

Robotic Grasping using Demonstration and Deep Learning

by

Victor Reyes Osorio

A thesis presented to the
University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2019

© Victor Reyes Osorio 2019

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The data collection process described in Chapter 3 and the procedure to replicate grasps described in section 5.1 had been published as an ArXiv preprint

- Rajan Iyengar, **Victor Reyes Osorio**, Presish Bhattachan, Adrian Ragobar and Bryan Tripp. A dataset of 40K naturalistic 6-degree-degree-of-freedom robotic grasp demonstrations. *arXiv preprint arXiv:1812.11683*, 2018.

For the data collection process, Rajan Iyengar developed the code for the NDI Polaris, gripper and processing the raw data to the reference frame of the workspace. I developed the code for the RGB-D cameras, integrating all the code into the data collection system, and designing the data collection procedure.

Abstract

Robotic grasping is a challenging task that has been approached in a variety of ways. Historically grasping has been approached as a control problem. If the forces between the robotic gripper and the object can be calculated and controlled accurately then grasps can be easily planned. However, these methods are difficult to extend to unknown objects or a variety of robotic grippers. Using human demonstrated grasps is another way to tackle this problem. Under this approach, a human operator guides the robot in a training phase to perform the grasping task and then the useful information from each demonstration is extracted. Unlike traditional control systems, demonstration based systems do not explicitly state what forces are necessary, and they also allow the system to learn to manipulate the robot directly. However, the major failing of this approach is the sheer amount of data that would be required to present a demonstration for a substantial portion of objects and use cases. Recently, we have seen various deep learning grasping systems that achieve impressive levels of performance. These systems learn to map perceptual features, like color images and depth maps, to gripper poses. These systems can learn complicated relationships, but still require massive amounts of data to train properly. A common way of collecting this data is to run physics based simulations based on the control schemes mentioned above, however human demonstrated grasps are still the gold standard for grasp planning.

We therefore propose a data collection system that can be used to collect a large number of human demonstrated grasps. In this system the human demonstrator holds the robotic gripper in one hand and naturally uses the gripper to perform grasps. These grasp poses are tracked fully in six dimensions and RGB-D images are collected for each grasp trial showing the object and any obstacles present during the grasp trial. Implementing this system, we collected 40K annotated grasps demonstrations. This dataset is available online.

We test a subset of these grasps for their robustness to perturbations by replicating scenes captured during data collection and using a robotic arm to replicate the grasps we collected. We find that we can replicate the scenes with low variance, which coupled with the robotic arm’s low repeatability error means that we can test a wide variety of perturbations. Our tests show that our grasps can maintain a probability of success over 90% for perturbations of up 2.5cm or 10°.

We then train a variety of neural networks to learn to map images of grasping scenes to final grasp poses. We separate the task of pose prediction into two separate networks: a network to predict the position of the gripper, and a network to predict the orientation conditioned on the output of the position network. These networks are trained to classify whether a particular position or orientation is likely to lead to a successful grasp. We also identified a strong prior in our dataset over the distribution of grasp positions and leverage this information by tasking the position network to predict corrections to this prior based on the image being presented to it. Our final network architecture, using layers from a pre-trained state of the art image classification network and residual convolution blocks, did not seem able to learn the grasping task. We observed a strong tendency for the networks to overfit, even when the networks had been heavily regularized and parameters reduced substantially. The best position network we were able to train collapses to only predicting a few possible positions, leading to the orientation network to only predict a few possible orientations as well. Limited testing on a robotic platform confirmed these findings.

Acknowledgements

I would like to thank my supervisor Bryan Tripp for his comments, mentorship and guidance over the course of my Master's. Looking back to my first day in the lab, I can see how much of a better researcher I have become due to his input.

I would also like to thank both John Zelek and Jeff Orchard for reading my thesis and providing me with feedback. Without their help, this thesis would not be what you see here. Another thanks is also due to Jeff Orchard for lending the NDI Polaris to be used in this project. Without this equipment it would not have been possible to do any of the work presented here.

I also thank my labmates for providing me with many insights, many laughs, and a lot of support throughout my Master's.

I also could not have managed to do the research here if it was not for the support of my friends and family. I owe them a huge debt of gratitude for being there when I needed them and for encouraging me to keep going when sometimes I did not think I could. Thank you.

Contents

| | |
|--|-----------|
| Author’s Declaration | ii |
| Statement of Contributions | iii |
| Abstract | iv |
| Acknowledgements | v |
| List of Figures | viii |
| List of Tables | ix |
| 1 Introduction | 1 |
| 1.1 Outline | 2 |
| 1.2 Contributions | 3 |
| 2 Background | 4 |
| 2.1 Grasping Literature | 4 |
| 2.1.1 Classical Approaches to Grasping | 5 |
| 2.1.2 Deep Learning Approaches to Grasping | 6 |
| 2.1.3 Programming by Demonstration | 9 |
| 2.2 Problem Statement | 11 |
| 3 Methodology | 12 |
| 3.1 Hardware | 12 |
| 3.1.1 Gripper | 13 |
| 3.1.2 Position Trackers | 15 |
| 3.1.3 Depth Cameras | 15 |
| 3.1.4 Robotic Arm | 17 |
| 3.2 Object Dataset & Grasping Taxonomy | 18 |
| 3.3 Data Collection Procedure | 19 |
| 4 Data Preparation | 23 |
| 4.1 Coordinate Transformations | 23 |
| 4.1.1 Palm with respect to Trackers | 24 |
| 4.1.2 Workspace with respect to Polaris | 25 |
| 4.1.3 Palm with respect to Workspace | 25 |
| 4.1.4 Camera with respect to Workspace | 25 |
| 4.1.5 Robot base with respect to Workspace | 26 |
| 4.2 Finding Final Gripper Pose | 26 |
| 4.3 Understanding the Data | 27 |

| | | |
|----------|--|-----------|
| 4.3.1 | Position Visualizations | 27 |
| 4.3.2 | Orientation Visualizations | 28 |
| 5 | Grasp Robustness | 33 |
| 5.1 | Replicating Grasp Trials | 33 |
| 5.2 | Statistically Estimating $P_s(Obj Perturbation)$ | 34 |
| 5.3 | Test Procedure | 35 |
| 5.4 | Results from Robustness Tests | 37 |
| 6 | Deep Learning Algorithm | 41 |
| 6.1 | Preparing Data and Quantizing | 42 |
| 6.1.1 | Quantizing Position | 42 |
| 6.1.2 | Quantizing Orientation | 42 |
| 6.1.3 | Preparing Images | 43 |
| 6.2 | Generating Realistic Failures | 43 |
| 6.3 | Loss Functions | 46 |
| 6.4 | Deep Learning Architectures | 47 |
| 6.4.1 | Prior Information | 47 |
| 6.4.2 | Model Fine Tuning | 48 |
| 6.4.3 | Architectures | 49 |
| 6.5 | Results | 53 |
| 7 | Conclusions | 58 |
| | Appendices | 66 |
| A | Mathematical Background | 66 |
| A.1 | Rotation Formalisms in \mathbb{R}^3 | 66 |
| A.1.1 | Rotation Matrices | 66 |
| A.1.2 | Rotation Vectors | 69 |
| A.1.3 | Rotation Quaternions | 69 |
| A.2 | Homogenous Transformations Matrices | 70 |
| A.3 | Deep Learning | 72 |
| A.3.1 | Multi-layer Perceptron | 72 |
| A.3.2 | Convolutional Neural Networks | 73 |
| A.3.3 | Residual Networks | 74 |
| A.3.4 | Other Layers | 75 |
| A.3.5 | Hyperparameters | 75 |
| B | Objects Used for Grasping | 78 |

List of Figures

| | | |
|-----|--|----|
| 3.1 | Picture of the test rig mounted in front of the UR 5 | 13 |
| 3.2 | Picture of ReFlex SF and 3D printed Shell | 14 |
| 3.3 | NDI Polaris with IR trackers | 15 |
| 3.4 | RGB-D Cameras | 16 |
| 3.5 | SR300 and Calibration pattern | 17 |
| 3.6 | UR 5 and ReFlex SF | 18 |
| 3.7 | Data Collection Physical Layout diagram | 20 |
| 4.1 | Gripper Axes | 24 |
| 4.2 | Position Histograms | 28 |
| 4.3 | Position Scatter Plots | 29 |
| 4.4 | 3D Scatter plots for pose of 3 Objects | 30 |
| 4.5 | Orientation Histograms | 31 |
| 4.6 | Orientation Scatter Plots | 32 |
| 5.1 | Probability of Grasp Success for Overhand Grasps | 38 |
| 5.2 | Probability of Grasp Success for Side Grasps | 39 |
| 6.1 | Example Results from Generating Realistic Failures | 45 |
| 6.2 | Position Architecture Diagram | 51 |
| 6.3 | Orientation Architecture Diagram | 52 |
| 6.4 | Position Network Training Curves | 53 |
| 6.5 | Orientation Network Training Curves | 54 |
| 6.6 | Position Histograms on Predicted on Training Set | 55 |
| 6.7 | Orientation Histograms on Predicted on Training Set | 56 |

List of Tables

| | | |
|-----|---|----|
| 3.1 | GRASP taxonomy grasps achievable with ReFlex SF | 19 |
| 5.1 | Quantifying Grasp Robustness for Position | 39 |
| 5.2 | Quantifying Grasp Robustness for Orientation | 40 |

Chapter 1

Introduction

A robot is a general purpose machine that can be programmed to perform a variety of tasks. However, unlike conventional tools or machines, a robot is capable of being repurposed for different tasks [50]. Robots have been used in industry for decades, performing many tasks faster and more accurately than a human being possibly could. Yet there are many more tasks that robots are not capable of doing that we would consider basic. This mismatch has to do with our ability to program these robots. Rote, repetitive tasks in a controlled environment are easy for us to program. As soon as the environment is not being controlled, the tasks become much harder to program.

Take for example the classic robotic task: pick and place. Under pick and place a robot is tasked with picking some object and accurately placing it in a different location. For a human, this task is quite easy, but for a robot a number of fundamental questions need to first be answered: where is the object? what trajectory needs to be used to approach the object? how are the joints going to be moved to fulfill this trajectory? how should the object be grasped? has the object been grasped properly? if not, how should this be fixed? etc. Even through all this uncertainty, industrial robots perform pick and place operations everyday. They rely on simplifications to their environment like making sure that the objects are always arranged in a specific order, or using strong backlights to allow computer vision systems to locate objects. However, there is still a strong desire to relax these constraints as it would make robots more useful in a wide variety of environments, like clinical or household settings. Primarily, there have been two challenges to allow robotic tasks like pick and place to be carried out in a wider range of settings: coming up with robust perception systems to locate the objects, and planning robust grasp locations on objects.

Until recently, computer vision systems were not capable of quickly and accurately identifying objects. However, in 2012 Krizhevsky et al. [30] presented AlexNet, a deep convolutional neural network trained on consumer hardware to almost match human level performance on ImageNet Large Scale Visual Recognition Challenge. The success of AlexNet spurred many other researchers to pursue deep learning to tackle many problems in computer vision. Deep learning computer vision systems now routinely outperform human performance on identification tasks.

However, deep learning has proven to not only be useful for computer vision tasks but across a wide variety of problems. Deep learning has been used to achieve state of the art performance in machine translation [4], speech synthesis [44], fraud detection [56] and of

course in robotic control [32] and grasp selection [36]. These grasps planners actually solve the perception and planning problems in one step. They are also designed to work even when they do not have access to 3D models of the objects they are trying to grasp. In fact they can plan grasp poses for objects they have never seen before.

Deep learning’s weakness is the sheer amount of data needed to train these systems. The ImageNet dataset consists of more than fourteen million images that had to be hand annotated so that a deep network could learn from them. Many of the modern deep learning grasping systems try to find ways to quickly synthesize a dataset [36], try to collect data autonomously for months [32], or train in simulation [57] before attempting their tasks.

We hypothesize that a dataset collected using human intuition might provide a stronger signal to learn how to perform grasping tasks. As stated previously, humans’ grasping ability is far greater than even the best deep learning grasping systems today. DexNet 2.0 [36], a modern deep learning grasping system achieves an 80% success rate in a controlled environment, when only one object is placed on a flat surface. Morrison et al. [41] won Amazon’s Robotic Challenge in 2017, with a grasp success rate of 63%. The task was to move a set of objects from one bin to another, a more complicated task since multiple objects were in the workspace at the same time. In both of these cases humans are not expected to face any difficulties, completing these tasks in seconds, highlighting the vast discrepancy between robotic and human grasping. It is important to note that this is not a failing on the robotic grippers available today, as during our data collection we found that our human demonstrators failed grasps only rarely, less than a hundred times in over forty thousand attempts. This is decidedly a control problem; the gripper is capable of executing the grasps but modern systems fail at controlling where to place the gripper and how to follow through. Having encountered many objects over years of experience, many of us do not even think about how to grasp new and unknown objects because we are already familiar with the constraints of the real world. This is why, we hypothesize human demonstrated grasps will provide a cleaner signal for a grasping system to learn to perform robust grasps. However, we do not want to deal with any correspondence problems between human hands and robotic grippers. Therefore, we devise a way to quickly and naturally collect grasp poses by holding the robotic gripper in one hand, and controlling the fingers using a joystick with the other hand. As mentioned previously, this data collection scheme allowed us to collect forty thousand grasps on over one hundred objects.

1.1 Outline

In the following chapters we present the work we have done on this project to collect this dataset, show its robustness and build a deep learning grasp pose planner. Appendix A covers relevant mathematical subjects that are useful for understanding this work.

In Chapter 2 we introduce our problem, and present a literature review of classical, deep learning and demonstration based approaches to grasping.

Chapter 3 we introduce all the hardware we used for this project: the gripper, robotic arm, camera and position trackers. We also go over the taxonomy we used to organize our grasps before delving in detail on how we conducted our human demonstrated data collection.

Chapter 4 we go over how we processed the raw data we collected into a useful dataset for

grasp pose training. Predominantly, we talk about how to perform reference frame changes from each frame in our setup and how to find the final grasp pose from our dataset of trajectories. We also show relevant plots to provide some intuition as to what the data we collected looks like.

After having collected the dataset, we empirically show that the grasps we collected are robust to perturbations in Chapter 5. We go into detail about how we replicated grasps so that they could be replayed on our robotic arm and how we collected data on the perturbed grasps. We then show results from these experiments.

Having shown that our dataset is robust to perturbations, we then explain our approaches to train a deep learning grasp planner in Chapter 6. First we describe preprocessing steps we took to prepare the data for the particular approach we settled on. We then go into some detail about what kinds of architectures, loss functions and hyperparameters we used for our training.

Finally, we provide concluding remarks in Chapter 7.

1.2 Contributions

We provide the following lists of contributions:

- A system that can be used to record reaching and grasping data in real time using a human operator's intuition to guide a robotic gripper.
- Empirical evidence for the robustness of the collected human demonstrated dataset to perturbations of up 2.5cm in position and 10° in orientation.
- An approach for how to train a deep network grasp planning system using the collected dataset, and intuitions on how to improve this approach.

Chapter 2

Background

Robotic grasping is a complicated task that has been an area of research for decades. Recently deep learning methods for grasping have become a popular area of research. Deep learning, however, depends on having large amounts of high quality data, but such a dataset does not yet exist. Large datasets of grasping data created from simulation do exist, but these suffer from not having realistic enough physics [36, 28]. DexNet 2.0, one of these systems, has an 80% chance of grasp success, while we expect humans to have perfect grasping ability under the same circumstances. Reinforcement learning approaches to grasping trained in the real world can produce very large datasets [32], over 800k grasps, but their rate of success is also around 80%. One possible reason for this is that these datasets do not have high quality labels. Human demonstrated grasps are considered the gold standard [28] for a grasping system, but in the past it has been labor intensive to collect enough human demonstrated grasps to train a deep learning model. In this work, we demonstrate how to collect such a dataset of high quality human demonstrated grasps in quantities large enough to train a deep learning model. This system is described in Chapter 3, while in this chapter we present a literature review of robotic grasping as seen through the lens of classical, deep learning, demonstration approaches to the field. Each of these approaches treats the grasping problem, quite differently, but they are all interrelated and frequently build on each other's work. For reference, relevant mathematical topics are covered in Appendix A. In the second section of this chapter we introduce our problem statement and demonstrate how we believe this project helps to address some of the shortcomings of modern approaches to grasping.

2.1 Grasping Literature

Robotic grasping is not a new discipline, but the modern approaches to robotic grasping are now quite divorced from the classical approaches. The oldest approaches (1980's - 1990's) are characterized by their reliance on calculating force closure characteristics [15, 40, 42, 21]. More modern approaches (2000's) [49] begin to lean on machine learning but still rely on hand crafted features. By contrast deep learning based grasping systems are more concerned with finding the appropriate representation [9] for the grasp pose and letting the system find its own features. Approaches where humans demonstrate grasps [23, 13, 20, 24] for data collection are not as common, presumably because of how time intensive collecting the data

can be. In the next few sections we present some of the work in these areas and comment on how they relate to our project.

2.1.1 Classical Approaches to Grasping

Perhaps the most intuitive approach to robotic grasping is to calculate exactly what are the forces and torques necessary to secure an object to a robotic gripper. Hanafusa and Asada [21] provide some of the earliest research into calculating stable grasps with robotic grippers. They propose a general definition for what constitutes a stable grasp: a grasp is stable if for any small perturbation the contact forces from the gripper provide a restorative force on the object towards the unperturbed state. This practical definition is easy to understand, but hard to quantify. Mishra et al. [40] use this definition to mathematically show that such stable grasps exist even in a friction free situation, and can even compute how many fingers are needed to execute these grasps on basic polygons. Knowing that such grasps exist, Ferrari and Canny [15] introduced a grasp quality metric that tries to encapsulate the stable grasp criteria given by Hanafusa. This quality metric came to be known as ϵ -metric, and is available in most robotics simulators to test for force closure.

Ferrari works in wrench space [5], where the force, \vec{F} , and torque, $\vec{\tau}$, acting on a body are represented as a six dimensional vector, $\vec{w} = [\vec{F} \ \vec{\tau}]^T$, called a wrench. The vector space of wrenches then represents all the possible force and torque combinations. Ferrari generalizes the forces the gripper is capable of exerting as a vector \vec{g} , which carries information about the normal forces each of the fingers apply to the object’s surfaces. From these two vectors, they posit a predicate, $\mathcal{A} : \mathcal{W} \times \mathcal{G} \rightarrow \{T, F\}$, which maps whether a given set of contact forces, $\vec{g} \in \mathcal{G}$, can resist a wrench, $\vec{w} \in \mathcal{W}$. They then propose the following grasp quality criteria, Q :

$$Q = \min_{\vec{w}} \max_{\vec{g} \in \vec{w}A} \frac{\|\vec{w}\|}{\|\vec{g}\|} \tag{2.1}$$

In this equation $\vec{w}A$ is used to represent the space of all contact forces that can resist the wrench \vec{w} .

This metric can be understood as the ratio of a wrench applied to an object being held in place by the fingers’ normal forces. Ideally we want $\|\vec{g}\|$, the contact forces, to be small but able to resist some amount of force/torque, $\|\vec{w}\|$. By maximizing the ratio over \vec{g} , we encourage minimizing the contact forces. However, by minimizing that over \vec{w} , we are searching for the smallest wrench that requires the most contact forces to overcome. In other words, this metric allows us to test for the worst case scenario. Ideally we want this metric to be as high as possible, since that implies that large magnitude wrenches are required to loosen the object, or that very small contact forces can be used to hold the object. Ferrari and Canny also work out a geometric way to calculate this constraint as the radius of the largest sphere that fits within the convex hull of the possible contact forces for a given wrench.

One of the prerequisites for this kinds of force calculations is access to accurate 3D models of the objects to be grasped. This condition is too restrictive for a general purpose grasping system, as it implies that it must know every single object that it might interact with. Saxena et al. [49] tackle this problem by building a system that does not require or attempt to build

a 3D model to grasp the object, thus showing that is possible to learn generic features that useful for grasping.

They approach the problem by training a logistic regression to recognize useful grasp points in an image. To do this they synthetically generated a set of 2500 examples from five object classes. Since these were synthetic, the grasp points were computed as the data set was made. They then manually engineered features that they believe would benefit their task. To decide if a given point is a good grasp point, they would crop a small rectangle around the point in question and apply their engineered features. This new feature vector would then be fed into a logistic regression algorithm that would classify the point as a good grasp point (1) or a bad grasp point (0). To actually execute the grasps they needed to know the 3D location of the grasp point, so they actually use multiple images of the same object from different views and use their feature vectors to try and identify the same grasp point from the different views. If successful they then triangulate the full 3D position of the grasp point. Since they have more information with two images they also combine the two sets of grasp point predictions using MAP to pick the best based on their observations. By training their classifier on tableware, they were able to demonstrate their systems ability to empty a dishwasher. This system still does not use deep learning, but it does start to lean more on machine learning techniques to identify useful features, and importantly showed that it was possible to learn grasping features.

All of the approaches described so far have been useful and have applications outside of research. However, they all suffer from the same key failure: it is very difficult to extrapolate to new and unseen objects. The force closure approaches have provided us with many of the analytical tools that are used in computer simulations, but as a general grasping system they fail because they depend on accurate 3D models of their objects, which will not be available a priori. Saxena’s work, though more general, still depended on hand coded feature vectors over a relatively small synthetic dataset. These feature vectors might fail to represent some object features which might be easily picked up by deep learning systems.

2.1.2 Deep Learning Approaches to Grasping

Deep learning approaches to grasping are relatively new, but they are now a very active area of research. This section will present the works of Mahler et al. [39, 36, 37, 38], Levine et al. [32], and Kappler et al. [28]. There are many other active researchers in the area, so we recommend Caldera et al. [9] who have compiled a review of deep learning approaches applied to grasping.

DexNet

The DexNet project [39, 36, 37, 38] is one of the better known deep learning grasping systems. Mahler et al. have published multiple articles describing this project. DexNet 1.0 [39] uses a variety of techniques to try to achieve force closure around an object. First an image of the object to be grasped is taken, and a set of grasps are sampled using the work of Smith et al. [51]. The algorithm has to decide from these grasps which is the one that is going to be attempted. It does this by treating this as a multi-armed bandit problem: from the sampled grasps (arms) it has to pick the one with the highest reward (probability of

success). Multi-View-CNN is used to build a feature vector that can be used to quickly find a similar object in DexNet 1.0’s database of over ten thousand 3D models (collected from academic datasets). Probability of success for the sampled grasps are then computed on the most similar objects in the database. The grasp with the highest chance of success is then sampled and it is computationally evaluated for force closure against the object placed in a pose that is slightly perturbed from the original. The probability of success for that grasp is then updated, that is the multi-armed bandit model updates its expected reward. This is repeated a set number of times before the model makes a decision. As DexNet 1.0 sees more data it can make better predictions about what kinds of grasps will be successful.

DexNet 2.0 [36] takes a different approach to grasp planning. Here, the robotic arm with the parallel jaw gripper is paired with a depth sensing camera capable of capturing point cloud data. Instead of using a multi-armed bandit to calculate probability of success, a grasp quality convolutional neural network (GQ-CNN) is trained to predict probability of success. The training set for this network is constructed by using traditional grasp quality metrics (like force closure) on a set of 1,500 3D object models. That is to say that the training set is bootstrapped by using other grasping methods that exploit the information from the 3D object models. The inputs to the GQ-CNN consist of a depth map and a corresponding point cloud. The depth map is aligned so that the potential grasp location is centered on the image and the parallel jaws come in from the sides. This makes actual representation of the grasp four dimensional: 2D object position of the center pixel (corresponding to position on a flat table), orientation of the gripper (taken from the rotation applied to align the image), and the distance of the gripper from the object. As the name implies the GQ-CNN does not try to calculate gripper poses but only evaluates them. From a sampled set of grasps, DexNet 2.0 always picks the one with highest score given by the GQ-CNN. DexNet 2.0, has a success rate of 80% for objects it has never seen before.

DexNet 3.0 [37] extends the same ideas to suction grippers. Suction grippers present a different challenge since they require relatively planar surfaces and are pressed directly onto the object. DexNet 4.0 [38] combines DexNet 2.0 and 3.0 in an ambidextrous robot in such a way that DexNet 4.0 can choose which of the two algorithms (arms) to use to maximize its probability of success. DexNet 4.0 also contributes a method to better transition performance of a robotic grasping system from simulation to real world grasps.

The DexNet project as a whole solves some of the problems present in early robotic grasping research. By using computer simulations and force closure calculations DexNet can learn to find its own features in the RGB-D input it is given. However, there is still room for improvement. Many of the grasping simulators available today do not handle friction in a realistic way. These unrealistic interactions are then carried into the data that DexNet uses to train. Furthermore, Kappler et al.[28] show that grasping ground truth based on human rated grasps outperform many of the force closure approaches used to train DexNet. This points to the need for collecting real world grasping data by demonstration, since it would be representative of real friction and would be based on human rated grasps.

Leveraging Big Data for Grasp Planning

We start by looking at the work of Kappler et al. [28]. In their article, they proposed a new kind of database of objects and grasp poses that could be used to train machine

learning models to predict whether or not a grasp was likely to succeed. They sought to answer two questions: what grasp quality metrics are useful for grasp planning, and how can deep learning be used for grasp planning. Kappler presented three different grasp quality metrics: ϵ -metric [15], physics based simulations, and finally crowdsource labels (human rated grasps). The use of human rated grasps was motivated by the need to have a gold standard rating for which to compare against. The physics based simulation involved trying a grasp on the object 3D model and then running a physics simulation, without gravity, only on that grasp. To arrive at a probability of grasp success they would slightly perturb the pose of the object and run the simulation again. They considered a grasp successful if over 90% of the perturbations resulted in success, while grasps with less than 10% chance of success are considered failures. Their human judging was done by leveraging a mechanical turk service. The tasks commonly seen in these services are things that are easy for humans to achieve, but hard to program. Kappler set up their task to judge whether images of grasps were stable or unstable. They made sure to include some pre-labeled grasps as a test so that they could filter out people spamming or clearly not understanding the task. Like with the physics based simulation, they considered a grasp a success if over 90% of the human judges rated the grasp as stable, while less than 10% would still count as failure.

They trained two models, a logistic classifier and a convolutional neural network. They used grasp heightmaps [24] (see section 2.1.3) as the input to both models. Kappler found that the CNN outperforms the logistic classifier as the number of object classes increases. Furthermore, they found that the models trained with the physics based labels consistently outperform the ones trained with the ϵ -metric labels. This shows us the need for labels derived from more realistic sources. Kappler et al. conclude that labels from physics based simulations are good enough to bootstrap learning. However, using human judges to build a gold standard grasping dataset, we believe, might lead to even better labels. The work of Levine et al. [32] is useful for understanding why we believe that what we need to collect is human demonstrated grasps, rather than just grasps under more realistic scenarios.

Learning Hand-Eye Coordination

Levine et al. [32] investigated learning hand-eye coordination on a robotic platform. In this context, hand-eye coordination refers to the ability of a system to be able to direct its end-effector based on visual input to accomplish a task, which in this case is a grasping task. They did so by training a convolutional neural network to predict which commands would lead the robotic system to a successful grasp, what they call visual servoing. The system itself would not be aware of the position of the robotic arm or the camera with respect to the workspace. The CNN would have to learn to associate the gripper and objects in the image with each other, but also with the possible commands that can be given to robotic system. The algorithm itself is split into two parts: a network that predicts the probability of a successful grasp given the current image and a randomly sampled command, and a separate algorithm that samples potential commands from the network and decides which to follow. One of the key aspects of this work is that it was carried out using multiple robotic systems simultaneously, each with different wear and tear and with slightly different camera positions, yet the system was trained using data from across all the robots. This serves to highlight the robustness of the system to different camera positions and emphasizes that the

CNN must have learned some representation that allowed it to identify the gripper and its location in space.

Their data collection functioned in the real world, but required at times eighteen robotic arms and took over three months. Of particular note, is that even though the training took place in the real world (as opposed to a simulation), the success rate for this system is still around 80%, about the same as DexNet. So Levine had both collected a large amount of data (over 800k grasps) and done so under real world conditions. This hints towards a need for higher quality labels, which Kappler et al. [28], believe can come from human rated grasps.

2.1.3 Programming by Demonstration

Having established the need for higher quality, human demonstrated grasps, we need to find a way to record these grasps. Programming robots by demonstration is yet another way to tackle the grasping problem. The main hallmark of this research is to figure out a way to transfer a human provided demonstration to a robot so that it can be executed. Argall et al. [2] provide a survey on using demonstration for robotic tasks. We recommend reading their survey for a more in depth look. Presented here are the works of Herzog et al. [24] and Granville and Fagg [13].

Learning of Grasp Selection Based on Shape-Templates

Herzog et al. [24] proposed learning to grasp by learning shape templates. Their learning of these templates was bootstrapped by a human demonstrator kinesthetically manipulating the robot to the desired position to execute a grasp. They describe their shape template as localized heightmap. A depth sensor looks out over the scene (a flat tabletop) and identifies the object to be grasped. To compute the grasp heightmap, they find a tangential plane to the object, and then calculate the distance from the points in the object point cloud to the tangent plane. If more than one point lands in the same bin on the tangent plane, the greatest distance between the two points is kept. The size of these grasp heightmaps (and therefore how big the tangent plane needs to be) is determined based on the robotic gripper being used. Their system would get the point cloud for an object, and begin to calculate grasp heightmaps. Each heightmap is compared against a database of grasps, and when a close match is found the robot attempts the pose associated with the matched template.

To bootstrap their database of grasps they have a human demonstrator physically move the robot's gripper to an area where a successful grasp can occur. The system then finds the closest tangent plane to the object point cloud, calculates and stores the heightmap and the gripper pose. Using only eighteen demonstrations they show their system is able to grasp a wide variety of objects because the grasp heightmaps encode object properties that generalize to other objects. Of interest is the way they extend their database of grasps by recording only heightmap-grasp combinations that fail. Their rationale is that their system's assumption is that if two grasp heightmaps are similar, then the same gripper pose should succeed for both heightmaps. However, if the system concluded two grasp heightmaps were similar, but then failed to grasp the object, then one of two things must be true: the demonstrated template is false, or the heightmaps were not similar to begin with. Since the demonstrated templates

are unlikely to be false, it must be the case that the heightmaps are not similar, so by adding it to the database, anything that matches to it can be ignored. The same logic does not extend to positive examples: two heightmaps that are different might still admit the same or different grasp poses. They use the example of a cup being face up as opposed to face down. From the top the heightmaps would look different but they both admit the same overhand grasp. They conclude that it might be possible to add more positive examples autonomously but care must be taken to not overwrite previous negative examples.

As a whole this work is impressive for introducing the concept of a grasp heightmap. Their ability to learn to avoid failing grasps based on experience is a very useful ability that not many other grasp systems possess. However, since the human demonstrator must move the robotic arm manually for every demonstration, the amount of time and effort required to collect training data for this system is quite high. The fact that using only eighteen demonstrations is enough to grasp a wide variety of objects, points to the idea that higher quality labels paired with a good representation is a good strategy to pursue for building a grasping system. In our case, by setting up an appropriate deep learning system we should be able to learn a good representation. To collect the human demonstrated grasps, we want to avoid having the human demonstrator move the robot's arm. Our goal is to have the grasping action happen as naturally and as efficiently as possible.

Learning Grasp Affordances through Demonstration

Granville and Fagg [13] collected data on grasp affordances using a human demonstrator performing natural grasps. This involved the human demonstrator wearing a special glove that was used to track the pose of the hand (position and orientation) with respect to the object at 15Hz. To make sure that the poses remained consistent with respect to some object frame, a tracker was also attached to the object that was being manipulated. Over a series of trials, the human demonstrator would proceed to grasp the object at any locations that felt natural. For example, a hammer might be grasped by any position along the handle, a spray bottle by the neck, a heat gun by the handle. Having collected a large series of possible grasp poses over the objects, they devised a compact representation for these by approximating the distribution of points using expectation maximization. They find good representations for their poses, but Granville does not test these on a robotic platform.

This work shows that it is possible to collect human demonstrated grasps at a reasonable speed, and that those demonstrations encode some generalizable information that might be used in other systems. However, this work suffers from the correspondence problem: robotic grippers are generally not articulated like human hands. This means that even though the position information might be valuable, exploiting the individual finger positions recorded here might not be possible, depending on the robotic gripper used. In our work, we instead directly use the robotic gripper, and have our operators manipulate the gripper to execute robust grasps.

2.2 Problem Statement

At this point, force closure approaches for grasping seem to have reached a plateau. More accurate physics based simulations can be made, but these systems tend to require 3D models of the objects being grasped which are not always available. In fact we know that such models are not required, since Saxena [49] showed that it was possible to train machine learning models in the absence of such 3D object models. Currently, deep learning seems to be the most promising path towards building general purpose grasping systems. DexNet 2.0 claims to have an 80% success rate [36] when shown objects it has not seen before. Levine et al. [32] showed that deep learning can learn to pick up arbitrary objects out of a bin when given sufficient data, also with roughly an 80% success rate on their own benchmark. We know that these robots are capable of moving with millimeter precision, which means that the problem is not having the ability to place the gripper, but knowing where to place the gripper. Since Levine et al. collected around 800k grasp examples, we also know that quantity of data might not be enough. The problem seems to be how to collect enough high quality training data.

Kappler et al. [28], showed that human rated grasps could be used to provide high quality labels for a deep learning system. The work of Granville and Fagg [13] showed that it is possible to collect grasping pose data in real time, but Granville opted to track a human hand directly. This introduces a correspondence problem: what is the best way to map the human hand, to a robotic gripper with fewer degrees of articulation, and often a very different shape. Herzog et al. [24] avoid this problem by having the human demonstrator move the robot manually to the graspable points of an object. However, this can again be time consuming.

We hypothesize that using human demonstrated grasps, collected with the robotic gripper itself, would allow us to have both high quality labels, and avoid the correspondence problem, and that in turn will lead to better deep learning models. In the following chapter, we show how we used a series of position trackers and 3D cameras, to be able to collect human demonstrated grasps at a natural pace for the human operators, and how we avoided the correspondence problem by having the human operators hold the robotic gripper directly.

Chapter 3

Methodology

The main thrust of the work is based on the idea that humans already possess a deep well of intuitive grasping knowledge. Humans routinely use visual perception to plan and execute grasps. By collecting data on how a human operator would execute a grasp using a particular robotic gripper we hope to capture this intuitive knowledge in our dataset. Then that dataset can be used to train a neural network algorithm to execute grasps on new objects. There are then two questions that we seek to answer: how to better collect human inspired grasps, and how to use them to train a neural network to execute novel grasps.

To answer these questions we put together a system that allowed us to quickly and accurately collect grasping data. For each grasp attempt, we collect multiple color images of the object, depth maps from the same cameras, and the final pose of the human operated robotic gripper. Once data is collected, we can then test the robustness of the dataset by replicating the same human grasps on a robotic arm, and also train a neural network to map images/depth maps to gripper poses.

In this chapter we will go over the methods and equipment that were used during the data collection portion of this project. First we will discuss the hardware that was used to collect and test the human demonstrated grasping data. The setup used involves a series of 3D cameras, position trackers, and robotic equipment. Afterwards we introduce the object dataset we collected grasping data on, the grasping taxonomy and the overall data collection procedure.

3.1 Hardware

To effectively collect grasping data based on human demonstration a collection of hardware was needed. Most of the hardware that we used could be bought off-the-shelf, but we wrote the software used to put all the aspects together. Since the goal was to collect human inspired grasping data using a robotic gripper, one of the first steps was to pick an appropriate gripper and design a human friendly way to control and handle the gripper. Capturing the pose of the gripper would be the next step. To capture the pose of the gripper we used the NDI Polaris along with two IR tracking crosses attached to the gripper. Along with capturing the gripper pose, we also require an image of the object we are attempting to grasp. These images were taken using two RGB-D cameras from two different perspectives. Finally, to

test the robustness of the collected data, and to test any new grasping algorithm we need to use a robotic arm that can be coupled with our gripper.

To support all of this hardware, we custom built a table out of 80/20 aluminum square tubing. This table was designed such that the cameras could be positioned, and adjusted to view the grasping area, and to allow for moving the whole table from the robotic arm stand to a more ergonomic position for data collection.

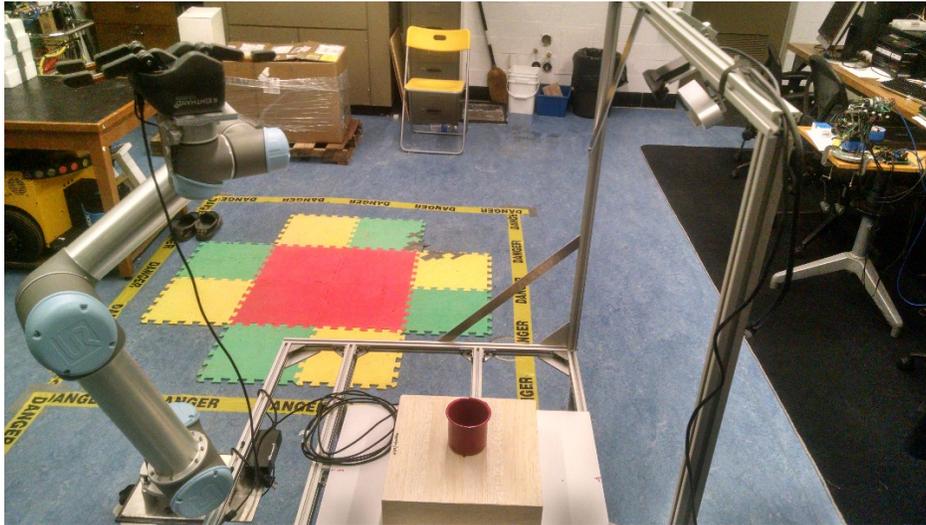


Figure 3.1: A picture of the the table and some of the hardware. Pictured here is the ReFlex SF mounted on the UR 5 robotic arm. A red cup from the YCB object dataset [10] is placed on the workspace wooden table. The ZED RGB-D camera is placed on the upper cross bar across from the robot. Not pictured here is the SR300 RGB-D camera, which is placed on one of the vertical aluminum bars across from the robot about a halfway up.

3.1.1 Gripper

We decided to use the ReFlex SF gripper from RightHand Robotics [47]. This gripper design is characterized by the use of the 3 under actuated fingers (two parallel, one opposing), along with four degrees of freedom, packed in a 3D printed shell. The four degrees are comprised of being able to rotate each finger individually, and additionally having the ability to scissor the two parallel fingers. The under actuated fingers make the gripper exhibit useful compliant properties. As the fingers close, the variable stiffness of the rubber along the fingers leads to a natural looking bend in each finger. If any opposing force is encountered the finger complies with the force and can wrap around the object causing the force.

Since the shell is 3D printed, we could print a new shell that allowed us to attach a simple handle, and tie points for the IR trackers used by the NDI Polaris. These tie points were chosen such that one of the trackers should always be visible for most poses we expected to do with the gripper.

Controlling the gripper’s fingers was done by use of a joystick. For the purposes of the grasping task, the three fingers were moved together in sync, reducing three degrees of

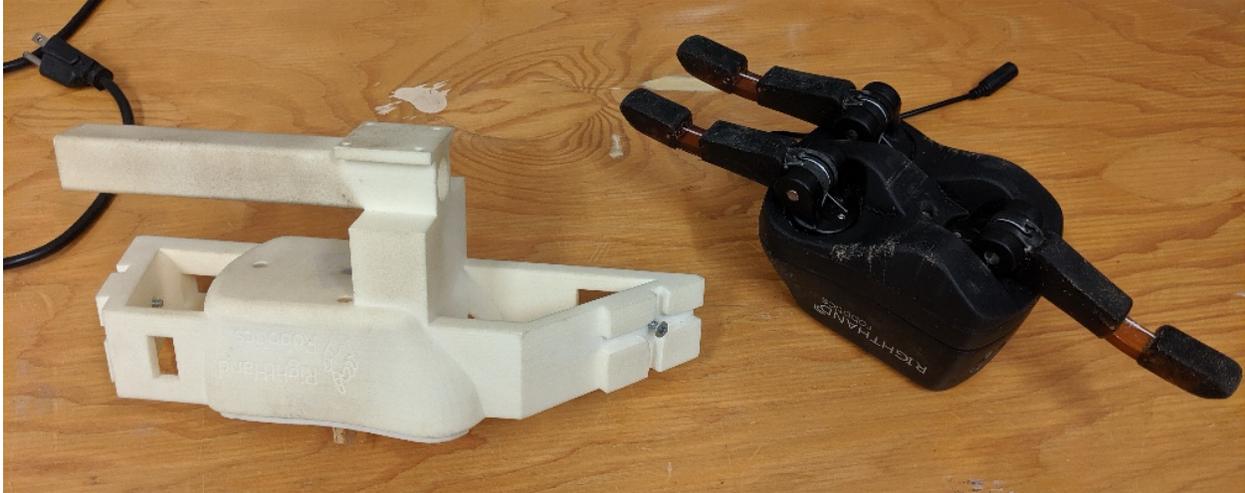


Figure 3.2: A picture of the shell (left) and the ReFlex SF (right). On the shell the visible screw is one of the locations where one of the trackers would be placed. The other location is on the shell under the handle, where another screw is visible.

freedom to one. The up-down axis on the joystick was used to control the three fingers, essentially controlling the finger opening. In the neutral position, the fingers are halfway between open and closed. Moving the joystick down opens the fingers, while up closes them. The left-right axis of the joystick was used to control the scissor degree of freedom. The neutral position keeps the fingers parallel, moving the joystick to the right forces the fingers to close, while moving the joystick to the left forces them to open. Since the ReFlex SF does not have absolute encoders for any of the motors, we calibrated the motors after every power-cycle. The calibrated position is fingers parallel, and open such that the fingers are flush with the palm. The finger position for each grasp is also recorded as a value relative to the calibrated motor position.

There were a couple of faults that we found with the ReFlex SF when we executed the data collection portion of the project. The fingers are connected to the motors with a length of nylon string, which over many cycles starts to wear out and will eventually snap. Though inconvenient, the repair is relative simple and can be done without any special tools. Frequently checking the state of the nylong strings was done to prevent a grasp failing due to unexpected string failure. The second fault has to do with the gear system used to transfer torque to the fingers scissor action. The gears themselves are 3D printed, and over time, and under stress, the teeth on the gears start to pulverize. The dust can become caught in between other teeth leading to motor shutdown (self-protection mechanism). In some cases the teeth snap and the fingers become loose and the scissor degree of freedom becomes unusable. The repair for this fault is more involved, but is also relatively easy to execute, provided that fresh gears are available. Unlike the nylong string, it proved to be too time consuming to check the gear before every data collection session, leading to some sessions ending early. We admit that we did push the limits the gripper's intended duty cycle. RightHand Robotics has addressed many of these issues in some of their newer grippers.

3.1.2 Position Trackers

To track the pose of the gripper we used the North Digital Imaging (NDI) Polaris. The Polaris consists of an IR source and two IR sensitive cameras set some known distance apart. By using special IR trackers, consisting of four IR reflective orbs, it is possible to track the full 6D pose of an object with respect to the Polaris' frame so long as the tracker is rigidly attached to the object. By using homogenous transformation matrices we can then obtain the pose with respect to any other frame we have measured.

The Polaris itself is capable of millimeter resolution in position and sub-degree resolution in orientation. It is also capable of recording data in real-time at greater than 20Hz. For each grasping trial we collect data not just on the final gripper pose, but also on the full trajectory towards the object. This data is not used in this project, though it is available.



Figure 3.3: NDI Polaris and 3D printed gripper shell with IR tracker. The Polaris has two cameras placed at either end of plastic shell. Around each camera there are a series of IR emitters that are used to illuminate the IR trackers.

3.1.3 Depth Cameras

To provide visual input to our neural network model, we used two RGB-D cameras. The RGB data is routinely used in CNNs used for grasping and object localization. Depth is becoming more common in grasping tasks, since it can be used to get local curvature of an object. The two cameras we used were the Intel RealSense SR300, and the StereoLabs ZED.

The SR300 is a combination of a high definition RGB camera along with an IR depth camera [25]. The depth camera works by shining structured IR light using a built-in emitter and then capturing the resulting patterns using an IR camera. Each camera can work independently of the other, though for most purposes both cameras run simultaneously. Since there are two distinct cameras, the images need to be rectified so that pixels in one image correspond to pixels in the other image. This operation can be automatically done by the software tools that come with the camera. Another important difference between the



Figure 3.4: Early version of the mounting mechanism for the RGB-D cameras. Above: SR300, which after some tests we found would be better placed closer to the workspace. Below: ZED camera.

cameras is that the depth camera only records at standard VGA resolution (640×480), while the RGB camera can record from VGA up to HD resolution (1920×1080). We record the images at HD resolution with the RGB camera, but when running any tests of algorithms we crop and scale so that the color image matches the depth image. The SR300 has an effective depth window from 20cm to 150cm, which is why it is placed closer to the objects across from the gripper operator. The center of the workspace is about 50cm away from the camera, with the closest spot being 20cm away and the farthest spot being roughly 70cm away. Importantly, depth is returned from the camera as a 16bit integer (max value of 65535), which makes the depth maps unsuitable to be saved in common image files (jpg, png, bmp), since those formats only have an 8bit channel depth. As a result the RGB-D images are stored as binary arrays on disk so as to not lose any resolution. Finally, it is important to realize that both the NDI Polaris and the SR300 use IR light, which means that as a result it is not possible to use both simultaneously. When we recorded images during the data collection process, the Polaris' emitter would be turned off while recording the RGB-D image of the object, and then the SR300's emitter would be turned off to allow the Polaris to function.

For the project presented here, we did not use the images captured by the ZED. We present the following information for completeness. The ZED camera achieves depth through a different means than the SR300. The ZED is a pair of RGB cameras positioned a set distance apart [54]. StereoLabs runs a proprietary algorithm on the two color images that come out of the cameras to produce a depth map. It is safe to assume that this algorithm works by doing some sort of feature matching across the two images and then by knowing the pixel distance between a particular feature in an image, and the distance between the cameras it is possible to calculate the distance to that feature. Both cameras can operate from WVGA (672×376) to a 2.2k (2208×1242) resolution. We found that the camera itself was very power hungry when operating at the max resolution which would lead to

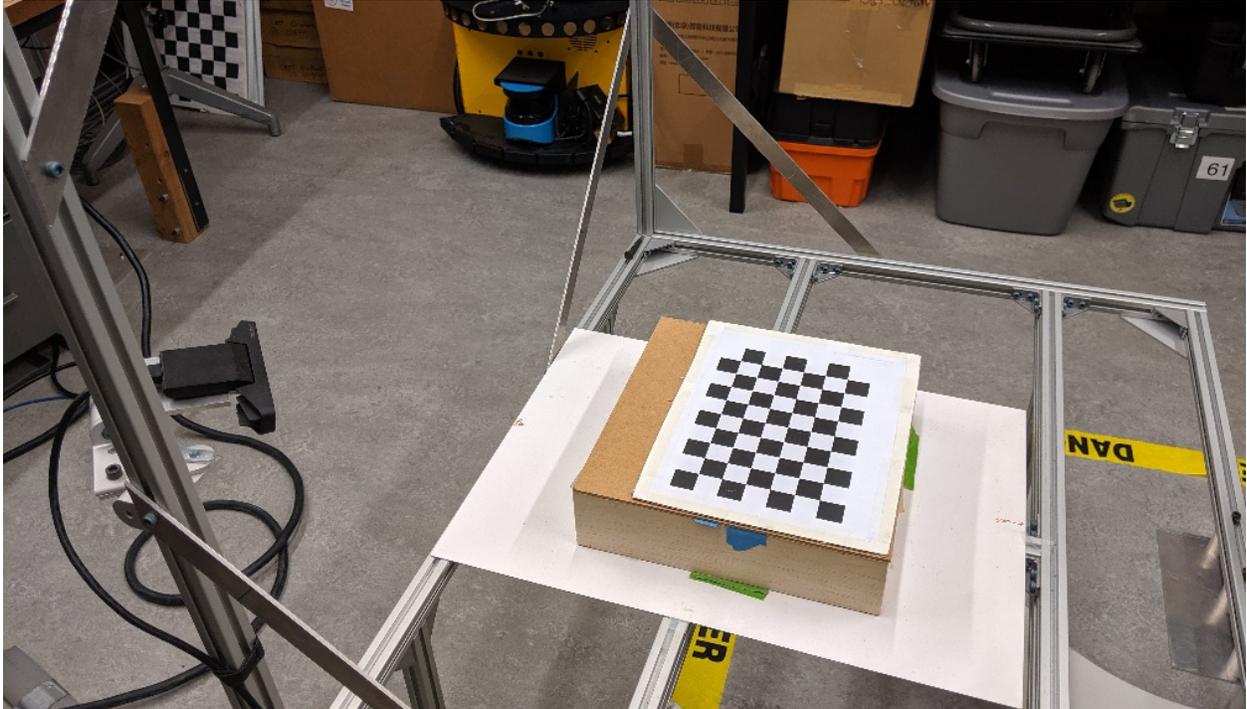


Figure 3.5: The final mounting position for the SR300 and the checkerboard pattern used to calibrate the position of the camera with respect to the workspace. The SR300 position is much closer to the workspace since it has a tighter range for depth sensing.

frequent shutdowns, therefore we recorded images at HD resolution. The effective depth range for the ZED is between 0.5m and 20m, where the distance between the cameras is the major constraint setting the range. Since the bottom of the range is so much higher than for the SR300, the ZED is placed in an overhead configuration, with the top of the workspace starting about 45cm from the ZED and the bottom of the workspace being about 80cm from the ZED. The ZED returns two images for each frame: a joint RGB image with both the left and right images, and a depth map. Again, the depth map is provided in a format that is not suitable for storage in traditional image formats, so it is stored as a binary array on disk.

3.1.4 Robotic Arm

To test the recorded grasps and the new grasping neural network we used Universal Robotics' UR 5. The UR 5 is a six degree of freedom, 850mm long, robotic arm that is designed to be used in situations that might see humans frequently entering its work space [48]. The arm itself is capable of achieving a repeatability of roughly 1mm, and can be programmed using a variety of interfaces. For quick testing and debugging, it is possible to use a tablet that is attached to the control box and to operate, program and manipulate the robot. For more complicated situations, it is possible to send commands to the control box the UR 5 is attached to from any computer with an ethernet connection. When performing any tests we use the second interface to automate all the movements of the robot. For the commands



Figure 3.6: The UR 5 robotic arm. The six degrees of freedom can be seen in this image. Each joint can move $\pm 360^\circ$ around its neutral position, allowing for a wide range of poses. Mounted on the UR 5 is the ReFlex SF gripper through an adapter plate.

themselves, after transforming any coordinates to the frame of the base of the robot using homogenous transformation matrices, we call on Universal Robots’ provided software for inverse kinematics. As a result, we generally have to be mindful of any robot motion as the inverse kinematics functions will sometimes return trajectories that seem more drastic than are required, or that might put the gripper in danger of collision with arm.

3.2 Object Dataset & Grasping Taxonomy

The set of possible grasps for a given object is dictated by not just the object but also by the gripper. As a result, we need to identify not just what objects are graspable, but also what kinds of grasps are possible with the ReFlex SF. We would also like to relate the grasps that we are attempting to realistic human grasps. Feix et al. [14] analyzed the state of grasp taxonomies in 2015 and provided a taxonomy of their own, GRASP, that encompasses the major grasps present in their review. Thirty-three different single hand holds were identified in their study. The conditions for a grasp to make it into their taxonomy were given as ensuring the object and hand remain rigidly attached no matter the pose of the hand (force closure). This means that grasps have to be robust to changes in applied force, predominantly gravity, and that only grasps performed with one hand would be considered. Feix provides a many different labels to categorize the thirty-three grasps presented, but we found that the most useful labels were to identify whether a grasp was a power or precision grasp. The definition of each can vary, but a power grasps predominantly involve the use of the palm as an opposing or virtual finger and can therefore exert more pressure on the object being grasped. A precision grasp only involves the finger and is associated more with fine motor control tasks, like writing.

| GRASP number | GRASP name |
|--------------|-------------------------|
| 3 | Medium Wrap |
| 12 | Precision Disk |
| 6,7,8 | Prismatic 4/3/2 Fingers |
| 11 | Power Sphere |
| 22 | Parallel Extension |
| 1 | Large Diameter |
| 28 | Sphere 3 Finger |
| 2 | Small Diameter |
| 13 | Precision Sphere |
| 10 | Power Disk |

Table 3.1: This table shows the GRASP [14] taxonomy numbers and names for the grasps we identified as being possible with the ReFlex SF. The order of the grasps corresponds to the frequency of use for each grasp observed by Feix et al. from footage of housekeepers and machinists.

Many of the grasps are not possible to execute with the ReFlex SF due to limitations of both the gripper itself, but also our chosen control scheme for the gripper (i.e. using the joystick). Out of the thirty-three grasps, we identified only twelve (see table 3.1) that we believe could be executed with our gripper. Feix also provided data on the frequency of each grasp from analyzing footage of housekeepers and machinists. Based on their table, we find that the twelve identified grasps cover 40% of the grasps used by these workers. During our data collection procedure, we note the GRASP number for the grasp we attempt on each object.

The grasp choice is not only influenced by the gripper, but also the object that is being grasped. The YCB object dataset [10] is an attempt to standardize the set of objects available to researchers to perform grasping studies. The objects in the dataset have been chosen to represent a wide range of objects across different domains. The main subject categories are: food items, kitchen items, tool items, shape items, and task items. Not all the YCB objects seemed graspable, given our constraints. Examples of objects in the YCB dataset we did not feel could be grasped with the ReFlex SF according to Feix’s criteria include: magazines, washers, dice, credit card, screws, cordless drill, and others. Many of these are simply too small to reliably grasp with the configuration we had set for the ReFlex SF, while others are too heavy to allow the kind of repetition we needed from human operators. To supplement the YCB dataset we collected common objects from our lab. This included items like: disposable coffee cups, water bottle, glue stick, flashlights, stapler, etc. The list of objects used is available in Appendix B. In total, 109 unique objects were used, with some objects having multiple possible grasps that were explored.

3.3 Data Collection Procedure

We projected that we would require roughly fifty thousand training examples to training a grasping neural network. Since we want to collect human inspired grasping data, we devised



Figure 3.7: Diagram showing a sketch of the physical layout of the space during data collection along with the coordinate axes of the Polaris (green rectangle), the workspace (blue square) and the operator (red circle). The Polaris line of sight is along the $-Z$ axes, which is why the reference frame is oriented the way it is. All reference frames are right handed, so the direction of the third axis can be inferred from the two shown. The RGB-D camera's position is also shown. It was located on a post looking out over the workspace.

a data collection procedure that we could use to minimize human effort and error, but also to avoid the human operators from performing the exact same grasp multiple times in a row. The data collection itself was as much a logistical problem as an engineering problem. To prevent repetitive stress injuries, a number of volunteers were recruited to spread out the number of trials each person would be conducting. For each of these volunteers an hour and half orientation session was made for them to familiarize themselves with the equipment and the way the ReFlex SF would control with the joystick. This orientation session would involve explaining the goals of the research and showing example grasps, followed by getting the volunteers to complete a set of dexterity based tasks with the gripper: unstacking and restacking cups, picking up small objects and depositing them in a container, and building towers out of cups.

Every data collection session was done by a pair of people so that one person could focus on grasping and one person could focus on setting up trials. An individual grasping trial

would proceed as follows. One person, the setter, would be in charge of setting the object in the middle of the workspace and making sure that the workspace was configured correctly. The other person, the operator of the gripper, would wait until the first person was done before pressing the start trial button on the joystick. When the button is pressed, the two RGB-D cameras record images and tag them with a unique ID to store on disk. Then the SR300 would be commanded to shut down its IR emitter and the Polaris would start up its own IR emitter. At that point, a prompt would show up on a screen across from the operator telling them that the trial had successfully started. The setter would then monitor a different screen with pose information being recorded by the Polaris for any blind spots or other irregularities. Once the operator had grasped the object, they would lift the object at least ten centimeters to demonstrate a solid grasp. At that point, the successful grasp button on the joystick would be pressed and the information from the Polaris would be tagged with the same unique ID as the images and saved. The operator would then release the object in the hands of the setter to set the next trial. If the setter notices any irregularities in the images, the pose information, or the operator does not successfully grasp the object, then the failure trial button would be pressed. The images and pose trajectory information would still be saved under a unique ID, but the trial would also be tagged as a failure. The average time to complete a trial was around six to seven seconds.

For each object/grasp combination three hundred grasps would be collected. The object was always placed standing up, where up was defined prior to the session starting, and in the center of the workspace. For each trial the object could be randomly rotated about the center of the workspace. The first hundred grasps would be collected with only the object in the workspace, but for the second set of one hundred grasps, an obstacle would be introduced to prevent the exact same grasp from being used multiple times in a row. The most commonly kind of obstacle was to place a tall, thin object between the operator and the object in question to force the operator to grasp the object by approaching from the sides instead of head on. For the last hundred grasps, a new obstacle would be introduced, again to prevent all the grasps from being too similar. The most common kind of second obstacle to be introduced would be any object that would block approaching from the operator's right as most operators were right-handed. The location of the obstacles described here merely represent the most common kinds of obstacles used. Many different placement and orientations were used to set up obstacles. The only constraint was that the obstacle must not occlude the object being grasped.

In a data collection session, only two to three objects would be grasped, around nine hundred grasps per session. Before each session we would identify the objects to be grasped and the GRASP number for the grasps that would be attempted. At the beginning of the session we would use the Polaris' pen tool tracker to get the exact position of the center of the workspace with respect to the Polaris. This would allow us to later build the homogenous transformation matrix that would take us from the Polaris reference frame to the workspace frame. During the session, the operator and setter would switch places every fifty grasps, and take a longer break at the end of every object (every three hundred grasps). Each session usually lasted two to three hours depending on how many breaks were taken and whether or not any problems arose during the data collection session.

It is interesting to note that most volunteers expressed concern about being able to grasp some of the objects before doing their orientation session. However, after only an hour most

people felt comfortable doing relatively dextrous and complicated tasks using the gripper and joystick. It was also interesting to witness human operators adjusting to the object being grasped. Generally any dropped object trials would only happen in the first ten to fifteen trials, if at all, for that particular operator, and afterwards the rate of failures due to dropping would fall to zero. We believe this speaks to the ability of human operators to leverage their experience to quickly adapt to the new situations being shown to them and formulate robust grasp planning under these circumstances.

In this chapter, we showed that it was possible to construct a system and procedure to allow for the efficient collection of human demonstrated grasps while also avoiding the correspondence problem. We not only collected final grasp poses, but also the trajectory of each individual grasp attempt over time. The dataset is available online at <https://dataverse.scholarsportal.info/dataset.xhtml?persistentId=doi:10.5683/SP2/1XRF9U>. In the next chapter we provide our data processing procedure for taking the raw data collected gaining insights from it. In chapter 5 we show that the data we collected is robust to perturbations, while in chapter 6 we go over how we trained our neural networks using this data.

Chapter 4

Data Preparation

After collecting data for three months, we ended data collection with 40,150 individual grasping trials over 109 objects. Due to the nature of our data collection, and the complexity of our setup, we reasonably expected some of the data to not be useable. In total we ended up with about 35,000 grasping trials, about 87% of the total amount of collected data. About half of the bad data was due to one of the SR300 or the Polaris not saving the data properly. Without both an RGB-D image and the final gripper pose, the trial is not useful for training a neural network. The other half of the bad data was discarded due to the gripper pose being non-sensical, most often with the pose being reported as being outside of the workspace we operated in. The two main reasons we expect these kinds of errors might have happened is due to IR interference on the Polaris (i.e. strong sunlight being visible even after covering windows), and also the times we might have operated the gripper into a blind spot for the Polaris.

Regardless, 35,000 trials should be enough to train a neural network model as a proof of concept. Over the next couple of sections, we will go over the different data preparation steps we took to build our final training set. First, we discuss all the different coordinate transformations that we require to shift the frames of reference to the ones that are convenient for this study. Then we describe the procedure we used to take the gripper trajectory and extract the final gripper pose at the time of the grasp. Finally, we present some figures to give insight into what information was captured through the data collection.

4.1 Coordinate Transformations

In appendix section A.2 we go over how to use homogenous transformation matrices to represent reference frames and to change the frames vectors are being described with respect to. In this section we show how we used these tools to calculate the pose of the gripper with respect to the center of the workspace and with respect to the SR300.

The Polaris calculates the pose of the IR trackers with respect to its own reference frame. This means that the raw data given from the Polaris can be used to construct $H_{Polaris}^{T1}$ and $H_{Polaris}^{T2}$. These are the matrices that describe the pose of tracker 1 (T1) and tracker 2 (T2) with respect to the pose of the Polaris. Alternatively, they are the matrices that can be used to describe a vector in the Polaris reference frame in the reference frame of one of the

trackers. Since the trackers move in space relative to the Polaris, both of these matrices are time dependent.

4.1.1 Palm with respect to Trackers

Since we are interested in the pose of the gripper, and not the trackers, we need to define the reference frame of the gripper and find the transformation matrix that will take us from the trackers to it. We define the reference frame of the gripper in the standard for robotics: the z-axis points out of the palm, the y-axis is perpendicular to it and parallel to the fingers, and the x-axis is chosen to make a right handed coordinate system. To find the center of the palm we used the 3D printed shell that the trackers would be attached to. Without mounting the gripper in the shell, we place one of the trackers in its final position, T_1 or T_2 . The other tracker is mounted in a special aluminum jig that when mounted to the shell places the tracker at the center of where the palm will be. We then use the Polaris to measure coordinates of both trackers. This means that we will end up with the following matrices: $H_{Polaris}^{palm}$, $H_{Polaris}^{T_n}$ (where n is either 1 or 2 depending on the tracker). Which means that we can then calculate:

$$\begin{aligned} H_{T_1}^{palm} &= H_{Polaris}^{palm} H_{T_1}^{Polaris} \\ H_{T_2}^{palm} &= H_{Polaris}^{palm} H_{T_2}^{Polaris} \end{aligned} \tag{4.1}$$



Figure 4.1: Figure showing the orientation of the reference frame centered on the palm of the ReFlex SF. The location of the origin is flush with the closest plane that can be placed on the gripper’s pads, and not with the center screw (1cm under the plane).

For each of the two matrices we will need to swap the placement of the trackers such that one of the trackers is in its final position while the other is on the palm position. Since the

gripper will be rigidly attached to the shell, these relationships will not depend on time. At the end of this process we will have the following constant matrices: H_{T1}^{palm} , H_{T2}^{palm} .

4.1.2 Workspace with respect to Polaris

Now we need to define the relationship between the reference frames of the Polaris and the workspace. We constructed the reference frame of the workspace such that the z-axis is pointing up, the x-axis is parallel with the z-axis of the Polaris, and the y-axis is chosen to form a right handed coordinate system. (See Figure 3.7)

We leveled the workspace and the Polaris to same orientation, which restrains two of the orientation degrees of freedom such that some of the axes are pointing along the same line. The third degree of freedom we restrain by making sure that both the workspace and the Polaris are square with respect to the same edge of the table. This alignment makes computing the orientation between the two frames simple. We just need to apply a rotation to align the z-axis (-90 degrees about the y-axis of the workspace on the Polaris frame). To find the translation we use the measurements we took before each data collection using the Polaris' pen tool. These measurements give us the location of the origin of the workspace relative to the origin of the Polaris. We found that these measurements could change as much as 5mm over the course of 24 hours even though both the Polaris and the aluminium rig were clamped to the same table. We never managed to identify the source of this shift. In our analysis we correct for the daily variation of the center, but assume that within the same data collection session the center of the workspace with respect to the Polaris does not appreciably shift. Using the previous measurements we can then construct $H_{Polaris}^{ws}$.

4.1.3 Palm with respect to Workspace

Using the above results, we can now find the pose of the gripper with respect to the center of the workspace.

$$H_{ws}^{palm}(t) = H_{ws}^{Polaris} H_{Polaris}^T(t) H_T^{palm} \quad (4.2)$$

Where in equation 4.2 T could be either $T1$ or $T2$. In practice, if we can see both trackers we take the pose to be the average of the two recorded poses, otherwise we just use whichever tracker happens to be visible.

4.1.4 Camera with respect to Workspace

We also want to be able to obtain coordinates with respect to the SR300 so that we can better train a neural network. To do so we leverage the OpenCV library [8], to handle finding intrinsic and extrinsic camera parameters so that we can track a checkered target (see Figure 3.5). We can place this target in our workspace, taking care to note the location of the origin of the reference frame of the target pattern with respect to the origin of the workspace. Since we can track the pose of the target, we can build H_{cam}^{ws} . Aside from using this transformation matrix during the data processing step, we also used it to ensure that the camera was still in the same place relative to the workspace after moving the aluminum table from the data collection setup to the testing setup.

4.1.5 Robot base with respect to Workspace

Finally, we need to find the pose of the workspace with respect to the reference frame at the base of the UR 5, since the robotic arm accepts commands in that reference frame by default. We easily find the pose by setting the robot to compliant mode and moving the end effector to the center of the workspace. We then line up the x of the end effector with the corresponding axis of the workspace. Note that since both the workspace and the end effector use right handed coordinate systems, and in this configuration their z-axis are anti-parallel, that it is only possible to line up either x or y, and not both. We then read the position of the workspace from the robot’s interface, and we calculate the rotation matrix required to line up the orientation of the reference frames. Using this information we build H_{ws}^{robot} , which we can use to convert poses to the default frame the robotic arm uses.

4.2 Finding Final Gripper Pose

It is important to understand that our data collection method yields gripper trajectories: the set of poses from the beginning of the trial until one of the end of trial buttons is pressed. During data collection we did not have a way of defining when the actual grasp happens, since we need to test that the grasp is successful. As a result we needed to come up with some algorithm to find the most likely time during the trajectory that the grasp occurs. We tried the following heuristics: find the time of the lowest height above the workspace, find the time of the last minimum in height above the workspace, find the time at which the gripper was closest to the origin of the workspace. We found that the third method seemed to work the best, but we will go over the rationale for all three.

To start, for all three of these methods we need the grasp trajectories to be in workspace coordinates. We can use the results from section 4.1 to achieve this. The first method came about when we did a small scale test before starting the full data collection regimen. For this test we used two objects: one that used an overhead precision grasp, and one that used power grasp from the side. In this small set, we found that plotting the z axis of the trajectories showed a high point at the beginning, a low point close to the end and then sharp up tick at the end. These three phases correspond, to the start of the trial (gripper held roughly chest height), the grasp point, and the grasping test. However, what we found was that during the course of the data collection, operators would not always start with the gripper at chest height. Due to the weight of the gripper and its bigger shell, the number of sheer repetitions, and the total time it took to perform one data collection session, many of the operators would place the gripper in various different poses to ease strain on their muscles. This meant that for many trials, the gripper starts below, or close to, the height of the workspace. For these trajectories, the first algorithm would return many poses outside of the workspace, or poses where the orientation was not even pointing towards the object. The solution was to find the last local minimum in the z axis trajectory. This meant that we would avoid the beginning of the trial, or any other strange dips that might have occurred during data collection and only keep the most likely dip that was due to the grasp.

However, after some thinking we settled on using the final approach: taking the time when the gripper is closest to the origin of the workspace. Since the objects are always

placed at the origin of the workspace, this method works not only for overhead grasps, but also side grasps. It also works no matter where the operator had placed the gripper at the beginning of the trial. It is also more robust since we depend not just on the z coordinate of the gripper, but also the x and y. Using the other two methods we found that we were throwing away a large number of trials since they seemed to be grasps that had not occurred in the workspace at all. With this final method we recovered many of those trials and were able to use them for further analysis.

4.3 Understanding the Data

Now that we have described the bulk of the data preparation, we dedicate some space to present some visualizations from the data to give a better sense of what was captured. Unless specified all the poses will be with respect to workspace reference frame, whose origin is where we placed all the objects to be grasped. We present the position and orientation data separately.

4.3.1 Position Visualizations

First we show histograms of the position variables in Figure 4.2. As a reminder, the reference frame for the workspace is laid out in such a way that positive x corresponds to the operator’s right, and positive y corresponds to the direction directly away from the operator. With this in mind, we see that there is a slight bias in the top histogram of Figure 4.2 for positive x. If there was no bias, we would expect the mean of the distribution to be centered at 0, but instead we see a mean of (5.1 ± 0.1) mm. We believe this is due to most of our gripper operators being right handed (one person out of five was left handed). This bias would obviously affect any models build using data. There is also a bias in the negative y direction. This is the side of the object that is closer to the gripper operator. Unless an obstacle was placed specifically to prevent it, most operators would grasp around the closest point they felt they could reasonably apply a successful grasp. Again this bias is worth keeping in mind for any systems or models built using this data. That is, the data is not position agnostic; just like most humans there is a preference for a particular side (right) and a particular location (closer to body).

Figure 4.3 shows all the pairwise scatter plots of the position variables, along with a 3D scatter plot of all three variables together. In the top left we have a view as would be observed looking down on the workspace from above. The main ball in the middle corresponds to a large portion of the grasps, while the bar below is a much lower proportion. That bar represents some of the side grasps that were done on the side of the workspace closest to the operator. There were much fewer examples of the same on the far side of the workspace; this position was uncomfortable to use the gripper given the handle that was designed and the overall weight of the gripper. Bottom left is the view as would be seen from the right of the workspace. In this view it is easier to see the bias towards grasps closer to the gripper operator. A small blob also becomes visible above most of the grasps. This corresponds to the few thin, tall objects we grasped, around twenty centimeters or more in height and less than five centimeters across. The top right view is the view as would be seen from the

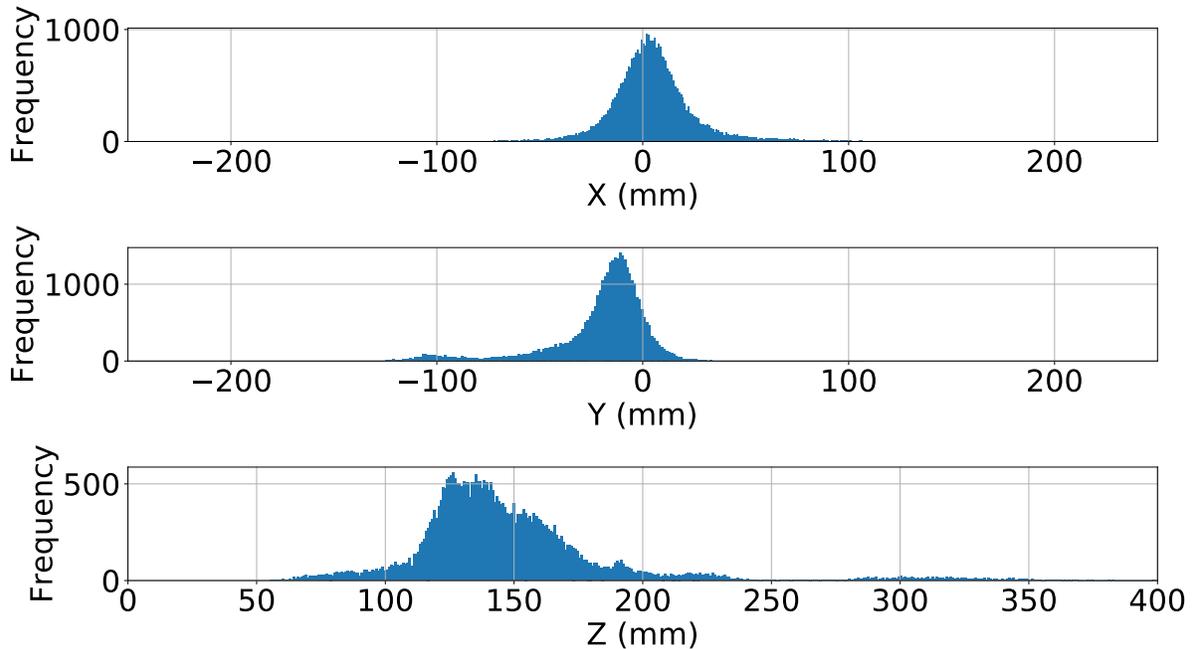


Figure 4.2: Histograms of the three position variables in workspace coordinates. All measurements are in millimeters, with the size of the workspace being $200\text{mm} \times 200\text{mm} \times 400\text{mm}$

operator’s side of the table. Here we again see the small blob above the larger proportion of grasps, and it is also more obvious that there is a right bias to the grasps (this can also be noticed from the top left but it is harder to pick out).

Figure 4.4’s left column shows position scatter plots for three different objects. We can see that the distribution of positions is conditional on the object identity. Object A is a blue plastic cup from a children’s toy set. This object is predominantly grabbed from the top, and we see that the scatter plot is fairly flat in the z axis, at around 150mm. On the x and y axis the position is biased left (-x) and towards the operator. The left bias is due to this object being grasped by the left handed operator. Object C is a Rubiks cube, this object was also grasped from the top, but it was about a centimeter shorter than the cup. We again see a flat distribution on the z axis, but this time it is centered around 130mm. The distribution along the x and y axes is more uniform with a right hand bias and bias towards the operator. Even though these objects were grasped similarly, their poses differ as a result of their own geometry, which is to be expected. Object B was a tall water bottle. For this object two sets of grasps were attempted, from the top and from the side. This is clearly seen in the position scatter plot as two different clusters of positions.

4.3.2 Orientation Visualizations

For the orientation degrees of freedom we have a lot more freedom to define which representation we want to use for visualization. We settled on using the Euler angles corresponding to pitch-yaw-roll (x-y-z) in the workspace coordinates. This extrinsic set of Euler angles

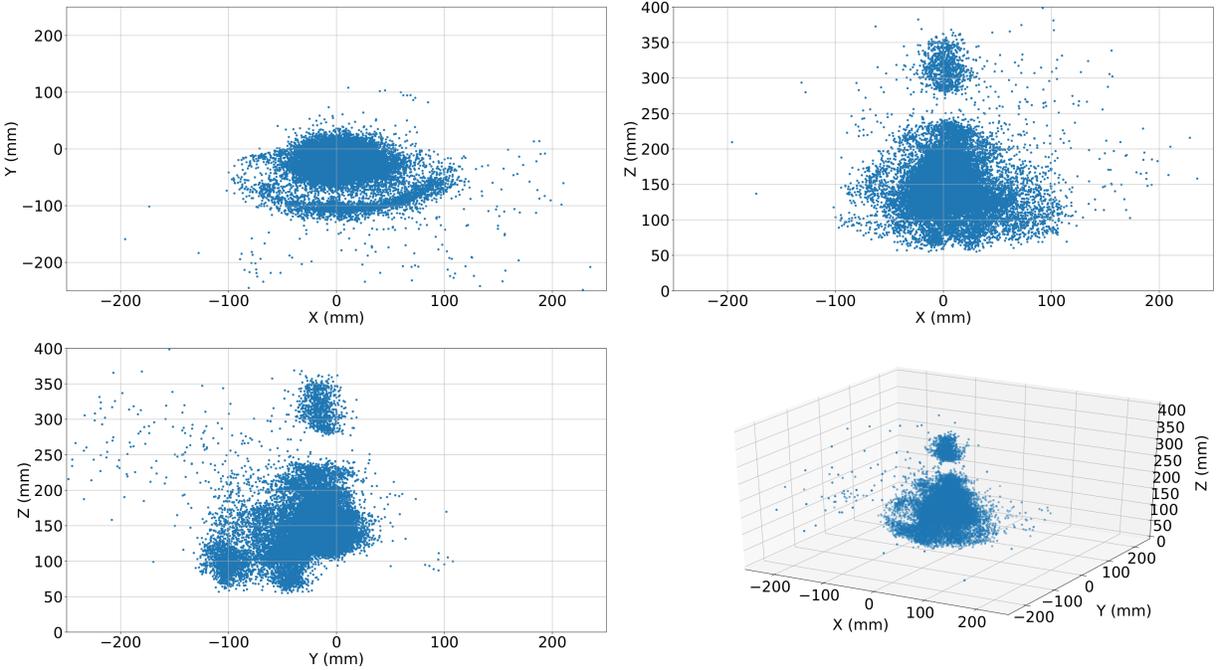


Figure 4.3: Scatter plots of the different combinations of position variables. Bottom right: A 3D scatter plot of all the position variables

is useful because it is easier to physically interpret what the angles correspond to in the physical world. The gripper’s starting position is with its axes (Figure 4.1) aligned with the workspace. To arrive at the final gripper orientation, first we apply a rotation about the workspace’s x axis, then we apply a rotation about the workspace’s y axis, and then a final rotation about the workspace’s z axis. The rotations are extrinsic, so they are easier to visualize since the rotation axis stay fixed as the gripper rotates. Other Euler angles could be chosen, but we found these to be the most convenient. If another set of Euler angles is chosen that would change the relationship between the different variables even though they would still be representing the same orientation.

Figure 4.5 shows the histograms for pitch-yaw-roll of the gripper with respect to the workspace coordinates. It is important to note that since angles are periodic, the peaks seen in the pitch histogram actually correspond to only one peak. Similarly, the roll histogram has three peaks, one of them being obscured by the fact that the histogram axis is not periodic. The pitch histogram being centered around π radians is consistent with the fact that the z-axis of the workspace points up, while the gripper needs to point down for most grasps; a rotation of π radians. The yaw histogram shows slight symmetry about the zero radians line. The first peak from the zero point corresponds to slight variations from having the gripper pointing straight down along the axis of the gripper’s fingers. The second peak (around 1.5 radians) corresponds to all the grasps we did from side (as opposed to overhand). It also highlights that most of the grasps we could execute with the gripper were overhand grasps even when we did not limit ourselves to only those grasps. There is some slight assymetry about the zero line, biased towards negative yaw. This is again the bias for right

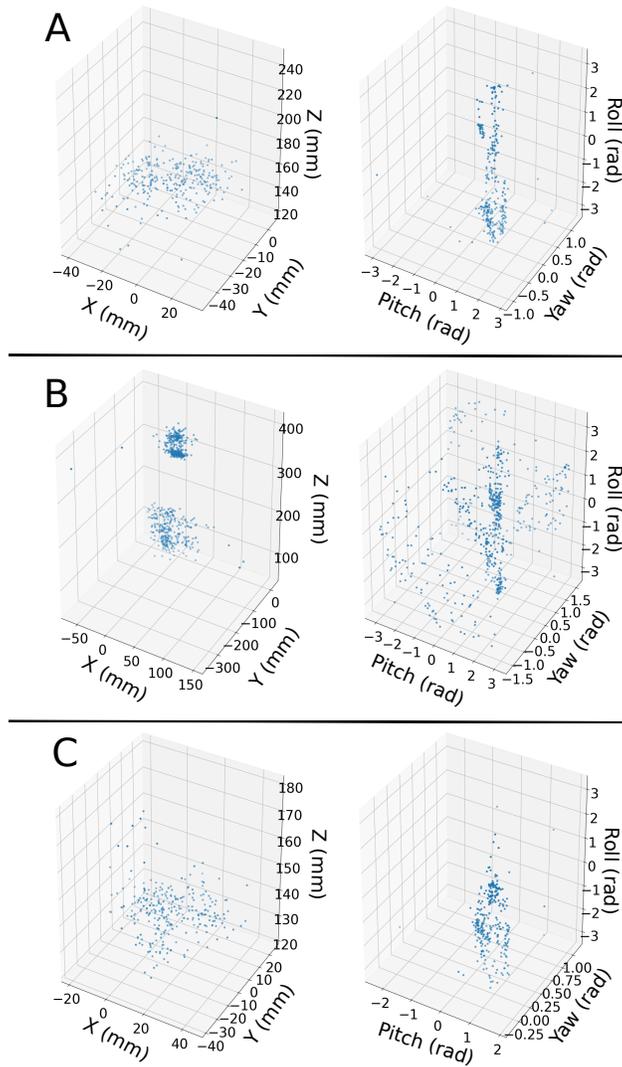


Figure 4.4: 3D scatter plots for the position and orientation of three different objects. The left column corresponds to the position variables, while the right is the orientaiton variables. Notice how both the position and orientation distributions change when conditioned on the object. **A**: Large Blue Cup. **B**: Water Botle. **C**: Rubiks Cube.

handed operators that we saw in the previous section. Finally for the roll histogram we see a pretty consistent run from about 2 radians, wrapping around to -1.5 radians, with a clear dip around zero radians. The run corresponds to the rotational freedom we experienced with most objects when performing grasps overhead. We could roll the gripper, sometimes freely for objects with cylindrical symmetry, and still perform a successful grasp. The bias away from the zero radian position has to do, again, with how uncomfortable such grasps would be. These would generally involve putting the gripper such that the two parallel fingers are pointing away from the operator, but the gripper would be lined up with the operator’s torso. The only way to realize that grasp would be to step aside, increasing the distance to the object and thus making the grasps harder, or to place the elbow pointing into the operator’s torso, with the forearm pointing straight ahead. So again as expected we see that our data collection managed to collect many of the problems and solutions faced by real grasp systems (i.e. humans) when operating in the real world.

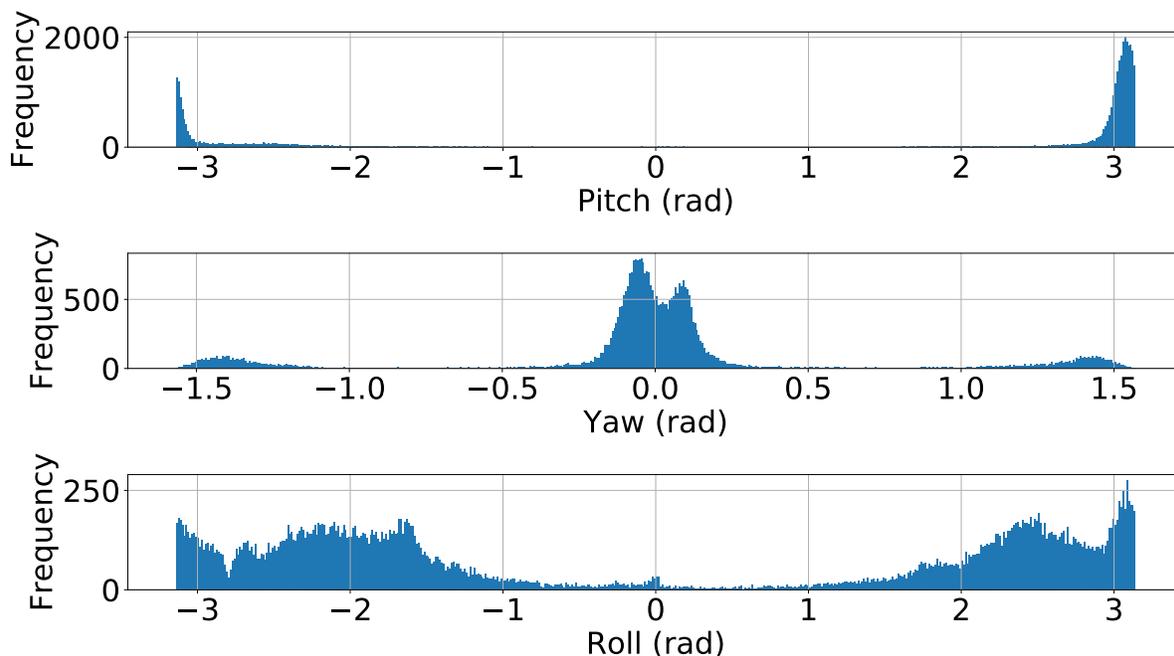


Figure 4.5: Histograms of the three orientation variables as the Euler angles corresponding to pitch-yaw-roll in workspace coordinates. All measurements are in radians.

As mentioned above, Figure 4.6 shows scatter plots of the pairwise combinations of pitch, yaw and roll. For these scatter plots, we slid the pitch and roll variables to place their peaks at the zero position. This operation does not break or generate any patterns that could be seen in the data, but does mean that the axis for the pitch and roll variables are off by an additive factor of $-\pi$. It is important to note that for all the scatter plots there seems to be more outliers. In this case by outliers we mean points that do not seem to line up with the main features we see in the data. The data is dominated by a cylinder like structured lined up with the roll variable. This is again due to the amount of freedom we had when collecting data to vary the angle about this axis. The column’s cross section extends some

distance from the center of the column. These combinations of pitch and yaw correspond to orientations that keep the gripper pointing relatively down and towards the objects in the center of the workspace. There are also thin wall-like features around yaw 1.5 and -1.5. These walls represent the planes that have combinations of variable consistent with side grasps. There are other faint patterns in the histogram, but they do not seem to be strong trends. These might be due to one or two objects needing some special combination of angles or an operator trying to collect more esoteric data.

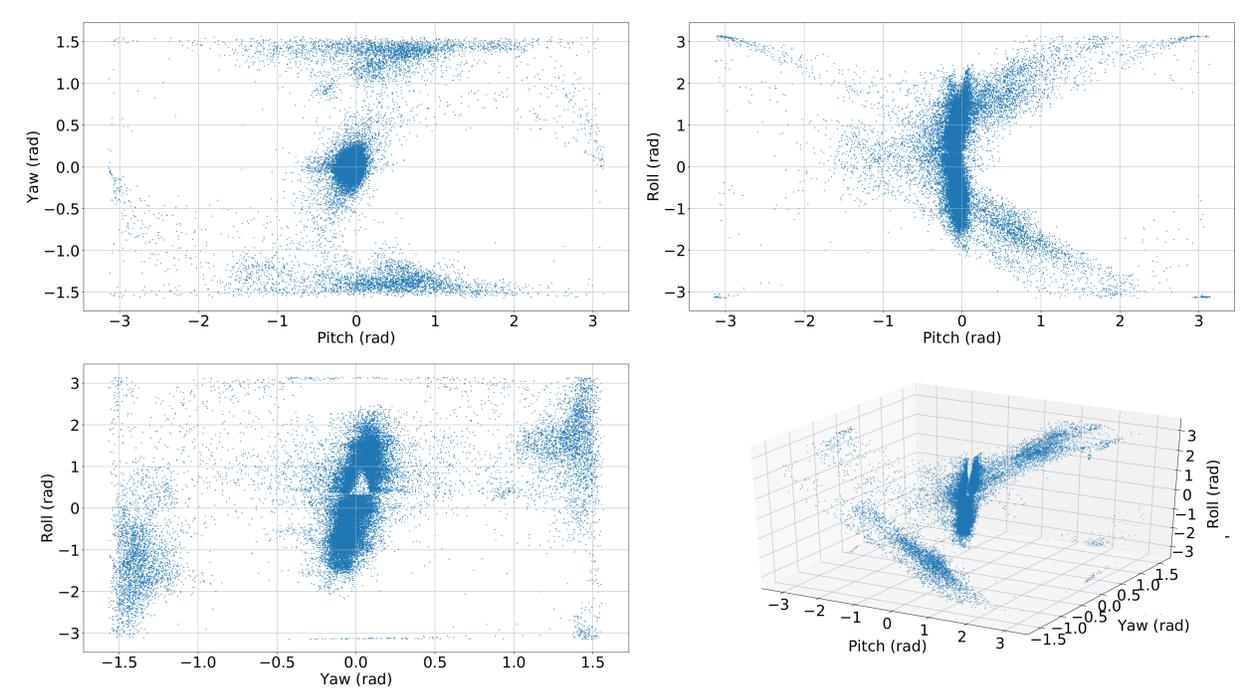


Figure 4.6: Scatter plots of the different combinations of orientation variables using Euler angles pitch-yaw-roll. Bottom right: A 3D scatter plot of all the orientation variables

Figure 4.4’s right column shows orientation scatter plots for the same three objects discussed previously. Again object A and C show similarities in their orientation scatter plots. This is to be expected as the orientation for the gripper for these grasps needs to be similar given how similar the object’s grasps are. Object B, the water bottle, shows a more complicated set of orientations. We can still recognize the same cylinder running along the roll axis that we see for the other two objects. These grasps correspond to the overhand grasps used for the water bottle. However we also see other orientations present in the scatter plot. These represent the side grasps that were also recorded for this object. Once again we see that the recorded poses are conditional on the objects themselves.

Seeing that the insights we can draw from the dataset seem to be consistent with our expectations, we move in the next chapter to showing that our grasps are robust to perturbations. We expect that if human demonstrations produces high quality grasping labels, that the grasps should be robust to some perturbations. Being able to resist perturbations is a useful property since errors can creep in from anywhere in the system, errors in object pose being the most likely to occur.

Chapter 5

Grasp Robustness

We hypothesize that by leveraging human intuition, the grasping dataset we have collected is robust to various kinds of perturbations. This is a desired property for any grasp; the more robust a grasp is the more likely the grasp is to succeed in the real world where conditions can change relatively quickly, or information might not be the most precise. To test the robustness of our grasps we seek to replicate the conditions during a grasp trial, and then perturb the final pose of the gripper in each of the six dimensions (three for position and three for orientation). We achieve this by using an image overlap method: our dataset image overlapping the live view of the camera. Aligning the two images should yield conditions relatively close to the original condition. To quantify how robust our grasps are we seek to find the probability of a successful grasp given a particular object and perturbation. We estimate this function by taking each perturbation test as Bernoulli trial, and bounding our confidence on the probability of a successful grasp. We find that our grasps are very robust to different kinds of perturbations.

5.1 Replicating Grasp Trials

The first step towards conducting a test of grasp of robustness is to be able to replicate the conditions during a trial as precisely as possible. This means that we need to replicate the location of the object with respect to the workspace accurately. The approach we take is to manually align the object using the image captured during data collection. This is done by displaying the image from the data collection and a live stream from the SR300 over each other at half intensity each. The object can then be manually aligned by having the live stream match the data collection image as closely as possible. However this depends on the alignment of the camera with respect to the workspace.

As mentioned in section 4.1.4, we can align the SR300 to the workspace with a decent amount of precision: about 0.1mm in each position axis, and 0.001 rad in each orientation axis. This means that in the worst case scenario we would be accumulating on the order of 0.2mm in the position axis and 0.002 rad in the orientation axis. We find this amount of error to be acceptable. With the camera aligned properly, we can use the superimpose image method described above.

We would like to also quantify the error we introduce by superimposing the images. This

is hard to measure, but we can get an estimate by getting multiple people to try to align the same image and measuring the difference between the two attempts. The way we choose to do this is to place a sheet of paper on the table in the workspace, and tracing the outline of the object after making an attempt to align the object. We can then look at the outlines of multiple attempts and measure the difference between the two. There are only three degrees of freedom when placing an object on a place: the x,y position and the orientation of the object. After measuring alignment across three different trials using this method we found on average a difference of 2mm in the position variables from trial to trial and a negligible (less than 2°) amount in the orientation variable. We found that in practice getting the alignment correct depended heavily on the object. Objects that have unique features, like visible handles or text, were much easier to align in both position and orientation. We believe that the error we accumulate by using this image superimposing approach is about 3mm in the position and on the order of 0.02 rad in the orientation.

The final tool needed to replicate a grasp trial is the UR 5. The UR 5 has a repeatability of 0.1mm [48]. Universal Robots does not provide the value for the repeatability in the orientation variables. Overall, this means that most of the error from trying to replicate a grasp comes from doing the image superimpose method. If we add all the sources of error together we expect that we should have an error of about 3mm in the position variables, and 0.02 rad for the orientation. We believe that error is low enough that we can claim that the grasps are being replicated with sufficient precision to conduct the robustness tests.

5.2 Statistically Estimating $P_s(\text{Obj}|\text{Perturbation})$

As a way of quantifying the robustness of our collected grasps, we want a way of estimating the probability of grasp success for a given a particular perturbation $P_s(\text{Obj}|\text{Perturbation})$. One way to do this is to model a grasp trial+perturbation with a Bernoulli distribution. The Bernoulli distribution is a simple discrete probability distribution that only tracks two outcomes: success or failure. The probability of success is usually denoted p , with the variance being given by $p(1 - p)$.

Using this model, we see that we want to estimate the value of p for a given object given a perturbation. Since we have multiple trials for each object, each with its own image we can replicate, we can estimate this probability by taking the marginal over all the images.

$$P_s(\text{Obj}|\text{Perturbation}) \approx \sum_{\text{Images}} P_s(\text{Obj}|\text{Perturbation}, \text{Image}) \quad (5.1)$$

This approximation becomes exact if we can cover all possible images that might have that object show up for grasping in our setup. By assuming that all images (i.e. conditions) are equally likely, we can replace the marginal with just an average over the images. Then taking a finite sum over the images we can get a decent approximation of the probability. However, replicating and running the trials is a time consuming process, so we are interested in minimizing the number of elements in that sum.

One way to do that is to specify what kind of confidence we require in our estimate of $P_s(\text{Obj}|\text{Perturbation})$. We can model our uncertainty that a given approximation to $P_s(\text{Obj}|\text{Perturbation})$ is correct by assuming that our error in estimating the probability is

normally distributed. We know that for a Bernoulli distribution the mean is given by the p . The error on the mean (also known as the standard error) is given by $\frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation of the data set and n is the number of samples. Then it becomes obvious that we can quantify our confidence on the approximation of $P_s(Obj|Perturbation)$ by using a normal distribution with mean $\mu_{grasp} = P_s(Obj|Perturbation)$ and standard deviation $\sigma_{grasp} = \frac{\sqrt{P_s(Obj|Perturbation)(1-P_s(Obj|Perturbation))}}{\sqrt{n}}$.

We need, however, to choose a threshold that we would like to meet; that is how much error are we comfortable tolerating. For the purposes of grasping we choose the following criteria: we will do at least 10 tests and will continue doing tests until we reach $n = 20$ or we have 95% confidence that the true value of p is not greater than 90%. We chose the value of 90% as that is the threshold Kappler et al. [28] set for a grasp to be considered a success.

$$\int_{-\infty}^{0.9} N(\mu_{grasp}, \sigma_{grasp}^2) dx > 0.95 \quad (5.2)$$

The confidence criteria can be summarized with the inequality in equation 5.2. If that inequality is true, then we stop doing tests and move on to the next perturbation or object. When the inequality is false we keep testing until we finish all twenty trials. This means that if the probability of success is greater than 90% we will at worst approximate the true to within 5%. There are two special cases. If in the first ten tests either none of the tests are successful or they are all successful then we stop testing and move on. These are special cases because they are when $\sigma_{grasp} = 0$. Since the normal distribution is not defined for zero variance, we have to handle this case separately. We assume that if ten grasps in a row were either success or failure, that it is unlikely that the real value of p is going to be much different.

Using these tools we can estimate the probability of a successful grasp given a perturbation, and also bound our confidence around that estimate. We can also save some time without sacrificing too much accuracy.

5.3 Test Procedure

Here we describe the test procedure in more detail. First we chose four objects from the YCB object dataset [10]: the foambrick, the purple toy arch, the bathroom cleaner and the sugar box. We chose these objects because they represent a mix of overhand and side grasps. We also want to make sure that our test and methodology can be replicated by other groups with the YCB objects.

After having selected objects, we sample twenty grasps for each object from our collected data set. Even though we sample grasp trials with obstacles, when we replicate the image, we do not bother replicating the obstacles. For each of the twenty selected grasps we then generate perturbations in each of the six pose dimensions. We do not mix dimensions when applying perturbations. That is, when we apply a perturbation on the x-axis, we do not apply any other perturbations. We apply seven levels of perturbation, with a logarithmic scale. For perturbations in the spatial dimensions, we apply the following perturbations: 5mm, 8.2mm, 13.6mm, 22.4mm, 36.8mm, 60.7mm and 100mm. The spatial perturbations

are applied with respect to the gripper's reference frame. For the orientation, we perturb by rotating about the coordinate axes of the gripper. We apply the following perturbations: 5° , 7.21° , 10.40° , 15.0° , 21.63° , 31.20° and 45.0° . At this point, we check that the generated perturbations will not collide with the table in the workspace. If they do happen to collide, we perturb in the opposite direction. The most common correction was to make sure that when the UR 5 executed the grasp, the robotic arm would keep the wrist high so as to not hit the table. To avoid the gripper colliding with the object, we always perturbed the z direction such that the gripper would move away from the object.

Then we move on to the tests. For each test, we would load one of the twenty images for a particular object and replicate the image as closely as possible. We would then trace the outline of the object on the workspace table so that we could quickly place the object back on the table after every test. We would then have the UR 5 replay final pose of the gripper, first with no perturbation then with perturbations in ascending order, first through the spatial variables then the orientation. The UR 5 does not follow the trajectory that we collected during the grasp trial, instead it only replicates the final pose. To approach the object we used the following procedure.

1. Move the gripper 20cm above the final pose
2. Rotate the gripper to the final pose
3. Back the gripper off 5cm in the negative z gripper direction
4. Lower the gripper 20cm
5. Cover the last 5cm in the positive z direction of the gripper
6. Close the fingers

After the fingers closed, we would be prompted to test the grasp. If the gripper did not make a good grasp with the object, we would skip the test and classify the test as a failure. If the gripper appears to have a decent grasp, then we would proceed with the test: lifting the object ten centimeters and shaking it back and forth five times. If the object is not dropped, then the test is successful. If the fingers of the gripper would drag or collide with the table or the UR 5 joints we would also consider that a failure. Regardless of whether or not the test was successful, the gripper would be backed away following the same procedure as above but in reverse.

After ten images with the same object, we apply equation 5.2 to decide whether we should continue testing that object-perturbation combination. After five more tests we would check again for any other perturbations that needed to be dropped (total fifteen), before doing the last 5 tests (grand total twenty). Since we performed the tests by starting with the small perturbations first, we could skip the larger perturbations if there was a failure early on (i.e. if a perturbation of 36.8mm did not work, then neither would 60.7mm or 100mm). In total it would take us a day to go through each object.

5.4 Results from Robustness Tests

Figures 5.1 and 5.2 show the results from our robustness testing. We have added logistic fits to all the plots, with the exception of yaw perturbations for the bathroom cleaner and the sugar box as these objects had a perfect success rate. Figure 5.1 shows the data for the purple toy arch and the foambrick; both of these objects were tested for perturbations against overhead precision grasps. As such, it is expected that perturbing the grasp in the z direction will cause a sharp decrease. However, we see that even perturbations of up to 10mm have a relatively high chance of success. Perturbations in the y direction fall off the most dramatically. The y direction is the direction that lies parallel with the fingers of the gripper. As a result, even moving a small distance in this direction might cause one or more of the gripper’s fingers to miss the object completely. Even then, the grasps performed on the foam brick are robust, achieving a hundred percent success rate for perturbations as high as 13.6mm. The purple toy arch has a more precipitous fall in success rate. Which is probably due to the arch shape of the object not having much surface area to grab after having moved even a small amount. The x direction lies along the long axis of both objects, which is why we see a gentler drop-off. It is still encouraging that perturbations on the order of a centimeter do not lead to absolute failure. The rotation degrees of freedom tell a similar story. Rotation about the z axis (roll) for overhand grasps leads to very little drop-off in success rate. The foambrick probably has a higher success rate due to its compliant nature compared to the hard plastic arch. Rotation about the y axis (yaw) seems to be able to withstand 10° without much of a drop-off, which again we find to be very encouraging. As far as pitch, the grasps associated with the foam brick seem able to take advantage of its compliant nature to go almost 40° without a decrease in success rate. The purple toy arch has a similar success rate up to about 10° . Overall these grasps seem to be quite robust. Changes of up to 10mm and 10° seem to have a pretty high chance of success. This means that systems built using this data might be able to function on conventional data and do not require high precision cameras or equipment to perform well.

Figure 5.2 shows the plots for the bathroom cleaner and the sugar box. Both of these objects were grasped from the top and from the side during the data collection sessions, but for these tests only grasps from the side were sampled. In analogy to the overhand grasps, where roll has a very slow decline due to the gripper placement, rotational perturbations along the y-axis (yaw) yielded no failures whatsoever. This is most likely due to the fact that both objects are tall and thus perturbations along the yaw axis still allow the gripper to find enough contact points to successfully grasp the object. Rotations about the x axis also yielded pretty high success rates: perfect scores even with perturbations of 20° . This is another axis about which the gripper can rotate, and still find enough contact points. These perturbations almost line up with a cylindrical symmetry; to achieve such symmetry it would be necessary to also perturb the spatial dimensions to keep the gripper pointing towards the object as it rotates. For the spatial variables we see even more robust performance, almost all the way out to four centimeters for both objects. This is to be expected as both objects were being grasped with power grasps, which place the palm of the gripper to the object, so there is a decent amount of perturbing that can happen before the fingers lose contact with the object completely. The sharp drop for perturbations along the y axis has to do with the overall length of the ReFlex SF’s fingers. At roughly 12cm in length, after being

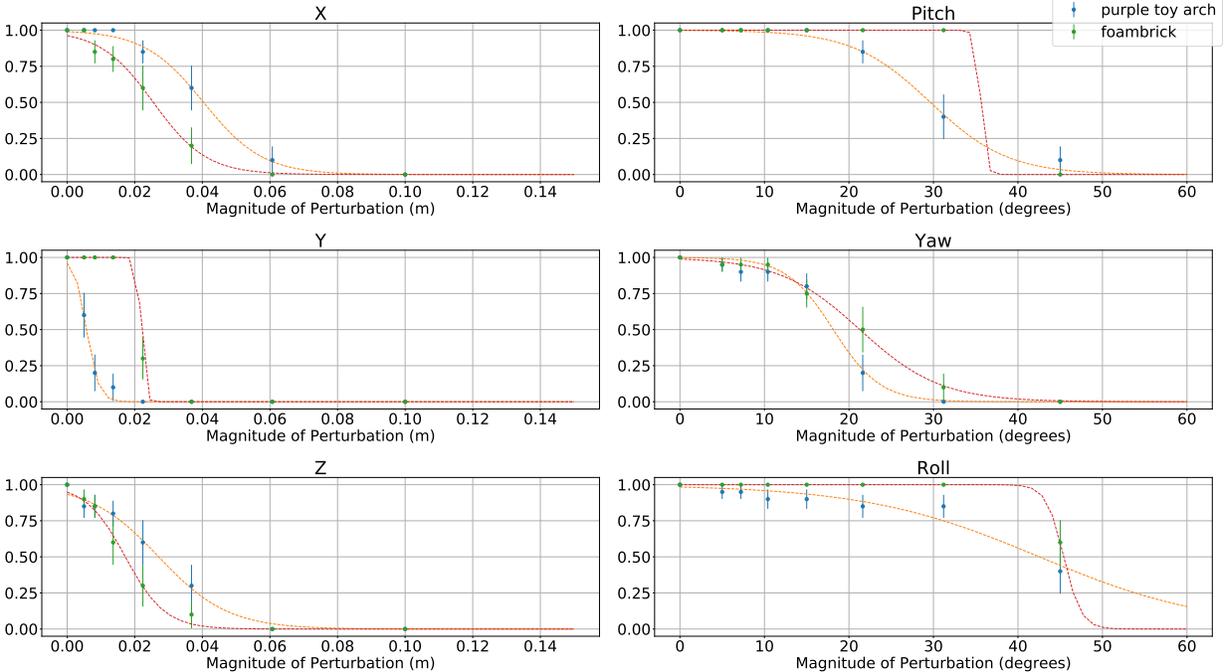


Figure 5.1: Plots showing the approximations of the probability of grasp success under various perturbations. Blue points correspond to the purple toy arch, while the green points correspond to the foambrick. Both of these objects are grasped with precision overhead grasps.

perturbed by six centimeters the fingers are more likely to push the object out of the way than wrap around them. In any case, we once again see that this data is robust to relatively large perturbations.

To quantify the robustness to these perturbations we use Kappler et al.’s [28] threshold criteria: anything with greater than 90% probability of success is considered a successful grasp. We then solve our logistic fits to find at which level of perturbation the grasp crosses the 90% threshold. Tables 5.1 and 5.2 show the results of taking the average across the set of objects grasped overhead with precision grasps and those grasped from the side with power grasps. We see that, except for the z axis, our grasps can resist perturbations of greater than 10mm. The z axis for precision grasps shows only the ability to resist up to 4.5mm, which is most likely due to the fact that these grasps are made only using the fingertips and backing away in the z axis would quickly lead to missing the object altogether. For the orientation perturbations we again see that our grasps can resist perturbations greater than 10°. We report N/A for the yaw perturbations of the power grasps since we never observed any of these grasps failing.

We conclude that the data set we have collected seems to be robust to perturbations. This is what we expect to find when we leverage human intuition to guide our data collection process. Even when the gripper itself is far from human-like, humans have developed a sense, over years of grasping objects, for what kinds of envelopments and enclosures are most likely to succeed in the real world. We believe that our dataset has captured this intuition to some

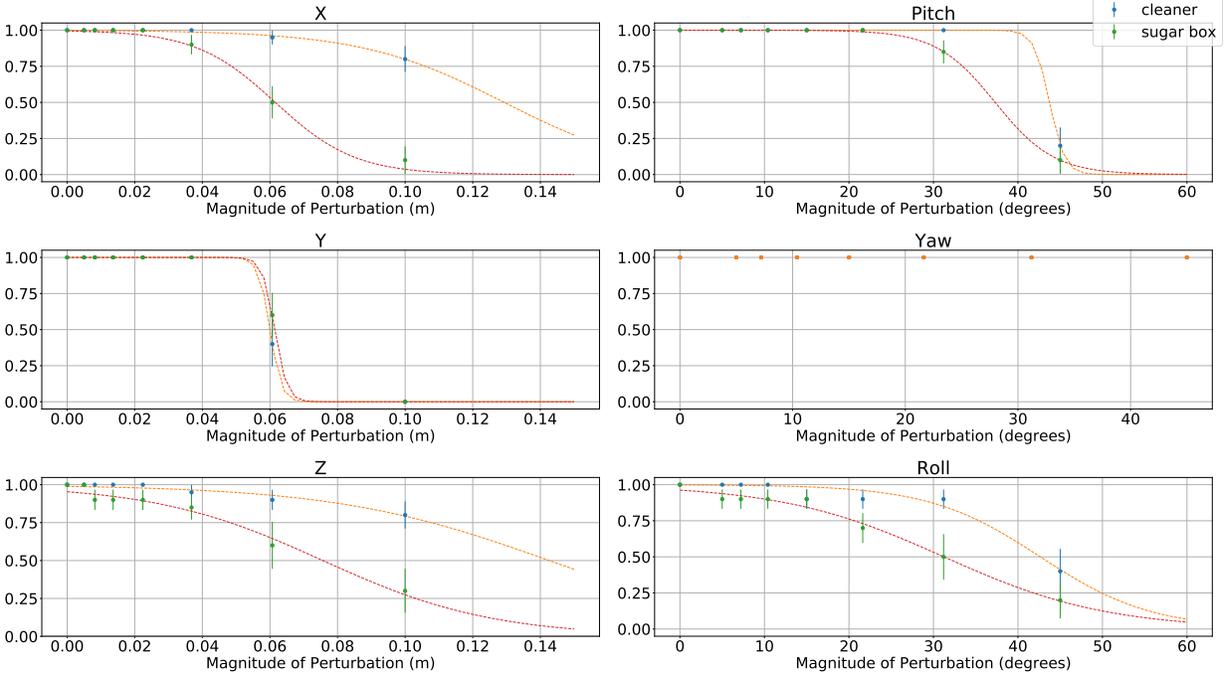


Figure 5.2: Plots showing the approximation of the probability of grasp success under various perturbations. Blue points correspond to the bathroom cleaner, green points correspond to the sugar box. Both of these objects are grasped from the side with power grasps.

| | X (mm) | Y (mm) | Z (mm) |
|-----------------|--------|--------|--------|
| Precision Grasp | 14.72 | 11.18 | 4.50 |
| Power Grasp | 59.16 | 56.91 | 46.77 |
| Overall Mean | 36.94 | 34.05 | 25.63 |

Table 5.1: Here we present the average across precision and power grasps for the point at which the logistic fits for the position data crossed the 90% threshold for a successful grasp.

extent, such that building grasping systems on top of it might lead to cheaper and more robust grasping systems. This is why in the next chapter we show our attempts at training a deep learning system to leverage this dataset to plan grasp poses based on the RGB-D images we collected along with the grasp poses.

| | Pitch (degrees) | Yaw (degrees) | Roll (degrees) |
|-----------------|-----------------|---------------|----------------|
| Precision Grasp | 27.20 | 11.44 | 31.49 |
| Power Grasp | 35.68 | N/A | 19.13 |
| Overall Mean | 31.44 | <11.44 | 25.31 |

Table 5.2: Here we present the average across precision and power grasps for the point at which the logistic fits for the orientation data crossed the 90% threshold for a successful grasp. We report N/A for the yaw of the power grasps since we never observed a failing grasp in those experiments.

Chapter 6

Deep Learning Algorithm

In chapter 5 we showed that our data set has managed to collect some features that lead to robust grasps. With this in mind, we then sought to build a grasping convolutional neural network that would leverage our data set to attempt grasps on novel objects. One of the fundamental problems that we need to address to build a grasping network is a way to build a one-to-many mapping. Neural networks are by design function approximators, which means that they are actually one-to-one mappings. However, any one object might admit vastly different grasps. As an example, a coffee cup, could be grasped by the handle, or from the top, or many other ways. We need to find a way to encode this idea into our network: one image can lead to multiple grasp candidates. We look to DexNet 2.0's [36] as an example of how to tackle this problem using a grasp quality network, in essence turning the problem into a classification problem: for a given a pose, what is the probability of success. However, we try to do a dense classification across a quantized set of possible poses, in a similar way to Zeng et al. [57] and Fischinger et al. [16]. We also do not restrict ourselves to only overhand grasps (three dimensions) and seek to create grasps in the full space of poses (six dimensions). To attempt to deal with the curse of dimensionality in this scenario, we train two networks: a position network and an orientation network. The position network consumes an image and produces a dense map of all the possible positions the gripper could be located in, while the orientation network consumes the same image plus the chosen position and produces a dense map of all the possible orientations. In essence we use our position network output to condition our orientation network. This allows us to reduce the complexity of our output space from six dimensions to two different subsets of three dimensions each. We expect that if our hypothesis is right that our grasping system should be able to approach, or surpass, an 80% success rate on unknown objects, which are the rates we see for Levine et al. [32] and Mahler et al. [36].

We tried a couple of different approaches, and many different architectures within those, but have yet to find any positive results. In the next couple of sections we will discuss the steps and experiments we have taken thus far. We will begin by explaining the extra steps we took to prepare the data set to be consumed by the neural network during training. Then we will cover our approach for creating realistic failures so that we might build a classifier based on our data of successful grasps. Afterwards we will cover the different architectures we tested and how we trained them. Finally we will deliver insights into why the approaches we have tried thus far have not delivered positive results.

6.1 Preparing Data and Quantizing

To prepare the data to be used in a grasping CNN, we do two things. First, we change the reference frame of our entire data set from the workspace frame to the camera frame. In section 4.1.4 we discussed how to find the transformation matrix that can do this. We want the data to be in the camera reference frame because we want the most direct correspondence between the receptive fields of the nodes in our CNN and the pose coordinates they are matching to in space. The second thing we want to do is shift all the positions in our data set five centimeters in the direction the gripper was facing at the time. We do this because we want the coordinates the network is associating with the grasps to be associated with the actual point the gripper’s fingers enclosed, rather than the palm of the gripper which does not always make contact. The choice of five centimeters is chosen as a rough halfway point down the length of the ReFlex SF’s fingers. For a lot of precision grasps the five centimeters brings the position we are training substantially closer to the object. For the power grasps, the shift of five centimeters might bring the position inside the object itself. We tried with and without this shift and found that using the shift led to slightly faster learning, and did not negatively affect the loss.

Turning the grasping problem into a dense prediction problem requires that we quantize the space of 3D poses. We know from section 4.3, that the data does not occupy all of the 3D position or 3D orientation space. Instead there are underlying structures that we can use to help us quantize the space.

6.1.1 Quantizing Position

In position space, we know that there is a concentration of points near the workspace origin. Since we are now working in camera coordinates, the relative location of these points has shifted, but they still have the same general shape. We can contain most of these points in a compact rectangular prism, whose axes follow the camera’s coordinate axes, and then quantize the space inside the prism into partitions at regular intervals. We assign each point inside the prism to some partition and use that as our quantized variable. Ideally, we want the partitions to be as small as possible, since the bigger they are the bigger the error we accumulate. We decided to drop the two hundred furthestmost points in each axis (x,y, and z), which was roughly equivalent to dropping outliers more than 3.5 standard deviations from the mean in each axis. This reduced the size of the prism substantially at the cost of only six hundred grasps (1.67% of the total data). We settled on using a prism with $8 \times 8 \times 8 = 512$ partitions, with each partition being roughly $1.5cm \times 1.5cm \times 2cm$. The worst case scenario is that of a data point at the corner of a partition, which would place it $\sqrt{0.75^2 + 0.75^2 + 1^2} = 1.45cm$ away from the center of the partition. According to our robustness tests, this is well within the range we expect our grasps to survive being perturbed.

6.1.2 Quantizing Orientation

Orientation space was not as easy to quantize since the features are distributed widely across the space. To quantize the space, we took a different approach, first we imagine adding nodes

evenly every $\frac{\pi}{32}$ radians. Then every data point is associated with the closest node, and we count how many points each node has accumulated. Finally we only keep nodes that have four or more data points associated with them. We then enumerate the nodes and use those labels as our quantized variables. After applying this algorithm we end up with 1160 nodes with four or more data points close to them, and dropped about one thousand points from our data set that were not close to these points, keeping about 97% of the data. As with the position quantization, $\frac{\pi}{32}$ radians in all three angles corresponds to a worst case of 9.73° which is within the range we expect a decent amount our grasps to survive any perturbations.

6.1.3 Preparing Images

Finally we need to prepare our images to be consumed by a CNN. Since we used the first two sets of residual blocks from a pretrained ResNet50 [22] model for most of our models, we need to use their preprocessing steps. These steps were already encapsulated in Keras [11], the neural network library that we used to implement our models. For the images themselves, we take the 640×480 size images and downscale them to 320×240 and then crop them so that only the workspace table and the space above it is visible. By coincidence, this lead to images being 224×224 pixels, the same size that our pretrained model was trained with. We applied a similar pattern to our depth channel, but also made sure to align the color and depth maps so that visual features would correspond to the same pixel locations.

We did another substantial change. To be able to use pretrained weights, we replaced the blue channel of our images with the depth channel of our RGB-D images [46, 12]. We tried a couple of architectures trained from scratch on all four channels and did not see any substantial benefits. We believe that it makes more sense to leverage the pretrained weights of the ResNet50 as a feature extractor. Even though the blue channel is now depth, there are still common features that the CNN will have learned to pick out, like edges and curves, that will still have the same basic composition in depth as they do in color.

6.2 Generating Realistic Failures

One of the basic approaches that we tried was to learn a CNN by providing the network with enough success and failure examples. However, we did not have a source of substantial failures to correspond to our data set of successful grasps. It is not enough to generate failing grasps if they are not realistic, as the classification problem would be too easy and the CNN would not generalize across to novel objects. As a result we sought a way to generate realistic failures to feed our network. We decided that the best way to do this would be to implement most of another grasping algorithm and to tweak it to produce realistic enough grasps, but that we were confident would be failures.

We decided that with our data set we could recreate the first part of Bohg et al 2011 paper [7]. In the paper depth maps are used to complete object point clouds under a symmetry assumption. These point clouds are then used to build high quality meshes which can then be fed into mature grasp planning software. We implemented the completion by symmetry, and the mesh reconstruction, but built a very simple grasping algorithm that could give us sufficiently realistic failures.

Completing the point cloud from a depth map involves the assumption that the part of the object not visible from the camera is similar to the visible part. The algorithm proceeds as follows:

1. Segment the table from the objects above the table surface
2. Identify the object in question (in our case the centermost object)
3. Find the weighted center of the point cloud, and pick the plane of reflection symmetry that would complete the back of the object
4. Generate a set of hypotheses for the plane of symmetry by varying the angle of the first plane, and its location relative to the point cloud center
5. Score the hypotheses and pick the best one from among them
6. Mirror the point cloud along the best plane of symmetry

The scoring mechanism is an integral part of how this algorithm works. Bohg et al. identified two conditions that would mean the symmetry assumption was being violated. These occur when a reflected point is either in front of the object it was reflected about, or it occupies some space that we already have depth data for (outside the space of the object). Both conditions essentially lead to violations of our prior knowledge about the scene, so they need to be penalized. Bohg et al. penalize the case where points end up in front of the object by a score proportional to the distance it would take to bring the point behind the object. The second case, a point outside of the object in an area that already has information, is penalized by finding the pixel euclidean distance to the object. In the best case scenario, both of these penalties are zero. We added a term to vary the relative importance between the terms and weighed penalties against points outside an object as being ten times worse.

For the mesh reconstruction we found the points that would make a convex hull from the point cloud and applied a Delaunay tessellation using SciPy [27]. This mesh is guaranteed to be completely closed, but it is not guaranteed to be smooth and it cannot preserve any internal structure since we use the convex hull as our starting point. We believe that this is not necessarily a bad outcome, since we wanted to create failing grasps. The meshes capture enough of the of objects' geometry that any grasps generated would not be completely non-sensical.

The final step was to build a simple grasp planner that would use these meshes to return poses for the gripper. We modeled the ReFlex SF as a three dimensional trapezoid with three twelve centimeter long fingers coming out of it. We randomly selected one of the facets making up the object mesh, and then randomly sampled a location in that facet. By finding the normal of the plane coming out of the triangle, we arrive at the orientation of the z axis of the gripper. We then pick a random number between 1cm and 12cm, and place the origin of the gripper at that distance along the facet's normal and intersecting random point that was selected earlier. The last degree of freedom, the rotation about the z-axis, is also decided by a random number. With all of these pieces, we can generate poses around the object meshes that look somewhat realistic as grasps. We add three more constraints however: 1) The gripper shell cannot be colliding with either the object or the table, 2) As

the fingers would be closing around the object, they cannot touch or drag on the table and 3) only poses that allow the fingers to provide opposing forces are allowed.

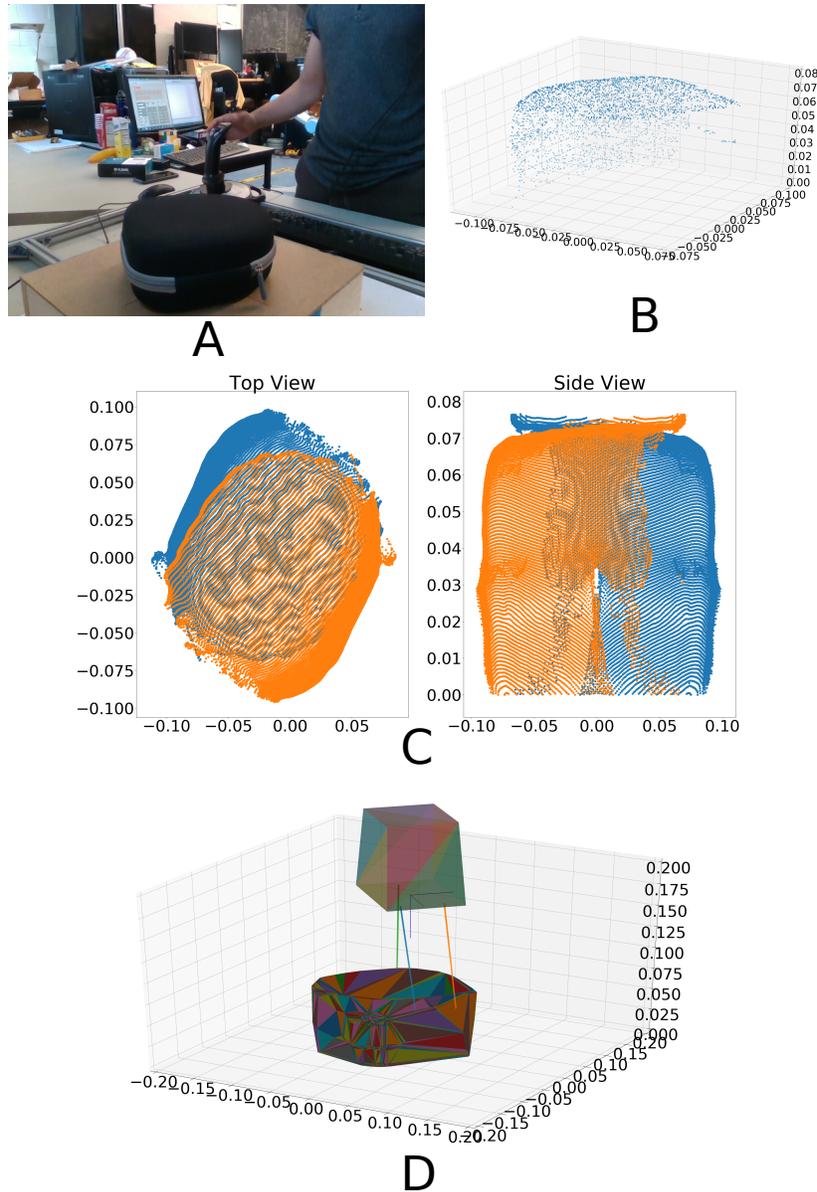


Figure 6.1: Panel showing some of the key steps for generating the realistic failures. All units in mm. **A:** RGB image of the headphones case being used in this example. **B:** The point cloud collected from the SR300's depth sensor. Notice how the back of the case is not closed. **C:** Scatter plots of the original point cloud (blue) and the mirrored point cloud (orange). There are still some points that stick out, but they seem to be minimized. **D:** The simple grasp planner showing the grasp pose on the completed mesh. Notice how the fingers barely make contact. This pose is likely not successful.

The poses generated from this procedure look at first glance as if they are reasonable,

but many of them would fail for small details. The biggest source we could identify for potential failures is that we do not check for force closure at any point, so many grasps might not be stable. Similarly, since we do not check forces or torques as the grasp occurs, the gripper might end up pushing the object out of reach before a full enclosure is reached. Another failing of this method is that it generates many grasps where the gripper is either too far or too close to the object. Being too far means that only the fingertips would make contact with the object, which in our experience does not provide enough force to secure many objects. We recognize that there is still the possibility that some of these grasps would be successful, but based on visually inspecting a large subset of them we do not believe that more than a small amount (10% 15%) would succeed in the real world. Differentiating our human demonstrated grasps from this synthetic dataset should still be a challenging task that forces the network to learn the subtle differences between the real and synthetic datasets.

To generate pairs of successful and failing grasps, for each grasp in our dataset and we run the algorithm above on the corresponding depth map. We then save the pose generated for each depth map along with the poses for our data set. This guarantees that for every grasp trial we have at least one failure to present to our grasp CNN. We did find about 500 cases where our algorithm did not find a pose within one thousand tries; we decided to drop those from our data set.

6.3 Loss Functions

We tested two different kinds of loss functions. The first is a more traditional kind of loss function, which requires the realistic failure grasps described above. This loss function is binary cross entropy (equation A.16 in appendix section A.3.1) applied only to the output node associated with our ground truth sample. This is equivalent to training a number of networks equal to the number of output nodes that share all weights except the last layer. Practically, we do this by calculating binary cross entropy individually across all nodes, but setting all the errors to zero for all nodes except the node we are currently training. That is, if our ground truth is that position (i, j, k) is a good position for grasping, we propagate gradients through the node corresponding to position (i, j, k) . The neighbouring nodes get an error of zero assigned to them, and thus no gradients are propagated through them. Without any failure data, the network could just minimize the loss by predicting high across all the nodes, so it is important that we use this synthetic failure data.

The second loss function is in some ways the opposite. For this loss function we only use the positive data we collected and none of the generated negative data. We apply binary cross entropy to each node individually and only the node corresponding to the positive sample we are currently training on will get the true (1) label associated with it. All other nodes get a false label (0). The most important difference is that now each individual node will see many more false labels than true labels. As a result we also combine the binary cross entropy with Lin et al.'s [34] Focal loss. We use the default parameters they presented in their paper ($\gamma = 2$). The other important aspect to remember when using this loss function is that the threshold between true/false is probably going to be below 0.5. This is not really a problem since we rank the outcomes from highest score to lowest score and take the best

score out of them. However, because this loss function penalizes a wide swath of grasps, and only encourages the one demonstrated grasps, it is possible that it will reduce the activation of grasps that might be just as likely to succeed as the demonstrated grasp. Since this data does not depend on the generated data, it makes less assumptions and is thus more likely to be stable.

6.4 Deep Learning Architectures

We now turn our attention towards describing the different architectures that we tested our problem against. The basic architecture that we theorized would perform the best was a ResNet [22] architecture. We did many tests using different variations of this architecture, but we also tested more classic CNN architectures. We spent the majority of our tests trying to get decent results out of the first network in our system, the position predicting network. While testing we realized that our data set of grasps is not evenly distributed across the 512 partitions, which leads to a bias in which partitions are most likely to be selected. We tried to use this prior information to our advantage to help the network train faster, but also tried methods to reduce the impact of this bias (namely focal loss [34]). We also spent some time tuning the hyperparameters in our network hoping to achieve better results. We found a combination of hyperparameters that seemed to lead to fast convergence with the networks we tried.

6.4.1 Prior Information

We want our position network to learn to predict $P(S|img, part)$, the probability of a grasp being successful given a partition and an image of the scenario presented. However, we know that not all positions are equally likely. As we saw in Figure 4.3, the position tend to form clusters. In fact, we found that providing no image to the grasping network could still lead to results that were much better than picking with a uniform random distribution which partition to place the gripper at. In fact, we found that the $\mathbb{E}[P(S|part)] \approx 78\%$, which is a fairly strong prior. This provides us with another benchmark to assess: we want our position prediction network to perform better than the prior distribution alone could.

It turns out that the $P(S|part)$ that the probability of success for one of the 512 of possible partitions tends to cocentrate on only some of the 512 partitions. We hypothesized that if we could give the network this constant information then it could concentrate on learning features that would allow it to map images to grasp success. We came up with three different ways to incorporate this information into the network.

Multiplication

We can use Bayes' Theorem to break down our problem. The common way of writing Bayes' Theorem involves only one conditional variable, whereas here we have two, the image and the partition. Since the prior information we have access to varies over the images, we leave

the partition conditional unchanged. This gives us the following equation:

$$P(S|img, part) = P_{prior}(S|part) \frac{P(img|S, part)}{P(img|part)} \quad (6.1)$$

Where we have labeled $P(S|part)$ with the "prior" subscript to highlight that we already have access to this information. Based on this formulation, we would require the grasping network to learn $\frac{P(img|S, part)}{P(img|part)}$. There are only two constraints on this value: it must be equal to or greater than zero, and when it is multiplied by the prior it must not result in a value greater than one. If the network can learn that, then we can calculate $P(S|img, part)$ easily by multiplication.

Log Addition

In some cases we might prefer to perform addition, rather than multiplication. In this case multiplication places different constraints on what the network's outputs can be, on top of routing gradients differently through the network. If we take the natural logarithm of both sides of equation 6.1, we get:

$$\log(P(S|img, part)) = \log(P_{prior}(S|part)) + \log\left(\frac{P(I|S, part)}{P(I|part)}\right) \quad (6.2)$$

Instead of asking the network to learn $\frac{P(I|S, part)}{P(I|part)}$, we can have it learn the log of that. This allows us to drop one of the constraints, since $\log\left(\frac{P(I|S, part)}{P(I|part)}\right)$ can take any value in $(-\infty, \infty)$. We still need the addition of that plus our log prior to be less than zero, so that if we exponentiate both sides again we still get a probability bounded by $[0, 1]$.

Residual Addition

Inspired by Zeng et al.'s Tossing Bot [57], we imagine that our network only needs to learn some residual information. That is our prior information is good enough to get us a decent guess and the network just needs to tweak that guess based on the image information presented.

$$P(S|img, part) = P_{prior}(S|part) + \delta P(S|img, part) \quad (6.3)$$

We briefly tried the three methods mentioned above and did not find any meaningful difference between them. We chose to stick with the residual addition method presented here because of its versatility and ease of implementation. Using any of the above three methods did reduce our training time considerably, leading us to believe that adding this prior information does indeed help the network find better features faster.

6.4.2 Model Fine Tuning

In appendix section A.3.5 we go over the different hyperparameters that are available for us to tune. Here we present the values we used for these hyperparameters. After some early testing, we found that using Adam with a learning rate of 10^{-5} seemed to be a good

match for our problem. All our layers except the output layers were set to LeakyReLU’s with $\alpha = 0.2$. We also noticed that our training curves would sometimes seem to oscillate and climb for an epoch before starting to decrease again. We found that by adding gradient clipping of 1, this problem went away. We set our batch size to 64, except in cases where our model would not run on our system due to memory errors. In those cases we decreased the batch size by a factor of 2 until the model would run. We also added an L^2 norm penalty on the weights, with the penalty parameter being set to 10^{-4} after some testing. For some tests we added dropout. The dropout rate would change from one test to another, but it was mostly left at 0.2.

6.4.3 Architectures

We built and tested a variety of different architectures, but they all had the same inputs and outputs. For the position network architectures, we designed networks that would consume a 224×224 image and output an $8 \times 8 \times 8$ volume, where each partition corresponds to a position in space, as described in section 6.1. Importantly, the individual partitions do not depend on each other. This is because, as mentioned previously, any one image of an object might admit multiple grasps. One partition might have as much of a chance of succeeding as another. The orientation network would consume the same image, plus a 3-tuple with the coordinates of the partition in the $8 \times 8 \times 8$ cube. It would then output a vector length of 1160 where again the individual nodes are not dependent on each other.

In this section we will discuss all the different model architectures that we tried training. We will not be showing results from all these architectures, as most of them did not seem to be converging towards good results. In section 6.5 we will show results for the architectures we felt performed the best.

Simple Architectures

We tried a few comparatively simple architectures. None of the following architectures had any pre-trained layers. One of the first networks we tried was a traditional convolutional neural network, replacing the max-pool layers with strided convolutions. Overall we had 8 conv-layers with 4 of those being of stride 2. The convolutional part was followed by a three layer MLP ending with an output of 512 nodes that was reshaped to the $8 \times 8 \times 8$ shape we need as an output. We varied the number of filters in the conv layers and the number of nodes in the dense section, but this method did not perform much better than just guessing based on the prior information.

Another network in this category was an all convolutional network [52]. This network consisted of 5 conv-layers with no max-pooling or strided convolutions in between. This was followed by a conv-layer with 28×28 sized filters and stride 28. By coincidence, $224 \div 8 = 28$ so by using filters of size 28×28 and a stride of 28 we can reduce our feature maps to $C \times 8 \times 8$. Following this with a 1×1 conv-layer with 8 filters and a sigmoid activation gives us our final $8 \times 8 \times 8$ cube. This network did not perform any better than guessing with prior.

We then moved to training more complicated networks based on the ResNet50 [22] architecture. We copied the architecture described by He et al., except we substituted the last three residual 2D convolution blocks with residual 3D convolution blocks. Since the output

of the network is inherently describing 3D space, this should allow for a more straightforward mapping from image to grasp probability. This model had on the order of 50 million parameters. We found that the model quickly overfitted and did not perform better than just using the prior information.

We then tried to reduce the size of the network described above by only keeping the first 10 residual blocks from the ResNet50 architecture and following that with one residual 3D convolution block to the output. By contrast, this network had around 1.5 million parameters. We still found this network to quickly overfit the training set before reaching performance better than just guessing based on the prior distribution.

Pre-trained Resnet style Architectures

Seeing that the previous networks had a propensity for overfitting, we moved to using some pre-trained layers in our networks to reduce the number of parameters. We used the pre-trained ResNet50 model provided by Keras [11], by reusing the first two layers plus the first ten residual blocks of the ResNet50 architecture, and freezing their weights. This model had been trained on the ImageNet dataset. We added more trainable residual blocks on top of this base.

Our first attempt was to add a strided convolution to reduce the size of the feature maps so that they would match the size of the next six residual blocks of the ResNet50 model and to then resize and reshape the feature map so that it would have the $C \times 8 \times 8 \times 8$ we need, where C is the number of channels of the map at that point in the network. We then followed that with three blocks of residual 3D convolutions and one 1×1 3D residual convolution to bring the size of the output to $1 \times 8 \times 8 \times 8$. To this output we added the prior, as discussed in section 6.4.1. This model still had on the order of 50 million trainable parameters, but it trained relatively quickly. However we found that the model would quickly start to overfit. Even by controlling the number of kernels, reducing the number of parameters to 7 million, we still saw overfitting. This led us to experiment with networks with fewer parameters organized in a different way.

Our next attempt was based on the second ResNet50 architecture we trained end-to-end, except we replaced the same layers mentioned above with pre-trained layers. This network still had on the order of 1.5 million parameters, but with only 16.5k trainable parameters, and we still found it to be overfitting. We decided then to try to change the architecture to see if we could encourage the network to learn the patterns we believe are there.

Our main change was to add different branches for different partition depths (see Figure 6.2), the z axis of the $8 \times 8 \times 8$ cube of predictions. This meant that pre-trained layers would act to extract useful features from the image, but one branch would be in charge of finding more features that would be useful to make predictions for its depth level. Each of the 8 branches consisted of a 2D convolution with a 1×1 filter followed by residual 2D convolution block. The feature maps from these branches are then concatenated together and one block of residual 3D convolutions is used before outputting. We found that this was the first model we trained that could perform better than just guessing based on the prior position information. It would still overfit, but we moved to trying to heavily regularize it.

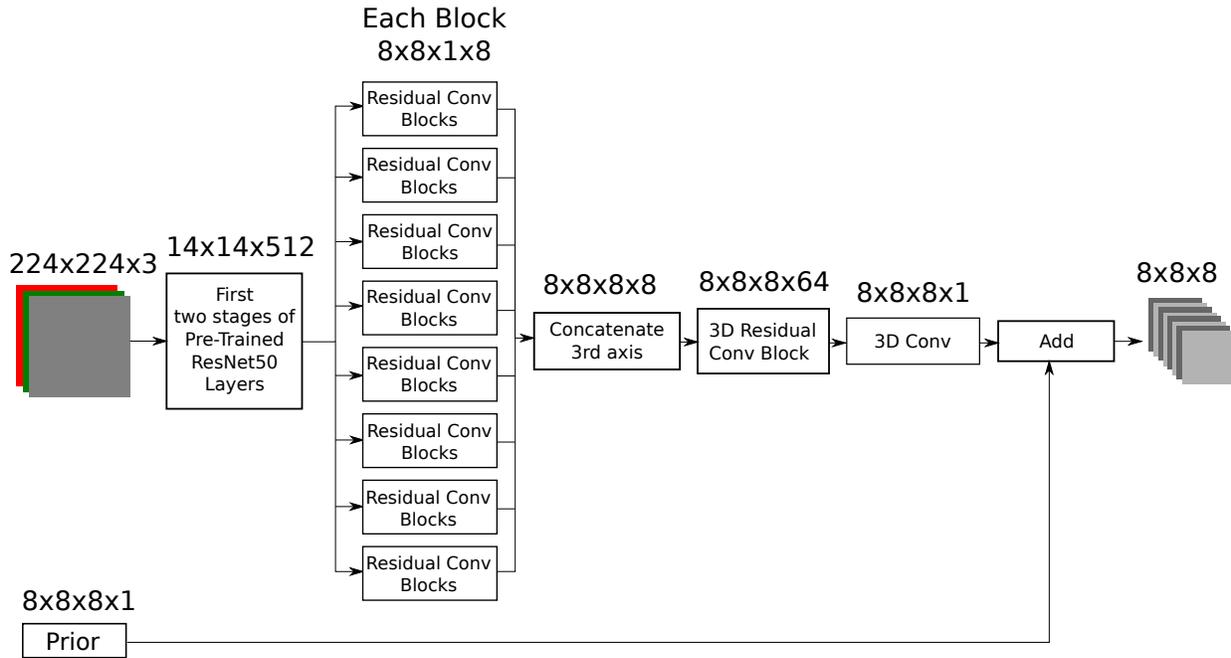


Figure 6.2: Diagram of the position network. Above each element the output size of the element is listed. The network first takes the RG-D input through the first two stages (first ten residual blocks) of a pre-trained ResNet50 [22] network. The output of the pre-trained layers is then passed into eight parallel paths. Each of these paths corresponds to one of the depths in the final output volume. Along each path, the feature maps are first passed through a residual convolution block and then an extra dimension is added so that when the feature maps are concatenated together the relationship of each bank of features as corresponding to one particular depth is preserved. Each of our residual blocks consists of five sets of bottleneck convolutional units, which in turn are set of three conv layers with kernel sizes of 3,1,3. After the bottleneck units a residual connection is made from the input of the block to the last layer of the block. These concatenated features are then passed through a residual 3D conv block, before finally passing through a 3D conv layer with a tanh activation. This result is then added to the prior distribution to arrive at a final output volume of $8 \times 8 \times 8$.

Rotation Architecture

We decided to base our rotation architecture on the ResNet50 architecture again. The important addition is that we need a way of mixing in which position we chose from the output of the position network. The easiest way that allows us to keep using the pre-trained layers is to combine the data somewhere after the pre-trained layers. We chose to add to the information to the feature maps as extra channels, an idea proposed and tested by Liu

et al. [35]. Each extra channel just holds an integer in each element. This integer represents the coordinates of the partition in the $8 \times 8 \times 8$ cube chosen by the position network.

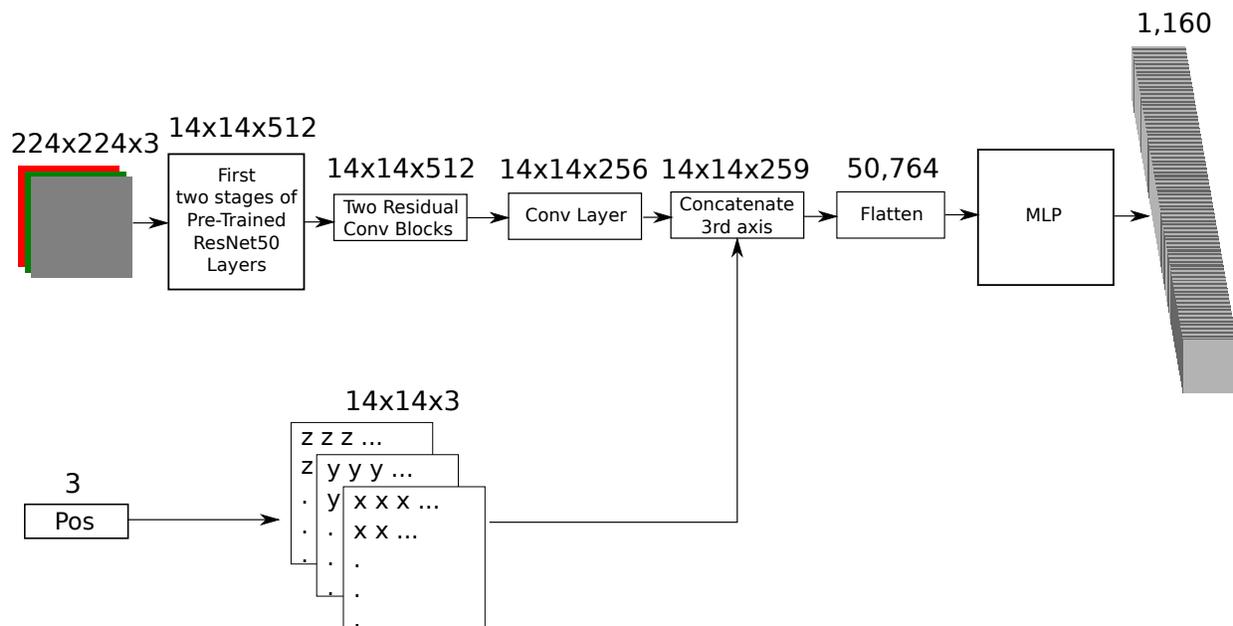


Figure 6.3: Diagram of the orientation network. Above each element the output size of the element is listed. The network first takes the RG-D input through the first two stages (first ten residual blocks) of a pre-trained ResNet50 [22] network. The output of the pre-trained layers is then passed into two residual conv blocks and one conv layer. In parallel, the position partition vector is expanded into three channels, with each channel holding just a copy of each element of the vector. These channels are then concatenated in the main network branch and the output of that is flattened. This vector is then passed into an MLP where the last layer is a 1160 dimensional vector, with each element corresponding to one of the quantized points in orientation space.

The network itself consists of the pre-trained ResNet50 layers followed by two residual blocks and the extra position channels being concatenated to the block of feature maps. Afterwards the whole block gets flattened and a series of dense layers is followed by the output. We found that this network was also prone to overfitting, so we heavily regularized by reducing the number of nodes in the fully connected part and using dropout between those layers.

6.5 Results

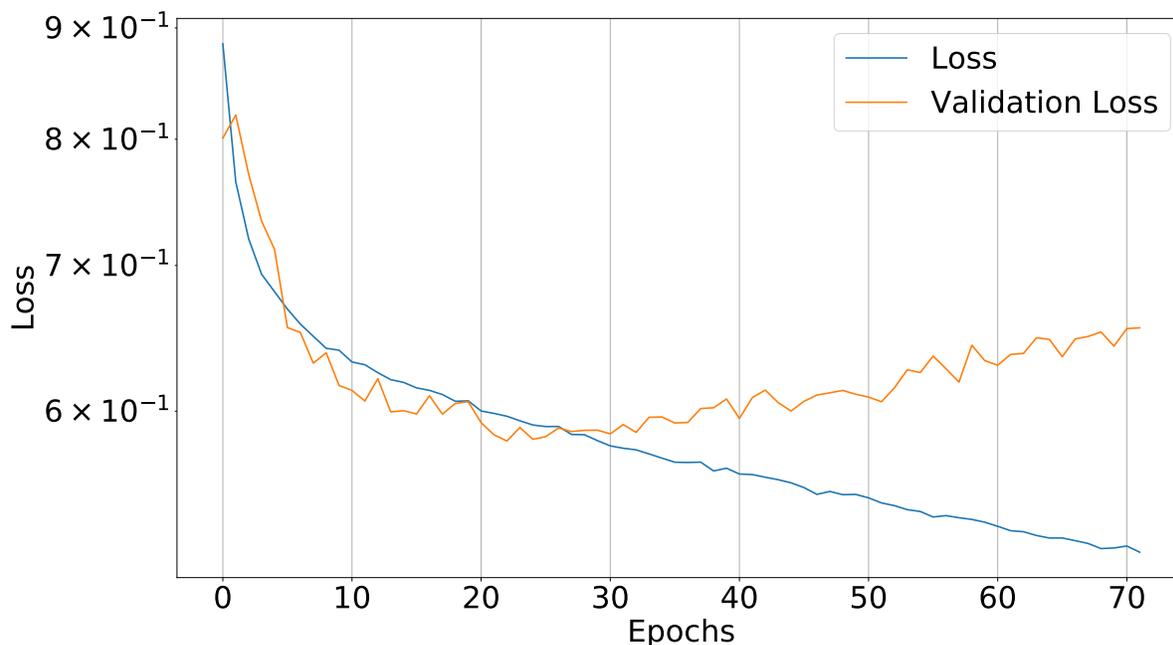


Figure 6.4: Training curves for the position network.

Here we present results from our tests training these grasping networks. First we want to make sure that the networks we are training are not overfitting. When a model is said to overfit, we refer to the model learning to replicate the training set without any regard for samples outside of it. That is, the model does not learn generalize across inputs, but merely learns how to map members of the training set to grasp poses. Ideally we want the model to generalize to any of the inputs that might be possible for our task, not just the ones we happened to collect. The most straightforward way to check if a model is overfitting is to split the data set into two parts: a training set and a validation set. The model trains on the training set, while the validation set is not used to make any training decisions. At set intervals, we test the model by using the validation set. If the model is not overfitting then the training and validation losses will be decreasing together. However, once the model starts to overfit, the validation loss will start to increase, even while the training loss continues to decrease. This is a clear indication that the model is starting to lose its ability to generalize even within the dataset we collected, hence we say it is overfitting.

We made two different validation sets: one contained images of objects that the network never saw during training, the other contained images that the network never saw during training (but the objects were seen during training). These two validation sets are meant to test the ability of the network to extrapolate to new objects, and interpolate known objects. To make sure that we got the best network we could out of each training run, we used a checkpoint system. Every time the network would reached a new low in the validation loss, we would checkpoint the model. This meant that even after the model started to overfit, we

would have access to the best model from each run. The results presented here used the prior information. The position network had to learn to correct the prior partition information by adding some value to it. These were also trained using the second loss function presented in section 6.3.

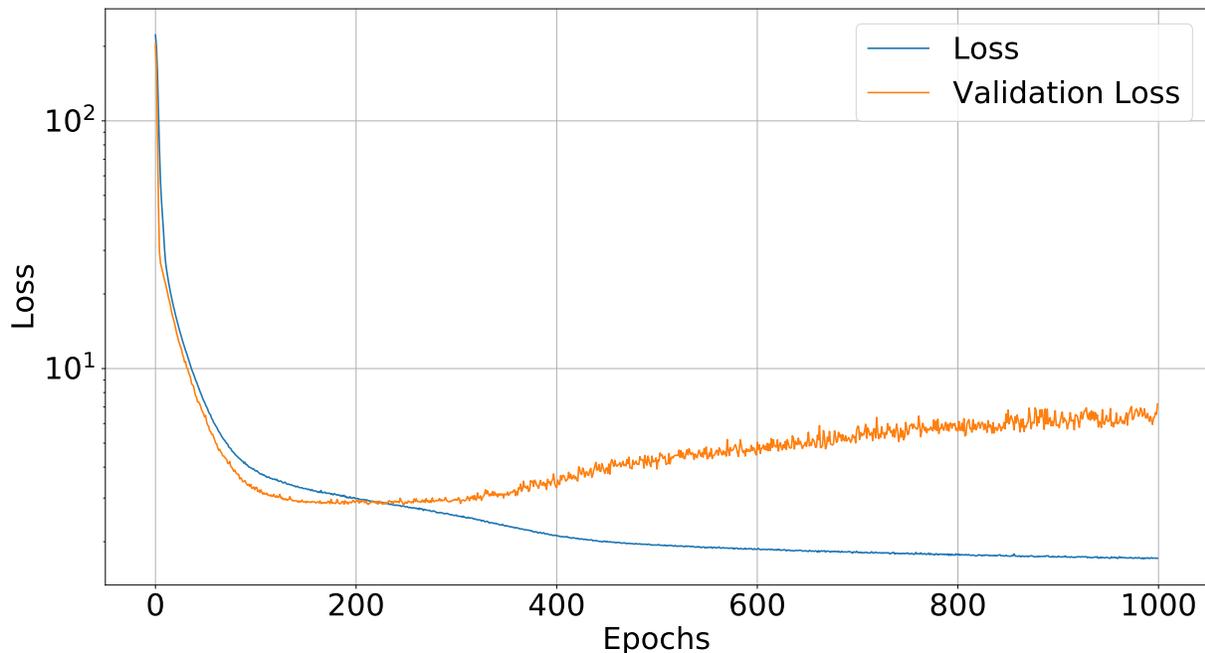


Figure 6.5: Training curves for the orientation network

Figures 6.4 and 6.5 show the training curves for the position and orientation networks. For these tests we combined both validation sets. The checkpoint system saved the position network at epoch 22, and the orientation network at epoch 212. We see that after these points the validation loss begins to climb, which is a clear sign of overfitting. We also measured $\mathbb{E}[P(S|part)]$ for the position network to be around 81%, which is better than what we expect we could do with the prior alone. This was the highest value we recorded before the position network started overfitting.

Having trained both models, we put them together into a grasp prediction system. This system would work by preprocessing a brand new image from the SR300 and then feeding it into the position network. From the position network we would get an output volume with shape $8 \times 8 \times 8$ and then proceed to find the coordinates for the highest partition, in essence finding the largest $P(S|img, part)$. We would then feed the orientation network the same preprocessed image plus the coordinates selected by the position network. The orientation network would then output a vector with each element denoting one possible orientation. We would take the largest element and find what orientation it corresponded to. Putting the position and orientation we would have a full pose that we could then test. There was the opportunity to do some optimization by re-using the feature maps coming out of the pre-trained ResNet50 layers for both the position and orientation networks, but we did not implement that at this time. This does not affect the pose results, but it might make the

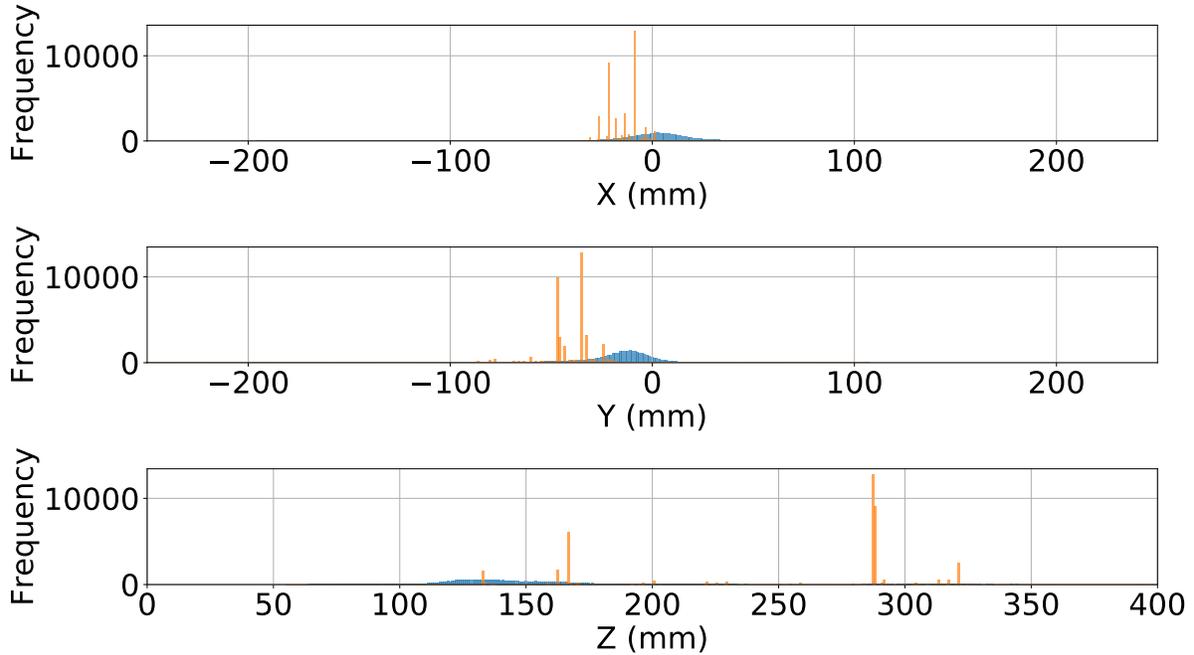


Figure 6.6: Position histograms made from predicting on the training data set using our system. The original histograms from the training set (blue) can also be found in Figure 4.2. The predicted position histograms (orange) are seen as being in small clusters with much higher density. Both the training and predicted histograms are plotted using the same bins. It is useful to remember that the space quantization took place in camera coordinates, which is why we the predicted set is not seen as having only eight bins. See section 6.1.1 for more details.

system faster and probably would use less memory.

As a sanity check for our system, we ran our entire training set of images through. Figures 6.6 and 6.7 show histograms for the different pose variables for the gripper. These plots are analogous to Figures 4.2 and 4.5. Regretably, these do not look good. It seems as if the network has collapsed its decision making to only a few positions, and then that cascades to only having a few possible orientations. Furthermore, it is not entirely clear why the predictions being made for the position have collapsed to only a few values. We do not necessarily expect the network to replicate the training set distribution; the network should be picking the best grasp for the image shown, which since some of the images are similar might lead to the same grasp being chosen. However, we would expect the network to have a range of behaviours and to at least demonstrate some of the biases we saw in the training set. The x axis predictions are not symmetric about zero, but are instead biased towards the negative side, which is the opposite of the training set. The y axis predicted values are biased in the right direction, but peak too late and do not show a trailing edge like the training set. Even worse, the z axis predictions place many samples at high z values, which is not seen at all in the training set. The orientation variables exhibit a lot of the same problems. Pitch is the best of the three, with at least the same basic shape. Yaw does not exhibit any

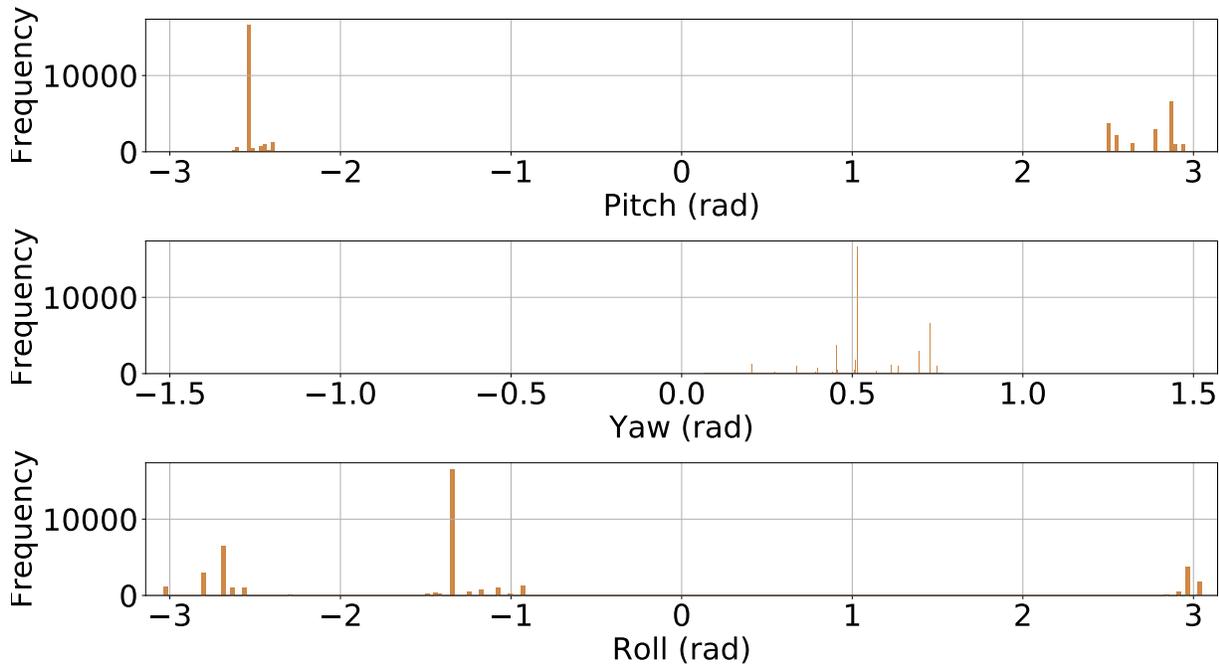


Figure 6.7: Orientation histograms made from predicting on the training data set using our system. The original histograms (blue) cannot easily be seen as the density of the predicted histograms (orange) is so high that the y axis scaling cannot show both at the same time. Compare to Figure 4.5, for the orientation histograms over the training set. The quantization for the orientation variables only kept nodes 1160 that had the highest density of data points around them. See section 6.1.2 for more details.

symmetry and it has peaks where there is next to no data in the training set. Running the system live on the UR 5 with novel objects has similar results. The gripper is placed in only a few location with little variation. In our limited tests, some grasps succeed but the vast majority drop the object due to not having a solid grasp on the object. The grasp success rate is decidedly much lower than the 80% goal we set out to achieve.

It is worth mentioning again that we used what we think were the best position and orientation network after having tested six different architectures, within which we tested increasing and decreasing the number of parameters, tweaking hyperparameters, increasing/decreasing regularization, and even generating our own negative data. The results we show here are also not due to overfitting to a few outliers. We selected our models such that we took lowest possible validation loss from the training runs. Even then, if the results were due to overfitting, we should not see peaks in areas where the data has no peaks.

We expected that our data set would allow us to train a full six dimensional grasping network. Even though we managed to get a slightly better position prediction accuracy than we could with the prior alone, our networks are having a hard time learning at all. There can be many different reasons for why the networks are having a hard time training. One possible reason might be that the representation we picked for our input image might not be conducive to learning. We tried both using RGB-D as separate channels, and dropping

the blue channel and replacing it with depth and using only RGB. It might be possible that some other combination might have been more successful. One particular representation that would require some feature engineering would be to provide grasp heightmaps [24], calculated from the depth map. This representation would involve a series of non trivial non-linear transformations that might not be easy for the network to learn, but which we know is a powerful representation. However, if we were to change the representation too drastically we would not be able to use a set of pre-trained network layers. Another possible area for improvement might be to try different, non-ResNet, architectures. We believe that we tried ResNet style architectures quite extensively, so perhaps other architectures might be able to better extract information from the RG-D images. One possible representation plus architecture combination that we did not try would be to provide the depth information as a point cloud volume [55] and to go directly to grasp pose prediction. The best explanation that has come to mind thus far is that we might be suffering from the curse of dimensionality. We need bigger networks to make better predictions, but the bigger the network the more data we need. Since the amount of data needed to tile the space increases exponentially, we would similarly need even more data. Other groups have tackled this challenge by limiting the complexity of their task space; usually by only making grasp plans that involve the gripper coming from above (two position variables) and only deciding the roll (one orientation variable) of the gripper, with a set pitch and yaw [36]. Predicting in the full six dimensional space is computationally prohibitive, which is why we split the problem in two. If we wanted to keep the same spatial resolution we have now and predict directly into the full six dimensional space, we would need more than half a million output nodes in our network. At this point, we believe that what we missed is just simply having more data, though it is unclear how much more data we would need.

Chapter 7

Conclusions

In the preceding chapters we introduced the work that we did towards creating a robotic grasping system using demonstration and deep learning. We hypothesized that leveraging human intuition to create a large dataset of grasps would lead to a dataset that would encode the years of human grasping knowledge that we have gathered from interacting with the natural world. This dataset would be robust, since human grasps tend to exhibit that property, and it could be used to train a deep neural network to grasp objects like humans do.

In Chapter 2 we covered the relevant literature required to place our work in context. We introduced the works of Mahler et al. on DexNet [36], Herzog et al. [24] and Kappler et al. [28], among others. Herzog showed that using as few as eighteen human demonstrated grasps, and clever shape descriptors, a flexible robotic grasping system could be built. Kappler built upon their work contributing a more robust grasp quality metric, and evidence for how neural networks could learn from large datasets. Mahler et al. have taken a different approach replacing the hand built shape descriptors with a grasp quality CNN that can be used to map directly from depth maps to probability of grasp success.

Chapter 3 introduced our hardware and methodology. We described our gripper, the ReFlex SF in detail, documenting many of its advantages but also its flaws. We also explained how our depth camera, the RealSense SR300, works and the steps we had to take to make sure that the IR emitters and receivers for both the SR300 and NDI Polaris would not interfere with each other. We finally went into detail about the procedure we used to collect over forty thousand human demonstrated grasps with their respective depth maps on 109 objects. This dataset is available online.

Having described the data collection process, Chapter 4 goes into how we transform the raw data from the data collection into a more compact grasp pose dataset. Primarily, we explain how to perform the various change of reference frames required to go from the pose of the tracking crosses with respect to the reference frame of the gripper, to the pose of the gripper with respect to the workspace reference frame. We then provide some insight into the data we collected by showing a series of histograms and scatter plots. We identify different biases in the data which were caused by that fact that our data came from human demonstrators which have their own biases. Some of these biases included the bias towards gripper poses to the right of the object (operators were mostly right handed), and a bias towards grabbing objects by the shortest path possible. We also showed that the data we

collected is also object dependent. Different objects exhibit different distributions because their geometry affects the poses the gripper can use to successfully grasp the objects.

A grasp that is robust, should still succeed even under uncertainty about object or gripper pose, which is a highly desirable feature. Chapter 5 described our robustness tests. We covered how we replicated individual scenes from our dataset and replayed them on our gripper. We also explained our statistical estimation strategy for estimating the probability of success given a particular object perturbation. This strategy allowed us to test the robustness of our grasps for four objects from our dataset. These objects encompassed both precision and power grasps both from the top of the object and from the side. We show that our grasps are robust to perturbations in most axes, with the position variables being robust to perturbations of up to 4.5cm for power grasps and up to 1cm for the precision grasps. The orientation variables show more resiliency being robust to up to 10° across all objects.

Finally we covered our experiments with training a deep grasping network in Chapter 6. We decided that we would split the grasp pose classification task into two parts: a position network, and an orientation network conditioned on position. To train our networks we had to first quantize position and orientation data we had collected. Working in the reference frame of the camera, to allow the CNNs to be able to associate features with physical space better, we partitioned the position data into 512 partitions arranged as an $8 \times 8 \times 8$ cube. The orientation data was clustered such that we could cover 97% of the data using 1160 nodes. We then described how we replaced the blue channel of the RGB-D images with the depth channel to arrive at a 3 channel feature map that we could feed into a series of networks using pre-trained ResNet50 layers. We then proposed two different ways to train these neural networks. One loss function could be used directly with our collected dataset, but might over penalize potentially successful grasps that we did not sample in our data. The other loss function would avoid this penalty but it could only train one partition at a time and it would require access to a source of negative examples. We created a source of negative examples by implementing the first half of Bohg et al.'s [7] pipeline, and substituting the second half with a simple convex mesh and a simple grasp planning heuristic. We then describe our approach to take advantage of the strong prior on the distribution of the partitions used to train the position network. We decided to mimic Zeng et al.'s [57] idea of residual physics and have our network learn an additive correction on the prior. We then described all the different network architectures we attempted to use to solve this problem. We found many problems trying to find networks that would not overfit, before eventually settling on using a network using pre-trained layers from ResNet50 and with a branching structure for each of the different levels of the z axis of the $8 \times 8 \times 8$ cube of partitions. We used a similar structure for the orientation network as well. Our results for these networks were not promising. Testing on the training set, we found that the position network had collapsed to only predict a few values with very high probability. This in turn would carry over to the orientation network and also cause it to predict a small number of different orientations. Even then, we found that the position network was performing slightly better than what we would expect from the prior alone on our validation set. Testing on the robot did not yield good results either. The system picks a small range of poses regardless of the object presented. The majority of grasps we tested resulted in failure, even when testing multiple times on objects that had been successfully grasped at least once, leading to a much lower success rate than the 80% we aimed for.

We conclude then on mixed results. We managed to collect a large dataset of human demonstrated grasps. These grasps seem to be robust to a wide variety of perturbations. However, we found training a neural network on this dataset to be a difficult task. One of the reasons we hypothesize learning this task is difficult is due to the curse of dimensionality. Predicting fully 6D poses might not be something that is feasible with our dataset. Other works use large synthetic datasets of tens to hundreds of thousands of grasps [39, 36, 28, 32, 57]. Even with this amount of data, some of them learn pose representations that are less than six dimensions [57, 36]. Yet, Hezrog et al. [24] manage to learn a flexible system using less than twenty demonstrations. The discrepancy might come from the feature representation that they use for their system. Calculating grasp heightmaps involve complicated, non-linear transformations that might not be the kind of approach the loss gradients guide our networks towards.

Potential future areas of work would involve continue to examine what kinds of neural network architectures can learn from our dataset, or what kinds of representations can be used to encourage our networks to learn to grasp.

Bibliography

- [1] G.B. Arfken, H.J. Weber, and F.E. Harris. *Mathematical Methods for Physicists*. Elsevier, 2005.
- [2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
- [3] A.P. Arya. *Introduction to Classical Mechanics*. Prentice Hall international editions. Prentice-Hall International, 1998.
- [4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [5] R.S. Ball. *The Theory of Screws: A Study in the Dynamics of a Rigid Body*. Hodges, Foster, and Company, 1876.
- [6] Mary L Boas. *Mathematical methods in the physical sciences; 3rd ed.* Wiley, Hoboken, NJ, 2006.
- [7] Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergström, Danica Kragic, and Antonio Morales. Mind the gap-robotic grasping under incomplete observation. In *2011 IEEE International Conference on Robotics and Automation*, pages 686–693. IEEE, 2011.
- [8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [9] Shehan Caldera, A. M. Rassau, and Douglas Chai. Review of deep learning methods in robotic grasp detection. 2018.
- [10] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, July 2015.
- [11] Francois Chollet. Keras. <https://github.com/fchollet/keras>, 2015.
- [12] Fu-Jen Chu, Ruinian Xu, and Patricio A. Vela. Deep grasp: Detection and localization of grasps with deep neural networks. *CoRR*, abs/1802.00520, 2018.

- [13] Charles de Granville, Joshua Southerland, and Andrew H Fagg. Learning grasp affordances through human demonstration. In *Proceedings of the International Conference on Development and Learning (ICDL06)*, 2006.
- [14] T. Feix, J. Romero, HB Schmiedmayer, A.M. Dollar, and D. Kragic. The grasp taxonomy of human grasp types. In *Human-Machine Systems, IEEE Transactions on*, 2015.
- [15] C. Ferrari and J. Canny. Planning optimal grasps. *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 2290–2295, 1992.
- [16] David Fischinger, Astrid Weiss, and Markus Vincze. Learning grasps with topographic features. *The International Journal of Robotics Research*, 34, 05 2015.
- [17] I.M. Gelfand. *Representations of the rotation and Lorentz groups and their applications*. (Pergamon Press book). Pergamon Press, 1963.
- [18] H. Goldstein, C.P. Poole, and J.L. Safko. *Classical Mechanics*. Addison Wesley, 2002.
- [19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [20] Abhishek Gupta, Clemens Eppner, Sergey Levine, and Pieter Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3786–3793. IEEE, 2016.
- [21] Hideo HANAFUSA and Haruhiko ASADA. Stable prehension of objects by the robot hand with elastic fingers. *Transactions of the Society of Instrument and Control Engineers*, 13(4):370–377, 1977.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [23] Micha Hersch, Florent Guenter, Sylvain Calinon, and Aude Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467, 2008.
- [24] Alexander Herzog, Peter Pastor, Mrinal Kalakrishnan, Ludovic Righetti, Jeannette Bohg, Tamim Asfour, and Stefan Schaal. Learning of grasp selection based on shape-templates. *Auton. Robots*, 36(1-2):51–65, January 2014.
- [25] Intel. Sr300 product datasheet. <https://software.intel.com/sites/default/files/managed/0c/ec/realsense-sr300-product-datasheet-rev-1-0.pdf>, Jun 2016. [Online; accessed 9-July-2019].
- [26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

- [27] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed `today`].
- [28] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. Leveraging big data for grasp planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311. IEEE, 2015.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.
- [32] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.
- [33] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *CoRR*, abs/1801.05134, 2018.
- [34] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [35] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9605–9616. Curran Associates, Inc., 2018.
- [36] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *CoRR*, abs/1703.09312, 2017.
- [37] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David V. Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. *CoRR*, abs/1709.06670, 2017.

- [38] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019.
- [39] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964. IEEE, 2016.
- [40] B. Mishra, J.T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 1987.
- [41] Douglas Morrison, Adam W Tow, M McTaggart, R Smith, N Kelly-Boxall, S Wade-McCue, J Erskine, R Grinover, A Gurman, T Hunn, et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7757–7764. IEEE, 2018.
- [42] Kiyoshi Nagai and Tsuneo Yoshikawa. Dynamic manipulation/grasping control of multifingered robot hands. volume 30, pages 1027 – 1033 vol.3, 06 1993.
- [43] Christopher Olah. Neural networks, manifolds, and topology, Apr 2014.
- [44] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [45] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1):145–151, January 1999.
- [46] J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1316–1322, May 2015.
- [47] RightHand Robotics. Reflex robotic gripper tech specs. <https://www.labs.righthandrobotics.com/reflexhand>. [Online; accessed 9-July-2019].
- [48] Universal Robots. Ur5 technical specifications. https://www.universal-robots.com/media/50588/ur5_en.pdf, September 2016. [Online; accessed 10-July-2019].
- [49] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.
- [50] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2010.
- [51] Gordon Smith, Eric Lee, Ken Goldberg, Karl Bhringer, and John Craig. Computing parallel-jaw grip points. 1999.

- [52] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.
- [53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [54] StereoLabs. Stereolabs zed tech specs. <https://www.stereolabs.com/zed/>. [Online; accessed 10-July-2019].
- [55] Jacob Varley, Chad DeChant, Adam Richardson, Avinash Nair, Joaquín Ruales, and Peter K. Allen. Shape completion enabled robotic grasping. *CoRR*, abs/1609.08546, 2016.
- [56] Yibo Wang and Wei Xu. Leveraging deep learning with lda-based text analytics to detect automobile insurance fraud. *Decision Support Systems*, 105:87–95, 2018.
- [57] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *arXiv preprint arXiv:1903.11239*, 2019.

Appendix A

Mathematical Background

As seen in Section 2.1 a variety of mathematical tools are required for the analysis of the different problems present in grasp planning. This section provides a brief introduction to the most important concepts required for understanding the work and results presented here. Most of the material is referenced from [50], [19], [1].

A.1 Rotation Formalisms in \mathbb{R}^3

There are a number of standard ways to express rotations in 3D space. Each of these formalisms has some advantages and some disadvantages. All the formalisms presented here were used during the course of this work. The main motivation for using rotation formalisms is to encode the relative rotation between different frames of reference throughout the experiments that will be presented later. An example, to fully describe the pose of a robotic gripper it is necessary to capture information about both its location and its orientation, which is most easily done by imagining that the gripper has its own reference frames and comparing to some standard frame.

In the following sections, rotation matrices will be introduced first along with some general rotation properties, followed by rotation quaternions. These two formalisms are the most used both for their interpretability and their ease of use. Rotation vectors will also be covered, but since their main strength is their compactness they are generally not widely used in applications.

A.1.1 Rotation Matrices

Rotation matrices are generally most people's only rotation formalism. They are easy to teach along with other linear transformations in introductory linear algebra courses. A rotation matrix, R , obeys three conditions:

- $R_{ij} \in \mathbb{R}$
- $R^{-1} = R^T$
- $\det(R) = 1$

These three conditions are where the general group of rotations, the Special Orthogonal 3 group (SO(3)) [17], gets its name from. All the rotation formalisms discussed here are representations of this group. There are three elementary rotation matrices in \mathbb{R}^3 , one for each coordinate axis [6].

$$\bullet R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\bullet R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\bullet R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The subscript for each of these matrices indicate which axis to rotate about. Applying $R_z(\theta)$ to a column vector would rotate the x and y components but leave the z component the same. Furthermore, as sanity check, each of the rotation matrices can be obtained from any other by sliding their components along the diagonal. For example, R_y can be obtained by shifting all the elements of R_x down and to the right, where any elements that end up outside the matrix wrap around.

From composition of these elementary matrices it is possible to arrive at any arbitrary rotation matrix. Euler angles are a way to arrive at any arbitrary rotation by using at most three elementary matrices [3]. There are two families of Euler angles: intrinsic and extrinsic. Intrinsic Euler angles compose their rotations with respect to the rotated frames. Extrinsic Euler angles compose their rotation with respect to some fixed world frame. The possible combinations of elementary rotation matrices is finite; there are 6 intrinsic combinations and 6 extrinsic combinations [18].

- Intrinsic: x-y-x, x-z-x, y-x-y, y-z-y, z-x-z, z-y-z
- Extrinsic: x-y-z, x-z-y, y-x-z, y-z-x, z-x-y, z-y-x

Taking z-x-z intrinsic combination, an arbitrary rotation matrix can be built by the following composition $R_z(\theta)R_x(\phi)R_z(\psi)$, where θ, ϕ, ψ are the Euler angles. Notice that the last rotation, the ψ rotation, is the rightmost matrix. This is because the rightmost matrix will align the axes for the second matrix and so forth. In essence, the order of the matrices will be the same as the order given above. For extrinsic combinations, like x-y-z, the order should be the reverse of what is shown above.

There is another way to build up rotation matrices. Imagine that two coordinate frames are given and that it is required to know the rotation matrix that can transform from one frame to the other. To build out this rotation matrix a method known as direction cosines can be used [1]. The idea follows from an understanding of how to build general transformation matrices using the projection operator.

Some notation needs to be introduced at this point for clarity. The unit vector \hat{x}_B^A is the x-unit vector of Frame B, written with respect to Frame A. That is, the superscript indicates which frame is being referenced, while the subscript indicates the original frame for that vector. \hat{x}_A^A would be $[1, 0, 0]^T$, but \hat{x}_B^A will be some linear combination of Frame A's basis vectors. The amount of \hat{x}_B^A lying along \hat{x}_A^A can be found by using the projection operator, which for unit vectors is just a dot product. Then it is possible to write \hat{x}_B^A , or any of frame B's axes, in terms of the basis vectors of frame A.

$$\hat{x}_B^A = (\hat{x}_B^A \cdot \hat{x}_A^A)\hat{x}_A^A + (\hat{x}_B^A \cdot \hat{y}_A^A)\hat{y}_A^A + (\hat{x}_B^A \cdot \hat{z}_A^A)\hat{z}_A^A \quad (\text{A.1})$$

$$\hat{y}_B^A = (\hat{y}_B^A \cdot \hat{x}_A^A)\hat{x}_A^A + (\hat{y}_B^A \cdot \hat{y}_A^A)\hat{y}_A^A + (\hat{y}_B^A \cdot \hat{z}_A^A)\hat{z}_A^A \quad (\text{A.2})$$

$$\hat{z}_B^A = (\hat{z}_B^A \cdot \hat{x}_A^A)\hat{x}_A^A + (\hat{z}_B^A \cdot \hat{y}_A^A)\hat{y}_A^A + (\hat{z}_B^A \cdot \hat{z}_A^A)\hat{z}_A^A \quad (\text{A.3})$$

After some inspection, it should be obvious that the coefficients above correspond to the elements of a transformation matrix. This matrix is given as follows:

$$R_B^A = \begin{pmatrix} (\hat{x}_B^A \cdot \hat{x}_A^A) & (\hat{x}_B^A \cdot \hat{y}_A^A) & (\hat{x}_B^A \cdot \hat{z}_A^A) \\ (\hat{y}_B^A \cdot \hat{x}_A^A) & (\hat{y}_B^A \cdot \hat{y}_A^A) & (\hat{y}_B^A \cdot \hat{z}_A^A) \\ (\hat{z}_B^A \cdot \hat{x}_A^A) & (\hat{z}_B^A \cdot \hat{y}_A^A) & (\hat{z}_B^A \cdot \hat{z}_A^A) \end{pmatrix} \quad (\text{A.4})$$

It is easy to show that this matrix is orthogonal. Since the basis vectors in each reference frame must be orthonormal, and each row is the projection of one of these vectors, then the elements of $R^T R$ will be the dot products of each row with every other row, which leads to the identity matrix.

It is also easy to show that this matrix has determinant 1. The determinant of a 3×3 matrix can be found via the vector triple product: $\det(R) = \vec{c}_1 \cdot (\vec{c}_2 \times \vec{c}_3)$, where \vec{c}_i are the column vectors of R . Since \vec{c}_i are orthonormal, $\vec{c}_2 \times \vec{c}_3$ must be a multiple μ of \vec{c}_1 , where $\mu > 0$ because \vec{c}_i form a right handed coordinate system. Then:

$$\begin{aligned} \mu|\vec{c}_1| &= |\vec{c}_2 \times \vec{c}_3| \\ \mu|\vec{c}_1| &= |\vec{c}_2||\vec{c}_3|\sin(\theta_{23}) \\ \mu &= 1 \end{aligned}$$

In this case, $\theta_{23} = \frac{\pi}{2}$ since \vec{c}_i form a right hand coordinate system, and the magnitude of \vec{c}_i is 1 since they are unit vectors. Since $\mu = 1$, the determinant is also 1. Since the direction cosine matrix obeys the three conditions required for a rotation matrix, it must be a rotation matrix.

In summary, there are many different ways to build rotation matrices. The two most important ones are using Euler angles, and direction cosine matrices. Rotation matrices are used often in applications mainly due to their ease of composition and application. Composing two rotations is as easy as matrix-matrix multiplication, while applying a rotation to a vector is matrix-vector multiplication. The main drawback is their size, relative to some of the other representations that are available, and that they can be hard to interpret.

A.1.2 Rotation Vectors

Rotation vectors are not as popular as rotation matrices, but they have some advantages. The easiest way to describe what a rotation vector is, is to go back to rotation matrices.

A rotation matrix will always have only one +1 eigenvalue. This eigenvalue corresponds to the eigenvector that is not changed at all by the rotation matrix. Intuitively, this vector must lie along the axis of rotation. The other two eigenvalues will be complex conjugates encoding the magnitude of the rotation, $e^{\pm i\theta}$. This means that the magnitude of rotation can be found through the trace by:

$$\begin{aligned} \text{Tr}(R) &= 1 + e^{i\theta} + e^{-i\theta} \\ \text{Tr}(R) &= 1 + 2 \cos(\theta) \\ \theta &= \arccos\left(\frac{\text{Tr}(R) - 1}{2}\right) \end{aligned}$$

Given the axis of rotation (three numbers) and the magnitude of the rotation is enough to reconstruct any arbitrary rotation. Often, the magnitude of the rotation is encoded in the length of the rotation axis vector. This is known as a rotation vector.

To apply a rotation using a rotation vector it is useful to decompose it into a unit vector, \hat{k} , and magnitude, θ . Then Rodrigues' rotation formula can be applied to any vector $\vec{v} \in \mathbb{R}^3$.

$$\vec{v}_{rot} = \vec{v} \cos(\theta) + (\hat{k} \times \vec{v}) \sin(\theta) + (\hat{k} \cdot \vec{v}) (1 - \cos(\theta)) \hat{k} \quad (\text{A.5})$$

However, the mapping is not unique. Any rotation (\hat{k}, θ) can also be achieved by $(\hat{k}, n \cdot \theta)$ or by $(-\hat{k}, n \cdot (2\pi - \theta))$ for integer n . For this reason, most people constrain rotation vectors such that $\theta \in (0, \pi]$.

There are two advantages in using rotation vectors, they occupy the smallest space possible, and they are the easiest formalism to visualize at a glance. Using only three floats it stores the same information a rotation matrix would store in nine. If the axis direction and the magnitude are kept separate, it is also easier to visualize what the rotation would do to a vector it is being applied on. The biggest disadvantage is that it is not easy to compose two rotations together. As a result, rotation vectors were mostly used for storage during the course of the work presented here. The space savings are not necessarily significant, 1.44MB in rotation matrices versus 480KB in rotation vectors over forty thousand grasps, but this coupled with the ability to quickly tell what a rotation is supposed to look like made troubleshooting and understanding the dataset much easier.

A.1.3 Rotation Quaternions

Rotation quaternions are another way of mathematically encoding rotations. Quaternions can be seen as an extension of complex numbers. They correspond then to four different kinds of objects: scalars, \mathbf{i} , \mathbf{j} , \mathbf{k} . The latter three obey the following relationship.

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \quad (\text{A.6})$$

A general quaternion is then of the form $q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ for $a, b, c, d \in \mathbb{R}$. Adding quaternions follows the same rules as regular addition, while multiplication must obey equation A.6 and is non-commutative. Conjugate quaternions can be calculated by analogy to the complex case, that is flipping the sign on the non-scalar components. The norm of a quaternion can also be found by analogy to the complex case as $\|q\| = \sqrt{qq^*}$.

All quaternions of unit length represent a rotation in \mathbb{R}^3 , which is why they are sometimes called rotation quaternions. There are many different proofs for how to build a mapping from unit quaternions to $\text{SO}(3)$ and many more proofs explaining the construction of rotation quaternions, but for the sake of brevity only a mechanistic explanation is given here. For more details see [18], [1].

To specify a rotation, a rotation axis (\hat{u}) and a magnitude (θ) are needed. Given both of those, a rotation quaternion can be built in the following way:

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right) \hat{u} \quad (\text{A.7})$$

$$\hat{u} = u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k} \quad (\text{A.8})$$

It is important to remember at this point that $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are still the complex numbers defined above and not cartesian unit vectors. To apply a rotation to a vector \vec{v} , first one must convert the vector to a pure quaternion $\vec{v} = 0 + v_1\mathbf{i} + v_2\mathbf{j} + v_3\mathbf{k}$ and then apply the following equation.

$$\vec{v}_{rot} = \mathbf{q}\vec{v}\mathbf{q}^* \quad (\text{A.9})$$

The resulting rotated vector will also be a pure quaternion. Special care must be taken to follow the commutator relations since the quaternion multiplication is not commutative. To compose two different rotations, all that must be done is to multiply the two rotation quaternions together. Again since quaternions are not commutative, the order of multiplication matters.

From the description of how to accomplish rotations with quaternions it should be obvious that they suffer from the same mapping problems as the rotation vectors, that is for every rotation there are an infinite number of ways to encode them. For that reason rotation quaternions generally are restricted to $\theta \in [0, \pi)$.

Rotation quaternions are usually taken to be the gold standard for rotations. They are easy to apply to arbitrary vectors by multiplication and can also be easily composed in the same way. Unlike rotation matrices they only require four numbers to represent a rotation as opposed to nine. Even though this is one more than rotation vectors, it is still quite efficient and is a decent trade-off for how easy they are to apply. The main disadvantages to rotation quaternions are how involved understanding the concepts can get, and that it is not easy to visualize what rotation any individual rotation quaternion represents.

A.2 Homogenous Transformations Matrices

Throughout the course of this work there is a requirement to interpret vectors recorded in one reference frame with respect to a different reference frame. The two reference frames

might differ in both the orientation of their axes, and the location of their origins. It is possible to perform the change of frame using the following equation:

$$\vec{p}^1 = \vec{o}_0^1 + R_0^1 \vec{p}^0 \quad (\text{A.10})$$

In this equation, \vec{p}^i is used to denote the vector \vec{p} with respect to reference frame i , \vec{o}_0^1 is the vector that goes from the origin of reference frame 0 to reference frame 1, and R_0^1 is the rotation matrix that takes a vector from the orientation of reference frame 0 to the orientation of reference frame 1. So in this equation, first the vector in reference frame 0 is rotated so that its orientation is consistent with frame 1, then the vector is translated so that its origin is consistent with frame 1. If multiple such transformations need to be completed, this equation is easily composed but becomes messy. It is possible to encode all the relevant information required for this transformation in a 4×4 matrix [50].

$$H = \begin{pmatrix} r_{11} & r_{12} & r_{13} & o_1 \\ r_{21} & r_{22} & r_{23} & o_2 \\ r_{31} & r_{32} & r_{33} & o_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{3 \times 3} & \vec{o}_{3 \times 1} \\ \vec{0}_{1 \times 3} & 1 \end{pmatrix} \quad (\text{A.11})$$

This method requires augmenting vectors to homogenous coordinates. Homogenous coordinates are used throughout computer vision and robotics. The easiest way to transform a vector to homogenous coordinates is to add an extra unit element. That is, $\vec{v} = [a \ b \ c]^T$ becomes $\vec{v} = [a \ b \ c \ 1]^T$. By augmenting input vectors to homogenous coordinates, it is possible to use the matrix in A.11 to perform the coordinate transformation. The resulting vector will also be in homogenous coordinates, but it is easy to return to \mathbb{R}^3 by dividing the first three elements of the vector by the fourth and then discarding the last element.

The inverse transformation is easy to find by inverting equation A.10. The equivalent homogenous matrix can then be read from the equation.

$$\vec{p}^0 = -(R_0^1)^T \vec{o}_0^1 + (R_0^1)^T \vec{p}^1 \quad (\text{A.12})$$

$$H^{-1} = \begin{pmatrix} (R_{3 \times 3})^T & -(R_{3 \times 3})^T \vec{o}_{3 \times 1} \\ \vec{0}_{1 \times 3} & 1 \end{pmatrix} \quad (\text{A.13})$$

The final element required is the ability to compose coordinate transformations. This can easily be done through matrix multiplication. Let there be 3 reference frames A, B, and C. Analogously to the rotation matrices, let H_B^A be the homogenous transformation matrix that takes a vector from frame B to frame A. Then let H_C^A be the matrix that takes a vector from frame C to frame A. If these are the only two matrices available, then it is still possible to find H_C^B by first going through frame A:

$$H_C^B = (H_B^A)^{-1} H_C^A \quad (\text{A.14})$$

This composition rule can be extended to as many homogenous matrices as are needed.

Homogenous transformation matrices form the backbone of most of the math undertaken during this project. The ability to change the perspective through which a vector is seen

is incredibly important, especially as it pertains to identifying the pose of the robotic gripper with respect to the various cameras used throughout the project. The most important concepts to take forward from this section are the facts that constructing these matrices requires knowledge of location and orientation of frames relative to each other, finding inverses is relatively straightforward, and that composition is as simple as matrix multiplication.

A.3 Deep Learning

After Alex Krizhevsky’s team won ImageNet’s Large Scale Visual Recognition Challenge in 2012 using deep convolutional neural networks, research on these techniques has greatly increased. Deep neural networks promise that by non-linearly combining simple stages (depth) task relevant information can be more easily extracted.

This subsection will serve as a primer and refresher of important topics in deep learning. It will not cover all topics however. In recent years Goodfellow et al. co-authored a comprehensive textbook introduction to the field [19]. This book is an excellent resource to delve into some of the details that will be skipped here.

For the purposes of the grasping project presented here, a brief discussion on multi-layer perceptrons and hyperparameters will be followed by summaries on convolutional neural networks and residual networks.

A.3.1 Multi-layer Perceptron

A perceptron is the first kind of neural network envisioned. This network only had two layers: an input layer and an output layer. Traditionally, perceptrons are used as binary classifiers, and importantly they represent a class of linear classifiers.

Implementing a perceptron is easy using matrix-vector multiplication. Let W represent the weight matrix, i.e. the connections between the input and output nodes. This means that W is size $n \times m$ where n is the output size, and m is the input size. Without too much effort it is also possible to add a bias term, \vec{b} . This term adds an extra degree of freedom that is often useful to classify data. The full perceptron equation is then:

$$o = W\vec{v} + b \tag{A.15}$$

The learning occurs by correcting the weight matrix based on the error observed at the output. This is done by measuring the performance of the model using a loss function, and then shifting the weights in the direction that will minimize the loss function. For binary classification problems, binary cross entropy is usually the loss function that is used.

$$BCE(t, o) = \begin{cases} -\log(1 - o) & \text{if } t = 0 \\ -\log(o) & \text{otherwise} \end{cases} \tag{A.16}$$

$$= -(1 - t) \log(1 - o) - t \log(o) \tag{A.17}$$

Where $t \in \{0, 1\}$ is the target value, and o is the perceptron output. The second equation above is merely a clever way to write an if-statement since t is a binary value. The overall loss function is then $\sum_i BCE(t_i, o_i)$.

The linearity condition means that this correction is easy to carry out computationally with a bit of vector calculus. However, the linearity condition means that non-linear tasks are intractable. The most famous example of this is the XOR problem [19].

The multi-layer perceptron (MLP) is the extension to the perceptron to deal with non-linear problems. Under this formulation a number of hidden layers are added between the input and output layers. The term hidden comes from the fact that the outputs of these layers are not observed, and are therefore hidden from view. It is not enough to just add more layers, since computationally this would just collapse to the same matrix that would be formed with a perceptron. To make the MLP non-linear, a non-linear function is added after each layer. Importantly the non-linear function needs to be continuous, and its first derivative needs to be at least piece-wise continuous so that the correction to the weights can be calculated using vector calculus.

The actual correction is calculated using the backpropagation algorithm. The algorithm uses automatic differentiation and differential calculus' chain-rule to recursively calculate the derivative of the error with respect to each of the weights. Details about how this algorithm can be implemented can be found in [19].

Research into what kinds of features MLPs learn in their hidden layers is ongoing. A decent intuition to carry forward is that the hidden layers non-linearly extract or combine features to build representations that can then be linearly classified by the weights connecting the last hidden layer to the output layer [43]. Manipulating the architecture of the network, by changing the number of layers, the size of the layers or even the connections between layers, can affect the kind and the effectiveness of the representations learned in the hidden layers.

Another way to affect the representations learned by the MLP is to manipulate the cost function to reward or penalize different aspects of the network. An example of such a change is to add an L^2 penalty on the size of the weights in the MLP. Such a penalty encourages the network to learn representations that give less weight to uncommon features in the training data [19].

A.3.2 Convolutional Neural Networks

Convolutional neural networks [31] (CNNs) are a special network architecture inspired by the primate visual network to deal with computer vision tasks. Convolutional layers are inspired by neurons in the V1 area of macaque monkeys. Neurons in this area react to particular visual features in the visual (receptive) field of the monkey. Importantly, even though different neurons cover different receptive fields, some of the neurons look for the same visual features, just in different parts of the visual field.

Convolutional layers (conv layers) aim to learn useful visual features as two dimensional filters that can be applied across the entire image. Like the neurons in the macaque visual system, only a small set of filters need to be learned at each layer. Composing convolutional layers means that the network can learn filters that build on top of more basic filters. In

this way, adding depth to a CNN can lead to a complex feature detector.

Traditionally, a convolutional layer is followed by a max pooling layer. These layers use small receptive fields, and reduce the size of the feature maps by only passing through the maximum value in the receptive field. The most common operation is to apply a 2×2 max pooling filter which would reduce the dimension of the feature maps by a factor of 2 in both height and width. These layers are also inspired by the primate visual stream. The neuron spikes generated by simple cells in V1 are received by complex cells and are pooled by them to make complex cells invariant to phase-shifts. These complex cells then provide (along with the simple cells) inputs for later layers.

Finally, after stacking a set number of conv and max-pool layers, a dense set of connections are used to make a prediction. This dense set of connections is merely a set of fully connected hidden layers like those found in an MLP. A useful way to think about this architecture is to think that the convolutional and max pooling layers act as efficient feature extractors that feed useful features into an MLP. Importantly, the features are chosen to minimize the cost function and are not decided *a priori*, which means that network can freely chose whatever features are most useful for the task at hand.

CNN architectures have proven to be incredibly efficient computer vision tools. Starting with AlexNet [30] in 2012, CNN architectures and their variants have consistently been pushing state of the art performance in various benchmarks. One particular variant used in this work the extension from two dimensional inputs to three dimensional inputs. In this case, the conv layer learns 3D filters (volumes) instead of 2D filters. Since the input is also three dimensional, the 3D filters can be applied in an analogous manner to the 2D case. This allows CNNs to be used in a wider set of fields than just image processing.

A.3.3 Residual Networks

Residual networks [22] (resnets) are another kind of network architecture. Residual networks are characterized for their use of skip or residual connections. These are connections between layers that skip one or more layers and are combined with the next layer through addition of features. Skip connections are used to combat a common problem in neural networks known as vanishing gradients.

Vanishing gradients tend to occur when networks are made to ‘be very deep. When a network is very deep, and the weights are randomly initialized, it is not clear what kind of low level features should be learned in the early layers of the network. As a result, these networks train very slowly, if at all. Mathematically, what is occurring is that the backpropagation algorithm assigns weight changes depending on the gradient of the loss function with respect to each individual weight. As the algorithm progresses recursively it is possible for the gradient with respect to early layers to have been multiplied by small magnitude values repeatedly to the point that the gradient “vanishes”. Since the gradients are so small, the network has a hard time training.

Skip connections help solve this problem by making a short path through which gradients can flow and help update early network layers. Another way to interpret skip connections is to realize that they can be seen as successive approximations from the input to the output. Residual networks have been shown to train even when they are very deep, hundreds to thousands of layers deep. The added depth allows for more complicated features to be

learned. Combining convolutional layers with residual connections leads to high performing neural networks for solving computer vision tasks.

A.3.4 Other Layers

There are many other kinds of specialized layers that are used throughout the literature. Presented here are ones that were used during the course of the project.

The BatchNorm layer [26] is a special layer used to correct internal covariance shift. During training, the distribution of activations for each hidden layer changes as the weights are updated. These shifts in the distribution of activations can make it harder for subsequent layers to train until the representations in the earlier layers start to stabilize. A BatchNorm layer normalizes the mean and variance of the distribution, and can also learn to scale and shift the distribution so that the new mean and variance helps subsequent layers in the network solve its task. Ioffe and Szegedy [26] demonstrate that using BatchNorm allows them to safely increase the learning rate of their networks leading to much faster training. In our work, we follow what is now a standard and add BatchNorm after every layer, unless we are using a Dropout layer.

A Dropout [53] layer is a stochastic regularization technique that randomly drops some of the nodes from the previous layer, by setting all their outputs to zero. The intuition behind this is to force the network to distribute its representations across the network since any one node might be lost at any time during training. Mathematically, Srivastava et al. claim that this is equivalent to approximating training exponentially many thin subnetworks and then combining their outputs at test time. The ratio of dropped nodes is set as a parameter of the network, p , with a typical value of 0.5. Recently, Li et al. [33] have explored the incompatibilities between Dropout and BatchNorm. In general, it is not recommended to mix both layers as Dropout will stochastically change the variance of the mini-batch, which as BatchNorm learns to correct during training leads to worse performance at test time. In our work, if BatchNorm is not being used, Dropout is used instead.

A.3.5 Hyperparameters

Finally there are a series of hyperparameters that need to be set or chosen while training neural networks. There are many hyperparameters that can be set, but for the purposes of this work, only a few are presented here.

The first hyperparameter usually introduced is the networks learning rate. The learning rate is the fraction of the correction from the gradient of the loss function, L , that will be applied to the weights when they are updated. The general update equation is given as:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}} \tag{A.18}$$

In this equation η is known as the learning rate, which is in \mathbb{R}^+ , though usually values in the range of 10^{-6} to 10^{-3} are used. The choice of the learning rate is important because it effectively provides a trade-off between speed of training and accuracy of inference. A low learning rate will take longer to train, as the network has to take more steps to get to a minimum, but it might reach a lower minimum than a higher learning rate. Low learning

rates might also more easily get stuck in gradient plateaus leading to long periods of almost no improvement. A high learning rate might take steps that are too big and never converge to a reasonable minimum. Tuning the learning rate for the particular architecture and task is an important step for training a neural network.

The next step would be to choose the kind of optimizer to be used. The classical optimizer to use is stochastic gradient descent (SGD), other optimizers tend to be modifications on top of SGD. SGD proceeds by first approximating the gradient with respect to the weights using a random sample from the training set and then using that sample to correct the weights (minimize the loss). Over time the entire training set is sampled and then the process starts again. This algorithm is easy to implement and is also surprisingly robust. It does require a bit more fine tuning; choosing the right learning rate becomes really important. A common addition to SGD is momentum [45]. This is an extra term to the update rule, with a corresponding parameter in $(0, 1]$, that keeps track of which direction past corrections to the weights were made in. This exponentially decaying memory allows the network to quickly move through gradient plateaus, or shallow local minima.

A more modern optimizer would be Adam [29]. Adam is built on top of stochastic gradient descent adding a per weight learning rate and updating these learning rates based on the magnitude of previous gradient approximations. This means that the network as a whole trains much faster as weights that need to change by a large amount will have higher learning rates than ones that only need a small correction. There is still a learning hyperparameter to set with Adam, which chooses the starting magnitude that is effectively changed as the optimization progresses. Adam adds two other parameters though: β_1 and β_2 . β_1 is used to set the exponential decay on the approximation of the mean of the gradients, while β_2 sets the exponential decay for variance of the gradients. These two parameters are sensitive to small changes, and it is in general recommended to use the default values in most deep learning packages ($\beta_1 = 0.9$, $\beta_2 = 0.999$).

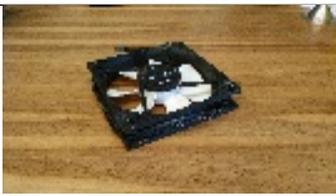
Regularizers are a common tool in machine learning. In the context of neural networks regularizers are implemented by adding an extra term to the loss function penalizing unwanted behaviours [19]. The most common kind of regularizer is applying an L^2 norm penalty to the magnitude of the weights. This encourages the network to avoid overfitting by using all the parameters more efficiently. Another common weight based penalty is the L^1 norm which encourages some of the weights to go to zero (sparsity). Both of these can be applied across the whole network or on specific layers. Regularizers can also be applied to the hidden layer activations. To control the strength of the regularizer, new hyperparameters are added to multiply with the penalties. Typical values are around 10^{-4} to 10^{-2} .

The final hyperparameter in this section will be the clipping ceiling. As a result of the weight update rule, it is possible for situations to occur where the optimizer can be stuck due to taking steps that are too large [19]. The classical example is encountering a gradient cliff. In this situation a plateau in gradient space is followed by a steep change in the gradient. Since the weight correction is based on the magnitude of the gradient, sometimes the gradient cliff will push the weights to over-correct and undo some of the progress that has already been made. Lowering the learning rate does not in general fix this issue since the cliffs could be very steep and it is not known beforehand how steep they might be. Additionally, lowering the learning rate slows learning as a whole. The solution is to clip weight corrections above a certain value, i.e. to clip the gradients. If the gradients behave nicely, the clipping will

not be triggered, but if the gradient is too large then it will be replaced with a predefined value and hopefully not undo some of the training. Clipping is not usually necessary, but an easy way to tell if it is needed is to look at the values of the gradients as the network trains. Alternatively, seeing spikes in the training loss is another sign that cliffs might be present.

Appendix B

Objects Used for Grasping

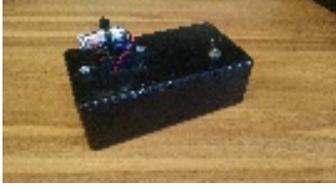
| Object Number | Object Name | Object Image |
|---------------|--------------|--|
| 1 | 120mmFan |  |
| 2 | aluminumFoot |  |
| 3 | arduinoBox |  |
| 4 | baseball |  |
| 5 | batteryPack |  |

| Object Number | Object Name | Object Image |
|---------------|------------------|--|
| 6 | blackFlashlight |  |
| 7 | blackboarderaser |  |
| 8 | blueJug |  |
| 9 | blueLid |  |
| 10 | blueSpool |  |
| 11 | blueTape |  |
| 12 | blueduploarch |  |
| 13 | carpetTape |  |

| Object Number | Object Name | Object Image |
|---------------|-------------------|--|
| 14 | casterBag |  |
| 15 | chipsCan |  |
| 16 | clamp |  |
| 17 | creativeBox |  |
| 18 | cylindricalSpacer |  |
| 19 | dLinkBox |  |
| 20 | duploCart |  |
| 21 | duploEye |  |

| Object Number | Object Name | Object Image |
|---------------|------------------|--|
| 22 | emptySpool |  |
| 23 | evenMetalShape |  |
| 24 | evenPlasticShape |  |
| 25 | eyeBox |  |
| 26 | flashlight |  |
| 27 | foamBrick |  |
| 28 | foamCorner |  |
| 29 | glueStick |  |

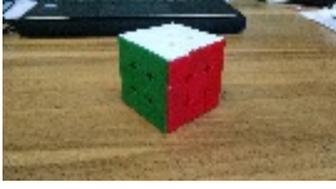
| Object Number | Object Name | Object Image |
|---------------|---------------|--|
| 30 | goPro |  |
| 31 | goProRemote |  |
| 32 | golfBall |  |
| 33 | gripTape |  |
| 34 | headphoneCase |  |
| 35 | innerFanTube |  |
| 36 | kleenexBox |  |
| 37 | labjack |  |

| Object Number | Object Name | Object Image |
|---------------|----------------|--|
| 38 | largeBlueCup |  |
| 39 | largeOrangeCup |  |
| 40 | largeScrewTray |  |
| 41 | largeYellowBox |  |
| 42 | laserBox |  |
| 43 | masterChefCan |  |
| 44 | measuringTape |  |
| 45 | medRedBox |  |

| Object Number | Object Name | Object Image |
|---------------|---------------|--|
| 46 | mediumTimmies |  |
| 47 | mustardBottle |  |
| 48 | myo |  |
| 49 | myoCase |  |
| 50 | orangeBox |  |
| 51 | outerFanTube |  |
| 52 | pipetteBulb |  |
| 53 | plasticApple |  |

| Object Number | Object Name | Object Image |
|---------------|--------------------|--|
| 54 | plasticBanana |  |
| 55 | plasticBox |  |
| 56 | plasticLemon |  |
| 57 | plasticOrange |  |
| 58 | plasticPeach |  |
| 59 | plasticPear |  |
| 60 | plasticPlaneBody |  |
| 61 | plasticPlaneWheels |  |

| Object Number | Object Name | Object Image |
|---------------|-------------------|--|
| 62 | plasticPlum |  |
| 63 | plasticRedCup |  |
| 64 | plasticSleeve |  |
| 65 | plasticSpiral |  |
| 66 | plasticStrawberry |  |
| 67 | puddingBox |  |
| 68 | purpleCup |  |
| 69 | purpleDuploArch |  |

| Object Number | Object Name | Object Image |
|---------------|-------------|--|
| 70 | racquetBall |  |
| 71 | redBowl |  |
| 72 | redCup |  |
| 73 | redJelloBox |  |
| 74 | rubberDucky |  |
| 75 | rubikBox |  |
| 76 | rubiksCube |  |
| 77 | scotchTape |  |

| Object Number | Object Name | Object Image |
|---------------|----------------|--|
| 78 | screwBox |  |
| 79 | screwTray |  |
| 80 | smallBlueCup |  |
| 81 | smallGreenCup |  |
| 82 | smallOrangeCup |  |
| 83 | smallPyrex |  |
| 84 | soccerBall |  |
| 85 | sodaCan |  |

| Object Number | Object Name | Object Image |
|---------------|-----------------|--|
| 86 | softScrub |  |
| 87 | softball |  |
| 88 | solder |  |
| 89 | soupCan |  |
| 90 | spam |  |
| 91 | squareContainer |  |
| 92 | stapler |  |
| 93 | sugarBox |  |

| Object Number | Object Name | Object Image |
|---------------|-----------------|--|
| 94 | tabletBox |  |
| 95 | tallPlasticTube |  |
| 96 | tennisBall |  |
| 97 | threadlock |  |
| 98 | timmiesCup |  |
| 99 | toyDrill |  |
| 100 | tunaCan |  |
| 101 | usbDock |  |

| Object Number | Object Name | Object Image |
|---------------|---------------|--|
| 102 | waterBottle |  |
| 103 | whiteBox |  |
| 104 | wineGlass |  |
| 105 | woodBlock |  |
| 106 | woodEvenShape |  |
| 107 | woodHandle |  |
| 108 | yellowBox |  |
| 109 | zedBox |  |