelf pre-trained grasp detector [95]. Kokic et al. [38] train a deep neural network (DNN) to detect affordances, object type, and object orientation given a point cloud and desired task. The output of this network is used to assign binary labels to each point in the cloud based on whether they afford the robot the ability to complete the given task. This affordance information, along with the object class and orientation, are then converted to contact and approach direction grasp constraints that describe how a grasp should be executed. Chu et al. [12] propose an affordance detection network that detects and segments objects. They show that while the network is trained on synthetic images, the domain transfer module enables it to perform well on real images, and it enables robots to plan task-oriented grasps.

One keypoint-focused line of work began with kPAM, where objects such as mugs are represented as semantic 3D keypoints [58]. After segmenting an object instance and detecting its class-specific 3D keypoints, an optimizer selects grasp and manipulation actions that move the keypoints to their specified targets. The abstracted manipulation actions are rigid transformations describing how the robot should move the object after executing a grasp with a provided grasp planner. kPAM 2.0 extends this semantic keypoint framework to perform more complex manipulation tasks, such as peg-hole insertion, with the addition of orientation [25]. Orientation simplifies the required problem definition for simple pick and place tasks, and enables actions to be represented as keypoint velocities or force/torque values to perform more complex manipulation. kPAM-SC builds upon the original kPAM formulation with a shape completion module [26]. The planner takes as input completed object models as well as semantic keypoints to better optimize physical feasibility.

Another keypoint-based work focused on tool use is KETO [75]. The keypoint generation network predicts grasp, function, and effect keypoints on tools for a given task. The keypoint detectors are trained through self-supervised exploration. Goals are represented with a given set of environment points. A set of robust grasp pose candidates are generated, and the one closest to the grasp keypoint is executed. Manipulation actions are selected that enable the tool to exert the desired force on the target using an optimization technique inspired by kPAM [58]. Initial sets of keypoints are generated with a heuristic to train a keypoint generation network, while a keypoint evaluation network is trained to predict whether manipulation defined by a given set of keypoints would result in task success.

Other recent data-driven approaches integrate DNNs into their frameworks in other ways. Kokic et al. [39] train a network to jointly predict human hand pose and configuration and object pose and shape from an RGB image. This network is applied to a large set of videos of humans completing tasks with tools such as knives to generate a dataset of task-oriented grasps on a variety of object instances. The resulting labeled human grasp dataset is used to train a second network, *TOG-T*, to predict the probability that a grasp at a given pose is suitable to complete a given task. A third network, *TOG-S*, is used to find a stable grasp in the region of grasps predicted to be suitable for the task by *TOG-T*.

Jang et al. [33] propose a dual-stream, end-to-end architecture to perform semantic grasping, where a user-specified object must be grasped from clutter. Given an RGB image, their network predicts the joint probability that an action leads to a grasp and that the grasped object is the one specified by the user. By keeping spatial and semantic information separate, this system optimizes both grasp stability and task success. The network is trained primarily with off-policy data, and some labels assigned by a human expert are required.

Detry et al. [17] detect grasp poses suitable for a given task with a two-step system. The first sub-system proposes a set of geometrically stable grasp candidates by matching object geometry with a set of grasp prototypes collected through kinesthetic teaching. The second sub-system consists of a CNN that predicts which pixels in an image contain part of an object where a grasp would be suitable for task completion. This network is trained with synthetic data generated with human-labeled object mesh models, where entire objects or partial segments provide suitable grasps for a given task.

Yang et al. [107] propose a system that performs task-oriented grasps on an object within a pile of clutter. Given a top-down depth image of a scene containing a pile of elongated tools such a knife, screwdriver, spoon, fork, wrench, and hammer, a DNN outputs a set of 3-DoF grasp candidate poses, grasp stability scores, and task scores using Conditional Random Fields. This architecture ensures the detected grasps are clear of occlusions and interference from other objects in the clutter pile.

Liu et al. [51] present a context-aware grasping engine that predicts whether a grasp is suitable for a given context. This context includes the label of the task being performed, a point cloud of the object, material information from a spectral sensor, and a task-specific object state. The context and set of grasp pose candidates are augmented with detected segmented part and grasp affordances to create a semantic representation that is then fed into a semantic grasp DNN.

Xu et al. [106] define affordances as the set of states in which a given parameterized skill is feasible. Parameterized skills are sequenced to form a plan, which reaches a goal affordance. In this model-based reinforcement learning framework, they train a latent dynamics model, which predicts future latent states given sampled skill plans, and an affordance prediction model, which predicts whether a latent state affords a given skill. The latent dynamics model is used as the deterministic transition model. The two models are trained jointly over multi-step sequences. The trained models are used to estimate the cost of a given plan for model-predictive control. The system is evaluated in several domains. The robot must use tools to retrieve blocks, then stack them, given grasp, place, hook, and poke parameterized skills. The second evaluation environment is a kitchen domain where the robot must prepare coffee and tea. The learned task-agnostic affordance representation is shared between tasks.

Murali et al. [65] present a large dataset of task-oriented grasps for 56 different tasks. This crowdsource-labeled dataset contains information on whether a given object is suitable for a given task, as well as whether each proposed grasp on suitable objects would afford a robot the ability to complete a task. Their proposed network estimates a grasp score from a point cloud with grasp candidate encoding and a semantic knowledge graph.

Wen et al. [102] propose CaTGrasp to pick small industrial objects from cluttered bins and place them precisely in receptacles. After segmenting a point cloud of a bin of uniform objects, individual objects are represented in a canonical representation. A neural network is trained to predict general grasp quality using simulated data. Affordance heatmaps are also generated using this simulated data; the contact areas of grasps that enable the robot to complete the placement task are tallied over all grasp attempts and converted to the canonical representation. Each grasp is assigned a task-relevance score using both the grasp quality measure and task relevance measure. Zhao et al.

[109] also perform task-oriented grasps for precise pick and place assembly tasks. They emphasize the importance of grasp robustness, precision, and task relevance when executing top-down grasps. They train a Grasp Quality Network to estimate grasp quality, a Grasp Displacement Network to predict post-grasp displacement, and an Insertion Quality Network to predict task appropriateness. These three networks are implemented in series to select a stable, precise, task-appropriate grasp.

The existing solutions to the task-oriented grasp pose detection system rely on biased human data, require large datasets of data collected from task executions, and use affordance assumptions to perform task-oriented grasping. In order to quickly learn which grasps enable a robot to perform a given task, I propose a task-oriented grasp detection framework that leverages a pre-trained general grasp detection network to learn from few labeled examples in Chapter 4.

2.3 Manipulation Policy Learning

2.3.1 Problem Definition

A policy informs an intelligent agent on which action it should take at some point in time given an observation or a representation of the state of the environment. As described in Section 2.1.1, in grasp pose detection, the robot executes a simple policy to perform a grasp at a given pose and then drop an object in a specified goal area. Given this desired pose to place the robot's end effector and the kinematics of the robot arm, inverse kinematics returns a set of joint states that bring the robot's end effector to the desired grasp pose. The robot uses motion planning to find a collision-free path to these joint goals, executes this trajectory, and closes its gripper to achieve a grasp. Inverse kinematics and motion planning are used again to drop the object at the known goal location, completing the task. With this simple policy, grasp pose detection is reduced to a computer vision problem.

However, pick and place is only one task that a robot must perform. Interacting with articulated objects such as doors, switches, and drawers, using a tool, and carefully manipulating delicate or deformable objects require more complex manipulation policies that vary based on visual, proprioceptive, and tactile sensor readings. Durative-contact manipulation policies are policies that complete tasks that involve robot manipulation where a robot must sustain contact with a manipuland throughout the course of manipulation, such as unlatching and opening a door while maintaining contact with the handle. Hand-coding these policies for each new complex task a robot may be introduced to is difficult and time-consuming for an expert human operator, and it reduces a robot's autonomy. Instead, these policies should be learned.

Formally, a policy π maps an encoding of the state of the agent's environment s to an action for it to take: $\pi : s \rightarrow a$. s could contain, for instance, visual information V, proprioception data q, and tactile data T. Though a robot can execute a simple grasping policy given joint state position commands, more complex policies require joint velocity and torque commands for a higher degree of control. The action a could be a combination of desired joint torques, velocities, or positions, and can also contain actions for a gripper. Controlling the torques or velocities directly enables robots to use the force required to open a heavy door while also manipulating delicate objects gently.

Learning these manipulation policies from scratch is difficult because the state and action spaces could be very large and contain continuous values, and the reward signal required for learning them could be sparse. Abstraction can alleviate this problem. An abstract state space is an augmented state space that may be more compact to ignore extraneous information and enable the agent to focus on learning only from task-relevant information. An abstract action space is, similarly, an augmented action space that provides the agent with a set of actions designed for the task at hand. In the context of manipulation policy learning, this action-space abstraction can consist of higher-level actions that, for instance, enable the agent to learn how to move the robot's end effector to complete a task. Learning how to move a robot's end effector to complete a task is an easier problem than learning how to move a robot's joints to move its end effector while also learning how the end effector should be used to complete a task. After defining the state and action space for an agent, reinforcement learning algorithms can learn a policy to complete a given task.

2.3.2 Related Work

In order to learn a manipulation policy π_M that maps a state representation s to an action a, a robot must gain experience through interacting with its environment. It must learn, for any given state, which action it should select to successfully complete a given task. π_M can be learned through reinforcement learning, where an agent is free to take actions in its environment to gather experience. A reward signal r is produced by a reward function $R : s, a, s' \to r$, and signifies whether the agent's task was completed, if it made progress towards its goal, or if it made a mistake. This function must be provided as the agent explores its environment. This reward function enables the agent to learn which actions to take to bring it closer to a goal state where cumulative reward is maximized [90].

More formally, reinforcement learning algorithms model a task as a Markov Decision Process (MDP), consisting of a state space S, action space A, reward function R, transition function T that represents the probability of reaching a state s' after taking action a in state s, and a discount factor γ that parameterizes how much an agent should discount reward received over time. This formulation adheres to the Markov property, which states that the reward received r and state reached s' after taking action a in state s is only a function of state s and action a and stochasticity and not the agent's history of states and actions. The environment changes over the course of a set of discrete time steps, and the agent can take an action at each of these steps. The policy π maps states to actions, or a probability distribution over actions $\pi(a|s)$ depending on the policy type. The return obtained during one episode, or sequence of time steps in which the agent attempts to complete a task, is the weighted sum of discounted future reward with discount factor γ:

$$\Sigma_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i). \tag{2.5}$$

The agent must learn a policy that maximizes the expected return over the course of each episode.

One traditional approach to policy learning is to learn a Q-function $Q : s, a \to q$, which maps a state s and an action a to a Q-value q. This Q-value represents the return expected when taking action a in state s and following a policy π until the end of the episode:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right). \tag{2.6}$$

The Q-function obeys the Bellman equation:

$$Q_\pi(s_i,a_i) = \sum_{s_{i+1} \in S} P(s_{i+1}|s_i,a_i) \left(R(s_i,a_i,s_{i+1}) + \gamma \sum_{a_{i+1} \in A} \pi(a_{i+1}|s_{i+1}) Q_\pi(s_{i+1},a_{i+1}) \right). \tag{2.7}$$

The agent could compute a greedy policy π by optimizing the Q-value over actions when in a state s. This Q-function could be computed in a tabular fashion for simple problems with discrete state and action spaces by tracking the rewards obtained over the course of many episodes attempting to complete the desired task and maximize return. Q-learning is an important algorithm that learns a Q-function [100]. The Q-learning update rule sequentially updates values in the Q-function table as the agent gains experience with the update rule

$$Q(s_t,a_t) \longleftarrow Q(s_t,a_t) + \alpha \Big(R(s_t,a_t,s_{t+1})) + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \Big). \tag{2.8}$$

This tabular Q-learning algorithm enables the agent to learn an approximation of the optimal Q-function Q^* over time in simple environments with discrete states and actions. However, this is not feasible for robot manipulation domains with large state spaces and continuous action spaces. Instead, Q-functions can be approximated using powerful deep-learning based function approximators.

Deep Q-Networks (DQN), introduced by Mnih et al. [61], are an influential approach to reinforcement learning that combine deep-learning techniques with Q-learning. Deep Q-Networks are Q-functions that map states and actions to returns that are approximated using DNNs. These networks are trained and evaluated in the popular Atari video game domains, where states are raw images of the video game screen, actions are combinations of the button and joystick inputs on an Atari 2600 controller, and reward is provided as the change in score between states. Deep learning enabled DQN to decipher important information like agent and enemy positions from the high-dimensional game images directly while also learning policies to achieve high scores in many of the Atari games. Deep Q-Networks are trained off-policy by collecting and sampling experience from a replay buffer of recent experience as the agent learns to play the game. The input to these networks are a sequence of game frames and the outputs are Q-values corresponding to each possible discrete action.

Though DQN is well-suited to Atari domains where the state space contains images and the action space consists of a discrete set of digital button presses, it is not well suited to robot manipulation where the action space is a vector of continuous values, such as joint positions, velocities, and torques. A naive solution to this would be to discretize a robot's action space. However, discretizing into large enough bins to enable precise manipulation for each of a robot's joints would lead to an intractably large action space. Furthermore, representing similar actions as separate discrete possibilities sacrifices structure in the action space. A groundbreaking work that addressed deep reinforcement learning with continuous action spaces was Deep Deterministic Policy Gradient (DDPG) [48]. DDPG leverages the dual-network actor-critic paradigm, where a policy or actor network is trained to suggest actions while the critic or value network predicts the utility of a proposed action with a DQN. Rather than select one discrete action from a set, the actor network is free to predict a continuous value for each element in an action vector. DDPG is evaluated on a variety of simulated tasks, including grasping and manipulation. Fujimoto et al. [24] improve upon the DDPG architecture with Twin Delayed Deep Deterministic Policy Gradient (TD3), which adds several enhancements to the continuous-space reinforcement-learning algorithm to improve performance. TD3 remains one of the top-performing continuous-action reinforcement learning algorithms.

Though these deep-learning based algorithms are necessary to learn a robust policy, abstraction is crucial. Rather than learn a single policy with robot-joint-level actions, several works utilize action abstraction to make learning more efficient. Sutton et al. [91] introduce the options framework, a general form of action abstraction that encapsulates a low-level policy with a set of states from which it can be executed and a condition for when it should terminate. Formally, an option consists of an option policy π_o that maps observed states to low-level actions, an initiation set I containing all states the option can be run from, and a termination condition β that defines when the option execution ends. A high-level policy can choose to execute an option from any state $s \in I$ in its initiation set, at which point the low-level policy takes control until the termination condition β is reached. This focused policy learns to execute a specific task within a specific region of the state space, and learning options can be more efficient than learning a single monolithic policy to select actions throughout the course of a task. For complex, multi-step manipulation tasks, such as tasks where an object must be grasped and then manipulated, a robot could more efficiently learn separate policies for grasping and manipulating. Importantly, abstract skills learned to complete one task can be re-used to bootstrap learning another; re-using generalized skills, such as grasping, is key to enabling robots to effectively learn multiple new tasks in novel environments. Bagaria and Konidaris [3] present a framework for learning and sequencing skills together through deep skill chaining. As an agent learns a continuous-control policy through DDPG, it begins to discover and construct options. It begins with an option that terminates at the goal, fine-tuning a policy that is optimal near the goal location. It then works backwards, chaining from one option to another by terminating one option when the next can be initiated. To more efficiently learn to chain options as they are learned, Bagaria et al. [4] improve upon this system with improved dual initiation-set classifiers, a goal-conditioned policy, and model-based reinforcement learning.

While these reinforcement-learning algorithms could be implemented on real robots directly, several approaches have evaluated other action abstraction methods geared towards learning manipulation policies. Khatib [36] propose operational space control, in which the agent selects high-level actions that change the end-effector pose and traditional control methods convert these commands to low-level control signals to send to a robot. Martín-Martín et al. [59] learn a policy using end-effector impedance control rather than one that directly modifies an arm's joint torques or velocities. VICES uses an impedance-based PD controller to convert desired changes in end-effector positions and velocities to joint torques to control the robot. A key insight they make is that the parameters for this controller can be updated on-the-fly and learned for each task as well. Learning changes in end-effector position and orientation while also learning to adjust the gains of the controller that effects these changes enables a robot to efficiently learn to perform constrained, contact-rich tasks while exerting minimal energy and remaining safe.

Besides these reinforcement-learning abstractions, other works have proposed frameworks for combining traditional robot control methods. Another action abstraction for robots is Dynamic Movement Primitives (DMPs) [81]. A DMP consists of a parameterized pose goal and a motion shape that is defined by a set of equations; a controller uses these equations to reach the goal under the given motion constraints. Riemannian motion policies (RMPs) are another popular formulation that map some task space to a robot's configuration space [76]. They allow a variety of motion primitives to be composed together into an RMP-tree, where each leaf node corresponds to a different task space. These trees enable motion planners, motion optimizers, motion controllers, collision controllers, learned policies, and other motion primitives to be synthesized and sent to a robot's low-level control interface. Shaw et al. [84] combine RMPs with a VICES-based controller to increase the safety of policies learned in the VICES action space.

Some works define methods to learn durative-contact manipulation policies with robots [44, 37, 27]. One issue with these works is that the agent must learn how to grasp objects as well as how to manipulate them after performing a grasp. Determining how and where to grasp an object in order to perform a task is no simple problem, as evidenced by the body of work detailed in Section 2.2.2 that learns task-oriented grasps on objects. Though the works in Section 2.2.2 propose methods for detecting task-oriented grasps, none learn both a task-oriented grasp classifier and a durative-contact manipulation policy. I posit that learning a task-oriented grasp classifier and a manipulation policy simultaneously will make learning more efficient, since the task success is dependent on both the grasp selected and the policy that performs the task. This is an entangled-learning problem because a task-oriented grasp is assigned a binary label based on whether the policy succeeds, but policy success is dependent on the initial grasp. I investigate this difficult joint-learning problem in Chapter 5.

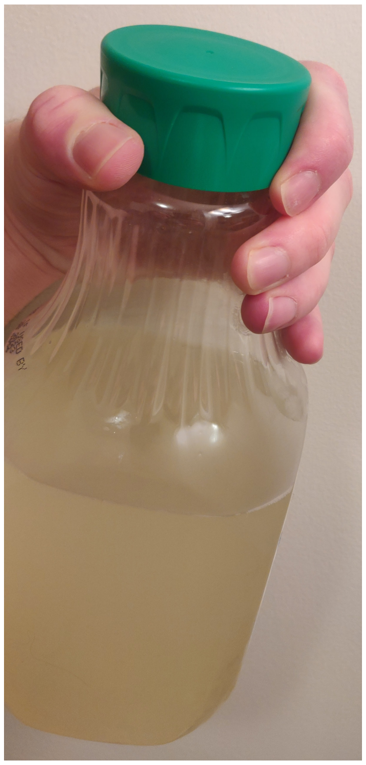
Chapter 3

Multi-Modal Grasp Detection

In order to jointly teach a robot to detect grasps and execute a durative-contact manipulation policy, I must first define a grasp detection framework. This chapter presents a general grasp pose detection system designed specifically to utilize the multiple grasp types many multi-finger robot grippers are capable of. This Multi-Modal Grasp Pose Detector takes as input a point cloud of a singulated object or cluttered scene along with a set of potential grasp poses and predicts the probabilities that each grasp type would succeed at each pose. Explicitly modeling multiple grasp types improved the object removal rate by 8.5% over the best performing baseline ablation during robot experiments in dense clutter.

3.1 Introduction

Humans use multiple types of grasps, depending on the object, the task, and the scene [21]. A human may perform a large-diameter power grasp to stably grasp a heavy jug, but a precision sphere grasp to lift a golf ball off the ground, as demonstrated in Figure 3.1. If the clutter around an object precludes one particular grasp type, humans simply switch to another. It is therefore natural that the ability to use multiple grasp modalities would substantially improve robots' ability to grasp a wide range



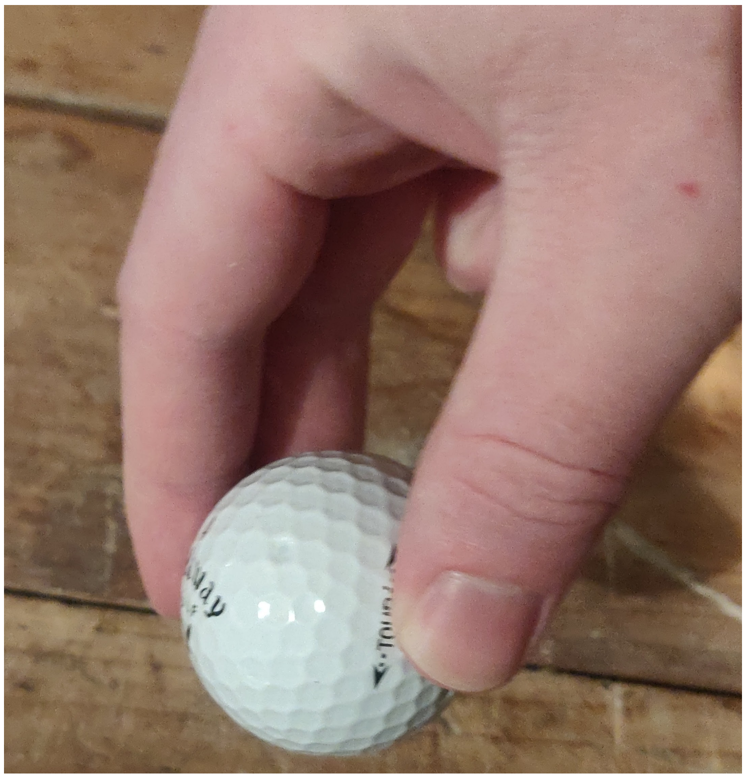
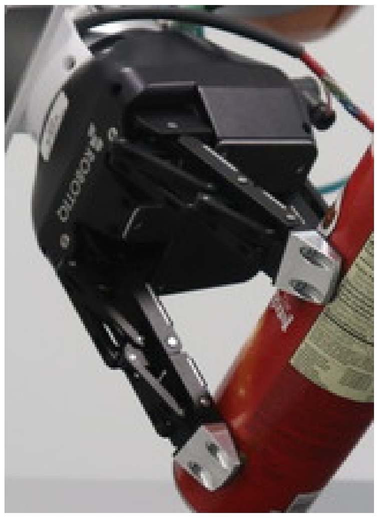


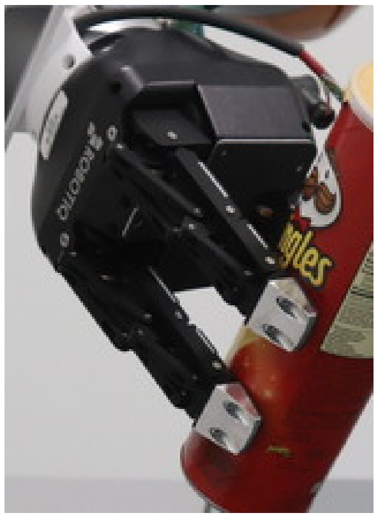
Figure 3.1: Two of the many grasp types a human may use. Stably grapsing a heavy jug necessitates a power grasp, while a precision sphere grasp lifts the small golf ball.

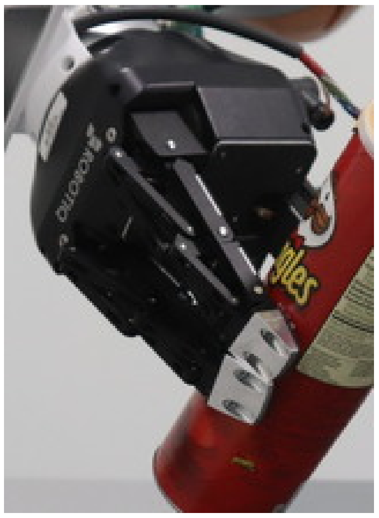
of objects, especially in dense clutter. However, state-of-the-art grasp detection systems typically detect pincher grasps exclusively, and are evaluated using small objects and two-finger parallel-jaw grippers [95, 55, 66, 32]. Existing grippers are capable of executing multi-finger dexterous grasps better suited to stably grasping both small and larger, heavier objects; the grasps a Robotiq 3-Finger Adaptive Gripper is capable of executing are demonstrated in Figure 3.2. Several grasp detection approaches are applicable to multi-finger grippers, but are only capable of performing one type of grasp [50], or do not explicitly model grasp type [34]. Some have taken grasp type into consideration, but are evaluated on singulated objects [52], rely on human-labeled data [15], or return fingertip placement for fully actuated fingers [98]. Furthermore, these systems are not evaluated in dense clutter.

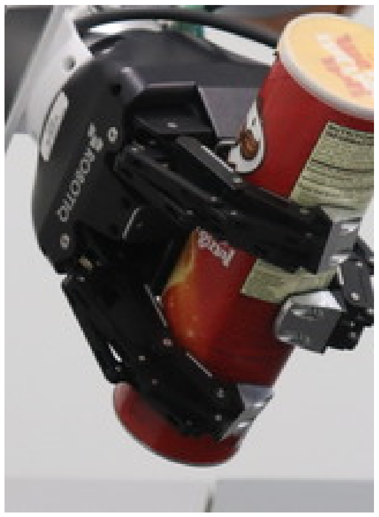
I propose a data-driven grasp detection framework, the Multi-Modal Grasp Pose Detector (MMGPD), that jointly predicts the grasp success probabilities of several types of grasps given partial depth data and a grasp pose. I train a DNN to perform this joint classification using a dataset containing grasp candidates generated from real point clouds and grasp labels generated in simulation. Given a point cloud—captured from an arbitrary number of depth sensors in arbitrary poses—along with a grasp pose, my network outputs a probability for each available grasp modality. These values reflect the probability that the corresponding type of grasp would succeed at the given pose.

I evaluate the system both in simulation and experimentally on a Robotiq 3-Finger Adaptive Gripper, which is shown in Figure 3.2. I first evaluate the system on a held-out test set from simulated data to show that the network efficiently learns to jointly predict grasp type when compared to a larger ensemble of networks. On a real robot, the system clears objects from cluttered tabletop piles containing objects of varying sizes. To show the usefulness of multiple grasp modalities in dense clutter, I compare against several ablations of my network capable of performing fewer grasp types, and find that a system capable of multiple grasp types clears more objects than baselines that use fewer.


(a) Wide Precision


(b) Basic Precision


(c) Pincher


(d) Wide Power

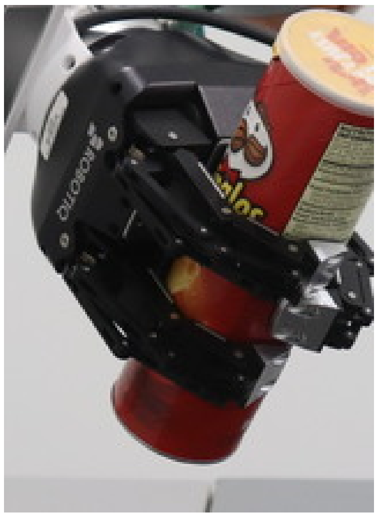

(e) Basic Power

Figure 3.2: The Robotiq 3-Finger Adaptive Gripper's five main grasp types.

3.2 Background and Related Work

As discussed in Section 2.1.2, with recent advances in deep learning, data-driven grasp detectors have proven effective for generating parallel-jaw grasps for two-finger grippers. Given visual information, these systems return an end effector pose at which an executed grasp would likely be successful. Most state-of-the-art parallel-jaw grasp detectors, such as those described by ten Pas et al. [95] and Mahler et al. [55], follow a two-step proposal-evaluation model. A proposal function $\mathbf{PROP} : P \rightarrow G$, implemented as a heuristic [95] or a generative network [64], first generates a large set of 6-DoF end effector poses $G \subseteq \mathbb{SE}(3)$ from a point cloud $P \subseteq \mathbb{R}^3$. A grasp evaluation neural network $\mathbf{EVAL} : g \in G \rightarrow [0, 1]$ then maps each g to a probability.

Another common approach is to train a neural network to predict optimal actions using a reinforcement-learning framework. Works such as Ibarz et al. [32] and Levine et al. [45] train their systems using real robot data, which is time consuming to produce. Though such systems can achieve state-of-the-art grasp success rates, their reliance on reinforcement learning makes them brittle; the same camera configuration used while training is required at test time. Furthermore, modifying the system to grasp a specified object is not straightforward as it is in proposal-evaluation systems, where the proposal step can be easily modified without adjusting a reward function or re-training. Though both of these types of systems enable two-finger parallel-jaw grippers to grasp some objects, these grippers are capable of executing only simple pincher grasps.

Data-driven grasp detection frameworks have also been applied to perform multi-finger dexterous grasping. However, these systems are either capable of performing only fingertip or precision grasps [83, 50], or use supervised [34, 53, 82, 87, 49] or reinforcement learning [103] to evaluate or predict wrist poses and finger pre-grasp parameters, but do not explicitly model grasp type.

A few recent works predict grasp stability for multiple grasp types. Lu and Hermans [52] train two classifiers to predict power and precision success probabilities from a shared embedding for a 4-finger Allegro Hand. Each classifier is evaluated separately on singulated objects on which power grasps are always preferred. As my system jointly classifies candidates of each grasp type at a given position, it returns both a predicted optimal grasp pose and type. I evaluate my system in a cluttered real-world scenario where multiple types of grasps can be necessary to clear the scene. Deng et al. [15] and Santina et al. [80] use human-labeled data to train a neural network to predict grasp type, while my system learns to use grasp types from simulated grasping data, avoiding human bias or error. Both systems are not evaluated in dense clutter. Varley et al. [98] employ a hybrid approach, using a DNN to guide fingertip placement and a grasp planning simulator to localize gripper placement. They define a set of canonical grasp types based on the most common finger pre-poses in their simulated training set. Planning fingertip placement can be impossible for underactuated grippers; to perform a power grasp with an underactuated gripper, each finger's more proximal links would make contact with the object first, making the final distal link placement less relevant. Their system is also not evaluated in dense clutter. Osa et al. [71] use hierarchical reinforcement learning to select a grasp type and grasp location. They maintain a dataset of successful grasps for each grasp type and match new point clouds to this dataset using ICP. Their system is not evaluated in clutter. As they employ a reinforcement-learning framework, a higher-level controller could not request a grasp type or target object as it could with mine. My approach is capable of executing multiple grasp types, allowing it to successfully grasp objects in a variety of real-world, cluttered experimental scenarios.

My system jointly predicts the independent probabilities that each grasp type would succeed at a given grasp pose. The necessity of multiple grasp types is shown through real robot experiments on scenes of cluttered objects. Most existing frameworks do not take grasp type into consideration, and those that do rely on biased human-labeled data, are incompatible with underactuated grippers, or train individual systems to evaluate grasp types from a shared representation while evaluating the systems separately. Mine is the first that explicitly models grasp modalities to more successfully clear cluttered scenes.

3.3 Learning to Detect Multi-Modal Grasps

Though parallel-jaw grasp detectors have proven successful, two-finger grasps can be insufficient when a robot deals with large, heavy objects. Grasp detectors designed for multi-finger grippers detect grasps of a single type, and those that can explicitly utilize multiple grasp types have not been proven to enable a robot to clear a pile of dense clutter. MMGPD demonstrates the usefulness of multiple grasp types when picking objects of varying sizes from piles of dense clutter using the proposal-evaluation paradigm commonly used in grasp detection systems [95]. The proposal function $\mathbf{PROP} : P \rightarrow G$ generates a set of 6-DoF end effector poses $G \subseteq \mathbb{SE}(3)$ from a partial point cloud $P \subseteq \mathbb{R}^3$. Figure 3.3 illustrates this process. Unlike the grasp evaluators $\mathbf{EVAL} : g \in G \rightarrow [0, 1]$ used in related works that map a grasp pose to a single probability [95, 55], my grasp evaluation neural network $\mathbf{EVAL} : g \in G \rightarrow [0, 1]^n$ maps each g to a vector of n success probabilities, each corresponding to a different grasp type. This architecture enables MMGPD to jointly predict the probabilities of success for multiple grasp types at a given g.

I generate grasp pose candidates $G \subseteq \mathbb{SE}(3)$ using the 6-DoF candidate generation algorithm $\mathbf{GEN} : P \rightarrow G$ proposed by ten Pas and Platt [94]: given a point cloud of an object or cluttered pile of objects represented as a set of 3D points $P \subseteq \mathbb{R}^3$, sample a subset $C \subseteq P$ of k grasp candidate centroid positions. Each $c_s \in C$ is assigned a single orientation $o_s \in \mathbb{SO}(3)$ based on the normals and curvature estimated at c_s; the gripper approach direction is anti-parallel to the estimated normal, and the gripper closes along the curvature. Similar candidates can be sampled by rotating the sampled orientation about the approach direction; I rotate by 90° to generate one additional pose. Finally, candidates causing the gripper to collide with P are pruned. The candidate generation algorithm returns a set of k proposed candidates $\mathbf{GEN}(P) = G$ where $g_s \in G$ and $g_s = \{c_s, o_s\}$.

The second phase of MMGPD evaluates each of the k proposed candidates $g_s \in G$. My Grasp Pose Classifier (GPC) deep neural network estimates success probabilities for each grasp type at each g_s, taking P and an encoding of g_s as input. As several recent papers have shown [16, 47], grasp

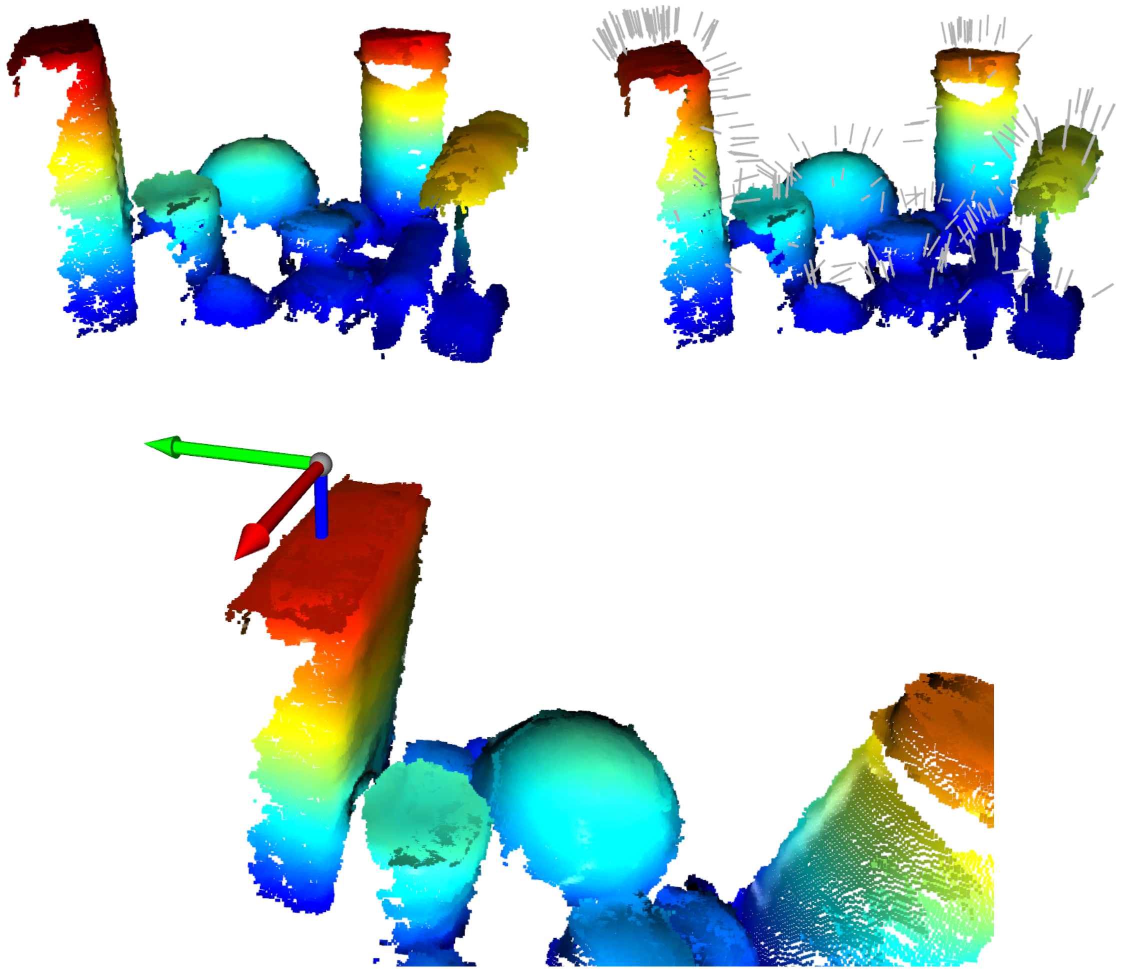


Figure 3.3: The grasp pose generation process. Given a partial point cloud of a real scene of cluttered objects (top left), the grasp pose generator samples a set of points to assign as candidate grasp centroids. Each of these sampled centroids is assigned an approach direction anti-parallel to the normal estimated at that point (top right). Each grasp centroid is also assigned a gripper closing direction that aligns with the estimated curvature at that point. In the bottom image, one candidate grasp pose is visualized; the approach direction is the blue axis and the gripper closes along the green axis.

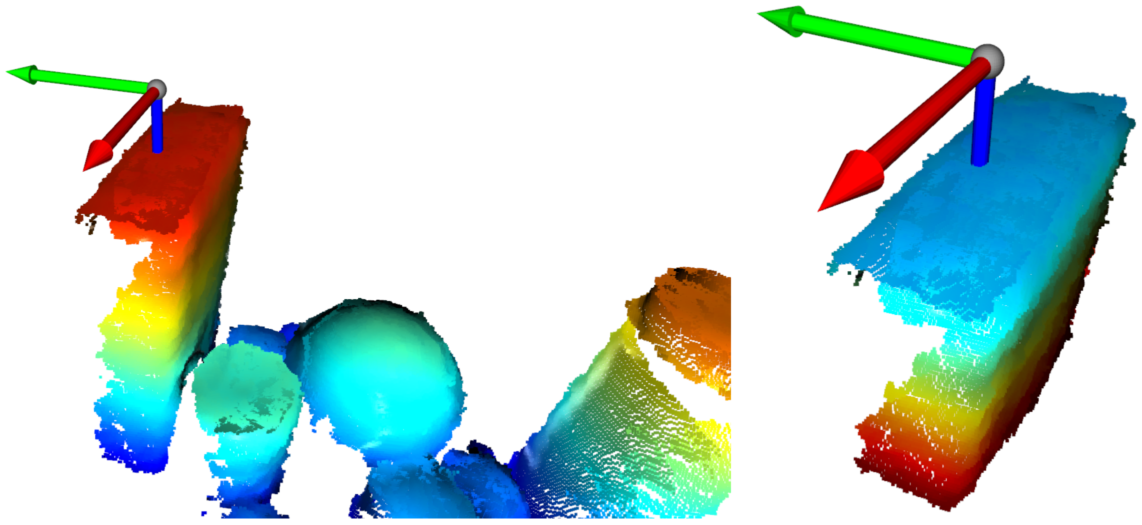


Figure 3.4: The left image shows a grasp pose g_s visualized on a partial point cloud P of a scene. The right image shows the point cloud P_s output by the grasp encoding function **TF**. This point cloud, which is passed as input to MMGPD's network, encodes g_s and the local geometry around it that the gripper may make contact with during a grasp attempt.

candidates can be efficiently encoded directly from point clouds recorded from arbitrary viewpoints using a PointNet-inspired architecture [74]. The encoding layers used in my network are based on the PointConv architecture [104]. I encode a candidate grasp pose by centering P at c_s and aligning P's orientation with o_s. I then crop all points outside the approximate grasping region, represented as a box around the fingers and the area they sweep through. This transformation $\mathbf{TF} : P, g_s \rightarrow P_s$ produces P_s, an encoding of a grasp pose and the object geometry local to it, as seen in Figure 3.4. This transformed, cropped cloud P_s representing a single g_s is then fed to the MMGPD's network, which is illustrated in Figure 3.7. Here, PointConv layer parameters listed are number of points, radius, sigma, and MLP sizes. The listed parameter for each fully connected layer is output size.

The encoding layers in MMGPD's neural network consist of four PointConv feature encoding layers. Following der Merwe et al. [16], I reduce the first layer's number of points from 1024 to 512 and the third layer's final multi-layer perception from 128 to 64 units. The output from the fourth encoding layer is then fed through a series of five fully connected layers with ReLU activations. The final fully connected layer outputs a logit pair for each of the n grasp types the gripper is capable

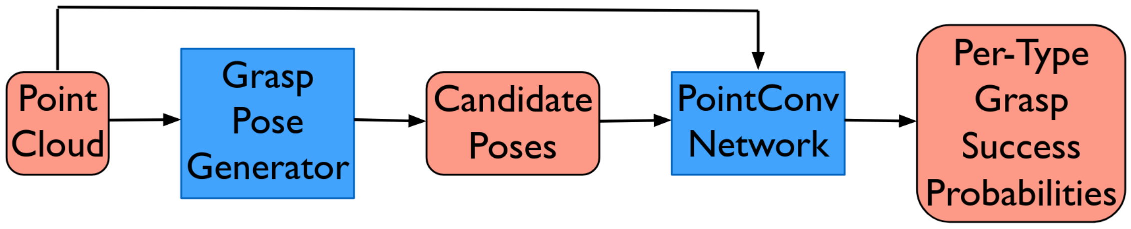


Figure 3.5: Multi-Modal Grasp Pose Detector system diagram. Orange boxes show inputs and outputs while blue boxes represent algorithms. The Grasp Pose Generator produces candidate grasp poses given a point cloud. The PointConv Network assigns each of these grasp poses a probability of grasp success for each grasp type, and also takes the point cloud as input. An example of these inputs and outputs is illustrated in Figure 3.6.

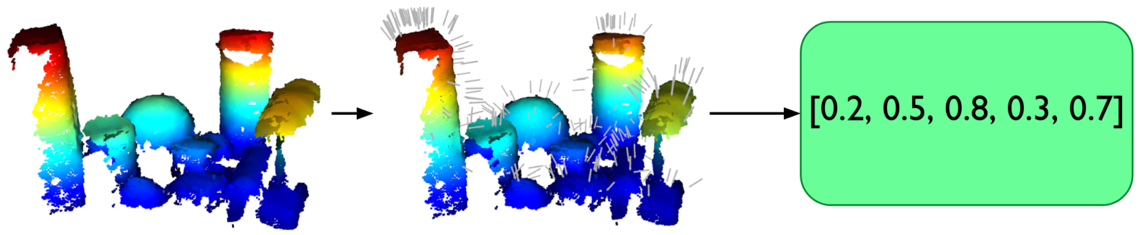


Figure 3.6: An example of the inputs and outputs to each Multi-Modal Grasp Pose Detector subsystem. The input to the system is a point cloud of the graspable objects in the scene, with the table plane and background filtered out. This cloud contains no semantic information that differentiates objects. The points in the example cloud shown are colored based on height. The grasp pose generator takes as input this point cloud and outputs a set of sampled candidate poses. The approach directions of each of these poses are illustrated as gray arrows in the second example cloud. The local geometry about each candidate pose is encoded by centering the point cloud at the grasp and cropping. This encoding is input into my PointConv network, which then predicts the independent probabilities that a grasp of each type would succeed at a given pose. In this example, the five probabilities shown in the green box correspond to the probabilities that a grasp of each of the five types would succeed for one potential candidate pose. Note that these independent probabilities need not sum to one.

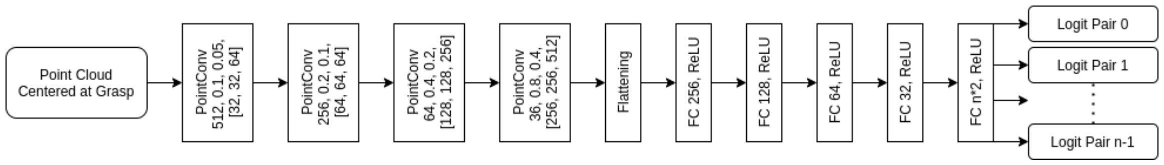

Point Cloud Centered at Grasp
PointConv 512, 0.1, 0.05, [32, 32, 64]
PointConv 256, 0.2, 0.1, [64, 64, 64]
PointConv 64, 0.4, 0.2, [128, 128, 256]
PointConv 36, 0.8, 0.4, [256, 256, 512]
Flattening
FC 256, ReLU
FC 128, ReLU
FC 64, ReLU
FC 32, ReLU
FC n*2, ReLU
Logit Pair 0
Logit Pair 1
Logit Pair n-1

Figure 3.7: Diagram of my Multi-Modal Grasp Pose Detector's neural network architecture. A partial point cloud containing local geometry centered at a grasp pose is input to the network. The PointConv layers at the head of the network learn to encode the input, and the fully connected layers at the tail end output a pair of logits corresponding to each grasp type.

of:

$$\mathbf{GPC} : P_s \rightarrow X_s \in \mathbb{R}^{n\times 2}. \tag{3.1}$$

I output two logits per grasp type in order to train the network to perform binary classification on each grasp type and predict whether a grasp of each type would succeed or fail. These logit pairs are passed through n independent softmax functions. The n resulting probabilities corresponding to positive labels, $\sigma(X_s)_{*,1} = s_s \in [0,1]^n$, can be interpreted as the probabilities that a grasp at g_s of the corresponding grasp type would succeed.

I train MMGPD's GPC to jointly perform n binary classifications using a summed cross entropy loss function. Joint binary classification is useful in cases where multiple entangled predictions are made from a single input source. By training a single network to perform joint binary classification, my system learns an embedding that efficiently encodes the information required to determine whether each grasp type succeeds given a cloud and grasp pose. Though joint binary classification has been proposed to solve problems such as emotion detection [31], mine is the first robotics application I am aware of that uses it. I define this summed cross entropy loss function (Equation 3.3) as a modified form of the standard cross-entropy loss function for m-class classification,

$$-\sum_{c=0}^{m-1} y_c \; log(b_c), \tag{3.2}$$

where y_c is 1 if c is the correct label for a given exemplar and 0 otherwise and b_c is the estimated probability that the exemplar is of class c. Given a labeled grasp exemplar $e = \{g, P, l\}$ where $l \in \{0,1\}^n$ and $P_g = \mathbf{TF}(P, g)$, I compute the summed cross entropy between l and $\sigma(\mathbf{GPC}(P_g)) = Z \in [0,1]^{n\times 2}$. The summed cross entropy loss for my joint binary classification problem is:

$$-\sum_{i=0}^{n-1}\sum_{c=0}^{1} y_{i,c} \; log(Z_{i,c}), \tag{3.3}$$

where $y_{i,c}$ is 1 if $c = l_i$ and 0 otherwise. Training details are found in Section 3.3.1.

In my experiments, given some P of an object or a set of objects, MMGPD's grasp proposal algorithm generates a set of grasp candidates $G = \mathbf{GEN}(P, k)$ where $|G| = k$. With its GPC trained to evaluate grasps for a specific gripper, the network predicts each candidate success probability vector

$$s_s = \sigma(\mathbf{GPC}(\mathbf{TF}(P, g_s)))_{*,1} \ \forall g_s \in G \tag{3.4}$$

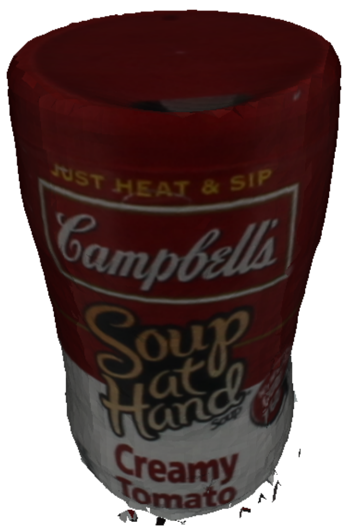
to get $S \in [0,1]^{k\times n}$ where $s_s \in S$. Finally, MMGPD selects the grasp pose $g_m = \{c_m, o_m\}$ and grasp type i_m corresponding to the maximum entry in S that is collision free. When executing this grasp, the gripper is first moved to a pre-grasp pose some distance d away from $\{p_m, o_m\}$ along the negative approach direction. Finally, the gripper is moved to $\{p_m, o_m\}$ and the fingers are closed to complete the grasp.

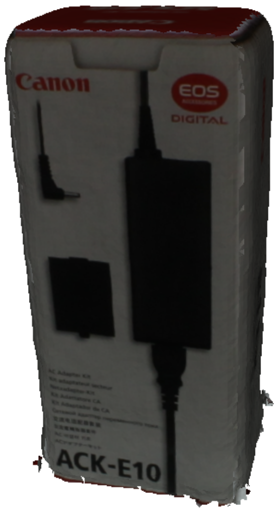
Figure 3.5 shows a block diagram of MMGPD's sub-systems with their inputs and outputs. Given a point cloud, the grasp pose generation algorithm generates a set of candidate poses. Each pose is passed through the PointConv network to predict its n per-type grasp success probabilities. Figure 3.6 illustrates an example of the inputs and outputs of each sub-system. A point cloud is used to generate a set of potential grasp poses (represented by gray arrows). The network predicts the probability that each grasp type would succeed for each grasp pose.

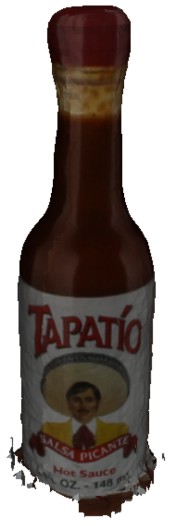
3.3.1 Dataset Generation & Network Training

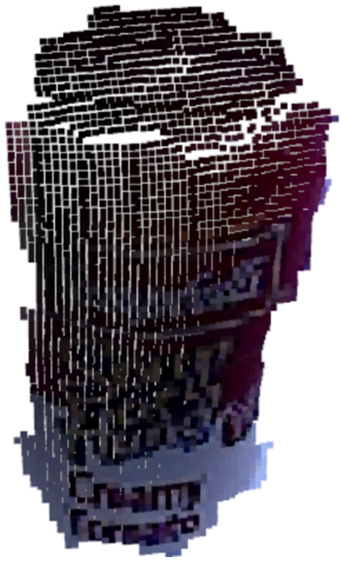
MMGPD's GPC requires a dataset of grasp exemplars E where $e = \{g, P, l\} \in E$ and $l \in \{0,1\}^n$ to train. The BigBIRD dataset [85] contains a set of real partial point clouds captured from 600 viewpoints on a set of common household products and a complete mesh for each object. A subset of point clouds and models of these objects can be seen in Figure 3.8. BigBIRD is a popular dataset for training grasp detection systems since no simulation-to-real transfer is required with real point clouds. My grasp dataset is generated from 20 BigBIRD objects and 12,000 point clouds. I generate a set of grasp candidates G from these clouds using **GEN**.

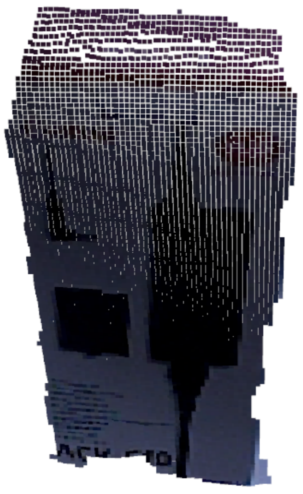
Each candidate is then assigned a label $l \in \{0,1\}^n$. A label is generated by attempting each grasp type i at g in simulation and recording whether or not the grasp succeeds. I perform grasps in the Drake simulation environment [93] to accurately simulate the Robotiq 3-Finger Adaptive Gripper


(a) Campbell's Soup Model


(b) Canon Box Model


(c) Hot Sauce Model


(d) Campbell's Soup Cloud


(e) Canon Box Cloud

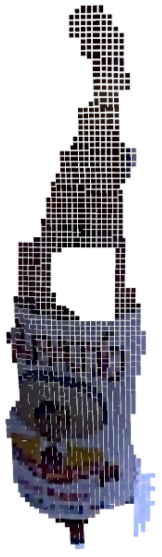
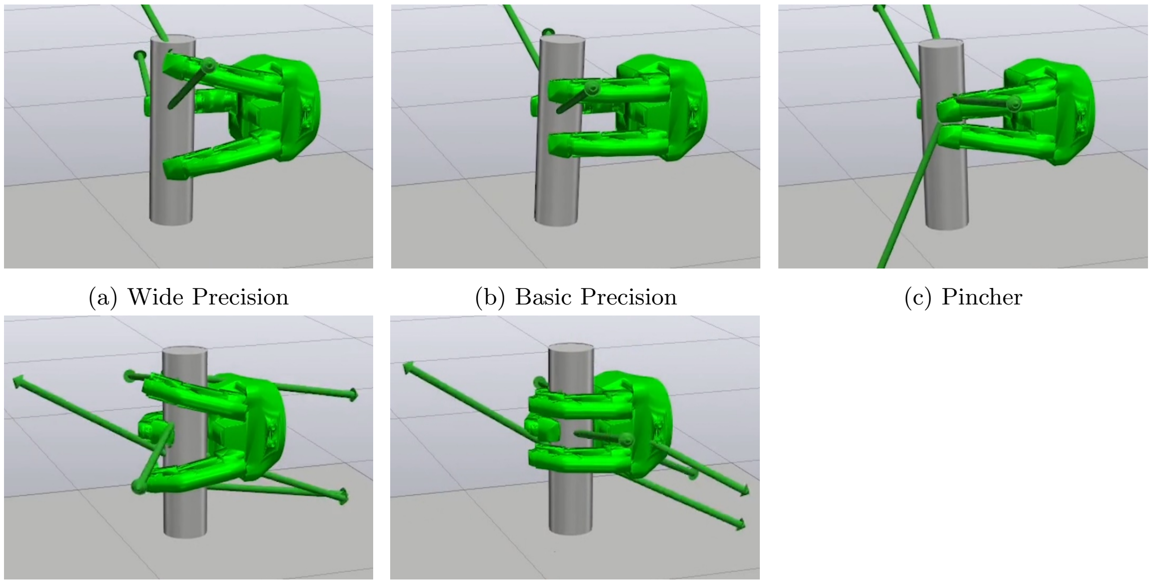

(f) Hot Sauce Cloud

Figure 3.8: Examples of complete models and partial point clouds of three objects from the BigBIRD dataset: a can of Campbell's Soup at Hand, a Canon charger box, and a bottle of Tapatio hot sauce.


(a) Wide Precision (b) Basic Precision (c) Pincher

(d) Wide Power (e) Basic Power

Figure 3.9: The Robotiq 3-Finger Adaptive Gripper's five main grasp types simulated in Drake.

and its grasp types, as shown in Figure 3.9. The simulated graspable object models are based on the complete BigBIRD object models. Each object model is placed on a plane, the gripper model is placed at the grasp pose specified by the sampled candidate g and grasp type i. After the gripper executes a grasp on the object and the plane is removed from the scene, l_i is set to 1 if the object remains between the gripper's fingers or 0 if it falls out of the grasp. Candidate grasps at which no grasp types cause a collision between the gripper and object or table before execution are kept in the dataset, and those that cause collisions are removed. The resulting dataset E contains over 36,000 grasp candidates, each assigned labels l generated by simulating over 180,000 grasp attempts using $n = 5$ grasp types. This dataset is not perfectly balanced between positive and negative labels; the percentages of positive labels for each grasp type are 79%, 43%, 81%, 73%, and 58%. Though my labels are generated on singulated objects, ten Pas et al. [95] showed that systems trained with grasps on singular objects generalize well to real-world clutter.

The network architecture described in Section 3.3 is then trained to perform joint binary classification using the summed cross entropy loss function defined in Equation 3.3. This network is implemented in TensorFlow and trained on a single GeForce GTX 1080 Ti. I use a learning rate of 10^{-5}, a batch size of 16, and the Adam optimizer to train the network. Like ten Pas et al. [95], the cloud input to my network is comprised of two point clouds; one cloud is the BigBIRD cloud used to generate a candidate, and the second is the cloud captured from the same viewing angle 54° away from the first. As the BigBIRD dataset is generated by capturing point clouds of objects from five fixed viewing angles and rotating the object in 3° increments on a turntable, these secondary clouds are available in the BigBIRD dataset. Since my network takes transformed and cropped point clouds P_s as input and classifies 6-DoF grasp candidates, the number of point clouds captured and their viewing angles are arbitrary. I choose to capture two point clouds at training and test time to sufficiently cover the workspace when generating grasp representations P_s, but the configurations need not be the same.

3.3.2 Grasp Types

A grasp taxonomy for any multi-finger robotic gripper could be derived to determine the number of grasp types n that it is capable of, much like the one Feix et al. [21] present to categorize 33 grasp types humans are capable of. Though my framework is compatible with any robot gripper and its n, I train and evaluate MMGPD with the Robotiq 3-Finger Adaptive Gripper, a 4-DoF, 11-jointed underactuated gripper. Unlike the fully articulated grippers used in related work [52], the Robotiq 3-Finger Adaptive Gripper is designed specifically to perform different types of grasps [78]. Each 3-jointed finger is controlled by one motor, and an additional actuator adjusts the orientation of the non-thumb fingers. To avoid self-collision, the gripper's controller allows for three discrete operating modes. The non-thumb fingers are parallel in **basic** mode, spread apart in **wide** mode, and brought together in **pincher** mode. The gripper is capable of performing two types of grips: a **fingertip** or precision grip occurs when the distal links grip an object, while an **encompassing** or power grip occurs when the proximal links first contact an object, causing the fingers to wrap around it. The grip executed depends on the distance from the gripper's palm to the target object. In basic and wide mode, both encompassing and fingertip grips are possible, while in pincher mode, fingertip grips emulate parallel-jaw grippers. These operating modes and grip types enable the gripper to execute five types of grasps, illustrated in Figure 3.2: **basic power** and **wide power** are useful for firmly grasping large objects, **basic precision** and **wide precision** can grasp objects from a surface, and a precision **pincher** that emulates a parallel-jaw gripper is useful for precise tabletop grasps on small objects.

The mechanical design of the Robotiq 3-Finger Adaptive Gripper enables it to execute multiple types of grasps with a single simple control policy without the need for tactile sensors or joint encoders. It enables one to parameterize grasp type over the operating mode and distance from the object to the gripper. To execute a grasp of a given type i_m at candidate $g_m = c_m, o_m$, I assign the gripper a pose $\{p_m, o_m\}$. I define p_m to be the position of the point at the center of the gripper's palm, as shown in Figure 3.10. The orientation of this pose is the same as the candidate's pose. As c_m is a point sampled from the cloud P, assigning $p_m = c_m$ would result in a collision. p_m is instead set some distance d away from c_m along the negative approach direction depending on the grip type required to execute a grasp of type i_m: $p_m = c_m - o_m d$. To execute an encompassing grasp, p_m should be close to c_m so the fingers' proximal links first make contact with the object and wrap around it. p_m should be sufficiently far from c_m during a fingertip grasp so the fingers' distal links would likely make contact with the object. I therefore set $d_e = 1.9$ cm to perform an encompassing grip when executing a basic or wide power grasp, and set $d_f = 8.22$ cm to perform a fingertip grip when executing a basic precision, wide precision, or pincher grasp.

3.4 Measuring Network Generalization

I first evaluate MMGPD by testing its performance on several held-out datasets. In the first scenario, the test set is comprised of 15% of the grasp candidates in my dataset, selected at random, while the remaining 85% are used to train the network. The second, more difficult scenario tests how

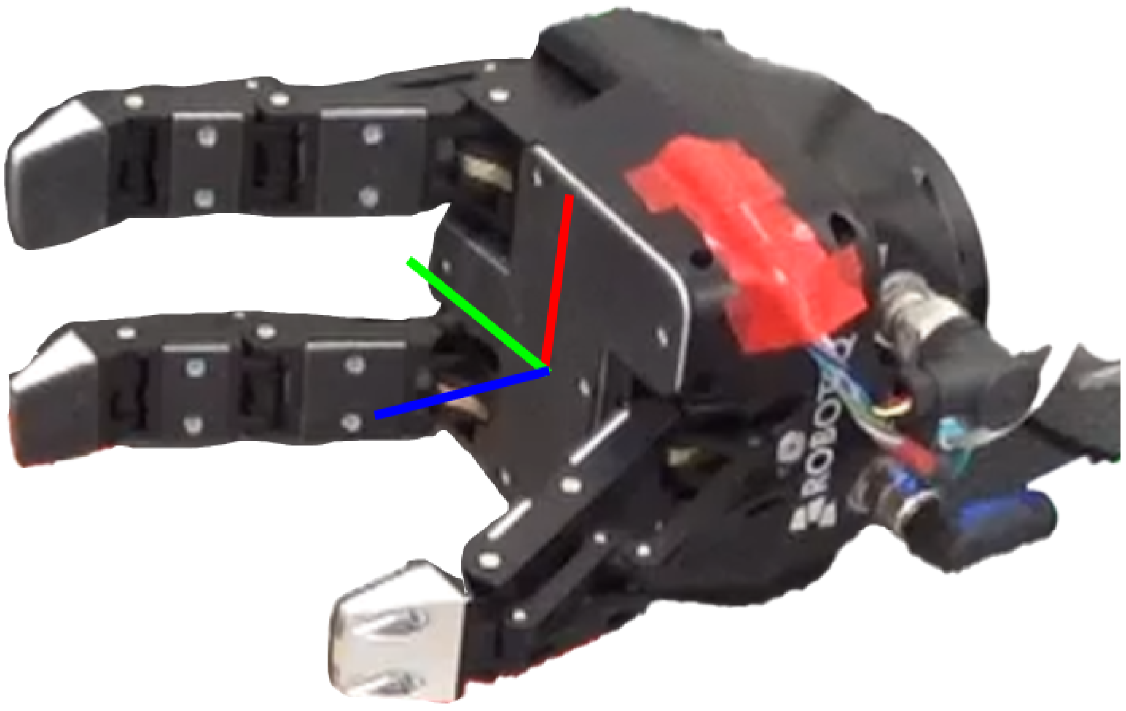


Figure 3.10: A robot gripper's fixed reference frame.

well the system generalizes to unseen objects. Here, the test set contains all grasp candidates from 15% of the objects in the dataset, while the grasps on the remaining objects are used to train the network. In each scenario, I compare my architecture that outputs $n = 5$ predictions per pose from a shared embedding (COMBINED) to a similar architecture that uses an ensemble of n individual deep networks to predict grasp success for each grasp type (SEPARATE). These individual networks are a naive approach that use n times as many parameters as the combined network, but provide an upper bound to compare my system against. Table 3.1 shows the test-set classification accuracies of my proposed architecture (COMBINED) and the ensemble of individual networks baseline (SEPARATE), trained and tested using the two dataset divisions. # Params shows the number of learnable parameters in each system. The final columns show average test-set accuracy, precision, and F1 score at convergence. Per-type accuracy is further broken down in Table 3.2. The final columns here show accuracy for all $n = 5$ grasp types—wide power, wide precision, basic power, basic precision, and pincher. These results are each averaged over three random seeds used to divide the dataset.

Split	Arch.	# Params	Avg Acc	Avg Prec	Avg F1
Rand.	COMBINED	10.4M	0.867	0.879	0.897
	SEPARATE	51.9M	0.871	0.878	0.9
Obj.	COMBINED	10.4M	0.829	0.869	0.881
	SEPARATE	51.9M	0.859	0.876	0.902

Table 3.1: Multi-Modal Grasp Pose Detector simulation performance

Split	Arch.	# Params	T1 Acc	T2 Acc	T3 Acc	T4 Acc	T5 Acc
Rand.	COMBINED	10.4M	0.913	0.802	0.908	0.824	0.886
	SEPARATE	51.9M	0.922	0.804	0.908	0.833	0.886
Obj.	COMBINED	10.4M	0.841	0.805	0.847	0.820	0.835
	SEPARATE	51.9M	0.915	0.818	0.854	0.838	0.869

Table 3.2: Multi-Modal Grasp Pose Detector simulation performance per grasp type

Though test-set accuracy demonstrates how well the system learns, when executed on a real robot, selecting false positives can cause a low grasp success rate. Since the system chooses to execute the one grasp with the highest predicted success probability, the grasp may fail if an incorrectly classified false positive is selected. False negatives, however, are not as detrimental to the grasp success rate since the system will choose only one of the many grasps expected to be successful. It is, therefore, also important to verify the system's precision and F1 score.

As COMBINED is trained to predict success for all grasp types, all statistics are reported at the epoch at which average accuracy across all grasp types is maximum; the individual grasp type accuracies are not necessarily maximum at this epoch. For SEPARATE, since each network is trained with only one grasp type, each accuracy, precision, and F1 score is reported at the epoch at which that grasp type's accuracy is maximum. The reported average accuracy is the average of these maximum accuracies. Despite this, when learning these binary classifiers jointly from a shared embedding, the average classification accuracy decreases by only 0.4% when test objects have been seen and 3.0% when the training and test object sets are exclusive. Furthermore, my system achieves a higher precision in the case where test candidates are selected at random. These experiments show that jointly training individual classifiers from a shared PointConv embedding enables my system to more efficiently classify grasp poses with a negligible loss in performance compared to a similar set of networks with five times as many parameters.

3.5 Clearing a Cluttered Table

I perform real-robot experiments to measure how much multi-modal grasps help to clear objects of varying sizes from a cluttered table. I capture two point clouds from fixed locations as shown in Figure 3.12, then remove all points within a threshold of the known table plane. As described in Section 3.3, I generate a set of 400 grasp candidate poses G using **GEN**, then check each for collisions, both in PyBullet with a simplified mesh and between the cloud and a simplified gripper model. Like ten Pas et al. [95], I also filter candidates with an insufficient number of points in the

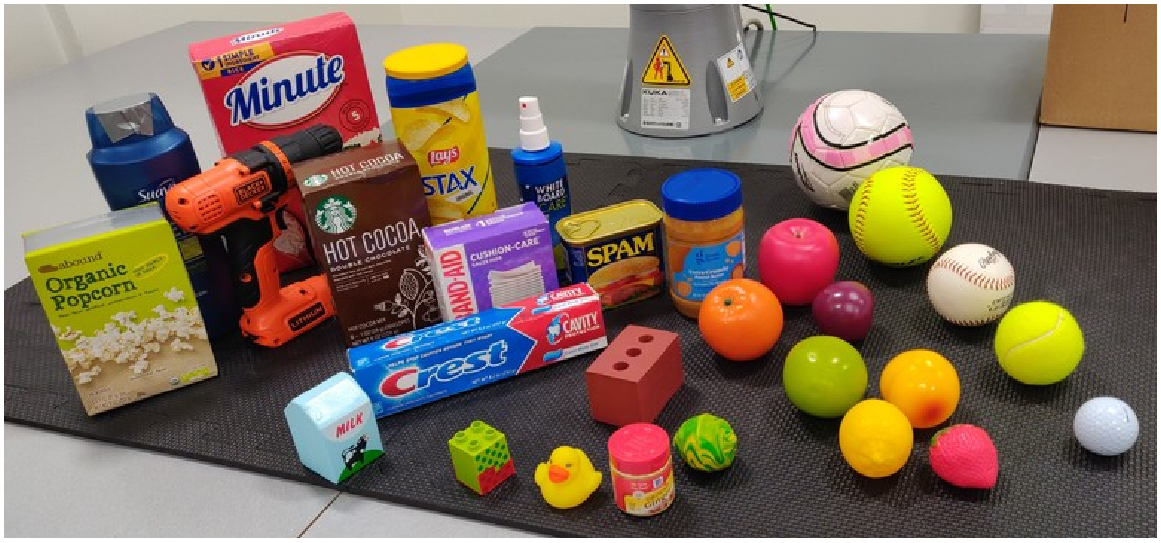


Figure 3.11: The set of objects MMGPD is tasked to grasp in the real-robot experiments. The six large objects on the left of the image are a shampoo bottle, a drill, a popcorn box, a cocoa box, a can of chips, and a box of rice. The 17 medium and six small objects on the right include fake fruit, sports balls, small toys, a box of toothpaste, a box of bandages, a jar of peanut butter, and a tin of Spam. Ten objects are selected from this set at random, and three random large objects are placed around a pile of the small objects to create a complex cluttered scene.

graspable region between the fingers. This is achieved by counting the number of points in the box each finger would sweep through as the gripper closes. I then evaluate the remaining candidates with MMGPD's trained GPC. As detailed in Section 3.3, MMGPD executes grasp g_m of type i_m with predicted success probability S_m with a Robotiq 3-Finger Adaptive Gripper. If the motion planner fails to find a path to g_m, MMGPD executes the next most likely to succeed grasp.

Because related works are evaluated with a variety of datasets and often simpler experimental scenarios, and implemented on different robot hardware, it is difficult to compare my system with them directly. To examine the benefits of multiple grasp types when grasping in dense clutter, I compare my system to two baseline ablations representative of related work whose deep networks have not been trained to assess all five grasp types. The first, 1Type, predicts only the probability that a pincher grasp would succeed, and is representative of systems designed for parallel-jaw grippers [95, 55]. The second, 2Type, predicts whether $n = 2$ grasp types, basic power and basic precision, would succeed; this is representative of the framework defined by Lu and Hermans [52].

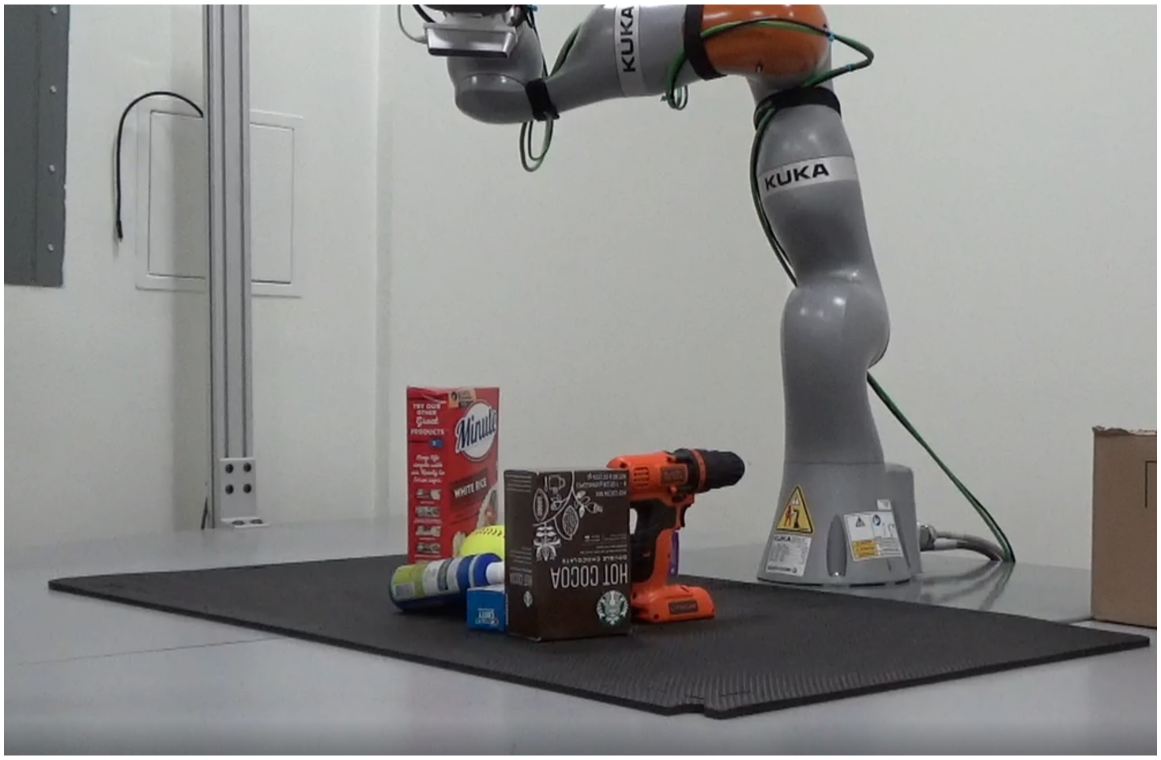


Figure 3.12: The depth sensor mounted to the wrist of the Kuka LBR iiwa 7 robot arm captures a depth image of a cluttered scene. The point cloud captured from this pose is fused with a point cloud captured at another pose and passed as input to MMGPD.

The full MMGPD system, 5Type, models all $n = 5$ Robotiq grasp types.

In each of my experimental trials, three large, upright objects are placed around a pile of ten small and medium-sized objects. This scenario is designed to challenge the system, as it may depend on all five grasp types to clear the table. An example of the clutter my system clears is shown in Figure 3.13. The objects used in these experiments, an augmented segment of the YCB dataset [10] containing six large, 17 medium, and six small objects, did not appear in the training set and can be seen in Figure 3.11.

My procedure follows that of ten Pas et al. [95]; a random selection of small and medium objects is placed in a box, shaken, and dumped into a cluttered pile on a table; large items are placed around the pile after dumping the box to ensure they remain upright. The system attempts to grasp objects until either 1) the same type of failure on the same object with the same grasp type fails three times

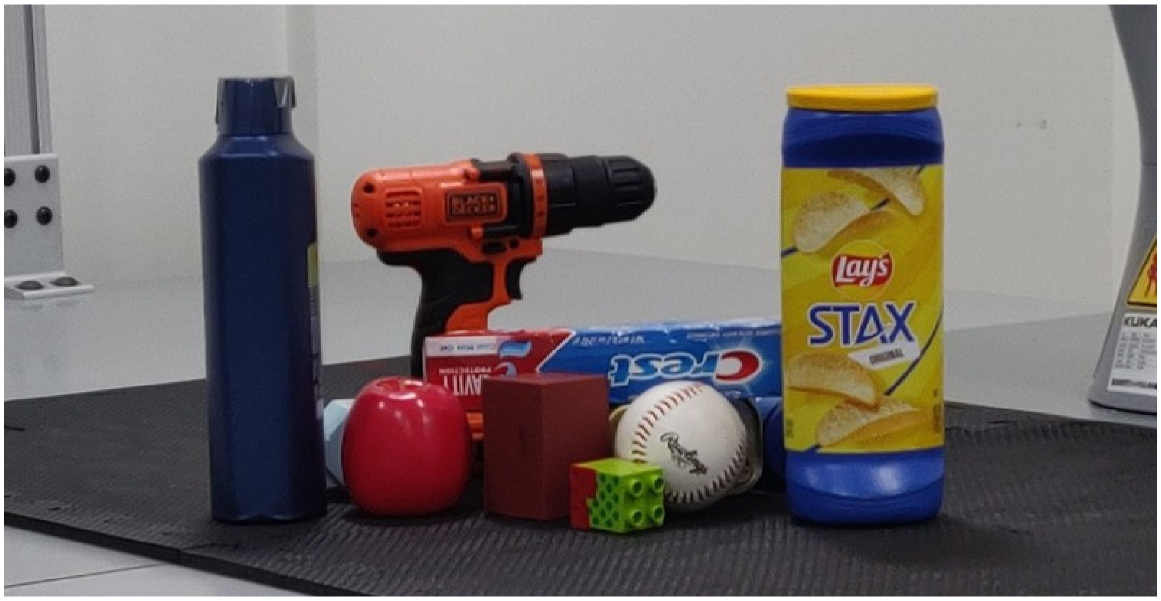


Figure 3.13: A cluttered scene the Multi-Modal Grasp Pose Detector is tasked with clearing.

in a row, 2) the system fails to generate reachable grasps in three consecutive attempts, or 3) all objects are removed from the table. If the system fails to find a feasible grasp or the motion planner fails to find paths to the top 25 grasps, I repeat the candidate generation process up to two more times, each time proposing twice as many candidates. A grasp is successful if one or more objects are lifted from the scene and moved towards a box, and do not fall from the gripper until the fingers are opened. If an object leaves the workspace during an unsuccessful grasp or during a grasp on a different object, it is placed back in the scene near where it left, abutting as many other objects as possible. If an unforeseen collision occurs during a grasp or placement execution, the disturbed objects are reset and the attempt is not counted. Each system is presented with the same objects as the other systems in each of the ten trials, but in a different random configuration. I report both the grasp success rate (number of successful grasps divided by number of attempted grasps) with number of attempts and overall object removal rate (number of objects removed from table at the end of a trial divided by initial number of objects). The results on the real-world experimental scenario averaged across ten trials are shown in Table 3.3.

	Success Rate	#Attempts	Removal Rate
5Type	0.808	125	**0.808**
2Type	0.692	120	0.654
1Type	**0.825**	114	0.723

Table 3.3: Multi-Modal Grasp Pose Detector real-world experimental performance

3.6 Discussion

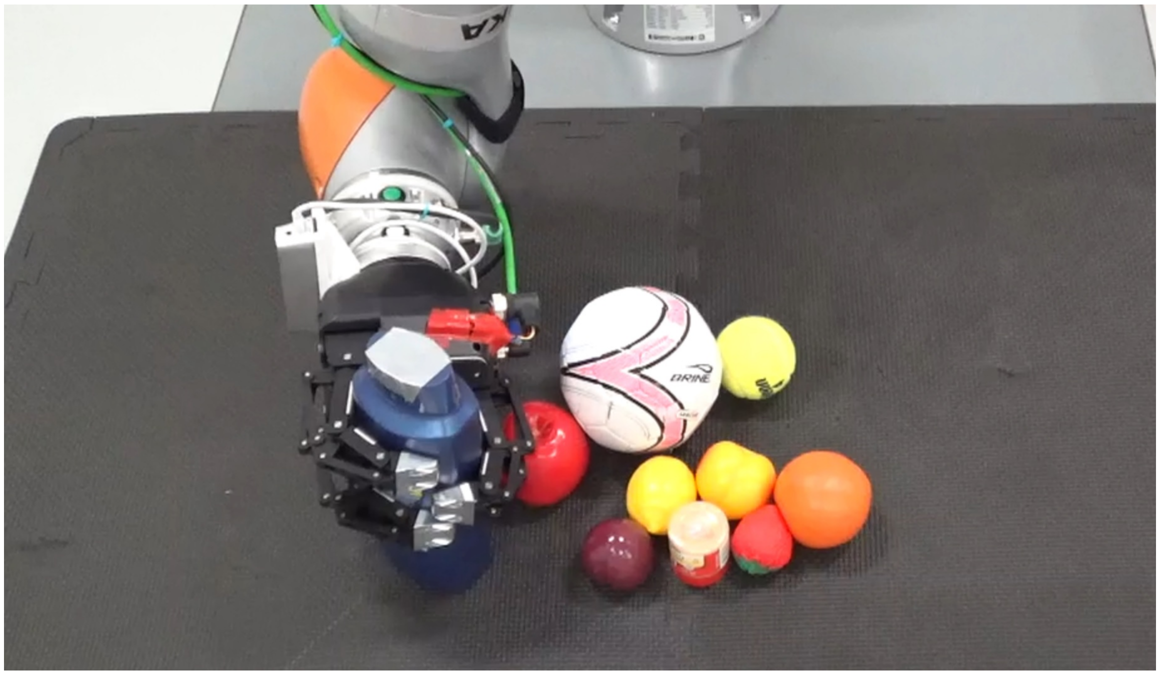


Figure 3.14: The Multi-Modal Grasp Pose Detector successfully lifts the shampoo with a power grasp.

As seen in Table 3.3, my system with access to all five grasp types, 5Type, outperforms the ablations of my system, 2Type and 1Type, when grasping items from cluttered scenes surrounded by large objects. 5Type achieved the highest object removal rate by successfully clearing nearly all scenes. In some trials, though **GEN** generated grasps on the small Lego or duck, the motion planner failed to find a path to these grasps. A common failure mode of the system, which is a common failure mode in other grasp detection systems [95], was that it attempted to grasp multiple objects at once. The 5Type system suffers from this issue more than the baselines because it has access to wide-type grasps that spread the fingers out and are more likely to contact multiple cluttered objects. The issue, illustrated in Figure 3.15, could be alleviated with an off-the-shelf depth image segmentation algorithm and an additional filtering step in **GEN**. Other failure modes of my system include false positives and objects moving as the fingers close. The system also struggled with the heavy shampoo bottle in two of the five trials it appeared in. Because my grasp evaluation network makes predictions based only on local geometry, it incorrectly predicted that unstable grasps would succeed. Since my system has no notion of grasp history, it became stuck in a local minima and unsuccessfully attempted similar grasps three times in a row, ending the trials. However, in three of the trials, it correctly used power grasps to stably lift the heavy bottle as shown in Figure 3.14.

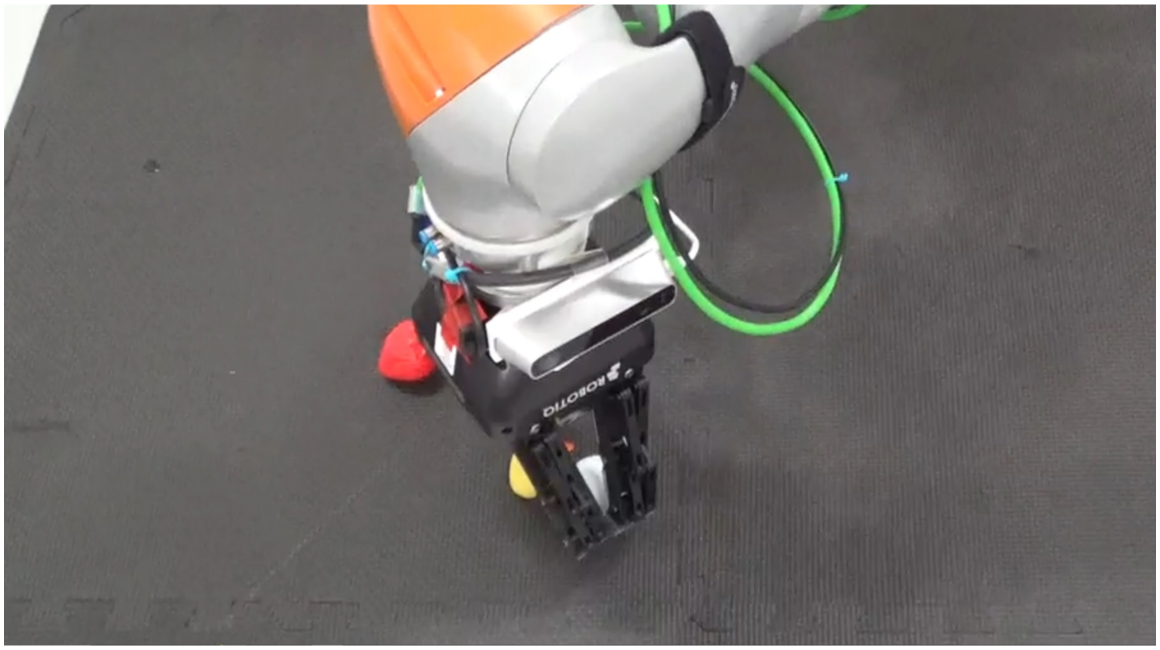


Figure 3.15: The Multi-Modal Grasp Pose Detector executes a grasp on two objects.

Since 2Type is incapable of performing pincher grasps, it often fails to find feasible grasp candidates on small objects that fit between the gripper's spread fingers. The system encountered difficulty with the rice box in one trial and the pear in another, objects that the 5Type system never failed to grasp. However, because this ablation did not have access to the weak but precise pincher grasps, it was able to successfully grasp the heavy shampoo bottle in three of the five trials it appeared in. The 1Type system outperformed the 2Type system in these experiments because pincher grasps succeeded on the small or medium objects that made up the majority of my object set. Even most of the large objects were light enough that they could be lifted with a pincher grasp. 1Type attempted to grasp multiple objects at once less frequently because the fingers are not spread apart during a pincher grasp. The 1Type system successively failed to lift the heavy shampoo bottle by its cap three times in three different trials, ending the trials prematurely and decreasing the object removal rate.

My 5Type system was able to choose applicable grasp types for the situation to clear each scene more completely. It used a wide power grasp to stably grasp the top of the drill, and another to perform a spherical grasp on the soccer ball, one of the largest medium-sized objects. It used basic power grasps when faced with the heavy shampoo. The 2Type system also selected basic power grasps for these objects. Overall, my 5Type system executed three wide power, 33 wide precision, six basic power, 38 basic precision, and 45 pincher grasps. Collision-free power grasps were rarely generated on the small and medium-sized objects because they lie close to the tabletop. As the large objects were often partially occluded by the adjacent clutter pile, their larger graspable areas were hidden. Though my 5Type system chose to use fingertip grasps in the many applicable scenarios, it used power grasps to lift the heavy objects the 1Type system often could not.

3.7 Conclusion

The simulation experiments presented in Section 3.4 show that my Multi-Modal Grasp Pose Detector is able to efficiently learn to jointly evaluate multiple grasp types for a given grasp candidate. My real-world experiments show that this architecture enables a robot equipped with a multi-finger gripper to more successfully clear a scene of cluttered objects of various sizes from a tabletop than systems that use fewer grasp types.

Chapter 4

Learning Task-Oriented Grasps from Limited Labeled Data

A given manipulation controller's task success is dependent on the initial state of a robot. When the robot is provided with grasping as a skill, this initial state is reached by executing a grasp at a given pose. In this chapter, I present a task-specific grasp classifier that predicts whether a grasp at a given pose would enable a given manipulation controller to successfully complete a task. This classifier learns from small amounts of labeled data to predict which grasps enable a task policy to succeed. This lightweight classifier is a key technological advancement required to simultaneously learn a manipulation policy and which grasps enable this policy to succeed. I investigate combining this classifier with a reinforcement-learning agent further in Chapter 5.

4.1 Introduction

The general grasp detector defined in Chapter 3 evaluates the quality of a grasp pose based on the visually detected geometry local to the grasp region. This information is sufficient for finding the most stable grasp in clutter, where the robot must grasp one of many objects and lift it upwards.

However, pick and place is not the only, or even the most common, reason for a robot to grasp an object. When completing more complex manipulation tasks, the local geometry is insufficient for classifying a grasp pose because grasp stability is a necessary but not sufficient metric for task-oriented grasping. While many grasps on a hammer would be stable, only the grasps on the handle that allow the head to face forward afford the robot the ability to complete a hammering task, as seen in Figure 4.1. The local geometry of grasp poses along the hammer's handle, or the grasp poses around the handle at a fixed height, would all appear visually similar or identical when passed to the general grasp pose detector. Instead, global object geometry should be considered when evaluating grasp poses for task-specific grasping.

Another roadblock encountered when modifying the generic grasp quality predictor to perform task-specific grasp classification is the amount of training data necessary to train a DNN. Over 180,000 grasp attempts were simulated to generate a dataset of labeled grasp poses. The sim-to-real gap was addressed by training with generic, singulated objects; as ten Pas et al. [95] showed, training on the local geometry of singulated objects in simulation transferred well to real-world grasping tasks. Gravity and contact physics for grasping a singulated object are modeled well by simulators. However, it is unreasonable to expect to have realistic physics models of each object a robot could encounter in the real world. Furthermore, a robot may interact with an object such as a heavy door for the first time in the real world. If all of its training experience was with lighter doors, or doors with different types of handles, a task-specific grasp classifier would fail. Therefore, the robot should learn to interact with each instance of a new object, whether it be discovered in simulation or in the real world. As obtaining 180,000 labeled examples in the real world is infeasible due to the time it would take, a task-oriented grasp detector should learn from few labeled examples.

Existing solutions to the task-oriented grasping problem require tens to hundreds of thousands of training examples, rely on human-labeled data, or make assumptions about the problem and the information available to the robot. Many existing solutions for task-oriented grasping rely on human-labeled data [39, 51, 17], which incorporate human biases and incorrect assumptions about

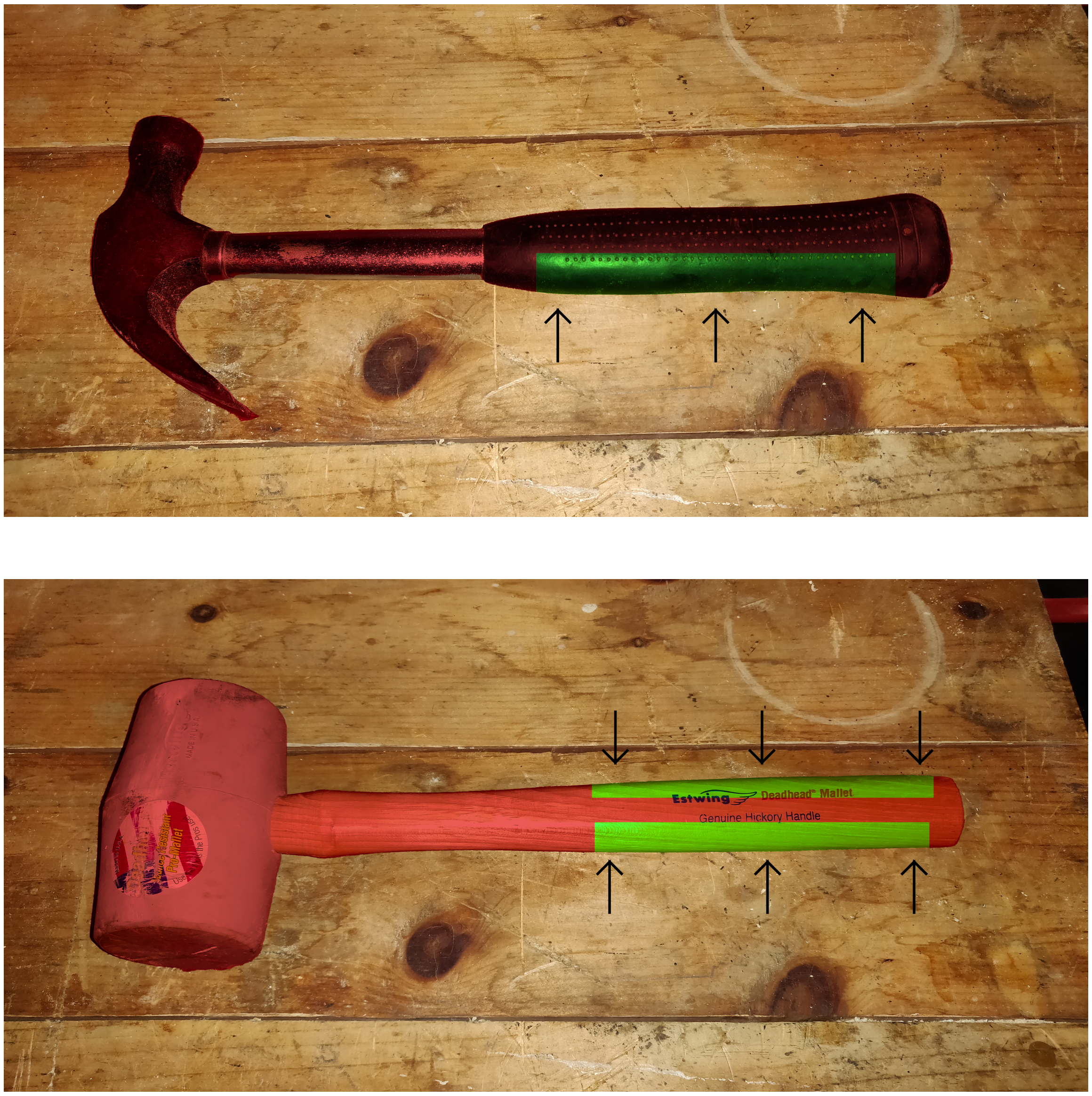


Figure 4.1: Task-oriented grasps for a hammer and a mallet. While grasps centered in the red areas could successfully lift the hammer or mallet, grasps centered in the green areas that align with the illustrated arrows would allow a user to use the tools for their intended purposes.

potentially unknown policies. Keypoint and affordance-based methods focus on segmenting objects and assuming grasps on the desired segment are appropriate for task completion [26, 75, 62]. Some recent deep-learning-based approaches have been proposed, though all require tens to hundreds of thousands of training examples [106, 102, 109, 18]. Two recent works leverage limited data learning techniques to grasp objects for general pick and place tasks, but do not perform task-oriented grasping [23, 29].

In order to quickly learn which grasps enable a robot to complete a given task, I propose the Augmented Task-Oriented Grasp Detection Network (ATOG), the first deep-learning-based approach that learns to detect task-oriented grasps with limited training data, while learning directly in the environment it is evaluated in. ATOG consists of a deep neural network with two branches. The first, based on a general grasp detection network, is pre-trained to predict whether a given grasp is stable from local geometry using a dataset of general grasps. This branch takes a partial point cloud of the object, centered and cropped at the given grasp pose, and encodes it using a PointConv-based architecture [104]. After training the general grasp detection branch of ATOG, the weights of this branch are frozen. The tail layers of the network are a small set of dense layers suitable to be trained with a small dataset. The second branch takes the 6-DoF grasp pose relative to the object, encoded as the position and quaternion in a fixed object frame. By providing the network with this pose, it can learn to detect which regions of the object provide grasps suitable for a task. These weights are adjusted on a per-task basis.

I evaluate ATOG in several simulated domains, illustrated in Figure 4.5, in which a robot arm performs a task with an articulated or free-body object. To generate a small dataset with which to train ATOG, a policy parameterized by its initial state (a grasp on the target object), is attempted at a small set of random and kinematically valid proposed grasp poses. Task success is recorded at each pose to train ATOG to perform binary classification. The network is updated each time a new example is added to the training set, and the framework is re-evaluated with a large held-out task-labeled evaluation set. ATOG's capabilities are also demonstrated with a real robot task.

I present four major contributions in this chapter. The first is the Augmented Task-Oriented Grasp Detection Network's architecture that combines object geometry with a grasp pose's relation to the object to predict whether the grasp would enable a robot to manipulate an object successfully. The second is a training procedure for ATOG that enables it to learn to predict task-oriented grasp success from a single-digit-sized training set. The third is a new metric that considers the precision and kinematic capabilities of real robots to evaluate ATOG's performance on a held-out test set. Finally, I demonstrate ATOG's capability to learn from small amounts of data by demonstrating learning on a real robot.

4.2 Background and Related Work

In grasp pose detection, as described in Sections 2.1 and 3.2, a robot is tasked with selecting a grasp pose g at which a stable grasp is likely to be formed given some representation of the scene. The robot would then execute a grasp at this pose, with the objective being to lift the object and place it in a goal location such as a box; grasp stability is the only metric optimized over. However, to perform manipulation tasks, robots must grasp objects in ways that are stable but also afford them the ability to complete tasks, as stability is a necessary but insufficient condition for completing a task with a grasped object. For instance, while a robot could stably grasp a hammer on the head, only specific grasps on the handle allow the robot to hammer in a nail. This more constrained grasp pose detection problem is called task-oriented grasp pose detection, and it is defined in detail in Section 2.2.1.

The goal of task-oriented grasp pose detection is to return a 6-DoF pose $g \in \mathbb{SE}(3)$ such that if a robot were to execute a grasp at g, it could successfully complete a downstream manipulation task. In this work, I assume that the pose g is in the robot frame, and that the robot can plan a path to a set of joint states (found through inverse kinematics) that place its end effector at the grasp pose. I also assume that the manipulation policy $\pi_{\mathbf{M}}$ that attempts to complete the task, parameterized by its initial grasp, is provided.

Existing approaches for task-oriented grasping can be broken into several categories: systems that require human-labeled data, optimization-based approaches, affordance and keypoint detectors, and deep-learning-based networks. However, ATOG is the first to train a deep network from small data sets to perform task-oriented grasp detection. A comprehensive review of existing task-oriented grasping approaches is presented in Section 2.2.2. In this section, I relate these task-oriented grasping works to ATOG specifically. Many approaches, both those that incorporate deep learning and earlier techniques, rely on humans to inform robots on which grasps are suitable for a given task. Some systems require a database of human-demonstrated task-oriented grasps, either to train a classifier or to use as a lookup table [19, 20, 46, 5, 39]. Others use a database of robot grasps with task labels annotated by a human expert [14, 6, 86, 33, 17, 65, 107, 51]. Some require rules or heuristics defined by human experts [73]. A major issue with human-labeled data is that it contains human biases. Without first-hand knowledge of the results of a grasp execution, grasps that a human thinks may enable a robot to complete a task could fail. Additionally, some tasks may prove too difficult for humans to accurately label because the manipulation controller or task policy could act counter to their intuition, especially when a sub-optimal controller performs the task after grasping. Rather than rely on human-labeled data, robots should learn through interaction with the environment in which they execute tasks through first-hand experience, like ATOG does.

Some optimization-based systems encode tasks as desired object wrenches [30] or trajectories [72]. The system described by Haschke et al. [30] is limited in that tasks must be encoded as single motions. Pardi et al. [72]'s framework selects a stable grasp that enables the robot to optimally move the target object along a specified trajectory without collisions. However, this system does not take properties such as a grasp's relation to the object as a whole into consideration. ATOG is compatible with tasks that cannot be encoded as single motions, and it learns to select grasps that stably grasp the object, avoid collisions during policy execution, and enable a given policy to complete the task.

Affordance-based approaches have become popular as deep neural networks have enabled accurate pixel-level classification algorithms. In many of these works, objects are detected and segmented in visual images, with each segment assigned a set of tasks for which a grasp would afford completion [62, 38, 12]. However, for complex manipulation tasks, it cannot be assumed that all stable grasps on a segment of an object afford the robot the ability to complete a task. Some stable grasps may be reachable but place the robot in an awkward configuration from which it is difficult to complete the task. Others may not enable it to complete the task as optimally as others. ATOG considers grasps across the entire object without making assumptions as to which segments afford the most optimal grasps. Similar to affordances, other systems detect keypoints on manipulable objects and plug these keypoints into their manipulation algorithms. The kPAM line of work involves detecting keypoints on objects that are used for planning [58, 25, 26]. KETO and affordance coordinate frame-based systems also rely on keypoints to perform manipulation [75, 11]. While keypoints enable these systems to optimize over manipulation actions directly, these approaches make simplifying assumptions about grasping.

Other recent data-driven approaches integrate deep neural networks into their frameworks in other ways. In some, a robot learns which grasps afford it a task through exploration within the task environment from scratch [106, 102, 109, 18]. However, like many other works cited here, these systems are deep-learning based, requiring tens to hundreds of thousands of training examples to learn to complete these tasks. When a robot in the wild is introduced to a new task, it must learn which grasps afford it to complete the task in a reasonable amount of time, perhaps 20 interactions.

Few works have studied grasping using limited data, and I am not aware of any that learn task-oriented grasp detection from small datasets. Fleytoux et al. [23] efficiently learn to classify top-down grasps using a deep auto-encoder and a Gaussian process classifier. They train a β-Variational Auto-Encoder to generate a small latent state representing top-down image patches centered at 3-DoF grasp poses. They then train a Gaussian process classifier to predict whether a given grasp would succeed from the latent representation corresponding to its image. Though this framework enables a robot to identify stable grasps on objects from different viewpoints using a single-digit-sized training set, it only evaluates the grasp stability from local geometry and does not perform task-oriented grasp detection.

Guo et al. [29] present IGML, a meta-learning framework that efficiently learns to grasp specific objects from clutter. The network is first trained with a large dataset of RGB-D images to detect 4-DoF grasps on desired objects at their specified grasping points. The network adapts when presented with a new object; given a small number of examples of grasps on a target object, the meta-learner is fine-tuned to perform grasps on the unseen object. This system requires a large amount of data on the grasping task for the initial training; the small dataset is only used to train the network on a specific new object. By contrast, ATOG learns a completely new task with very few task-labeled examples.

ATOG learns task-oriented grasps using data collected in the task domain directly by the robot, simulated or physical. This system does not rely on biased human data, segmented affordance models, or deep learning architectures that require large amounts of data. This system enables a robot to learn task-oriented grasps from small amounts of data and robustly manipulate novel objects as quickly as possible.

4.3 Augmented Task-Oriented Grasp Detection Network

The previous work listed in Section 4.2 is lacking because it does not enable a robot to learn which grasps afford it the ability to complete a task using small amounts of data from the robot's environment. ATOG, by contrast, quickly learns which grasp poses enable it to complete a task using an augmented pre-trained grasp quality network. It learns task-oriented grasps using data collected in the task domain directly by the robot, simulated or physical. This system does not rely on biased human data, segmented affordance models, or deep learning architectures that require large amounts of task data; instead, it learns task-oriented grasps from small amounts of data, allowing it to robustly manipulate novel objects as quickly as possible.

The key insight behind ATOG is that a stable grasp is a pre-requisite for manipulation. Stable grasps can be learned effectively from simulated datasets, even when evaluated on real-world objects that differ from the training objects [95]. ATOG takes advantage of this by bootstrapping its learning process with a pre-trained general grasp classifier, enabling it to fine-tune its weights given single-digit numbers of task-specific training examples.

ATOG consists of a deep neural network trained in two steps. First, a PointConv-based architecture learns to predict whether a robot can execute a stable grasp at a given pose. This network is trained with a large dataset of grasp attempts generated in simulation. To learn which grasp poses afford a robot to complete a specific task, the pre-trained PointConv branch of the network is frozen, and the remainder of the network is trained to predict whether the desired task would succeed given the grasp pose in a fixed object frame and the probability that a stable grasp could be executed at the pose.

ATOG follows the common two-step proposal-evaluation paradigm: to select a grasp, a grasp generation algorithm proposes a set of potential candidate grasps and ATOG evaluates each of these candidates and to determine which has the highest probability of success. My generator **GEN** : $P \to G$ is adopted from ten Pas and Platt [94], and is described in detail in Section 3.3. A set of grasp poses $G \subseteq \mathbb{SE}(3)$ is proposed from a partial point cloud of the graspable area of the target object $P \subseteq \mathbb{R}^3$; this cloud is first cropped to remove the background and ungraspable parts of the target objects, such as the door frame. **GEN** posits that a grasp pose could be centered at any point $p \in P$. Each p is assigned an orientation o such that the gripper approaches anti-parallel to the estimated normal at p, and the gripper closes along the curvature estimated at p.

4.3.1 ATOG Network Architecture

Figure 4.2 shows the ATOG network diagram. The general grasp stability evaluation branch, highlighted in yellow, predicts whether a grasp pose will enable a robot to lift an object given a patch of the point cloud centered at the grasp pose. This PointConv architecture is based on the networks

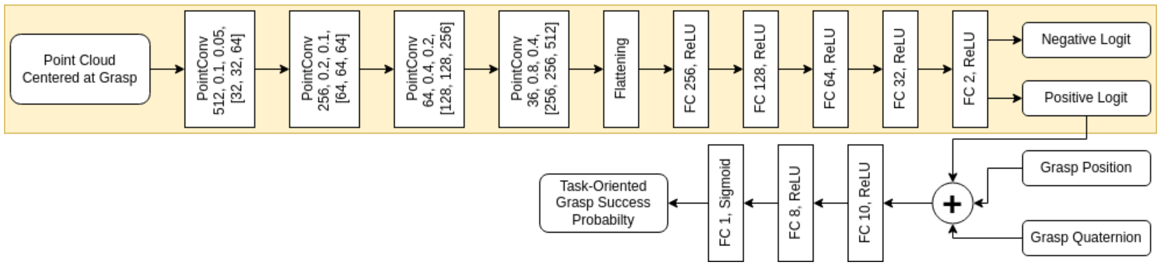

Point Cloud Centered at Grasp
PointConv 512, 0.1, 0.05, [32, 32, 64]
PointConv 256, 0.2, 0.1, [64, 64, 64]
PointConv 64, 0.4, 0.2, [128, 128, 256]
PointConv 36, 0.8, 0.4, [256, 256, 512]
Flattening
FC 256, ReLU
FC 128, ReLU
FC 64, ReLU
FC 32, ReLU
FC 2, ReLU
Negative Logit
Positive Logit
Grasp Position
Grasp Quaternion
FC 10, ReLU
FC 8, ReLU
FC 1, Sigmoid
Task-Oriented Grasp Success Probabilty

Figure 4.2: A neural network architecture diagram of my Augmented Task-Oriented Grasp Detection Network. Highlighted in yellow is the pre-trained branch that predicts general grasp success given a point cloud centered and cropped at the grasp pose. The softmax of the positive logit from this branch is concatenated with the grasp position and grasp quaternion in the fixed object frame. The tail of the network then predicts whether a manipulation policy could complete a task from the given pose.

proven effective for grasp stability prediction by der Merwe et al. [16] and the Multi-Modal Grasp Pose Detector defined in Chapter 3 [13], though I do not train it to complete object models or use multiple types of grasps. The output from this branch of the network can be interpreted as the probability that a grasp is stable enough to lift an object. This value is then concatenated with the pose of a task-oriented grasp in a fixed object frame, represented as a position vector and quaternion as shown in Figure 4.3. The grasp pose in a fixed frame is useful for task-oriented grasping because the policy is expected to perform similarly on certain types of stable grasps on certain segments of the object, enabling the network to learn to implicitly segment the object and cluster grasp poses. This concatenated vector is fed to the three dense layers that compose the tail of the network. These layers learn whether a grasp at the given pose enables the robot to complete the task given a grasp quality prior prediction.

4.3.2 Efficiently Training ATOG

First, the PointConv branch of the network is isolated and trained to perform general grasp detection. ten Pas et al. [95] showed that grasp detection frameworks perform better when the training set contains the same class of objects as the training set. As each of the manipulation tasks involves

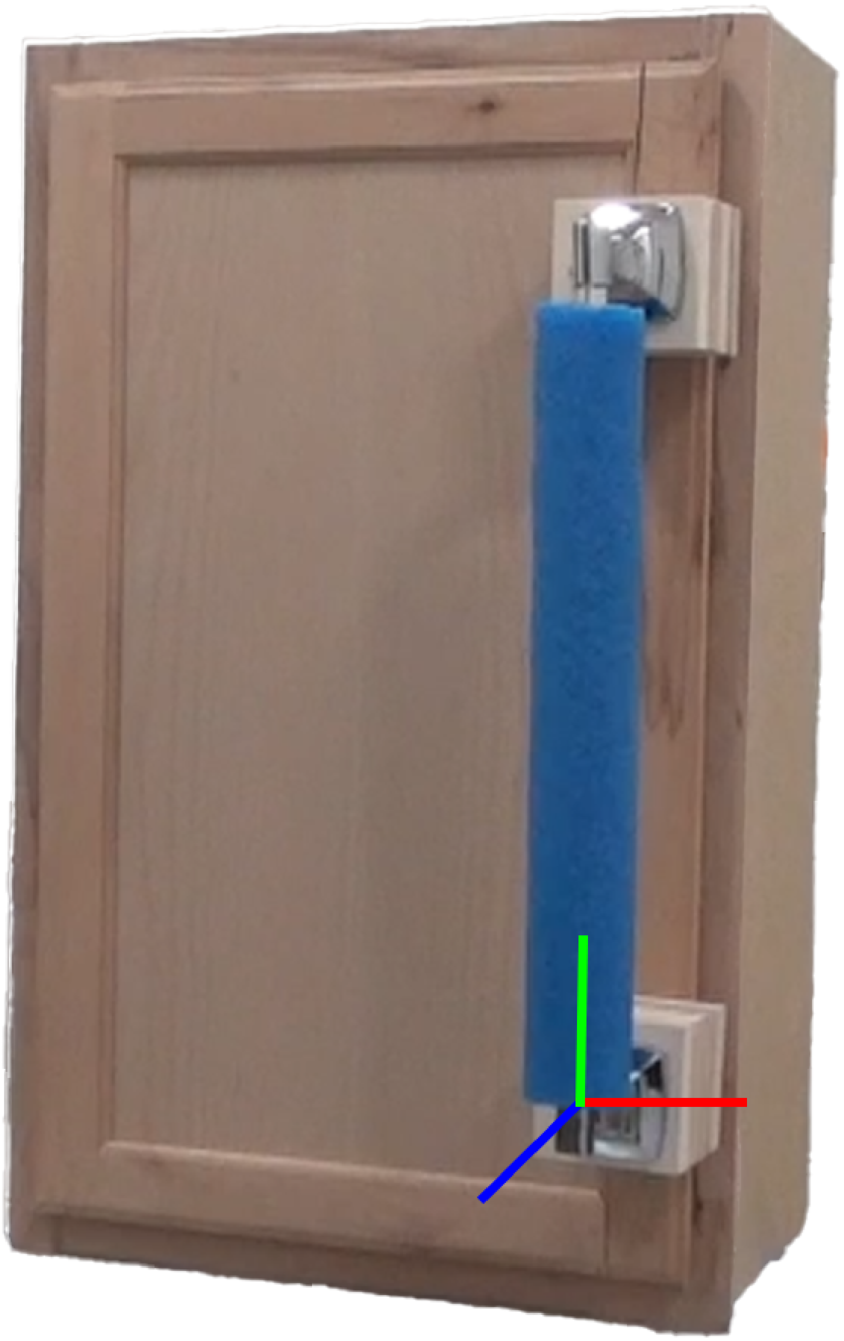


Figure 4.3: An example of the fixed frame of reference for the door handle. All poses passed to an ATOG classifier trained to open this door are represented in this frame.

grasping cylindrical objects, I train the PointConv branch of ATOG with data obtained from attempting grasps in simulation on a set of 10 cylinders with varying heights, radii, and masses. Each cylinder is simulated in MuJoCo [97], where partial point clouds are generated from two simulated depth sensors. Grasp candidate poses are generated with **GEN**, then the robot attempts a grasp at each collision-free candidate to record a binary success label. The PointConv network is trained with over 25,000 labeled grasp poses and their simulated point clouds for 50 epochs.

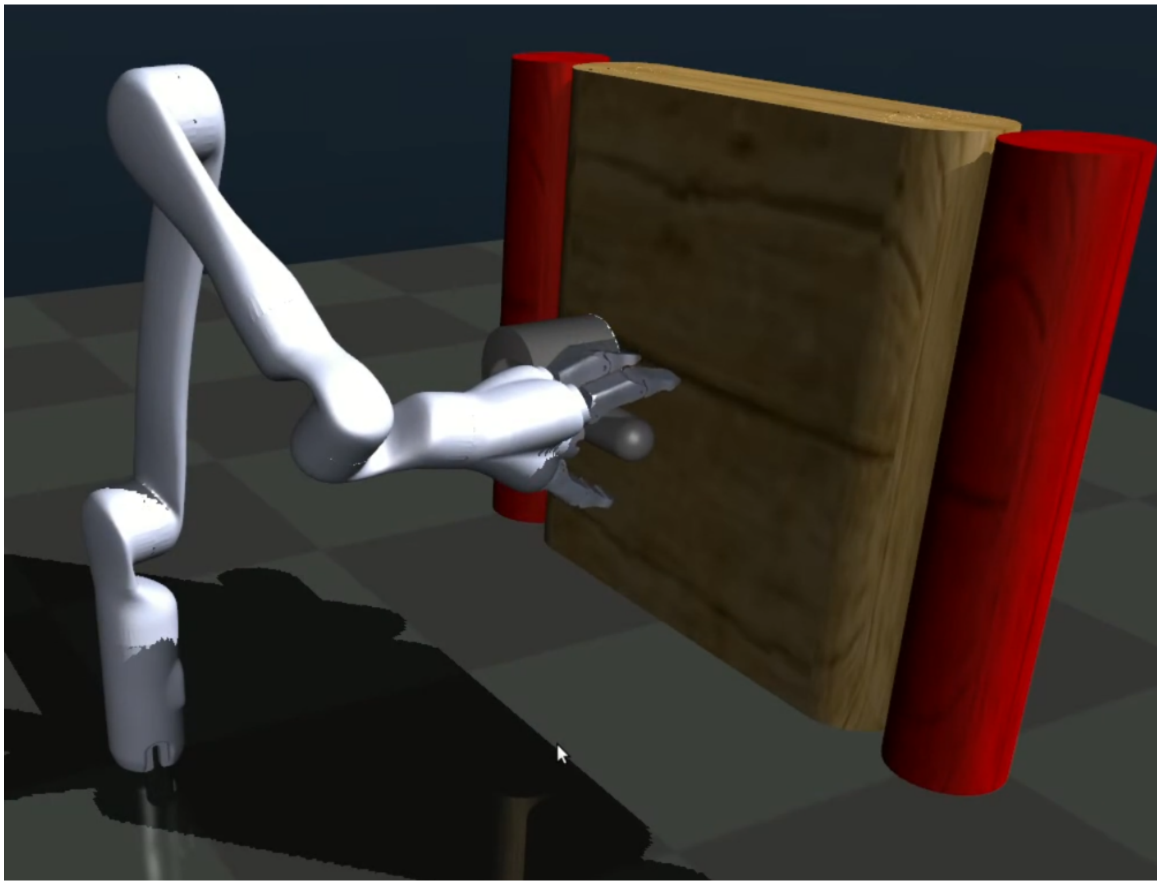
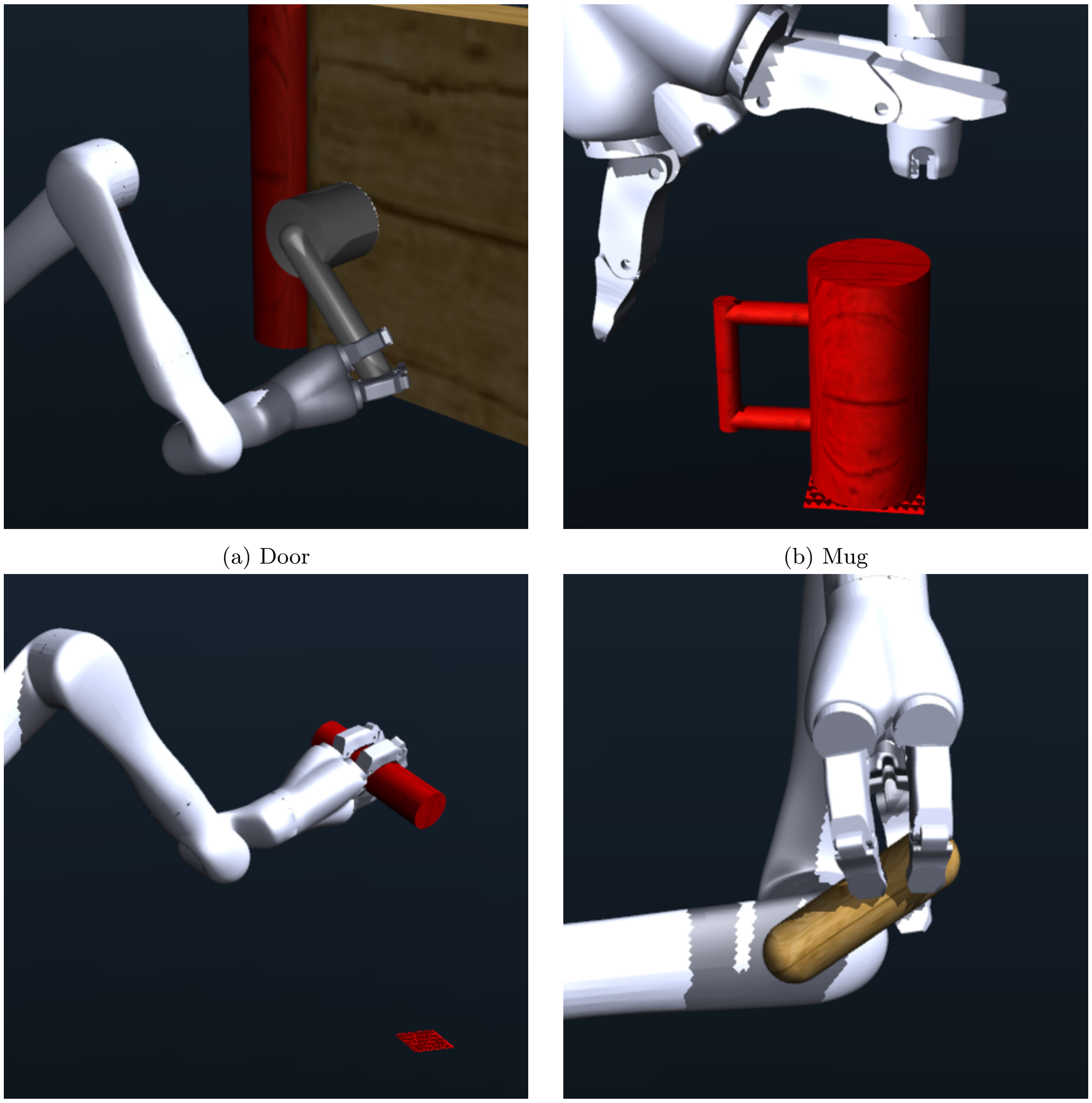


Figure 4.4: A simulated robot arm attempting a grasp on a door handle in the MuJoCo simulation environment.

When training the full network to predict task-oriented grasps, the pre-trained PointConv branch's weights are loaded and frozen. This drastically reduces the number of learnable parameters in the network, enabling ATOG to learn from limited labeled data.

4.4 Experiments

My experiments aim to illustrate: 1) how quickly ATOG learns, and 2) how ATOG performs against several baselines, including a deep learning approach, using a metric relevant to robot manipulation. To verify the algorithm, I evaluate it in several simulated environments, shown in Figure 4.5.



(a) Door (b) Mug

(c) Pitcher (d) Switch

Figure 4.5: Four simulated tasks that my Augmented Task-Oriented Grasp Detection Network is evaluated on. The robot arm must rotate a door handle, lift a mug, lift and pour from a pitcher, and flip a large switch upwards.

4.4.1 Environments

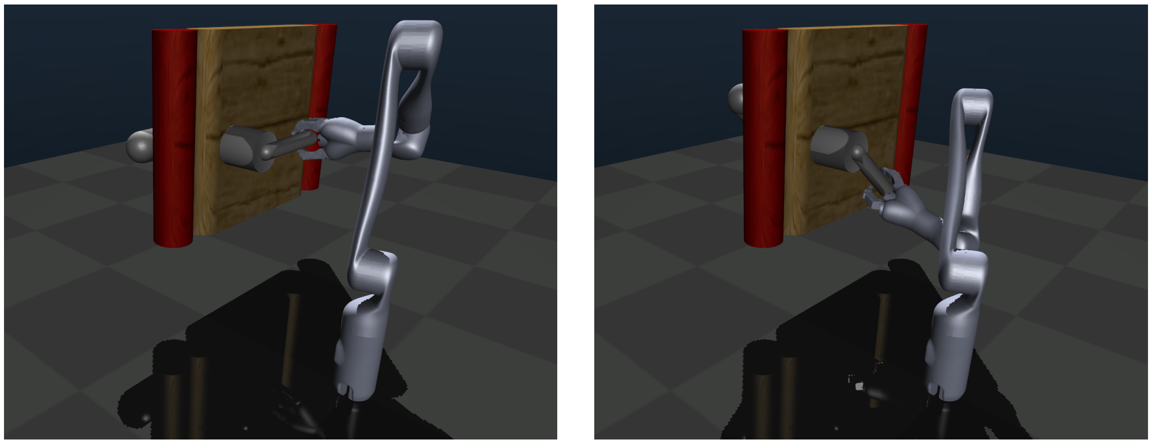
Each domain consists of a MuJoCo simulation environment with a Kinova Jaco arm and an articulated or free-body object [97]. A policy parameterized by the grasp pose is provided; the robot first executes a grasp at the given pose, then executes the policy. These policies are represented by several waypoints in end effector space, which are planned to then attained with a PD controller.

First, a robot arm is tasked with grasping an elongated door handle and rotating the handle past 0.6 radians. An image of the robot opening the door is shown in Figure 4.6. In my provided policy, the system receives as input a segmented point cloud of the handle in a fixed frame, captured from two camera views. When attempting a grasp candidate, the robot motion plans to a location a fixed distance away from the provided grasp centroid. It then moves to the grasp centroid and executes a grasp, closing the fingers until contact is made, and moving an additional fixed number of steps afterwards to ensure grasp success. Next, the gripper attempts to move to a pose rotated about the known axis of rotation in the handle frame. This task is difficult because though an initial grasp may be stable, as the gripper manipulates the handle, the grasp could break contact. Some grasps are better suited to enable the robot to rotate the handle without slipping.

In the next task, the robot must successfully pour liquid from a container. I do not simulate the liquid itself; rather, the robot must lift a pitcher above 0.15 meters, and rotate it after grasping so its opening faces below the horizontal, and any liquid inside would begin to pour out. The liquid container is implemented as a simple cylinder. Similar to the previous task, when executing a grasp at a pose, the end effector first moves to a pre-grasp location before executing the grasp. In this case, contact with the container before grasp execution could knock over the container and cause the task to fail. The robot must grasp the container, lift it upwards, and rotate it past a threshold. This task is difficult because the object is free floating. Grasps near the top or bottom of the container may be stable for lifting, but the container could slip or get water on the gripper if the grasp is not in the center.

The third environment tasks the robot with flipping a large switch. This task differs from the door opening task because the switch must be rotated past 0.75 radians. While a grasp could be stable and allow the robot to grasp the switch and initiate the rotation, the initial configuration of the arm at the time of the grasp has more of an influence into whether the task can be completed.

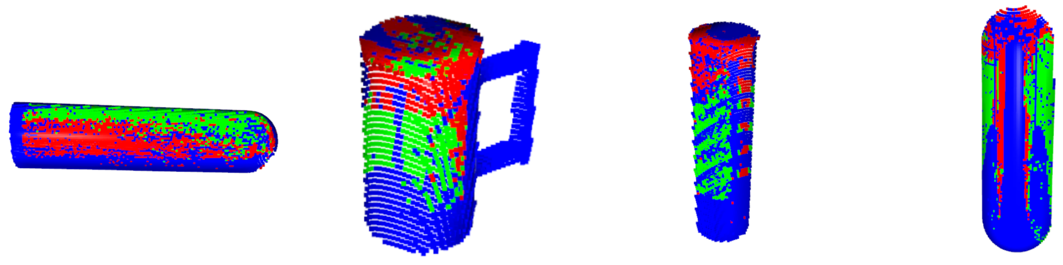
For the final task, the robot must lift a mug to a height of 0.15 meters. The challenge in this task lies in selecting a stable grasp, since the handle could obstruct grasps that would be otherwise stable.



(a) Initial state (b) Final state

Figure 4.6: Simulated door task. Figure (a) shows the robot after executing a grasp on the door handle before executing its manipulation policy. Figure (b) shows the robot after executing its manipulation policy and turning the door handle.

Manipulation tasks may vary in difficulty, and policies are not guaranteed to be optimal, so it cannot be expected that a real-world dataset would be balanced. Of all the 1,000 to 4,000 kinematically reachable candidates generated by **GEN** in each domain, the respective percentages of positive examples are 51.76%, 35.40%, 40.00%, and 63.04%. These distributions are illustrated in Figure 4.7. Note that ATOG never sees all of these examples at once; it receives one at a time, up until it has a training set containing 50 examples in the experiments.



(a) Door (b) Mug (c) Pitcher (d) Switch

Figure 4.7: Partial point clouds of each simulated task object. The points are colored based on the result of a grasp centered at the corresponding point. Blue grasps are kinematically infeasible, green grasps enable the robot to successfully complete the task, and red grasps are feasible but do not enable the robot to complete the task.

4.4.2 Procedure

When presented with a new object to manipulate, the system first generates a partial point cloud of the object, crops the cloud to remove all points besides those corresponding to the target object, then transforms the point cloud to a fixed object frame where the object is centered at the origin and aligned with the axes. **GEN** then generates a set of potential grasp poses. Those that are kinematically infeasible due to collisions or joint limits are pruned. In simulation, 20% of these examples are held out and placed in a test set, while the remainder are placed in the training pool. A random grasp is selected from this pool and the grasp and task policy are executed to generate a binary success label, which is added to the training set. ATOG is trained on this set, then the performance metrics are computed using the success predictions ATOG makes on the test set. A new random grasp is then attempted and added to the training set, and the network is trained again, initialized with the weights from the previous training cycle. This procedure continues until the training set contains 50 examples.

The PointConv sub-network of ATOG is first trained for 50 epochs using a learning rate of 1e-5, batch size of 16, and 1,024 points in each point cloud. The tail end of ATOG is trained with an Adam optimizer with a learning rate of 0.001. For each example added to the training set, the optimizer parameters are reset but the network is initialized with the weights from the previous training iteration. The network is trained for 20 epochs each time, with a batch size of 32. The Pose-Net baseline is trained with the same parameters in the same fashion.

The PointConv baseline is trained for 20 epochs for each example added to the dataset with a learning rate of 0.001, batch size of 16, and 1,024 points in each point cloud. The training data is repeated to fill each batch in cases where the size of the training set is not divisible by the batch size.

4.4.3 Baselines

I compare ATOG against two baselines. Pose-Net is an ablation of ATOG where the PointConv quality prior is not included in the input to the tail of the network; it predicts whether a grasp enables a task from its pose in relation to the object without the predicted quality prior. It is trained using the same procedure as ATOG. The other baseline, PointConv, is the PointConv branch of the network trained to predict task-oriented grasp success directly from local geometry. PointConv's pre-trained weights are loaded before fine-tuning on a specific task.

4.4.4 Evaluation Metrics

This is not strictly a learning problem; the goal is that the grasp with the highest predicted probability of success is suitable for the task, stable, and reachable. Therefore, I introduce a new metric, Probability of Manipulation Success (PoMS). PoMS takes a robot's precision and possibility of unreachable grasp poses into consideration when evaluating the grasp poses the network would select to execute. The system need not successfully classify every example in an evaluation set. The robot is provided with a large set of potential grasp candidates to execute, and it selects the one with the highest predicted probability of task success. However, this selected grasp could be kinematically infeasible in the presence of obstacles. Therefore, the set of grasps that are far from each other with the highest predicted probability of task success must enable the robot to complete the task; the robot could select the next grasp in the set if the previous grasp proves unreachable. Furthermore,

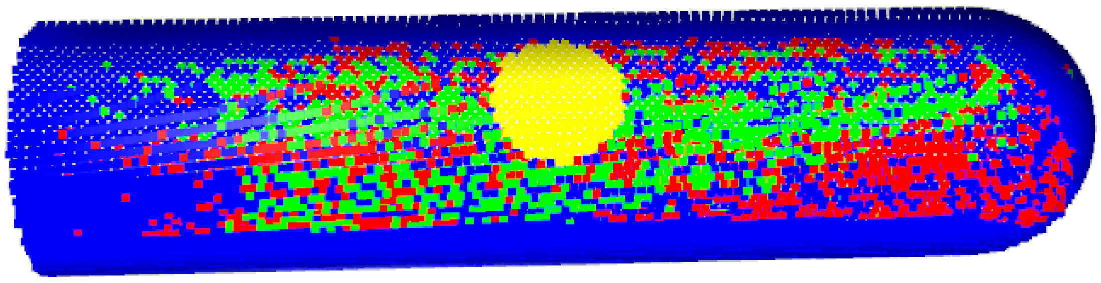


Figure 4.8: The point cloud of the door handle illustrating the PoMS metric. Green points show grasps that enable the robot to complete the task, red are grasps at which the robot fails to complete the task, blue represent unreachable grasps, and all points in yellow fall within $\epsilon = 10cm$ of the $m = 1$ top grasp predicted by the trained network.

the robot is not guaranteed to execute a grasp exactly at the desired point. The robot's precision, noise in the point cloud, and slight changes in object position during grasp execution could cause the executed grasp to differ slightly from the selected grasp. PoMS captures the probability that a grasp within a small radius ϵ of the m top grasps selected by the network would succeed:

$$PoMS = \frac{1}{|\mathbf{G}|} \sum_{1}^{|\mathbf{G}|} SUCCESS(g \in \mathbf{G}), \tag{4.1}$$

where $\mathbf{T}$ is the set of grasps in the held out test set, $\mathbf{G}_{top} = \text{argmax}_g^m \, ATOG(g \in \mathbf{T})$ is the set of m grasps with the highest predicted ATOG scores that are at least 3ϵ from all other poses in $\mathbf{G}_{top}$, $\mathbf{G}$ is the set of grasps in $\mathbf{T}$ within ϵ of any of the m grasps in $\mathbf{G}_{top}$, and $SUCCESS(g)$ is 1 if the given grasp pose enables the robot to complete the task and 0 otherwise. The PoMS metric is illustrated in Figure 4.8. This best encapsulates the probability that a grasp executed by a robot would succeed.

In order to gauge the success of learning, I also report the test-set accuracy as the size of the learning set increases. As the robot executes only one of the top grasps, it is important that ATOG classifies minimal false positives. Therefore, I report precision as well.

4.5 Initial Simulation Results

Results of the simulation experiments are shown in Figure 4.9. My PoMS metric computed on the held-out test set, test-set accuracy, and test-set precision are plotted for each of the four simulated domains. Average values across all training set sizes are reported in the legend. I use $m = 3$ and $\epsilon = 1cm$ when computing PoMS, with all poses in $\mathbf{G}_{top}$ spaced at least 4ϵ for the door, pitcher, and switch, and 3ϵ for the mug when the feasible grasping region is closer together. Works like ten Pas et al. [95] and Liang et al. [47] demonstrate that deep-learning-based systems trained on point clouds and labels generated in simulation well without the need for sim-to-real transfer techniques when introduced to the real world. Though ATOG is evaluated in simulation, it would seamlessly transfer to a real robot because it operates on point clouds and is built off of systems demonstrated to perform well in the real world when trained with simulated data [16, 13].

4.5.1 Discussion

ATOG most clearly outperforms the two baselines in the pitcher environment. The average PoMS is 6.5% higher than the closest baseline, while accuracy is 6.2% higher than the baselines, and precision is 4.5% higher. The exceptional performance on the pitcher task is due to the utility of the predicted general grasp success probabilities. The PointConv branch of ATOG was trained to predict whether grasps on cylinders successfully lift them; this task is similar to the pitcher task, where after lifting the cylindrical pitcher, the robot pours out its contents with an additional rotation. ATOG performs well in the switch domain as well, beating baselines by 1.9%, 0.5%, and 0.8% for the respective metrics. The capsule-shaped switch is lifted and rotated upwards, and the probability scores predicted by ATOG's general grasp branch are informative towards task success, though not as strongly as in the pitcher domain. In the door and mug domains, ATOG performs well, while the PointConv baseline struggles to predict successful grasps; ATOG's average PoMS is 27.4% higher than PointConv's in the door domain and 7% higher on the mug. This could be because the general

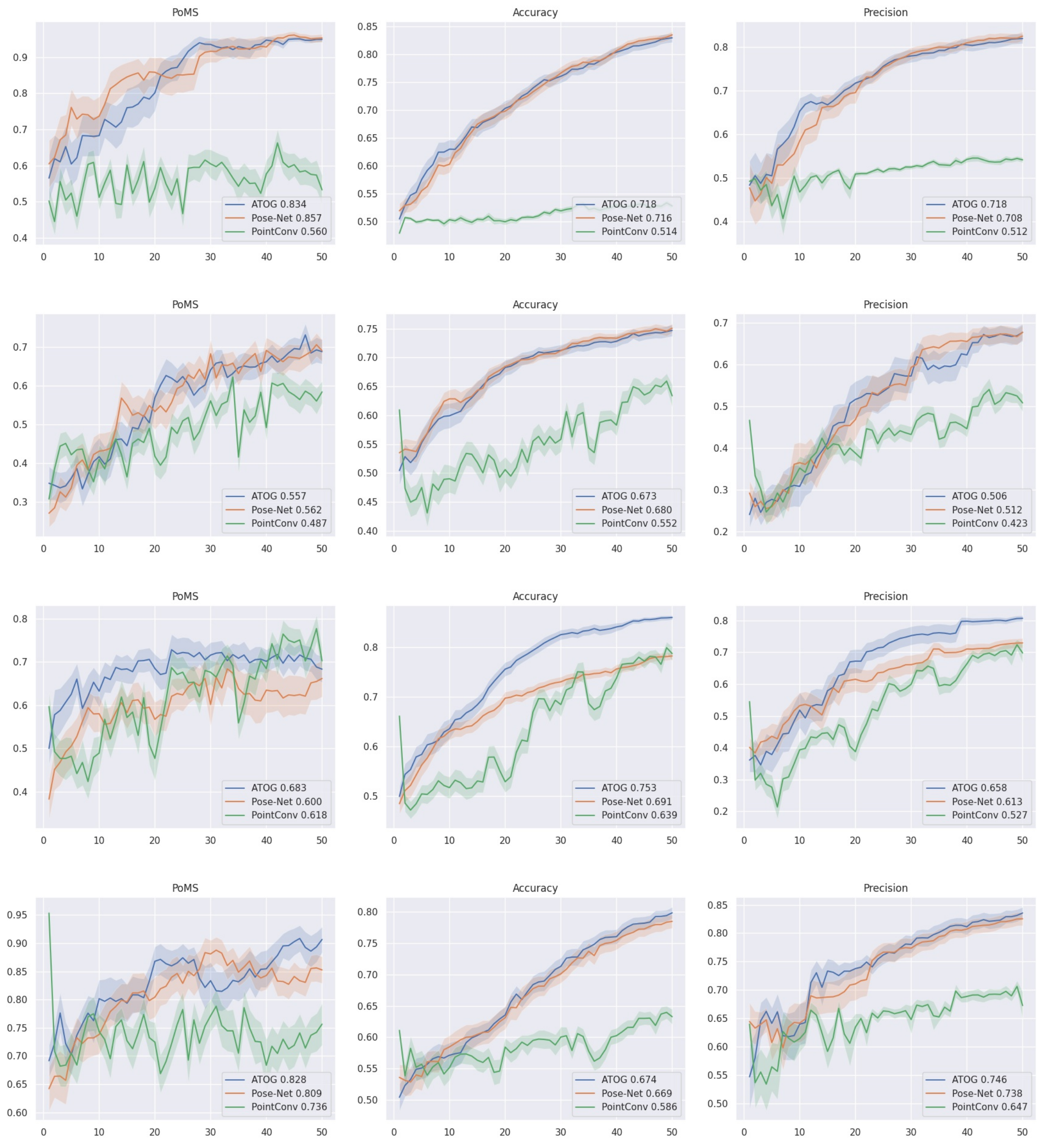


Figure 4.9: Simulation results for the four simulated domains (door, mug, pitcher, switch). PoMS, Accuracy, and Precision on the held-out test set are reported. The x-axes show the number of examples in the training set. Standard error computed over 30 random seeds is shaded. Average values across all training set sizes are reported in each plot's legend. These plots show the performance of my ATOG system, the Pose-Net ablation, and the PointConv ablation.

grasp detection results are not as informative to these tasks. The door handle, for instance, contains many grasps that look stable from local geometry, but fail to afford the robot the ability to complete the task because of kinematic failures during policy execution. Though the uninformative grasp stability information causes PointConv to perform poorly, ATOG learns to ignore it and focus on the pose's relation to the object that Pose-Net learns from. ATOG's average PoMS is 0.5% higher than Pose-Net's on the mug and 2.3% lower than Pose-Net's on the door.

As a whole, ATOG quickly learns to select good grasps. ATOG achieves 80% PoMS within 20 training examples for door, 10 for switch, 70% PoMS for the pitcher within 20 training examples, and 60% PoMS within 20 for the difficult mug, where most of the generated grasp poses are negative examples. ATOG achieves higher average metric scores and learns as fast as, if not faster, than the other approaches.

Though my ATOG classifier and Pose-Net ablation outperform the PointConv baseline, they perform similarly on three of the four tasks. In an attempt to determine why ATOG does not definitively outperform Pose-Net in all domains, I perform a series of additional experiments.

4.6 Further Investigation

There could be several reasons why ATOG does not significantly outperform Pose-Net in all domains. One potential explanation is that ATOG's point cloud branch was trained only with cylindrical objects, to which it overfit, resulting in a sensitive branch of the network. This PointConv branch could provide useful grasping information about cylinders, but performance may degrade when the objects contain other geometry not included in the training set. This could explain why ATOG outperforms Pose-Net when predicting task completion in the cylindrical pitcher domain, but the PointConv branch does not provide useful grasping information for the capsular door and switch handles or more complex mug. Another possibility is that the grasping and lifting task from which the PointConv branch's labels were generated differs too much from the manipulation tasks. When grasping and lifting free-body objects, the robot's grasp must be strong enough to lift the object while overcoming the force of gravity acting upon the object. If the object slips from the grasp, even slightly, the object could fall and the attempt would be considered a failure. Conversely, slight slippage may not hinder the robot while it manipulates an articulated object's handle; in fact, the gripper could slip towards a more stable or suitable grasp pose during task execution while attempting to open the door or flip the switch. ATOG could outperform Pose-Net on the pitcher task because grasp stability is likely a much more important metric to consider when evaluating grasp poses for grasping-based tasks.

In order to evaluate these two hypotheses, I perform an additional set of experiments. To determine if ATOG's PointConv branch overfit to cylindrical objects, I compare ATOG's performance on the door and switch tasks when the PointConv branch is trained on the original cylinder grasping dataset with an ablation of ATOG with its PointConv branch trained to grasp capsules. I further ablate the original training set by training the PointConv branch on both grasping datasets containing both cylinders and capsules. In the second experiment, I evaluate whether the PointConv branch of the ATOG network is more useful on the door task when slippage is considered task failure.

4.6.1 Retraining the PointConv Branch

This experiment compares the performance of four classifiers: my ATOG system, my Pose-Net ablation without a PointConv branch, the PointConv baseline that predicts task success from the grasp-centered point cloud alone, as well as the new ATOG Retrained with its frozen PointConv branch trained on grasping data for either capsules or both capsules and cylinders. Data is generated with the same grasp pose generation and grasp execution policy as ATOG trained on cylinders; the graspable capsules are the same sizes as the original cylinders with a half-sphere on their top ends, and are upright in the simulation environment. The capsule and cylinder dataset contains 50,000 examples. ATOG Retrained is trained using the same hyper-parameters and random seed averaging as the original ATOG described in Section 4.4.2. The PointConv branch that predicts grasp stability is trained from scratch on these additional datasets, but ATOG's tail is trained on the original task-labeled datasets.

The performance of the ATOG retrained with capsule data is shown in Figure 4.10. These results, showing PoMS, accuracy, and precision on the door and switch, are the same as the results from Figure 4.9 with an additional line showing the performance of the retrained ATOG. Similarly, Figure 4.11 shows performance when ATOG is trained on both the cylinder and capsule grasping datasets.

Figure 4.10 and Figure 4.11 show that retraining ATOG's PointConv branch on either of these additional grasping datasets does not significantly change the system's performance. When trained on only capsules and evaluated in the door opening domain, ATOG's average PoMS increases by 3.3%, its average accuracy increases by 0.2%, and its average precision increases by 0.2%. In the switch domain, ATOG's average PoMS increases by 0.9%, its average accuracy remains the same, and its average precision decreases by 0.1%. When trained on both capsules and cylinders, ATOG's average door-environment PoMS increases by 0.3%, but average accuracy and precision decrease by 0.4% and 0.3%. In the switch environment, average PoMS decreases by 1.1%, and average accuracy and precision each decrease by 0.2%.

Though ATOG performed slightly better in the door and switch environments when trained on capsules alone, modifying ATOG's grasp quality branch's training set had little effect on performance. This shows that ATOG's similar performance to Pose-Net in the door and switch domains was likely not caused by the PointConv branch over-fitting to its training data.

4.6.2 Grasp Slippage as Failure

Next, I evaluate whether ATOG's lack of performance boost over Pose-Net on the door task is the result of grasp slippage tolerances allowing task completion with less stable grasps. As ATOG's PointConv branch is trained to predict general grasp stability, perhaps this information is less useful for the door task where the task may still succeed even when the robot's gripper slips away from the

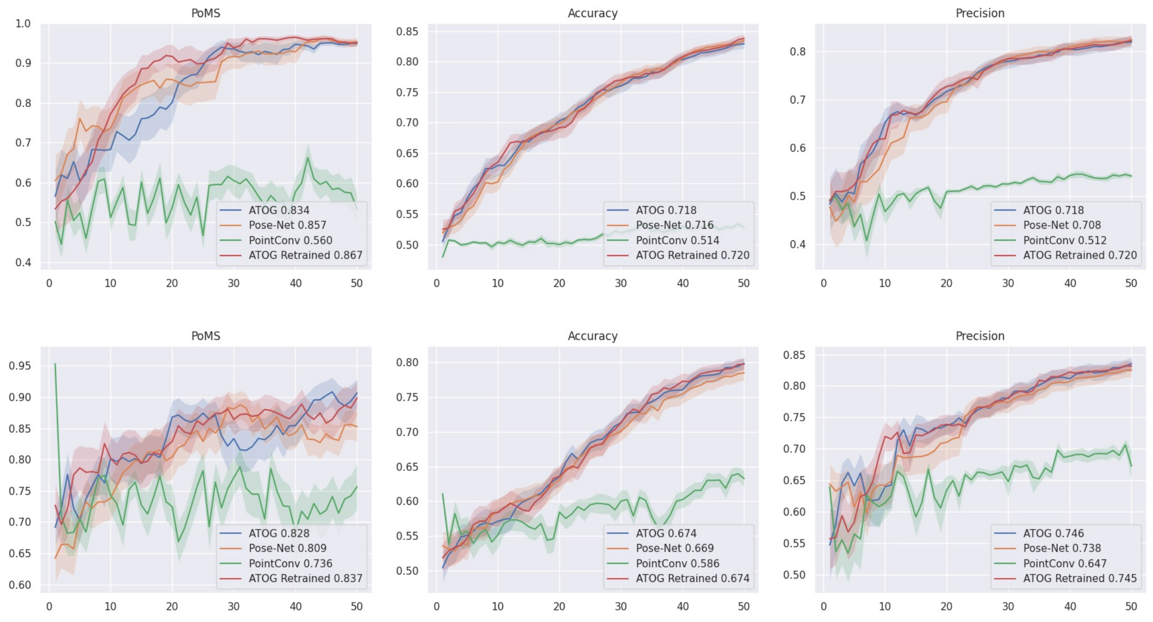


Figure 4.10: PoMS, Accuracy, and Precision results in the door and switch domains. The results from Figure 4.9 are repeated and include a new line showing the performance of ATOG when the PointConv branch is trained to detect grasps on capsule-shaped objects.

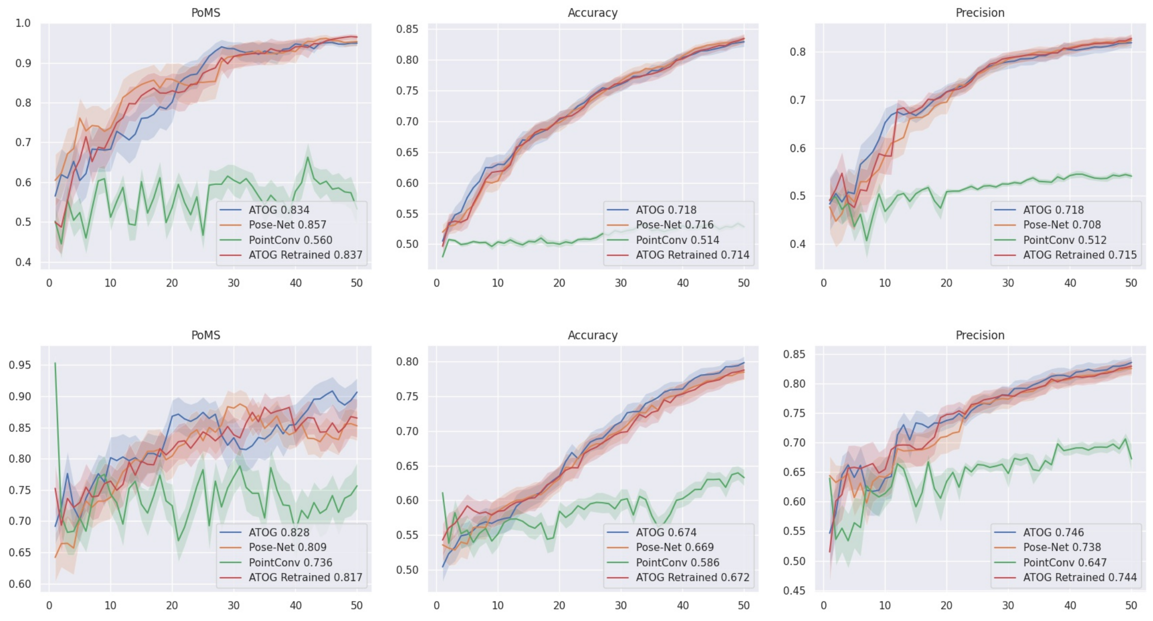
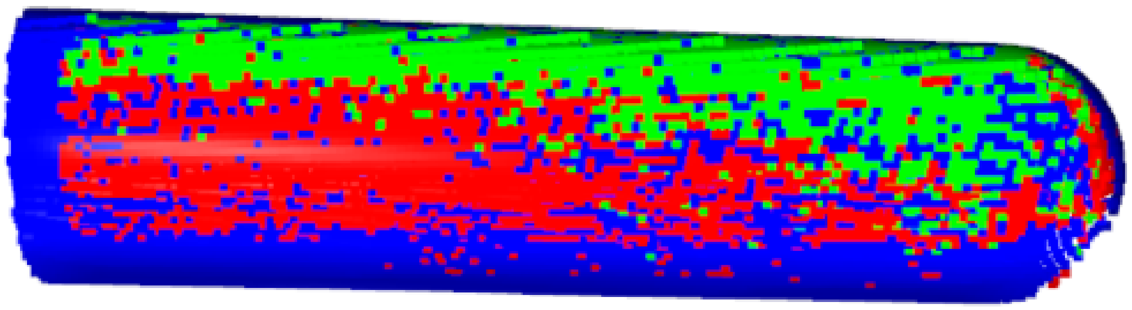
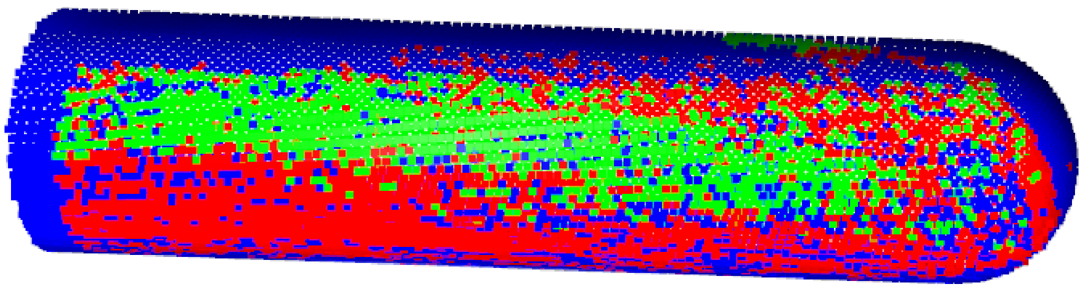


Figure 4.11: PoMS, Accuracy, and Precision results in the door and switch domains. The results from Figure 4.9 are repeated and include a new line showing the performance of ATOG when the PointConv branch is trained to detect grasps on both cylindrical and capsule-shaped objects.



(a) Original door handle labels, where grasp slippage does not result in failure.



(b) Updated door handle labels, where grasp slippage does result in task failure.

Figure 4.12: Door handle labels. Grasps centered at green points enable task completion; grasps centered at red points do not enable task completion; grasps centered at blue points are unreachable. Grasp orientation is not visualized here.

original grasp. Here, I generate a new task-labeled dataset in which a trial is considered a failure if the gripper moves more than 10 centimeters from its original position on the handle. As seen in Figure 4.12, this eliminates a large set of positive examples from the right end of the handle; in the original dataset, the robot was able to complete the task after executing a grasp in this area despite the grasp slipping during the execution of the manipulation policy.

In order to determine if ATOG can outperform Pose-Net when slippage is taken into consideration, I retrain ATOG and Pose-Net with this new dataset following the procedure detailed in Section 4.4.2. The results of this experiment are shown in Figure 4.13. Here, ATOG achieves an average PoMS 1.3% higher than Pose-Net, but average accuracy and precision 0.2% and 1.2% lower.

Comparatively, when trained on the original dataset, as shown in Figure 4.9, ATOG achieves an average PoMS 2.3% lower than Pose-Net but average accuracy and precision 0.2% and 1.0% higher on the door task. This modified dataset is more difficult to classify. When trained on the original data where slippage is not considered a failure, both systems can label the held-out test set with nearly 85% accuracy and over 80% precision after being trained on 50 examples. When trained and tested on this modified dataset, both systems only achieve 75% accuracy and nearly 65% precision. Furthermore, PoMS drops from 95% on the original dataset to 60% when slippage is considered a failure. One cannot make a direct comparison between performance on these two datasets since the labels of both training and testing examples changed when slippage is considered failure. However, it is noteworthy that the poses of grasps that enable the system to complete the door opening task are easier for a classifier to distinguish than the grasp poses that allow for task completion without slipping.

As ATOG's performance did not increase over Pose-Net's on this new dataset, adding the non-slip requirement to the task success metric did not enable ATOG to better leverage information from its general grasp quality PointConv branch. This is likely because a stable grasp for lifting an object and fighting gravity is too different from a stable grasp that enables the robot to open the door without slipping. As seen in Figure 4.12b, grasps on the door could slip during manipulation because the palm was placed on the spherical end of the handle; when force was applied to rotate the handle or pull the door open, the gripper likely slipped up the handle. By contrast, when grasping an object, it is more important for the fingers to wrap tightly around the object because the force of gravity moves the object away from the gripper and palm.

Overall, these experiments have shown that both our Pose-Net and ATOG networks efficiently learn which grasps enable a policy to complete a task when trained on very small datasets, ranging from 1 to 50 examples. The comparisons between ATOG and Pose-Net show that predicted grasp quality helps the network predict whether the pouring task will succeed since pouring is very similar to the pick and place task used to generate labels to train the quality branch of the network. This

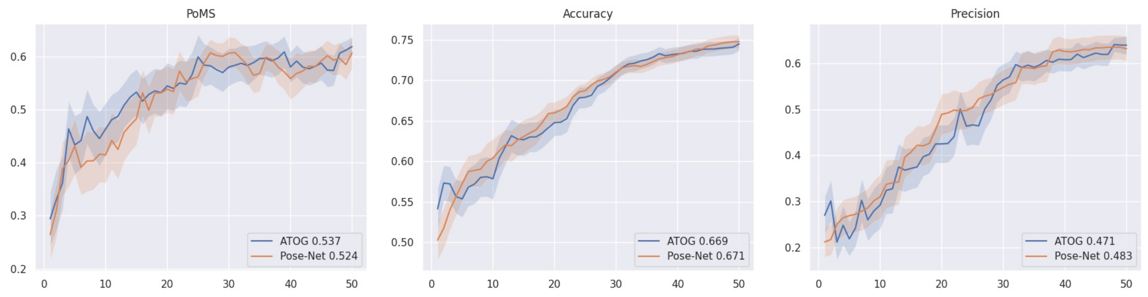


Figure 4.13: PoMS, Accuracy, and Precision results in the door domain. Here, the task success metric is modified so a trial is considered a failure if the gripper moves away from the grasp pose during the execution of the manipulation policy. Results from ATOG and the Pose-Net ablation are shown.

shows that integrating predicted grasp quality labels into the input to a task-oriented grasp classifier is useful when the task requires lifting free-body objects, but less useful when the task is a more complex manipulation task that requires interacting with a complex articulated object. However, both my ATOG and Pose-Net ablation outperform a PointConv baseline that considers the object geometry local to the grasp pose.

4.7 Real Robot Demonstration

I demonstrate ATOG's ability to learn task-oriented grasps in a real-robot domain: opening a cabinet. A Kuka LBR iiwa 7 robot arm with a Robotiq 3-Finger Adaptive Gripper is presented with a standard kitchen cabinet with a towel rack handle shielded by a pool noodle affixed to the front. The robot must open the cabinet past a fixed threshold for the task to be considered a success. The grasp-dependent policy provides a trajectory comprised of five waypoints that rotate the gripper about the door's known axis of rotation by 90°. The arm attempts to execute this trajectory using MoveIt's Cartesian planner, moving through each available waypoint and stopping when it reaches the final goal or cannot move beyond 40% of the way to the next waypoint in the chain because of collisions or joint limits. In this demonstration, the robot learns from several small datasets containing five to 15 training examples and is evaluated based on its performance when its top three selected grasps are executed.

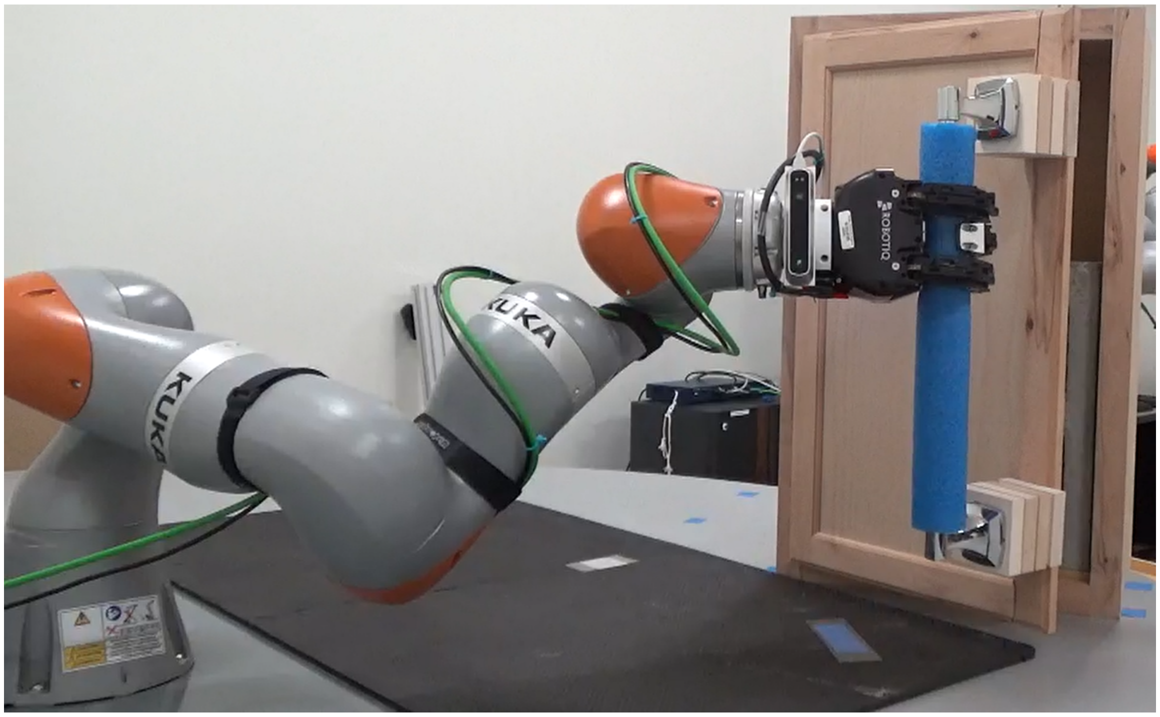


Figure 4.14: A robot learning which grasps enable it to successfully open the cabinet.

4.7.1 Procedure

The arm first moves to two fixed positions to capture a point cloud of the scene from two views on either sides of the cabinet. The combined partial point cloud is cropped, once using the known location of the handle to generate a cloud of graspable points and again to generate a full cloud of the scene from which a mesh is generated for collision detection. As the cabinet is placed in the same pose during each trial, the same cloud is used for the entire demonstration. This reduces processing time and ensures grasps are not repeated in the datasets. Next, **GEN** generates a set of potential grasp poses G using the point cloud of the graspable handle. These are power grasps, in which the palm moves close to the the grasp point so the fingers can wrap around the object and stably hold it in place. Each pose is checked to confirm that it is collision-free using models of the gripper and table and a mesh of the cabinet generated from the captured point cloud. In the demonstration, collision checking reduced the number of feasible grasp candidates from 17,324 to 6,379. These poses can be seen in Figure 4.15.

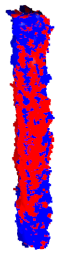


Figure 4.15: A point cloud showing all points on the graspable area of the door handle. Grasps centered at the blue points are in collision while grasps centered at red points are feasible.

Each $g \in G$ is assigned a pre-grasp pose $10cm$ behind the grasp pose along the grasp axis. The set of feasible grasp candidates is narrowed down further using MoveIt and a MoveIt environment including a model of the table, arm, and cabinet. Candidates for which MoveIt cannot plan a path to the corresponding pre-grasp pose, or cannot plan a Cartesian path that makes it 70% of the way from the pre-grasp to grasp pose are pruned. Five grasp poses are selected at random from the remaining set of feasible grasps with which to initialize the training set. For each grasp, the arm executes the plans to move to the pre-grasp pose, then the grasp pose, then closes the gripper. It then attempts to execute the grasp-dependent plan to open the cabinet door. After moving as far along this path as possible, the gripper opens. If the door is beyond a pre-defined threshold, the attempt is considered a success. Otherwise, it counts as a failure.

After executing all five grasps to generate a labeled dataset, ATOG is trained. The PointConv branch is pre-trained with the same dataset of simulated cylinder grasping data used to train the networks used in simulation, and as with the simulation experiments, the weights are frozen after training. The network is trained for 100 epochs with a batch size of five and a learning rate of 0.001. This trained network is evaluated by attempting the top three grasps predicted by the network and measuring the percentage of success. This is analogous to the PoMS metric used in the simulation experiments.

After evaluating the network trained with five labeled examples, five additional examples are selected at random and the grasps and policies are executed with the robot to generate labels. ATOG is trained from scratch with the new dataset of 10 examples, and the three grasps with the highest predicted probability of success are again executed to evaluate the system. This is repeated once more to generate a training set with 15 labeled examples and evaluate a network trained on this set. In all of these experiments, the training examples are kept from the test pool, and all executed evaluation grasps must be at least $4cm$ from the nearest evaluation grasp that has already been executed.

4.7.2 Results

Three of the first five training grasps are successful, seven of the first 10 are successful, and nine of all 15 are successful. For the network trained on five examples, the second selected grasps enables the robot to open the cabinet, though the first and third choices do not, the first narrowly missing the goal. All three grasps selected by the network trained with 10 examples succeed. The first and third grasps selected by the network trained with 15 examples succeed.

4.8 Conclusion

In order to learn or execute complex manipulation policies, robots must first perform task-oriented grasps on objects. A robot must learn which task-oriented grasps enable manipulation success as quickly as possible to limit expensive manipulation time with objects in the real world. My ATOG deep network leverages a pre-trained general grasp quality classification branch to learn which grasps enable task success with few training examples. ATOG outperforms two baselines in Probability of Manipulation Success, and classification accuracy and precision. Though adding infrastructure to generalize between object instances or integrating active learning with ATOG could be interesting extensions, the most useful extension would be to combine ATOG with a

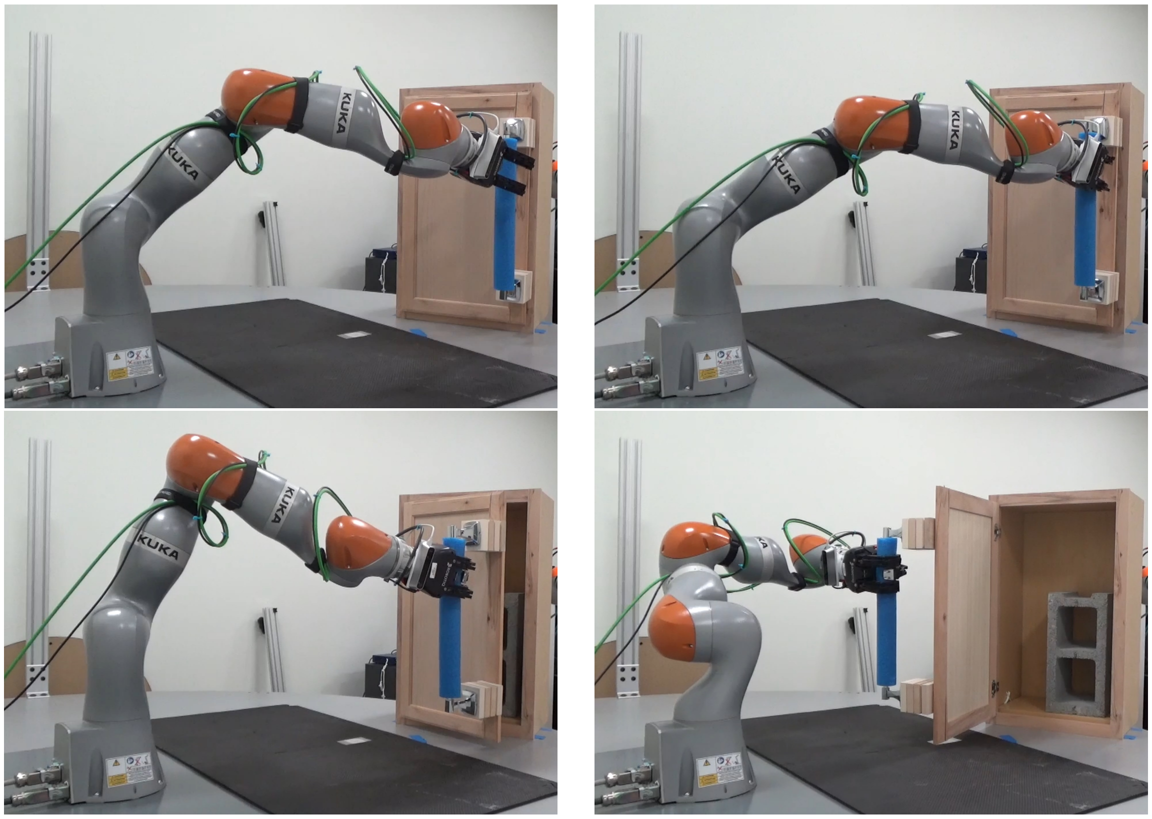


Figure 4.16: A real robot executing a hand-coded door opening policy after using ATOG to learn which grasps enable task success.

reinforcement-learning agent to more efficiently learn complex manipulation abilities without the need for hard-coded policies.

Chapter 5

Jointly Learning a Task-Oriented Grasp Classifier and a Task Policy

One major assumption that the ATOG system defined in Chapter 4 made is that the manipulation controller π is provided for all tasks; the system must only predict which grasps would enable the controller to complete the task. However, such a controller would not be known to the robot when it interacts with a novel object for the first time. Instead, the robot should be provided with the ability to learn new manipulation policies that take grasps into consideration. In this chapter, I propose an algorithm for jointly learning a manipulation policy and a task-oriented grasp detector.

5.1 Introduction

I have shown in Chapter 4 that a manipulation controller, which defines a control policy for a robot arm to complete a task, alone is not sufficient for task completion. Policy success is also dependent on the initial state of the system. The manipulation controllers provided to the system in Chapter 4 are instantiated after a grasp has been attempted; the initial state of such a controller is a grasp pose $g \in \mathbb{SE}(3)$. The problem of task-oriented grasping was simplified in Chapter 4. Assuming a manipulation controller was provided, the task-oriented grasp classifier defined in Section 4.3 predicted whether a given grasp would enable the controller to complete the task. However, it is unreasonable to assume a robot is provided with a manipulation controller for any task it might be faced with. Hand-engineered controllers require time from an expert programmer and lack the robustness of a learned control policy. For example, opening a door is a complex manipulation task that requires a robot to grasp the handle, rotate the handle, then pull or push the door open. An engineer could hard-code a policy using a pre-defined grasping skill, end-effector position control, and information about the door's axes of rotation to control the robot to grasp the handle, rotate the handle by moving the end-effector to a fixed opening pose, then moving to a fixed pose to open the door. This hard-coded policy would prove to be too brittle when either goal pose was unreachable from the initial grasp, the door re-latched while trying to open it, the gripper slipped off of the handle while manipulating the door, or the robot was unable to follow the interpolated path between goal poses. Instead, a learned policy would encounter these difficult situations during training and learn how to rebound from them. To enable a robot to be truly autonomous, its manipulation controllers should be learned for each new task.

Learning manipulation policies is an active area of study. This task is challenging because of the sparse reward signal robots receive as they learn; without a hand-tuned reward that coaxes the robot toward the goal, robots often struggle with the large state and action spaces manipulation frameworks induce. If a robot is given full control over each of its joints, even reaching a specified end-effector pose is difficult. I posit that empowering a robot with a grasp controller to bootstrap the learning process will make learning a manipulation policy possible for difficult durative-manipulation problems. Furthermore, by selecting grasps that enable the robot to complete the task, learning a manipulation policy that begins once these grasps have been executed is more efficient. However, learning a control policy while simultaneously learning which grasps enable task completion is a difficult entangled learning problem because the initial state during policy learning is determined by the grasp detector, while data used to train this grasp detector is labeled with an evolving policy.

Robots deployed into novel or changing environments must be able to adapt and learn to interact with new objects quickly. Traditional reinforcement-learning techniques require large amounts of data and hundreds of robot hours to learn from real-world data. Systems such as those defined by Ibarz et al. [32] begin with no notion of grasping or other abstract actions; they learn policies to move the robot that act on visual information alone. However, problems such as motion planning in robotics have existing solutions that allow a robot to reliably move its end effector to a desired location [35]. Such controllers allow robots to execute grasps, and are often utilized to execute grasps in generic grasp-detection systems [95, 55]. These grasp detectors provide robots with abstract grasp actions, where a grasp can be executed at a given pose without learning a controller to reach that pose. Abstract actions that take advantage of existing solutions must be utilized to learn from as few labeled examples as possible.

I propose an algorithm for jointly learning a task-oriented grasp detector and a manipulation policy to complete a task. This algorithm combines the lightweight task-oriented grasp classifier described in Chapter 4 with a more complex manipulation policy. To learn the manipulation policy, I employ a state-of-the-art reinforcement-learning algorithm, TD3 with Hindsight Experience Replay [24, 2]. After selecting an arbitrary grasp with the task-oriented grasp classifier and executing this grasp using a given grasp controller, the manipulation policy takes over and attempts to complete the durative-contact manipulation task, such as flipping a switch or opening a door. In order to select a grasp to be executed, I simultaneously train a classifier to predict whether a given grasp would enable the manipulation policy to complete the task. As this classifier requires fewer than 20 examples to accurately classify grasps, as demonstrated in Chapter 4, it quickly informs the manipulation policy on which grasps may afford it the ability to complete the task. Furthermore, my algorithm handles the entangled joint-learning problem using a weighting scheme that down-weights examples based on how much the policy has changed since they were observed. I call this combination of a task-oriented grasp detector and a reinforcement learning agent a Grasp-Aware Reinforcement-Learning Agent (GARLA).

I evaluate GARLA in two simulated domains, opening a door and flipping a switch, by comparing the episodic task success as the system trains. I compare my algorithm and an ablation where task-oriented grasp classifier examples are not weighted against two baselines that also combine a grasp selector and executor with a manipulation policy agent. In the lower-bound baseline, the agent learns to complete the manipulation task while selecting any proposed grasp at random. To estimate an upper-bound on my algorithm's performance, the oracle baseline selects from a small set of grasps that afford a fully trained policy the ability to complete the task while learning its manipulation policy. I find that when an agent jointly learns a manipulation policy and which grasps afford it the ability to successfully execute this policy to complete a task, the agent outperforms a system that learns to manipulate an object after executing a random grasp pose. This agent can even outperform a system that executes grasps only at a small set of "good" poses that enabled a previously trained system to complete the task successfully.

I describe two major contributions in this chapter. The first is a method with which a task-oriented grasp classifier can be combined with a manipulation policy learner to jointly learn to select grasps and manipulate an object. The second is a weighting system that helps the task-oriented grasp classifier ignore labels assigned by an outdated version of the manipulation policy. I verify that combining these two contributions enables a robot to learn to complete complex manipulation tasks more quickly and successfully than baseline ablations.

5.2 Background and Related Work

In order to robustly manipulate the world around them and complete new tasks in novel environments, robots must be able to learn manipulation policies. Formally, a policy $\pi : s \rightarrow a$ returns the action or a distribution of actions a a robot should take given the state or some observation of the state of its environment s. The Multi-Modal Grasp Pose Detector introduced in Chapter 3 could be considered a policy, as it can greedily return the grasp pose and grasp type

$a_g = (g_i \in \mathbb{SE}(3), t_i \in \{1, ..., N\})$ that correspond to the highest predicted success probability for all reachable proposed grasps given a point cloud of the scene $s_P \subseteq \mathbb{R}^3$. After the robot executes this action with its provided grasp controller, then lifts the object and drops it at the goal location using a hard-coded controller, it requests another reachable grasp from MMGPD to continue to clear the cluttered scene. The hard-coded manipulation policies that were engineered to complete tasks in Chapter 4 return actions in the form of trajectories of goal end-effector poses $a_m = (p_i \in \mathbb{SE}(3)\ \forall\ i\ \in\ T)$ that the system computes given initial end-effector poses $s_p \in \mathbb{SE}(3)$. In both of these cases, part or all of the manipulation policy is hard-coded and hand-tuned by an expert programmer to perform one very specific task. These policies provide a robot with a long sequence of steps to perform, which are not robust to disturbances or errors that may occur during execution. Robust manipulation policies should provide a robot with constantly updating actions that guide the robot to its goal and change as the states of the robot and manipuland change. Therefore, the policy π should return an action a that moves the robot towards its goal, and the robot should call upon π throughout the course of its manipulation attempt to receive updated actions. Robot manipulation tasks should therefore be defined as Markov Decision Processes (MDPs) with discrete time steps, as first discussed in Section 2.3.1.

An MDP is a framework for defining tasks as processes broken into discrete time steps where an agent takes actions $a \in A$ in an environment that consists of states $s \in S$ while receiving rewards $r \in R$ [90]. Importantly, these processes obey the Markov property, which states that the state reached and reward obtained after taking action a in state s only depend on the previous state s and action a, though this transition could also include stochasticity. This implies that there is no memory or history of actions that determine the next state and reward. To model robot manipulation as an MDP, the state and action spaces and reward function must be defined. Additionally, an algorithm to learn a policy π is necessary. These algorithms are reinforcement learning algorithms, where the agent explores its environment and learns π from its experience over time.

5.2.1 Reinforcement Learning in Continuous Action Spaces

Deep learning has enabled major breakthroughs in important problems such as grasp detection [28] and image classification [41]. It has also powered recent advances in reinforcement learning. Traditional reinforcement learning approaches include Q-learning and SARSA [100, 79]. These approaches focus on learning a Q-function $Q : s, a \rightarrow q$ that maps a state-action pair to a Q-value q, a score indicating how much discounted reward the agent would expect to obtain by taking action a from state s. Given a Q-function, a greedy policy π selects the action a^* that optimizes the expected return at the current state s. A tabular Q-function is represented as a table that explicitly tracks the Q-values for all possible combinations of states and actions. These traditional Q-function approaches work well on simpler problems with small state and action spaces, but struggle in large state spaces, such as when the state is an image. Function approximation is necessary in these cases, and deep neural networks have proven to be powerful function approximators.

Deep learning first made a major breakthrough with Deep Q-Networks (DQN), as introduced in Section 2.3.2. Deep Q-Networks represent Q-functions as DNNs, and achieved state-of-the-art performance in learning to play Atari video games [61]. A Deep Q-Network takes in a state, represented as a small series of sequential frames from the Atari game that the agent is learning to play. It outputs a Q-value for each of the available actions, which consist of a varying number of button combinations depending on the game. With a single pass through the network, DQN predicts the Q-values for all actions at the given state. This combination of deep learning and reinforcement learning enabled the agents to learn efficiently from a very large state space, but was only compatible with a small set of discrete actions.

The Deep Deterministic Policy Gradient (DDPG) method revolutionized reinforcement learning in large, continuous action spaces [48]. This advancement is vital for manipulation policy learning, as robots' actions can consist of continuous desired positions, velocities, and torques. DDPG consists of a critic network that estimates the Q-value for a given state-action pair as well as a policy network that suggests an action to take in a given state. This DNN pairing enables the agent to learn to sample a continuous action that optimizes the Q-function. Other works such as TD3 have iterated on the DDPG design to achieve even better performance when learning in continuous domains [24]. TD3 introduces several enhancements to DDPG; TD3 represents its Q-function with a pair of Deep Q-Networks, reduces the policy update frequency, and adds noise to the target to avoid overfitting. As I detail in Section 5.3.2, GARLA uses a modified version of TD3 to efficiently learn robust manipulation policies.

5.2.2 Robot Manipulation Policy Learning

The works described in Section 4.2 focus on selecting grasps that enable robots to complete a given task, but do not simultaneously learn a manipulation policy like GARLA does. Those that rely on human-labeled data or human guidance to train a system to predict whether grasps enable task completion assume that a manipulation policy is given and focus solely on learning to detect task-oriented grasps [39, 17, 107, 51]. Optimization-based approaches [72, 30] and affordance-based approaches [12, 26] similarly detect task-specific grasps and assume a policy is provided. Several do use deep learning, but do not simultaneously learn a grasp classifier and a manipulation policy. Fang et al. [18] present TOG-Net, a deep neural network that jointly predicts grasp stability, task-oriented grasp quality, and the manipulation parameters necessary to complete a task. However, in the two presented tool-based tasks, the manipulation parameters that this policy takes as input are simply the initial position and orientation of the gripper at the start of a given manipulation trajectory; the gripper moves a fixed distance along a fixed axis when performing a sweeping task, and the wrist joint rotates by 90 degrees when hammering. Xu et al. [106] plan using a relaxed definition of affordances that also considers whether the affordance enables the completion of a task during some future step. Their policies are comprised of simple motion planning-based skills that are parameterized by grasp and place positions or trajectory start and end points. Their system consists of a latent dynamics neural network that predicts future states and an affordance model that estimates the cost of a given plan. These simple manipulation chains can perform simple tasks, but GARLA is capable of learning complex manipulation policies from scratch with 6-DoF grasps. Zhao et al. [109] focus on insertion tasks. They train a cascaded sequence of neural networks to predict, for a given 4-DoF grasp, general grasp stability, post-grasp displacement, and the probability of a successful task completion. Their Grasp Quality Network and Grasp Displacement Network are trained simultaneously, and the Insertion Quality Network is trained after that. They do not learn an insertion policy, but estimate the position of the object in the gripper with the post-grasp displacement, then generate a motion plan to complete the task. Wen et al. [102] learn an object-class representation in simulation that is well suited to predict the probability that a grasp leads to task success across object instances, but do not learn a task policy. GARLA is the first I am aware of that learns both a manipulation policy and a task-oriented grasp detector.

State-of-the-art robotic reinforcement learning works learn policies to complete a variety of manipulation tasks, but I am not aware of any that explicitly learn a grasp classifier while learning a manipulation policy. Levine et al. [44] present a neural network that predicts actions in the form of a robot end-effector configuration and joint torques directly from images using guided policy search. This system does not predict grasp poses that enable task completion, but learns from two to three given initial grasp states. Khazatsky et al. [37] use foresight to predict goal configurations for a variety of tasks, then learn to complete new tasks. These tasks include grasping; as grasping is not provided to the system as an explicit skill, they learn grasping as part of the policy learning process. Gu et al. [27] similarly learn grasping as part of the manipulation policy and assume that the gripper is near the manipuland at the start of each episode, but do not learn a grasp classifier. Frameworks that rely on efficient action abstraction to learn manipulation policies such as VICES do not explicitly consider grasping [59]. Other works that utilize frameworks like DMPs [81] or RMPs [76] to compose skills, such as the system proposed by Shaw et al. [84], do not explicitly learn a grasp detector and manipulation policy simultaneously. GARLA is the first system that leverages grasp classification and policy learning to efficiently learn to complete durative-contact robot tasks.

5.3 Learning a Durative-Contact Manipulation Controller

My solution to the durative-contact manipulation learning problem consists of several disparate sub-systems. I call this system a Grasp-Aware Reinforcement-Learning Agent (GARLA). Framing the reinforcement-learning problem properly with the right level of abstraction is essential for the agent to tractably learn to solve the task. The selection of a reinforcement-learning algorithm also dictates the agent's ability to quickly learn to complete the task correctly. Integrating a grasping skill bootstraps the learning process, and a task-oriented grasp classifier enables the agent to select initial states that enable it to solve the problem. This combination of a grasping skill with a learned policy is inspired by the options framework, the hierarchical reinforcement-learning framework in which a high-level policy can choose to pass control to one of several low-level policies depending on the state s of the environment. In order to train the reinforcement-learning agent while jointly using its experience to train a grasp classifier, I present a weighting scheme that enables the grasp classifier to ignore examples with outdated labels.

5.3.1 Durative-Contact Manipulation Reinforcement-Learning Agent

In order to frame durative-contact manipulation as a reinforcement-learning problem, I first concretely define my agent's state space S and action space A, as well as a reward function R. The system described in this work solves durative-contact manipulation tasks, where a robot must first successfully grasp an object, then perform a downstream task while maintaining the grasp. In these durative-contact manipulation tasks, task completion is represented as a binary success metric $r \in \{0, 1\}$. The agent is provided this success metric as a sparse reward: one if the task is completed successfully, and zero otherwise. A dense reward could encode additional information, such as the distance from the object's current state to its goal state or the distance from the robot to the object. This dense reward would guide the robot through the process, providing partial reward for making progress towards the goal. However, such a reward signal would have to be hand-engineered for each new task the robot must solve.

The agent's state space is comprised of the knowledge about its environment. At the lowest level, a robot arm's state is described by its joint positions q and joint velocities $\dot{q}$, with its joint accelerations $\ddot{q}$ derivable as the derivatives of the joint velocities [88]. The gripper's state similarly contains proprioceptive finger joint positions and velocities. Additionally, tactile sensors placed on the links of each of a robot's fingers explicitly inform the agent of how much contact each segment of the gripper is making with the environment. In order to learn from observations made about the environment, the agent's state must also contain information about the object to be manipulated. This observation could consist of images of the scene captured by the robot, and the agent could learn to encode these images using convolutional neural network techniques, such as those that proved successful for learning to play Atari games [61]. To simplify the learning problem, a pre-trained neural network that estimates the states of objects could be employed to predict the state of an object. For instance, Abbatematteo et al. [1] present a system that predicts the position and orientation of an object's axis of articulation. Pose detection systems such as Xiang et al. [105]'s PoseCNN similarly estimate the pose for free-body objects. This lower-dimensional vector contains all the information that the agent would have to learn to extract from an image while simultaneously learning the task; providing object state information directly simplifies learning by greatly reducing the size of the state space. Concretely, my durative-contact manipulation agent receives observations from the state space

$$S = \{q_r,\ \dot{q_r},\ q_g,\ \dot{q_g},\ q_o,\ \dot{q_o},\ t\}, \tag{5.1}$$

where $q_r \in \mathbb{R}^n$ is the joint positions for an n-jointed robot arm, $q_g \in \mathbb{R}^m$ is the joint positions for an m-jointed robot gripper, $q_o \in \mathbb{R}^k$ is the joint positions for an object with k points of articulation, $\dot{q}$ is a set of joint velocities, and $t \in \mathbb{R}^j$ is the readings from a set of j tactile sensors. An example state is illustrated in Figure 5.1. Alternately, if the manipuland were a free-body object and not an articulated object, $q_o \in \mathbb{SE}(3)$ would be its pose in some fixed frame of reference.

At the lowest level, a robot arm's joints are each actuated by a motor, the input to which is

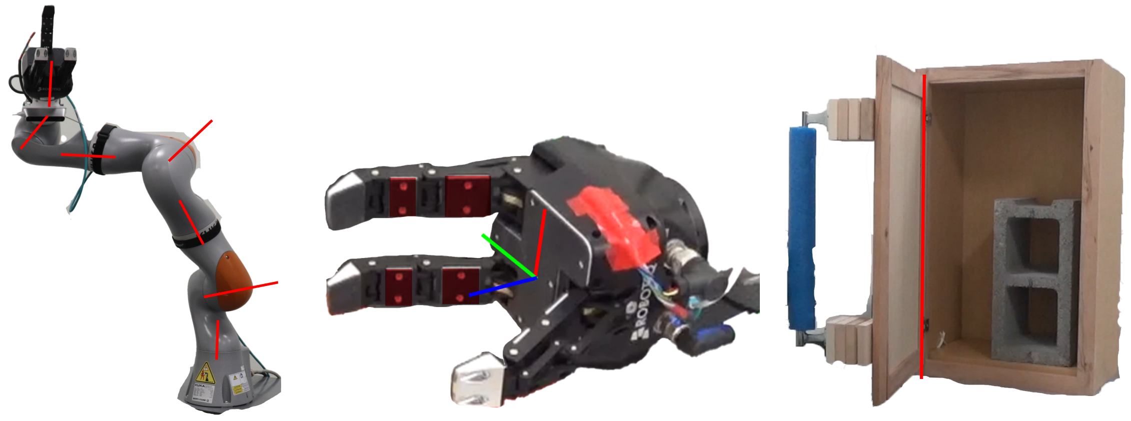


Figure 5.1: An example state that GARLA could encounter. The robot arm's joints' axes of rotation are displayed in the first image. The gripper's frame is represented as a set of three colored axes in the second, along with tactile sensors highlighted in red. The articulated cabinet door's axis of rotation is represented by a red axis in the third image.

electrical current. An agent could be trained to select the amount of electrical current to pass to each motor at each timestep to complete a task. However, such an action space places an unnecessary burden on the agent to not only learn precisely how electrical current relates to the robots's motions, but also how to chain motions together to complete the task. Instead, action abstraction must be employed at several levels. The action space can be simplified, making learning easier by providing the agent with ways to intuitively interact with its environment, by leverage existing control systems and motion planning techniques. Existing robot arm control systems consider many complex variables, such as friction, backlash, compliance, joint coupling, flexibility, actuator and drive-train dynamics, and impedance [88]. An example of one layer of abstraction is a control system that takes each joint's desired position, velocity, and torque as input and outputs changes in electrical current to each individual motor to drive the system to the goal while considering disturbances and external sources of error. Such a system could be abstracted further; rather than train an agent to predict how to optimally change the robot's joint positions, velocities, and torques, the agent could work directly in the end effector frame to predict optimal changes in end effector positions. This control signal would be converted to joint commands using the kinematics of the arm, then to low-level electrical impulses to control the arm directly.

GARLA makes decisions in the robot's end-effector space using operational space control [36]. As the state-of-the-art Variable Impedance End-Effector Space (VICES) work demonstrated, selecting the correct action space for the task makes learning more efficient and policies more robust [59]. Operational space control, also known as end-effector space control, is a robot action space augmentation that enables the reinforcement-learning agent to output changes in end effector position and orientation while a fixed controller converts goal poses to low-level control commands. The VICES action abstraction is similar, but augments the action space with the gain parameters of the underlying low-level robot controller to learn both how to change the end effector's pose and how much force to apply when moving to this pose. My agent could use VICES to learn both the high-level control commands and low-level controller gains to modify the controller during task execution. However, as GARLA is agnostic to the exact action-space abstraction and the tasks I evaluate it on do not require precise impedance control, I choose the less complex operational space control to make policy learning easier. Formally, the actions available to the agent are

$$A = \{\Delta p_x,\ \Delta p_y,\ \Delta p_z,\ \Delta o_x,\ \Delta o_y,\ \Delta o_z\}, \tag{5.2}$$

where $\Delta p_n \in \mathbb{R}$ represents a change in end-effector position by Δp_n meters along axis $n \in \{x, y, z\}$ and $\Delta o_n \in \mathbb{R}$ represents a change in end-effector orientation, represented by Euler angles, by Δo_n radians about axis $n \in \{x, y, z\}$.

At each time step of a durative-contact manipulation episode, the agent is provided with the robot's and object's states. Given this information, it must decide how to change its end effector's position and orientation. Once the agent's policy has selected an action, this high-level action is converted to a low-level action for the robot to make. First, an impedance controller computes the end-effector force and torque required to obtain the desired change in end-effector position and rotation while maintaining the desired stiffness. These desired end-effector forces and torques are converted to desired joint torques using the robot's Jacobian matrix. These output torques are then fed to the robot's low-level controller. Enabling the agent to learn exactly how to move its end effector to achieve a goal makes for a simplified learning problem, as the robot need not learn both how to control its joints to move its end effector and how to move its end effector to achieve the goal.

5.3.2 Learning a Policy to Perform Durative-Contact Manipulation

Using the sparse reward signal that requires no hand engineering, the state space $S = \{q,\ \dot{q},\ t\}$ containing robot proprioception, object state, and tactile sensor readings, and the abstract operational space control action space $A = \{\Delta p,\ \Delta o\}$, that enables the agent to learn in the robot's workspace directly, GARLA is equipped to learn durative-contact manipulation policies. To learn these policies, I employ a state-of-the-art continuous-control reinforcement-learning framework: TD3 [24] with Hindsight Experience Replay [2].

As detailed in Section 5.2, the reinforcement learning framework enables an agent to learn a policy π that maps states s to actions a. Given an observation that encodes the state of the robot and its environment, the policy predicts which action the robot should take to move towards its goal. At each discrete time step t in an episode, the trained policy informs the robot on how to act. During learning, the agent learns its policy from sequences of states, actions, subsequent states, and rewards. Since GARLA's action space is continuous and not discrete, it requires a reinforcement-learning algorithm with continuous control. DDPG, introduced in Section 2.3.2, is a powerful model-free continuous-control reinforcement learning algorithm [48]. GARLA's learning algorithm is based on DDPG, and it takes advantage of several additional advancements proposed by Fujimoto et al. [24] and Andrychowicz et al. [2].

The Deep Deterministic Policy Gradient agent consists of two DNNs: the Deep Q-Network scores pairs of states and actions, while the policy network predicts which action is optimal given a state. The Deep Q-Network is a critic that evaluates state-action pairs, and the policy network is an actor network that suggests an optimal action given some state. This actor-critic pairing makes selecting an action tractable, as computing the optimal action at every time step by optimizing the Q-network is computationally expensive. DDPG is an off-policy algorithm; all experiences are added to a replay buffer containing tuples of the agent's current state s, action taken in that state a, subsequent state reached s', and reward obtained upon reaching that state r. Both the actor and critic networks are updated at each time step by randomly sampling a new batch of training data from the replay buffer. The agent discards old experience when the buffer becomes too full, enabling the actor and critic networks to be updated with recent data. The size of this replay buffer is a hyper-parameter that requires careful tuning; a small replay buffer causes the agent to overfit to recent experience, while a buffer that is too large makes the agent slow to learn.

DDPG learns its critic DQN using mean-squared Bellman error. The mean-squared Bellman error $MSBE$ for a batch containing N experiences of the form $(s_i,\, a_i,\, r_i,\, s_{i+1})$ is

$$MSBE = \frac{1}{N}\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2, \tag{5.3}$$

where the target is

$$y_i = r_i + \gamma Q(s_{i+1}, \pi(s_{i+1})), \tag{5.4}$$

the discount factor hyper-parameter is γ, the Deep Q-Network is Q, and the policy network is π. The mean-squared Bellman error teaches the critic network to recursively estimate the return for a state using the reward obtained during a transition and the expected return for executing the remainder of the policy from the newly reached state.

DDPG updates its policy network using the gradient of the policy. As the Q-function is differentiable with respect to action, the policy network is updated using gradient ascent. The gradient of the reward J with which the policy network's parameters are updated for a batch of N experiences is

$$\nabla_{\theta^\mu} J = \frac{1}{N}\sum_{i=0}^{N-1} \nabla_a Q(s,a)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\mu}\pi(s|\theta^\mu)|_{s=s_i}. \tag{5.5}$$

This update does not affect the Deep Q-Network's parameters, only the policy network's. The policy is deterministic; the policy network predicts one action given a state. In order to force the agent to explore the environment, DDPG adds Ornstein-Uhlenbeck noise to the output of the policy network when selecting an action in order to randomly explore states that the evolving policy network may not predict to be optimal. This exploration strategy is important, as it enables the agent to search the state space for paths to the goal.

DDPG leverages target networks to make learning more stable. The target actor and critic networks use the same DNN architectures as the actor and critic networks, but learn to slowly track the original networks. DDPG learns using targets that also depend on its Deep Q-Network's weights; with a single actor and single critic network, learning is often unstable. Instead, the weights of the target networks are updated at each learning iteration to slowly track the learned actor and critic networks. Using these target networks to compute the targets for the Deep Q-Network to learn, as calculated in Equation 5.4, makes the networks less likely to diverge.

Though Lillicrap et al. [48] showed that DDPG successfully learns to solve many continuous-control robotic tasks, Fujimoto et al. [24] proposed several extensions; their updated Twin Delayed Deep Deterministic Policy Gradient (TD3) contains three modifications to further enhance performance and achieve state-of-the-art results. To avoid over-estimating the target, TD3 learns a pair of clipped Deep Q-Networks to estimate Q-values. The second enhancement is to simply reduce the frequency of policy network updates. The third involves adding additional noise to the target so the policy network does not overfit to errors produced by the Deep Q-Network.

Like the DDPG architecture it is built off of, TD3 is an off-policy algorithm that learns from a replay buffer. Rather than learn a single Deep Q-Network using experiences in this buffer and update a single delayed target Q-network, TD3 learns two Deep Q-Networks simultaneously. Clipped Double-Q learning learns a pair of twin Q-functions and takes the minimum Q-value in Bellman error loss in order to avoid overestimating the target. Formally, the target for the first Deep Q-Network can be expressed as a modified version of the target defined in Equation 5.4 for one experience tuple $(s_i,\, a_i,\, r_i,\, s_{i+1})$:

$$y_i = r_i + \gamma \min_{q=1,2} Q_{\theta'_q}(s_{i+1}, \pi_\phi(s_{i+1})). \tag{5.6}$$

Here, $Q_{\theta'_1}$ and $Q_{\theta'_2}$ are the two target Deep Q-Networks and π_ϕ is the policy network. Target overestimation error is propagated through the policy network update, but underestimation error is not; therefore, is is preferred to underestimate the target return. Furthermore, minimizing the Q-values enables the agent to focus on states with low variance in the value estimates, which adds stability to the policy network.

In TD3, the policy and target networks are updated less frequently than the Q-functions because policy updates change the targets and make learning difficult if they are updated too often. With a less frequent policy update computed using Q-values with lower variance, learning is more stable. The frequency with which to update the policy and target networks is a hyper-parameter.

The final modification TD3 adds on top of DDPG is noisy target actions that enable the policy to better ignore errors in the learned Deep Q-Network. To ensure that the policy does not overfit to errors in the Q-value estimates, TD3 adds an additional regularization term to the target update to smooth the value estimate. The noisy target estimate is computed by adding some randomly sampled noise ϵ to the target value computed for some experience $(s_i,\, a_i,\, r_i,\, s_{i+1})$:

$$y_i = r_i + \gamma Q_{\theta'}(s_{i+1}, \pi_{\phi'}(s_{i+1}) + \epsilon). \tag{5.7}$$

ϵ is sampled from a noise distribution with mean 0, standard deviation σ, and clipped at $\pm c$. Though DDPG adds noise to the policy when selecting an action to explore, TD3's additional noise helps the policy network avoid overfitting to a potentially erroneous Q-function during learning.

Two of TD3's enhancements can be summarized in the target function

$$y_i = r_i + \gamma \min_{q=1,2} Q_{\theta'_q}(s_{i+1}, \pi_{\phi'}(s_{i+1}) + \epsilon), \tag{5.8}$$

while the third simply reduces the frequency of the target and policy network updates. These three straightforward additions significantly boost the performance of a deep reinforcement-learning agent in continuous domains.

GARLA integrates another state-of-the-art advancement in reinforcement learning, Hindsight Experience Replay (HER) [2], with TD3 to efficiently and robustly learn manipulation policies. HER is a simple algorithm that can be implemented on top of off-policy reinforcement learning algorithms that learn from a replay buffer. HER is particularly useful for scenarios with sparse rewards. It is a form of reward abstraction where the agent still only receives a sparse reward signal from the environment when it completes the task, but internally learns to achieve different goals in the state space. HER does not require an expert programmer to implement a specialized shaped reward for each particular task; the agent is intrinsically rewarded for reaching different states in the state space.

HER follows standard off-policy learning algorithms and adds experience to the agent's replay buffer. During learning, it adds standard experience tuples $(s_i,\ a_i,\ r_i,\ s_{i+1})$ it encounters to the replay buffer after augmenting them with the goal g. The augmented experience tuples are therefore $(s_i,\ a_i,\ r_i,\ s_{i+1},\ g)$. In durative-contact manipulation tasks, this goal is the desired position of the manipuland. For each experience trajectory obtained throughout the course of an episode, HER also adds additional experiences to the replay buffer where the goal is not the true goal of the task, but instead the manipuland position in the final state reached in the episode, g_s. In these modified experiences, the reward is one if the manipuland had reached this final sub-goal state g_s. GARLA's state space contains the manipuland position, so the sub-goal g_s is extracted directly from the final s_{i+1} observed during an episode. Note that the agent receives a positive reward in some states even though they have not achieved the true goal of completing the task.

By adding experiences from both trajectories, at least half of the experiences in the replay buffer have the true goal g. The remainder have goals that the policy was able to reach successfully. With this positive reward, the agent can learn to reach some other state, even if it is not the goal or even necessarily closer to the goal. By learning from trajectories where the agent fails to complete the task, GARLA learns to make progress towards states that it reached during previous exploration attempts and more quickly expand the area that it explores. This enables it to reach the goal that it is interested in faster.

If an agent that uses the state and action spaces defined in Section 5.3.1 and a sparse reward signal was tasked to learn a durative-contact manipulation policy using TD3 and HER, it would first spend time learning to find, approach, and grasp an object before learning to complete the downstream task. Learning these prerequisite skills would be time consuming and potentially impossible with a sparse reward and large state space. Instead, the state-of-the-art reinforcement-learning agent should only begin learning once some grasp has been made on the object using an existing pose-based grasp controller similar to the ones described in Chapter 3. Further, as shown in Chapter 4, a task controller's success is a function of the initial grasp state the system begins in. Therefore, the grasp selection method should be learned.

5.3.3 Learning a Task-Oriented Grasp Classifier in-the-Loop

With a state-of-the-art reinforcement learning algorithm and a learning environment with an intelligently selected action abstraction, my agent can learn a manipulation policy to complete a task when instantiated after a grasp is executed. To select a grasp, I propose that the agent also learn a task-oriented grasp classifier like the ATOG described in Chapter 4. Rather than learn to reach the gripper towards the graspable object, align the gripper with the object, close the fingers, then perform the durative-contact manipulation, an intelligent agent should utilize existing grasping technology to perform a grasp and only learn to manipulate the object after a grasp. Even if learning to reach, grasp, and manipulate was feasible with only a sparse reward, learning only to manipulate will enable the agent to better focus on a single task rather than waste valuable time learning to solve tasks that already have solutions. As the agent learns to complete the task, each episode generates a datum of the initial grasp pose and whether the task succeeds, which is added to the classifier's training set. By updating this classifier as the policy evolves, the agent can focus on learning a policy that completes the task from a focused and related subset of the potential grasp poses at its disposal.

This combination of a grasp classifier and a reinforcement learning policy that begins once that grasp has been executed is inspired by the options framework for reinforcement learning [91]. As described in Section 2.3.2, an option consists of a policy π, an initiation set I of states the option can be run from, and its termination condition β that determines when control should be passed back to the global agent. Bagaria and Konidaris [3] model the initiation set as a classifier $I(s)$ that predicts whether a given state belongs in an option's initiation set. The parallel between the use of a lightweight classifier to classify states to begin an option policy from and the use of a task-oriented grasp classifier to select grasps at which to begin a manipulation policy from is obvious. A manipulation policy and option policy are both executed or learned after the system reaches a state in the initiation set. The initiation set classifier and grasp pose classifier are both trained with data generated as the policy learns. Therefore, the framework presented in this chapter could be extended to non-robot domains, or domains where multiple skills must be chained sequentially.

In many manipulation tasks, a robot can successfully complete the task by performing a grasp and executing its manipulation controller while maintaining the grasp. Therefore, the initial states of the given π can be modeled as a grasp executed at a given pose g. As defined in Section 3.3, these grasp poses are defined by a position and orientation. Therefore, a task-specific grasp detector must return a pose at which the given manipulation controller can be executed, as the ATOG system described in Chapter 4 did. The robot will execute this pose with its given motion planner, then will execute the manipulation controller.

The smaller and more task-relevant action space introduced by operational space control makes learning to complete complex manipulation tasks more feasible, but it does not completely eliminate the difficulty induced by the sparse reward. As the agent explores the environment, it must still control the end effector to reach the graspable object, execute a grasp, and manipulate the object before receiving a positive reward signal. Integrating a grasp detection system and grasp controller with the manipulation control policy further injects structure into the problem. Durative-contact manipulation tasks can be solved by chaining together a general grasping skill with a manipulation skill. Rather than begin learning the manipulation control policy with the arm in an arbitrary position away from the manipuland, the system should first grasp the object, then hand control over to the manipulation policy.

The input to this grasp detection sub-system is a point cloud P. As the pose of each depth sensor is known relative to the robot, this point cloud P, which is initially represented in the robot's frame, can be transformed to a cloud P_o in a fixed object frame. This ensures that all points are represented relative to a fixed pose of the object, either given or detected through a computer vision algorithm, so the cloud is invariant to changes in the object's pose. The grasp proposal algorithm **GEN** defined in Section 3.3 first generates a set of potential candidate grasp poses in the object frame G_o from the point cloud: **GEN** : $P_o \rightarrow G_o$. The task-oriented grasp classifier must then predict whether a given grasp will enable the manipulation controller π to successfully complete the task. In other words, the task-oriented grasp classifier must determine if a given grasp pose is in the manipulation controller's initiation set. I define this task-oriented grasp classifier as **TOGC** : $g_o \in G_o \rightarrow [0, 1]$.

The proposed **TOGC** does not operate on local or global visual information, P_o. As each **TOGC** predicts whether one particular task can be completed at a grasp on one particular instance of an object, and these grasp candidates G_o are represented in the object frame, this decision can be made from the poses G_o alone. This is a reasonable assumption, especially for home robots that would interact with the same instances of objects repeatedly. For the door opening task, though the same general door opening policy could be used on all doors in a house, the controller's initiation set would differ from door to door because of factors such as the weight, handle location, and size of the doors. Therefore, a different **TOGC** would be trained for each door. The poses passed to **TOGC** are vectors of size seven that contain a grasp's Cartesian position and its orientation represented as a quaternion. This is similar to the input passed to the tail end of the ATOG network defined in Chapter 4, but it does not contain the grasp stability prediction from the PointConv branch of the network, as this additional information did not prove useful for solving the door and switch tasks. The network architecture is also similar to the tail end of the ATOG network; three fully connected layers predict task-oriented grasp success probability.

Concretely, my Grasp-Aware Reinforcement-Learning Agent combines this **TOGC** grasp classifier with a reinforcement-learning agent as follows. At the beginning of an episode, the system captures a point cloud of the scene. The system is evaluated in simulation only because reinforcement-learning algorithms are data hungry, so this point cloud is always simulated, though the system is compatible with real point clouds as well. The grasp pose generation algorithm **GEN** defined in Section 3.3 proposes a set of potential grasp poses G given the point cloud in the object frame. Initially, the agent selects one of these grasp poses at random. If the **TOGC** classifier had been trained at least once during a previous episode, the classifier predicts the probability of task success for each of these grasp candidates. These normalized predicted probabilities seed the random grasp selector, enabling the system to attempt grasps at poses that are more likely to enable the robot to complete the task. After selecting a grasp and executing it using the same controller used in Chapter 4, control of the robot is handed over to the reinforcement-learning agent. The agent executes its policy until the episode terminates when the robot achieves task success or exceeds the step limit. After an episode terminates, the system stores the new example in the **TOGC** classifier's training set. This data point consists of the grasp pose g_e at which the system began the episode and a binary label l_e that is 1 if the robot completed the task in this episode and 0 otherwise. As the simulated point cloud in the object frame is the same during each episode, the output of **GEN** is always the same, and the set of potential grasp poses is discrete. If the agent attempted to complete the task from grasp pose g_e during a previous episode, the system simply updates the corresponding label l_e already present in the training set, as labels could change as the policy evolves. If this training set contains at least 10 positive and 10 negative examples, the system retrains the **TOGC** classifier for 10 epochs. As described in Section 5.3.2, the system also retrains the reinforcement-learning agent. This process is repeated during each episode, with each new experience updating the classifier and the classifier dictating how grasps are selected.

5.3.4 Weighting Task-Oriented Grasp Labels

Though the Grasp-Aware Reinforcement Learning Agent that uses the components described in the last three sections could likely learn to solve manipulation tasks faster and with higher accuracy than baseline agents, the solution discussed so far ignores an important aspect of this problem. The agent learns a policy that often begins at a grasp candidate with a high estimated probability of task success, but these predictions are made with a classifier trained on data labeled by the policy. This is an entangled joint learning problem because the agent trains two neural networks simultaneously and the labels used to train each system depend on the performance of the other. The reinforcement-learning policy's success depends on its initial grasp pose, as Chapter 4 illustrates; its initial grasp pose is determined by the grasp classifier. The agent learns the grasp classifier using labels that its manipulation policy generates; as the policy changes and the robot learns to complete the task, initial task completion labels become incorrect. The initial policy acts at random, and successful task completion is unlikely. The agent labels all of these training examples as bad grasps, even though they may be good grasps that enable a trained policy to complete the task. Rather than retrain a classifier with these stale examples, an optimal agent should retrain its classifier while focusing on examples that agree with its current policy.

A simple weighting scheme could involve only training on recent examples. However, tuning this parameter would be difficult and the agent would likely throw out too many or too few examples. Furthermore, even labels generated with an older policy have some use. If an agent is able to initially solve the task from some grasp state using a sub-optimal policy, it may be able to solve the task from that grasp with a more robust policy. Another simple strategy that ignores older labels during classifier training is to keep only the latest label for each example. This is not possible in continuous domains or a real robot where the number of possible grasp poses $G \subseteq \mathbb{SE}(3)$ is infinite. To completely account for the shifting distribution of grasp classifier training labels, my agent must be equipped with a more complex method of ignoring stale data. Specifically, the network should focus on learning to classify examples labeled with a policy close to the current one by loss-function weighting.

Training a DNN classifier to focus on some examples more than others is simple. As discussed in Sections 3.3 and 5.3.3, the grasp classifier is a binary classifier that predicts whether or not a given grasp would succeed. The cross-entropy loss function that a binary grasp classifier is trained with defined in Equation 3.2 is

$$-\sum_{c=0}^{m-1} y_c \; log(b_c) \tag{5.9}$$

for a particular example where y_c is 1 if c is the correct label for a given exemplar and 0 otherwise, b_c is the estimated probability that the exemplar is of class c, and $m = 2$ in the two-class binary classification case. Specifically, b_c is the softmax of the logits output by the classifier. When training a classifier on a dataset D, the loss over all examples $d \in D$ is

$$-\sum_{d=0}^{|D|-1} \sum_{c=0}^{m-1} y_{d,c} \; log(b_{d,c}). \tag{5.10}$$

In order to have the classifier focus on some examples more than others during training, each example can be assigned a weight $w_d > 0$. The weighted cross-entropy loss function is

$$-\sum_{d=0}^{|D|-1} w_d \sum_{c=0}^{m-1} y_{d,c} \; log(b_{d,c}). \tag{5.11}$$

Each training datum now consists of the classifier input, a label, and a weight. If all weights are set to one, $w_d = 1 \; \forall \; d \in D$, the weighted cross-entropy loss function is reduced to the standard cross-entropy loss function. Examples with weights $0 < w_d < 1$ are downweighted, while the loss is amplified for examples with weights $w_d > 1$. These weights affect the loss during training, but do not affect the predictions at inference time.

With this weighted loss function, GARLA is equipped to train its task-oriented grasp classifier given grasp examples with binary labels and weights. The approach I take is inspired by Natarajan et al. [67], who examine classification with noisy labels and relate uncertainty to the probability that a noisy label could flip. I define a weighting function **WeightGrasp** $: q_g, l_g \rightarrow w_g$ that takes in the current Q-function's scores of the grasp pose g, q_g, and the latest received binary task completion label when the agent began its policy execution from grasp g, l_g. **WeightGrasp** returns a weight with which the grasp example g is weighted in the classifier's training during the next training iteration. Intuitively, if the Q-function's evaluation of a grasp state does not match the most recent label obtained, the uncertainty is high and the example should be down-weighted when training the grasp classifier.

Concretely, GARLA first assigns each grasp pose g in the classifier's training set with the full reinforcement-learning agent state encountered at the end of a grasp execution but before the reinforcement-learning policy takes control, s_g. As these grasps had been attempted previously, these states are known. These states contain the robot and object states, as described in Section 5.3.1. Next, GARLA predicts the optimal action corresponding to the grasp state s_g using its policy network:

$$a_g = \pi(s_g). \tag{5.12}$$

GARLA calculates the Q-function score q_g for a grasp g by estimating the Q-score for the $(s_g,\, a_g)$ pair with its Deep Q-Network:

$$q_g = \mathbf{DQN}(s_g, a_g). \tag{5.13}$$

WeightGrasp then assigns weights to a grasp example with q_g and the label l_g. It calculates a labeling threshold t_D for a training set D as the median across all Q-values q_g:

$$t_D = \operatorname{median} q_g \;\forall\; g \in D. \tag{5.14}$$

This weighting function is compatible with distributional Q-functions, which predict a distribution of scores for a given state-action pair rather than a single value [7]. With distributional Q-functions, each q_g is a vector of values instead of a single score. In the distributional case, t_D is similarly calculated as the median of all values in all Q-value distributions. For each example g, the flipping mass fm_g is calculated as the sum of all Q-values in the distribution that, when thresholded, do not match the latest ground-truth label:

$$fm_g = \sum_{i=0}^{|q_g|-1} \begin{cases} q_{g,i}, & (q_{g,i} < t_D \text{ and } l_g = 1) \text{ or } (q_{g,i} > t_D \text{ and } l_g = 0) \\ 0, & \text{otherwise} \end{cases} \tag{5.15}$$

A grasp's flipping mass is large if, on average, the corresponding Q-values do not agree with the latest ground-truth label. A grasp's flipping probability is the ratio of its flipping mass to the sum of all values from its Q-value distribution:

$$fp_g = \frac{fm_g}{\sum\limits_{0}^{|q_g|-1} q_{g,i}}. \tag{5.16}$$

When the critic is implemented as a standard Q-function and not a distributional Q-function, this flipping probability will be one if the Q-value is below the dataset's median and the grasp is labeled as a success or if the Q-value is above the dataset's median and the grasp is labeled as a failure, and it will otherwise be zero if the thresholded Q-value aligns with the most recent label. Finally, **WeightGrasp** computes a grasp's weight w_g as the reciprocal of this flipping probability ratio plus a small constant c for stability:

$$w_g = \frac{1}{fp_g + c}. \tag{5.17}$$

With a standard Deep Q-Network critic, this weight will be close to one for grasps whose Q-values do not match their labels, and c^{-1} for those that do. The resulting weight will be larger for grasps whose corresponding Q-value distributions align with their latest ground-truth labels. This weighting scheme enables GARLA to focus on grasp examples whose labels its Q-function is more certain of when learning a grasp classifier, allowing the classifier to ignore examples that were likely labeled using a stale policy.

5.4 Experiments

Though Chapter 4 has shown that my task-oriented grasp classifier can be trained with as little as 20 labeled examples, reinforcement-learning algorithms require much more data to learn complex manipulation policies. Therefore, I train and evaluate GARLA exclusively in simulation environments where episodes run quickly without human intervention or manual labeling. The two domains in which I evaluate this agent are the door opening and switch flipping environments introduced in Section 4.4 and simulated in MuJoCo [97]. These tasks challenge the agent to learn the relationship between its actions and an articulated object; the single-step switch task requires the arm to rotate the switch about its base, and the more complex two-step door task requires the arm to sufficiently rotate the door handle before pulling the door open.

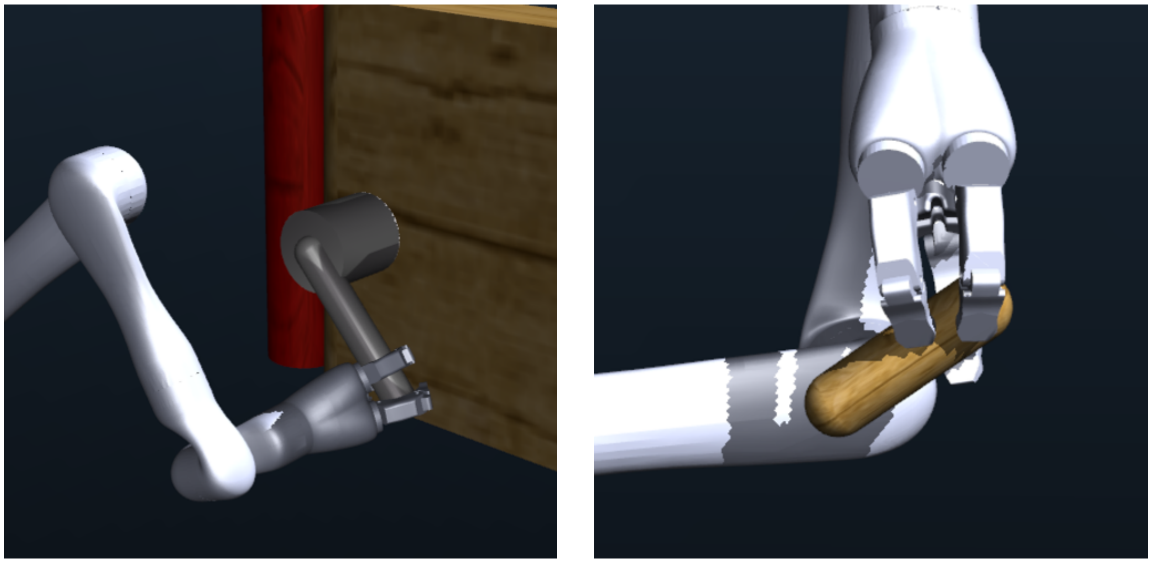

(a) Simulated Door Domain (b) Simulated Switch Domain

Figure 5.2: The two simulated tasks the Grasp-Augmented Reinforcement-Learning Agent must learn to complete.

As discussed in Section 5.3.1, GARLA receives a sparse reward in these domains; 1 at the time step when the task succeeds (at which point the episode ends), and 0 for all others. In the door domain, the positive reward is triggered when the door is pulled towards the arm by 0.5 radians. The switch must be rotated upwards by 3 radians. If the agent fails to complete a task after executing a fixed number of actions, the execution is considered a failure.

The agent is equipped with a simulated Kinova Jaco arm to solve the two tasks. This arm has six articulated joints, and each of its gripper's three fingers has two articulated joints. Each finger's two linkages are equipped with a simulated tactile sensor array. The door consists of a rotatable handle and an articulated door that can be opened once the handle is rotated by $\pm\, 90°$. The switch has one axis of articulation. Therefore, the state space defined in Section 5.3.1, $S = \{q_r,\, \dot{q_r},\, q_g,\, \dot{q_g},\, q_o,\, \dot{q_o},\, t\}$, consists of vectors of the following sizes: $|q_r| = 6$, $|q_g| = 6$, $|q_d| = 2$, $|q_s| = 1$, $|t| = 6$, where q_d and q_s are the q_o for the door and switch domains respectively.

I compare the performance of my proposed GARLA against three baseline ablations. Each of these ablations learns the manipulation control policy from the same state and action spaces using the same hyper-parameters; the methods to select the grasp poses to begin the manipulation execution from differ. GARLA selects from the proposed grasp pose candidates at random, with the probability of selecting any grasp being the normalized probability that the grasp enables the agent to complete the task as predicted by the task-oriented grasp classifier trained with weighted examples. The unweighted classifier ablation uses the same grasp selection strategy, but its task-oriented grasp classifier is trained to predict task success without weighted exampled. The most simple of the ablations selects from all feasible grasp candidates uniformly at random. It utilizes the operational space control action space and grasp-bootstrapped reinforcement-learning framework, but does not take advantage of a task-oriented grasp classifier. The final ablation agent is provided with "oracle" grasps: the final 15 grasps that another agent was able to complete the task from during its training in a previous run. It selects uniformly at random from this small set of good grasps, and should provide an upper bound on performance.

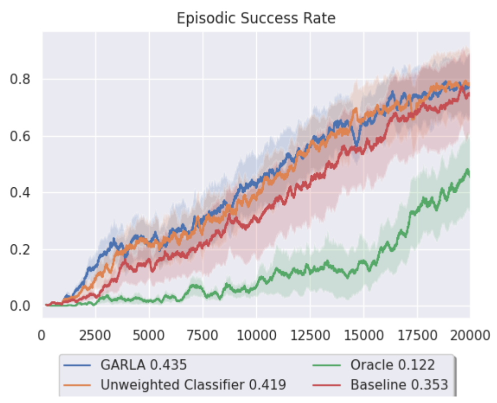
For each task domain, I generate a simulated point cloud of the graspable area and generate a set of grasp candidate poses using the grasp pose generation algorithm **GEN** defined in Section 3.3. As in Chapter 4, unreachable grasp poses are removed from the set of grasp poses. In each episode of an experimental trial, the agent begins by selecting a grasp using its grasp selection algorithm described in the preceding paragraph, then executing it with the grasp controller used in Chapters 3 and 4. Once the selected grasp is executed, GARLA passes control to its manipulation policy.

All agents' manipulation policies train with actor and critic learning rates of 3×10^{-5} for the door domain and 3×10^{-6} for the switch domain. Their reward discount rate γ is 0.99, TD3 target action noise with standard deviation 0.2 is clipped at 0.5, and exploration noise is generated with standard deviation 0.1. The policy network and target networks are updated with every 2 Deep Q-Network updates and the batch size is 256. Each agent is trained with the same five random seeds for a fair comparison, and all reported results are averaged over these seeds. Each agent is given 20,000 episodes to learn the task. When learning a grasp classifier, the grasp pose and binary success label based on whether the task succeeded are added to the training set after each episode. If that grasp pose had already been attempted, the old label is over-written. Once the training set contains at least 10 positive and 10 negative examples, the classifier is trained for 10 epochs after each episode with the current training set. The probability that every grasp would succeed is predicted with this retrained classifier, and these probabilities are then used to randomly select the next grasp to attempt.

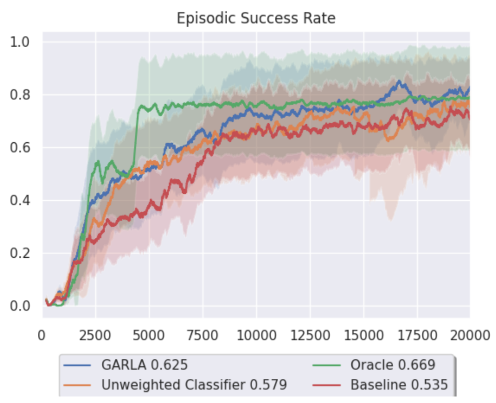
In order to evaluate my system, I measure each agent's performance as they learn. To visualize average performance as each agent learns, I plot the fraction of episodes that resulted in task success over the past 200 episodes. This averaged task success metric shows how well each agent learns as the training episodes roll out. Though the agent does randomly explore during some time steps due to TD3's exploration noise, causing sub-optimal behavior, all agents explore with the same likelihood, so exploration should not impact performance when comparing the agents. Examining average task success as a function of episode enables one to determine when and how quickly an agent learns to complete the task, as well as how an agent performs at the end of its training. Figure 5.3 shows the performance of the four agents on the door and switch domains.

5.4.1 Discussion

In the switch case, all agents learn to perform the task quickly; Figure 5.3b plots the agents' performances. As expected, the agent with access to oracle grasps performs the best, spiking to 75% task success around 4,000 episodes and approaching an 80% success rate after 20,000 episodes. The



(a) Door domain results.



(b) Switch domain results.

Figure 5.3: Reinforcement-learning agents' performances on the door and switch tasks. The plots show success rate averaged over the past 200 training episodes vs. episode number. Error over five random seeds is shaded.

smoothed success rate averaged over all episodes is 66.9%. GARLA with my weighted classifier performs very well; it attains a 60% success rate after 6,000 episodes and surpasses 80% after 20,000. Its average smoothed success rate is 62.5%. Though, on average it does not perform as well as the agent with access to the oracle grasps, GARLA efficiently learns which grasps are good while learning to complete the task, and achieves a higher smoothed success rate than the oracle agent after 20,000 episodes. The unweighted classifier ablation achieves a success rate over 75% after 20,000 episodes. While it learns about as quickly as the GARLA agent, it completes the task less often, and does not achieve a smoothed task completion rate above 80%; on average, its 57.9% success rate is 4.6% lower than GARLA's. The agent that selects grasps at random performs the worst, but it does learn to complete the task. It takes 7,500 episodes to reach an average success rate of 60% and the final success rate does not approach 80%. Its average smoothed success rate of 53.5% is 4.4% lower than the agent with an unweighted grasp classifier and 9% lower than GARLA's. These results support my hypothesis that an agent that learns with state and action spaces designed for manipulation tasks and a state-of-the-art reinforcement-learning algorithm can learn an effective durative-contact manipulation policy, but that integrating a task-oriented grasp classifier and weighting this classifier both improve performance. Additionally, empowering a reinforcement-learning agent with a weighted grasp classifier enables it to learn to complete the task more quickly than the baseline agents, but not as quickly as an agent given knowledge about known optimal grasps.

The door task is more difficult since it is a two-part task with a handle rotation sub-goal, but the agent is only rewarded for opening the door past the given threshold. As Figure 5.3a shows, my GARLA agent achieves the best performance, reaching a 60% success rate around 13,000 episodes, a final task success rate near 80%, and an average smoothed success rate of 43.5%. The ablation agent with an unweighted grasp classifier also performs well; it reaches the 60% success rate milestone at the same time as GARLA, and its final performance is also about the same. Its averaged smooth success rate of 41.9% is 1.6% lower than GARLA's. The baseline ablation agent that selects grasps at random also learns to complete the task, achieving a smoothed success rate of 60% around 16,000 episodes and a final smoothed success rate close to 75%. Its average smoothed success rate of 35.3% is 8.2% lower than GARLA's. Interestingly, the agent that selects from the small set of oracle grasps at random does not perform well. It takes 16,000 episodes to reach a 20% task success rate, and its smoothed success rate only approaches 50% after 20,000 episodes. Its average smoothed task success rate of 12.2% is 31.3% lower than GARLA's. One possible explanation for this is that the other agents that learn from the full set of grasps are more robust; if their grasp slips during task execution, the resulting grasp pose would likely still be similar to a grasp pose encountered during training, and the policy could recover and complete the task. If an agent that only begins the task from a small set of states slips from a grasp pose, it would likely never have seen the resulting grasp pose before, and the agent would be unable to recover and complete the task. Another explanation is that the policy learned from a small set of grasps is brittle and only succeeds for a subset of the good grasp poses. The set of oracle grasps that work well for a previous agent could be a subset of the good grasps in a multi-modal distribution. It could be too difficult to learn a policy that succeeds on a subset of this distribution. Learning a grasp classifier while learning the manipulation policy enables the agent to select which grasps to focus on while its policy evolves over time. Though the oracle agent does not provide an upper bound on performance as expected, my GARLA agent learns to complete the door task well, learning a robust policy while focusing on good grasps. In this environment, relying on a set of grasps that enabled another agent to complete the task is not sufficient; to learn a robust policy, the agent must be allowed to explore the graspable region and learn which grasps enable its evolving policy to complete the task.

5.5 Conclusion

In order to learn complex manipulation tasks, reinforcement-learning agents must be equipped with action space abstractions crafted specifically to enable learning in the robot's end effector space. Combining existing sub-systems such as a kinematic solver and a motion planner to perform grasping as a skill bootstraps the learning process so the agent need not re-learn to achieve basic preconditions. The method I have defined in this section combines the lightweight task-oriented grasp classifier from Chapter 4 with a reinforcement-learning agent, enabling the agent to learn a manipulation policy while learning which grasps lead to task success. To solve this entangled learning problem, the agent learns to update its notion of a good grasp by weighting this classifier's training examples.

Chapter 6

Conclusions

To move robots from controlled manufacturing facilities where they perform repetitive tasks to unstructured and variable home environments where they may be asked to performed complex tasks, they must be imbued with the ability to learn to complete new tasks using unseen objects on-the-fly. The most vital skill robots require to learn to manipulate novel environments is the ability to stably grasp objects of varying shapes and sizes. Once they can stably grasp objects, they must reason about how to best grasp objects in order to perform downstream tasks with them. Learning to perform these downstream tasks can be very difficult. To efficiently learn to complete new tasks, robots must simultaneously learn which grasps enable them to complete the task.

To conclude, I have introduced a novel neural network architecture and learning process that predict grasp quality for each potential type of grasp for a general picking task. Then, I extended this method to train a task-oriented grasp classifier given a policy and small amounts of labeled data by leveraging a pre-trained grasp quality prediction branch. Finally, I have shown how this task-oriented grasp classifier can be used as a manipulation option's initiation set classifier to efficiently learn complex manipulation tasks by selecting optimal grasps at which to begin manipulation. With these abilities, a robot could be introduced to a new environment and quickly learn to manipulate novel objects using structured actions. The experimental results I have provided support my thesis statement:

With a classifier that learns from limited data and weighting that considers uncertainty, we can train a task-oriented grasp classifier jointly with a manipulation policy to efficiently learn to manipulate objects.

6.1 Contributions

In this dissertation, I have presented three major contributions. Though grasping is a difficult problem, breaking it down into a grasp detection problem and solving it with computer vision techniques, then leveraging standard motion planning algorithms to execute grasps enables it to be solved effectively. Existing systems provide methods to assess grasps for simple grippers, but do not explicitly consider different types of grasps. In Chapter 3, I present a Multi-Modal Grasp Pose Detector that predicts where and with what grasp type a robot should attempt a grasp on a singulated object or a pile of cluttered objects. A neural network architecture that operates directly on point clouds and predicts the probabilities that each grasp type succeeds at a given pose is defined. By explicitly modeling grasp type and using all five grasp types a Robotiq 3-Finger Adpative Gripper is capable of, I showed that a real robot can remove more objects from clutter. Efficiently removing objects from clutter is a vital skills robots must possess, especially when picking items from warehouse bins or unpacking groceries in a home.

Though grasp stability is important when a robot selects a grasp for a more complex manipulation task, it must also consider whether a grasp is suitable for the task. I study this task-oriented grasp detection problem in Chapters 4 and 5. My second major contribution, presented in Chapter 4, is a specialization of a generic grasp detector, where the robot must grasp an object to complete a certain task. This Augmented Task-Oriented Grasp Detection Network predicts grasp stability with a pre-trained grasp stability branch. Since generating labeled manipulation attempts with a real robot is costly due to long execution time and safety hazards, this grasp detector must learn from as small a set of labeled examples as possible. ATOG's tail, therefore, is a lightweight neural network that predicts whether a grasp enables a robot to complete a task based on the grasp pose and the output of the grasp stability branch of the network. This learning framework enables the robot to quickly learn which grasp poses allow it to complete a task for an individual instance of an object with a given policy.

Grasping alone enables robots to pick objects, but more complex manipulation tasks necessitate more complex manipulation policies. In Chapter 5, I combine the lightweight task-oriented grasp classifier from Chapter 4 with a state-of-the-art reinforcement-learning agent to learn complex manipulation policies and grasp-selection classifiers simultaneously with a Grasp-Aware Reinforcement-Learning Agent. Since the learned manipulation policy assigns labels with which the grasp classifier is trained in this joint learning problem, I present a weighting scheme for the grasp classifier that enables it to accurately classify grasp examples as the policy evolves. As demonstrated experimentally with several simulated manipulation tasks, integrating a grasp classifier with a reinforcement-learning agent enables the agent to learn to complete a task faster and more accurately.

6.2 Future Directions

There are several natural extensions to the systems described in this dissertation, as well as planned future works that build upon these systems. Importantly, each of my three presented frameworks is modular. With my Multi-Modal Grasp Pose Detector and Augmented Task-Oriented Grasp Detection Network, an algorithm first proposes a set of potential grasp candidates from visual information, then the candidates are evaluated, with one being selected for execution. This modularity enables these systems to be easily integrated with algorithms that further constrain grasp selection by masking the inputs or outputs. For instance, a computer vision algorithm or user could define one object in particular to pick from a pile of clutter, and a multi-modal grasp detector could only consider grasp poses on that object. Delicate areas of objects could be identified, and grasps could be avoided in these areas to avoid damage. Prior information about which parts on an object can be stably grasped can be fed to a task-oriented grasp classifier to learn even more quickly. A grasp detector would be a useful module in other systems, such as an object retrieval framework like the one proposed by Nguyen et al. [69] or a system that also learns to utilize primitive nudging actions to separate clutter before grasping like the one described by Tang et al. [92]; I have collaborated on both of these works.

These trained quality or task-oriented grasp detectors and learned manipulation policies could be used in a task and motion planning context. A trained classifier could be implemented as an abstract, parameterized action available to a task and motion planner. The frameworks proposed here are crucial precursors to a larger system that will chain multiple durative-contact skills together to perform even more complex manipulation tasks.

Bibliography

[1] B. Abbatematteo, S. Tellex, and G.D. Konidaris. Learning to generalize kinematic models to novel objects. In *Proceedings of the 4th Annual Conference on Robot Learning*, Oct 2020.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, , and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[3] A. Bagaria and G. D. Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020.

[4] A. Bagaria, J. Senthil, M. Slivinski, and G. D. Konidaris. Robustly learning composable options in deep reinforcement learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2161–2169, 8 2021.

[5] R. Balasubramanian, L. Xu, P. Brook, J. Smith, and Y. Matsuoka. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *IEEE Transactions on Robotics*, 28:899–910, 2012.

[6] Y. Bekiroglu, D. Song, L. Wang, and D. Kragic. A probabilistic framework for task-oriented grasp stability assessment. In *2013 IEEE International Conference on Robotics and Automation*, 2013.

[7] M. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458, 2017.

[8] J.L. Blanco-Claraco. A tutorial on **SE**(3) transformation parameterizations and on-manifold optimization. *arXiv preprint arXiv:2103.15980*, 2022.

[9] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014.

[10] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P Abbeel, and A. Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36:261–268, 2017.

[11] X. Chen, K. Zheng, Z. Zeng, S. Basu, J. Cooney, J. Pavlasek, and O. C. Jenkins. Manipulation-oriented object perception in clutter through affordance coordinate frames. *arXiv preprint arXiv:2010.08202*, 2021.

[12] F. Chu, R. Xu, and P. Vela. Learning affordance segmentation for real-world robotic manipulation via synthetic images. *IEEE Robotics and Automation Letters*, 4(2):1140–1147, 2019.

[13] M. Corsaro, S. Tellex, and G. D. Konidaris. Learning to detect multi-modal grasps for dexterous grasping in dense clutter. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4647–4653, 2021.

[14] H. Dang and P. Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1311–1317, 2012.

[15] Z. Deng, G. Gao, S. Frintrop, F. Sun, C. Zhang, and J. Zhang. Attention based visual analysis for fast grasp planning with a multi-fingered robotic hand. *Frontiers in Neurorobotics*, 13:60, 2019.

[16] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *2020 IEEE International Conference on Robotics and Automation*, pages 11516–11522, May 2020.

[17] R. Detry, J. Papon, and L. Matthies. Task-oriented grasping with semantic and geometric scene understanding. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[18] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research*, 39(2-3):202–216, 2020.

[19] D. Faria, R. Martins, J. Lobo, and J. Dias. Extracting data from human manipulation of objects towards improving autonomous robotic grasping. *Robotics and Autonomous Systems*, 60(3):396–410, 2012.

[20] D. Faria, P. Trindade, J. Lobo, and J. Dias. Knowledge-based reasoning from human grasp demonstrations for robot grasp synthesis. *Robotics and Autonomous Systems*, 62(6):794–817, 2014.

[21] T. Feix, J. Romero, H. Schmiedmayer, A. M. Dollar, and D. Kragic. The grasp taxonomy of human grasp types. *IEEE Transactions on Human-Machine Systems*, 46:66–77, 2016.

[22] C. Ferrari and J. Canny. Planning optimal grasps. In *1992 IEEE International Conference on Robotics and Automation*, May 1992.

[23] Y. Fleytoux, A. Ma, S. Ivaldi, and J.B. Mouret. Data-efficient learning of object-centric grasp preferences. *arXiv preprint arXiv:2203.00384*, 2022.

[24] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

[25] W. Gao and R. Tedrake. kpam 2.0: Feedback control for category-level robotic manipulation. *IEEE Robotics and Automation Letters*, 6(2):2962–2969, 2021.

[26] Wei Gao and Russ Tedrake. kpam-sc: Generalizable manipulation planning using keypoint affordance and shape completion. In *2021 IEEE International Conference on Robotics and Automation*, pages 6527–6533, 2021.

[27] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation*, pages 3389–3396, 2017.

[28] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 598–605, 2016.

[29] W. Guo, W. Li, Z. Hu, and Z. Gan. Few-shot instance grasping of novel objects in clutter. *IEEE Robotics and Automation Letters*, 7(3):6566–6573, 2022.

[30] R. Haschke, J.J. Steil, I. Steuwer, and H. Ritter. Task-oriented quality measures for dextrous grasping. In *2005 International Symposium on Computational Intelligence in Robotics and Automation*, pages 689–694, 2005.

[31] H. He and R. Xia. Joint binary neural network for multi-label learning with applications to emotion classification. In *Natural Language Processing and Chinese Computing*, pages 250–259, 2018.

[32] J. Ibarz, D. Kalashnikov, P. Pastor, M. Kalakrishnan, D. Quillen, A. Herzog, S. Levine, V. Vanhoucke, E. Holly, E. Jang, and A. Irpan. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of the 2nd Annual Conference on Robot Learning*, Oct 2018.

[33] E. Jang, S. Vijayanarasimhan, P. Pastor, J. Ibarz, and S. Levine. End-to-end learning of semantic grasping. In *Proceedings of the 1st Annual Conference on Robot Learning*, Nov 2017.

[34] D. Kappler, J. Bohg, and S. Schaal. Leveraging big data for grasp planning. In *2015 IEEE International Conference on Robotics and Automation*, pages 4304–4311, May 2015.

[35] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[36] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[37] A. Khazatsky, A. Nair, D. Jing, and S. Levine. What can i do here? learning new skills by imagining visual affordances. In *2021 IEEE International Conference on Robotics and Automation*, pages 14291–14297, 2021.

[38] M. Kokic, J. Stork, J. Haustein, and D. Kragic. Affordance detection for task-specific grasping using deep learning. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 91–98, 2017.

[39] M. Kokic, D. Kragic, and J. Bohg. Learning task-oriented grasping from human activity datasets. *IEEE Robotics and Automation Letters*, 5, 2020.

[40] D. Kragic, A. Miller, and P. Allen. Real-time tracking meets online grasp planning. In *2001 IEEE International Conference on Robotics and Automation*, May 2001.

[41] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[42] O. Kroemer, S. Niekum, and G. D. Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of Machine Learning Research*, 22, 2021.

[43] L. Ku, E. Learned-Miller, and R. Grupen. Associating grasp configurations with hierarchical features in convolutional neural networks. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2434–2441, 2017.

[44] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[45] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37:421–436, 2018.

[46] Y. Li, J. Fu, and N. Pollard. Data-driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):732–747, 2007.

[47] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, and J. Zhang. Pointnetgpd: Detecting grasp configurations from point sets. In *2019 International Conference on Robotics and Automation*, pages 3629–3635, May 2019.

[48] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

[49] M. Liu, Z. Pan, K. Xu, K. Ganguly, and D. Manocha. Generating Grasp Poses for a High-DOF Gripper Using Neural Networks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1518–1525, Nov 2019.

[50] M. Liu, Z. Pan, K. Xu, K. Ganguly, and D. Manocha. Deep Differentiable Grasp Planner for High-DOF Grippers. In *Proceedings of Robotics: Science and Systems*, July 2020.

[51] W. Liu, A. Daruna, and S. Chernova. Cage: Context-aware grasping engine. In *2020 IEEE International Conference on Robotics and Automation*, pages 2550–2556, 2020.

[52] Q. Lu and T. Hermans. Modeling Grasp Type Improves Learning-Based Grasp Planning. *IEEE Robotics and Automation Letters*, 4:784–791, 2019.

[53] Q. Lu, K. Chenna, B. Sundaralingam, and T. Hermans. Planning Multi-Fingered Grasps as Probabilistic Inference in a Learned Deep Network. In *International Symposium on Robotics Research*, pages 455–472, 2017.

[54] J. Lundell, F. Verdoja, and V. Kyrki. Robust grasp planning over uncertain shape completions. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1526–1532, 2019.

[55] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, Juan A., and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Proceedings of Robotics: Science and Systems*, July 2017.

[56] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg. Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning. In *2018 IEEE International Conference on Robotics and Automation*, May 2018.

[57] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26), 2019.

[58] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. Kpam: Keypoint affordances for category-level robotic manipulation. In *International Symposium on Robotics Research*, 2019.

[59] R. Martín-Martín, M. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg. Variable impedance control in end-effector space. an action space for reinforcement learning in contact rich tasks. In *Proceedings of the International Conference of Intelligent Robots and Systems*, 2019.

[60] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122, Dec 2004.

[61] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836.

[62] R. Monica and J. Aleotti. Point cloud projective analysis for part-based grasp planning. *IEEE Robotics and Automation Letters*, 5(3):4695–4702, 2020.

[63] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019.

[64] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Oct 2019.

[65] A. Murali, W. Liu, K. Marino, S. Chernova, and A. Gupta. Same object, different grasps: Data and semantic knowledge for task-oriented grasping. In *Proceedings of the 4th Annual Conference on Robot Learning*, Nov 2020.

[66] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target-driven object manipulation in clutter. In *2020 IEEE International Conference on Robotics and Automation*, pages 6232–6238, June 2020.

[67] N. Natarajan, I. Dhillon, P. Ravikumar, and A. Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems*, Dec 2013.

[68] R. Newbury, M. Gu, L. Chumbley, A. Mousavian, C. Eppner, J. Leitner, J. Bohg, A. Morales, T. Asfour, D. Kragic, D. Fox, and A. Cosgun. Deep learning approaches to grasp synthesis: A review. *arXiv preprint arXiv:2207.02556*, 2022.

[69] T. Nguyen, N. Gopalan, R. Patel, M. Corsaro, E. Pavlick, and S. Tellex. Robot Object Retrieval with Contextual Natural Language Queries. In *Proceedings of Robotics: Science and Systems*, July 2020.

[70] V. Nguyen. Constructing force-closure grasps. In *1986 IEEE International Conference on Robotics and Automation*, Apr 1986.

[71] T. Osa, J. Peters, and G. Neumann. Experiments with hierarchical reinforcement learning of multiple grasping policies. In *2016 International Symposium on Experimental Robotics*, 2016.

[72] T. Pardi, R. Stolkin, and A. Ghalamzan E. Choosing grasps to enable collision-free post-grasp manipulations. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 299–305, 2018.

[73] M. Prats, P. Sanz, and A. del Pobil. Task-oriented grasping using hand preshapes and task frames. In *2007 IEEE International Conference on Robotics and Automation*, pages 1794–1799, 2007.

[74] C. Qi, H. Su, K. Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jul 2017.

[75] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation*, pages 7278–7285, 2020.

[76] N. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[77] J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation*, pages 1316–1322, 2015.

[78] Robotiq. *3-Finger Adaptive Robot Gripper Instruction Manual*, 2019.

[79] G. Rummery and M. Niranjan. *On-line Q-Learning using Connectionist Systems.* University of Cambridge, Department of Engineering, 1994.

[80] C. D. Santina, V. Arapi, G. Averta, F. Damiani, G. Fiore, A. Settimi, M. G. Catalano, D. Bacciu, A. Bicchi, and M. Bianchi. Learning from humans how to grasp: A data-driven architecture for autonomous grasping with anthropomorphic soft hands. *IEEE Robotics and Automation Letters*, pages 1533–1540, 2019.

[81] S. Schaal. *Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, 2006.

[82] P. Schmidt, N. Vahrenkamp, M. Wächter, and T. Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. *2018 IEEE International Conference on Robotics and Automation*, pages 6831–6838, May 2018.

[83] L. Shao, F. Ferreira, M. Jorda, V. Nambiar, J. Luo, E. Solowjow, J. A. Ojea, O. Khatib, and J. Bohg. Unigrasp: Learning a unified model to grasp with multifingered robotic hands. *IEEE Robotics and Automation Letters*, 5:2286–2293, 2020.

[84] S. Shaw, B. Abbatematteo, and G. D. Konidaris. Rmps for safe impedance control in contact-rich manipulation. In *2022 International Conference on Robotics and Automation*, 2022.

[85] A. Singh, J. Sha, K. Narayan, T. Achim, and P. Abbeel. Bigbird: (big) berkeley instance recognition dataset. In *2014 IEEE International Conference on Robotics and Automation*, pages 509–516, May 2014.

[86] D. Song, K. Huebner, V. Kyrki, and D. Kragic. Learning task constraints for robot grasping using graphical models. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[87] F. Song, Z. Zhao, W. Ge, W. Shang, and S. Cong. Learning optimal grasping posture of multi-fingered dexterous hands for unknown objects. In *2018 IEEE International Conference on Robotics and Biomimetics*, pages 2310–2315, Dec 2018.

[88] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005. ISBN 9780471765790.

[89] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation*, pages 13438–13444, 2021.

[90] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[91] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[92] B. Tang, M. Corsaro, G.D. Konidaris, S. Nikolaidis, and S. Tellex. Learning collaborative pushing and grasping policies in dense clutter. In *2021 IEEE International Conference on Robotics and Automation*, 2021.

[93] R. Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL `https://drake.mit.edu`.

[94] A. ten Pas and R. Platt. Using geometry to detect grasp poses in 3d point clouds. In *2015 International Symposium on Robotics Research*, 2015.

[95] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36:1455–1473, 2017.

[96] A. ten Pas, C. Keil, and R. Platt. Efficient and accurate candidate generation for grasp pose

detection in se(3). In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5725–5732, 2021.

[97] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[98] J. Varley, J. Weisz, J. Weiss, and P. Allen. Generating multi-fingered robotic grasps via deep learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4415–4420, Sept 2015.

[99] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. Shape completion enabled robotic grasping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2442–2447, 2017.

[100] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[101] J. Weisz and P. Allen. Pose error robust grasping from contact wrench space metrics. In *2012 IEEE International Conference on Robotics and Automation*, May 2012.

[102] B. Wen, W. Lian, K. Bekris, and S. Schaal. Catgrasp: Learning category-level task-relevant grasping in clutter from simulation. *arXiv preprint arXiv:2109.09163*, 2021.

[103] B. Wu, I. Akinola, and P. Allen. Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1789–1796, Nov 2019.

[104] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2019.

[105] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Proceedings of Robotics: Science and Systems*, July 2018.

[106] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. In *2021 IEEE International Conference on Robotics and Automation*, 2021.

[107] C. Yang, X. Lan, H. Zhang, and N. Zheng. Task-oriented grasping in object stacking scenes with crf-based semantic model. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6427–6434, 2019.

[108] D. Yang, T. Tosun, B. Eisner, V. Isler, and D. Lee. Robotic grasping through combined image-based grasp proposal and 3d reconstruction. In *2021 IEEE International Conference on Robotics and Automation*, pages 6350–6356, 2021.

[109] J. Zhao, D. Troniak, and O. Kroemer. Towards robotic assembly by predicting robust, precise and task-oriented grasps. In *Proceedings of the 4th Annual Conference on Robot Learning*, Nov 2020.

[110] Y. Zhou and K. Hauser. 6dof grasp planning by optimizing a deep learning scoring function. *Robotics: Science and Systems Workshop on Revisiting Contact - Turning a Problem into a Solution*, 2017.