Design and Analysis of Active Vision Methods for Robotic Grasping of Novel Objects

by

Sabhari Natarajan

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

by

August 2021

APPROVED:

Professor Berk Calli, Thesis Advisor

Professor Jing Xiao, Thesis Committee

Professor Michael Gennert, Thesis Committee

Abstract

In this project, multiple heuristic-based and data-driven active vision strategies are presented for viewpoint optimization of an arm-mounted depth camera to aid robotic grasping. These strategies aim to efficiently collect data to boost the performance of an underlying grasp synthesis algorithm. An open-source benchmarking platform was created in simulation (https://github.com/galenbr/2021ActiveVision), and an extensive study for assessing the performance of the proposed methods as well as comparing them against various baseline strategies was performed. The experimental study was done on a Franka Emika Panda robot with a two-fingered parallel jaw gripper. In these experiments, an existing grasp planning benchmark in the literature is utilized. With these analyses, we were able to quantitatively demonstrate the versatility of heuristic methods that prioritize certain types of exploration, and qualitatively show their robustness to both novel objects and the transition from simulation to the real world. The heuristic-based and data-driven methods were also compared in terms of their execution times and efficiency. We identified scenarios in which our methods did not perform well and present a discussion on which avenues for future research show promise.

Acknowledgements

I would first like to thank my thesis advisor Prof Berk Calli. His office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this research to be my own work but steered me in the right direction whenever he thought I needed it.

I would also like to thank Galen Brown, a Ph.D. student at WPI, who was also a part of this research work and the research scholars at Manipulation and Environment Robotics Laboratory (MER Lab) for their support.

I would also like to acknowledge Prof Jing Xiao and Prof Michael Gennert as committee members of this thesis, and I am gratefully indebted to them for their very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my family and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic grasping is a vital capability for many tasks, particularly in service and warehouse robotics. The problem of grasp synthesis is to determine how to position the robotic fingers on an object's surface so that a stable grasp can be achieved. If the shape and pose of the object is known to the robot accurately, various optimization techniques can be used to determine the finger contact locations on the object. Alternatively, if all the possible objects are known a priori, a database can be created with models of all objects along with several stable grasps, and an approach similar to Huebner et al. (2009) can be followed for localizing and identifying the object. However, in many scenarios of service robotics or warehouse robotics, the object variety is immense and the object models are not available to the robot a priori. For such applications, vision-based grasp synthesis methods are presented (Du et al., 2021), which utilize partial and noisy object data. Most of

This thesis is partially based on a manuscript that is published in Frontiers in Robotics and AI: "Natarajan, S., Brown, G., and Calli, B., 2021. Aiding Grasp Synthesis for Novel Objects Using Heuristic-Based and Data-Driven Active Vision Methods. Frontiers in Robotics and AI, 8."

The work in this thesis was supported in part by the National Science Foundation under grant IIS-1900953.

these grasp synthesis algorithms use data from a single viewpoint to synthesize a grasp (Caldera et al., 2018), i.e. they use only a single image of the object. Such an approach attempts to create a single, master algorithm that is useful for all objects in all situations. Nevertheless, these algorithms tend to suffer when the viewpoint of the vision system is different from the images used in the training set (Viereck et al., 2017). Additionally, many graspable objects have observation angles that are "singular" from which no grasp can be synthesized: For example, if an object has only one graspable surface, which is self-occluded from the current viewpoint of the vision system, the grasp synthesis algorithm would either fail to find any grasps or would need to rely on assumptions that might not always hold, and therefore lead to an unsuccessful grasp. These drawbacks of the single viewpoint approaches can be addressed via active vision frameworks, i.e. by actively moving the vision system to collect more data about the target object. Active vision is also used by humans in their day-to-day activities including grasp synthesis (Findlay and Gilchrist, 2003), suggesting that active vision is a viable solution to address the drawback of single viewpoint approaches.

1.1 Problem Statement

We aim to develop active vision strategies that can efficiently collect data with brief motions and allow the grasp synthesis algorithms to find sufficiently good grasps as quickly as possible. It is shown in the grasping literature that even algorithms tailored for single viewpoints can have a substantial performance boost even with very simple data collection procedures (Viereck et al., 2017). Utilizing active vision for robotic grasping has several avenues for optimization: the exploration algorithm, the data analysis, and the grasping algorithm. The major focus of this work is on

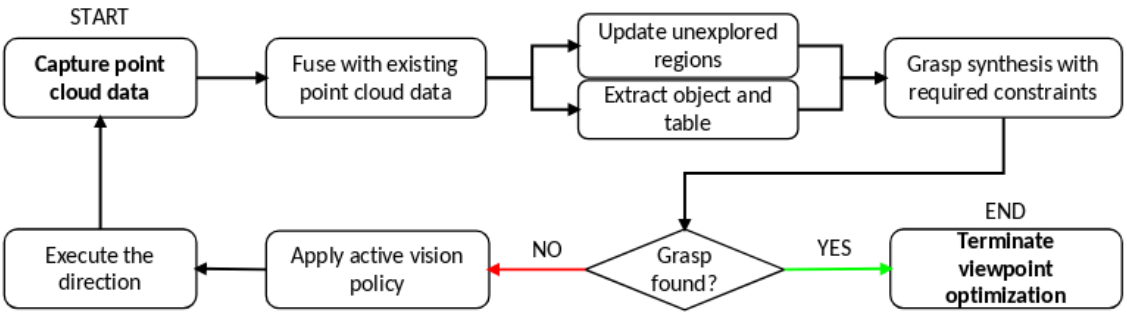


Figure 1.1: The active vision-based grasp synthesis pipeline

the exploration algorithm.

1.2 Overview

The active vision-based viewpoint optimization pipeline for grasp synthesis is represented in Fig. 1.1. The steps of this pipeline follow an iterative process until a grasp is found. The first step of an iteration is collecting the point cloud data about the environment using the vision system. In subsequent iterations, the point cloud data collected is fused with the point cloud data captured in previous iterations, thereby increasing our knowledge of the object and the environment. This point cloud data is processed using segmentation techniques to extract the points corresponding to the table and the object. Using this table and object points, a point cloud data is generated, to represent the unexplored regions in the environment i.e. the regions which have not been explored by the vision system yet (these processed point clouds are used in the grasp synthesis algorithm and active vision policies which will be explained in the further chapters). Then, an attempt is made to synthesize a grasp with the available point cloud data, and if it fails, the active vision policy is called to guide the vision system to its next viewpoint after which a new iteration starts until a grasp has been found. Each of these steps will be explained in the upcoming

chapters.

1.3 Contribution

- Two novel heuristic-based policies for viewpoint optimization.
- A self-supervised and a novel Q-learning-based policies viewpoint optimization.
- Three baseline policies for benchmarking the heuristic and data-driven policies.
- An open-source simulation platform (`https://github.com/galenbr/2021ActiveVision`) to develop new active vision algorithms and benchmark them.
- Extensive simulation and experimental analysis, assessing and comparing the performance of 5 active vision methods using objects from the YCB dataset (Calli et al., 2015).
- Use grasp planning benchmark (Bekiroglu et al., 2020) to assess the quality of grasps in real world experiments.
- Comparison study to another active vision-based algorithm (Arruda et al., 2016), which provides, to the best of our knowledge, the closest strategy to ours in the literature.
- Tools for visualizing the point cloud data collected and the path taken by the various policies.

1.4 Outline

The thesis is outlined as follows:

- **Chapter 2** is a literature review on various approaches for robotic grasping.
- **Chapter 3** describes the workspace and the hardware setup used.
- **Chapter 4** is focused on the point cloud processing techniques for simulation as well as additional processing for real world scenario.
- **Chapter 5** explains the grasp synthesis algorithm used in this study.
- **Chapter 6** focuses solely on the active vision policies developed i.e. Baseline, Heuristic, and Data-driven ones.
- **Chapter 7,8** explains the various simulation and real world tests performed along with a discussion on the results and the conclusions.

Chapter 2

Literature Review

Adapting robotic manipulation algorithms to work in an imperfect and uncertain world is a central concern of the robotics field, and an overview of modern approaches is given by Wang et al. (2020). It focuses on three major uncertainties; Geometric uncertainty (shape, size, etc.), Physical uncertainty (mass, friction, etc.), and completely unknown objects. Under each, the research done has been classified into feature sensing (estimating object properties and localization) and robotic grasping (grasp synthesis). Du et al. (2021) also gives a detailed review of the researches done towards vision-based grasping. They are segregated into the three key tasks which are object localization, object pose estimation, and grasp estimation. Object localization provides the regions of the target object in the input data. The object pose estimation refers to estimating the 6D object pose and the grasp estimation includes 2D planar grasp methods and 6DoF grasp methods. The focus on grasp synthesis using data-driven methods has been constantly increasing and Caldera et al. (2018) discusses the various approaches along with their performances over the past 10 years.

2.1 Grasp Synthesis

Much of the research on robotic grasping does not attempt to move the vision sensor and focuses on single image grasp synthesis. Also, the grasp synthesis algorithms differ based on the configuration of the gripper. Richtsfeld and Vincze (2008) uses range images from a fixed laser scanner to scan the object on a table. Based on the shape of the object, the grasp points for the gripper (3D model of a hand) are generated by using brute force search and collision checks. This study was done in simulation and showed good results for a wide range of objects. Saxena et al. (2010)'s work on grasping objects in cluttered environments acknowledges the problem of accurately sensing a complex 3D environment, but attempts to overcome it by storing prebuilt 3D models of the environment and using them to better analyze a single stereo-vision image rather than by collecting more information. With the environment information known, the objects are segmented and pre-trained models are used to find grasp for the objects. In a similar vein, Zheng et al. (2018) approaches industrial grasping for robotic welding by trying to accurately map known features to objects instead of by trying to collect more data to resolve ambiguities in the images. The environment here is a set of planar objects on a conveyor, for which the height of objects is mapped using a vision system after which a heuristic approach is used to find the grasp.

A typical and upcoming approach in the literature is to train a neural network to produce grasps by annotating individual images with grasp candidates. This neural network-based approach has been done using both 2D-based and 3D-based images as seen in Pinto and Gupta (2016), Chu et al. (2018) and Mahler et al. (2017) among many others. These neural network-based methods are trained with data from a single viewpoint which restricts their use for our application where data is

fused from multiple viewpoints. Learning-based algorithms have also be attempted in the past like Salganicoff et al. (1996), which uses Interval Estimation (IE) exploration heuristic with the ID-3 inductive learning algorithm to learn grasps for a two-fingered gripper. Even in tasks peripheral to grasping, like shape and pose estimation, considerable work has gone into more refined algorithms and machine learning strategies for extracting information from a single 2D image without attempting to capture more images or optimize the viewpoint. Morales et al. (2001) discusses the heuristic way of generating antipodal grasps based on the 2D features of unknown objects as seen by a top-down camera. Kurenkov et al. (2018) uses 3D data deformation to learn a model for 3D reconstruction through deformation. The network takes an image input, searches the nearest shape template from a database, and deforms the template to match the query image. Zhang and Cao (2019) focuses on 6D object pose refinement from noisy depth images. It also compares its methods with other common algorithms for pose refinement which are the ICP-based algorithms and optimization-based algorithms.

2.2 Active vision

Our research focuses on the problem of collecting new data to improve processing outcomes. Active vision has been applied to many aspects of machine vision, but often with the explicit goal of achieving a complete 3D model of an object rather than our objective of acquiring just enough data about the object to synthesize a sufficiently good grasp. Khalfaoui et al. (2012) uses the Mean Shift technique to construct the first set of candidates for the Next Best View (NBV). Weights are assigned to each NBV to pick the best one for scanning the object. Daudelin and Campbell (2017) also uses a similar approach but uses a mobile robot for this

purpose. It integrates an object probability characteristic to determine sensor views that incrementally reconstruct a 3D model of the object.

Even in the narrower domain of active vision for grasp synthesis, not all work relates to our concerns. For instance, Fu et al. (2019)'s study on industrial grasping uses active vision to assist feature identification of known objects, but with the explicit goal of maximizing grasp precision rather than minimizing information collected. For the use of active vision to address grasping using incomplete information, there has been research into both algorithmic (Calli et al., 2011; Arruda et al., 2016) and data-driven methods (Paletta and Pinz, 2000; Viereck et al., 2017; Calli et al., 2018b; Rasolzadeh et al., 2010), with more recent works tending to favor data-driven approaches (Caldera et al., 2018). In particular, the work in Viereck et al. (2017) demonstrated that active vision algorithms have the potential to outperform state-of-the-art single-shot grasping algorithms. Rasolzadeh et al. (2010) had an interesting approach with a multi-camera setup. A wide field of view camera was used for the initial localization of the object and then a near field of view camera mounted on a different manipulator was used to focus on the localized region to collect detailed information to generate a grasp.

Calli et al. (2011) proposed an algorithmic active vision strategy for robotic grasping and extending 2D grasp stability metrics to 3D space. As an extension of that work (Calli et al., 2018b), the authors utilized local optimizers for systematic viewpoint optimization using 2D images. Arruda et al. (2016) employs a probabilistic algorithm whose core approach is the most similar to our heuristics presented in Section 6.2. Our approaches differ in focus since Arruda et al. (2016) selects viewpoints based on estimated information gain as a proxy for finding successful grasps, while we prioritize grasp success likelihood and minimizing distance traveled. In our simulation study, we implemented a version of their algorithm and included it in

our comparative analysis.

The data-driven approach presented in Viereck et al. (2017) avoided the problem of labeled data by automating data labeling using state-of-the-art single shot grasp synthesis algorithms. They then used machine learning to estimate the direction of the nearest grasp along a view-sphere and performed gradient descent along the vector field of grasp directions. This approach has the advantage of being continuous and fast but did not fit in our discrete testing framework. All data-driven methods analyzed in this work utilize a similar self-supervised learning framework due to its significant easiness in training.

One of our data-driven active vision algorithms utilizes the reinforcement learning framework. A similar strategy for active vision is used by Paletta and Pinz (2000) to estimate an information gain maximizing strategy for object recognition. We not only extend Q-learning to grasping but do away with the intermediary information gain heuristic in reinforcement learning. Instead, we penalize our reinforcement approach for each step it takes that does not find a grasp, incentivizing short, efficient paths. Two of the data-driven methods in this work are based on the general strategy used in Calli et al. (2018a), which presented a preliminary study in simulation. In this work, we present one additional variant of this strategy and present a more extended simulation analysis along with real world experiments.

Gallos and Ferrie (2019) focused on object classification rather than grasping and influenced our theoretical concerns and experimental design to some extent. Their paper argues that contemporary machine learning-based active vision techniques outperform random searches but this is too low a bar to call them useful and demonstrates that none of the methods they implemented could outperform the simple heuristic of choosing a direction and moving along it in large steps. Virtually all active vision literature like de Croon et al. (2009) and Ammirato et al.

(2017) compares active vision approaches to random approaches or single-shot state-of-the-art algorithms. While there has been research on optimality comparison in machine vision Karasev et al. (2013), to the best of our knowledge, it has never been extended to 3D active vision, much less active vision for grasp synthesis. Our simulation benchmarks are an attempt to not only extend their approach to grasping but to quantify how much improvement over the best-performing algorithms remains possible.

Chapter 3

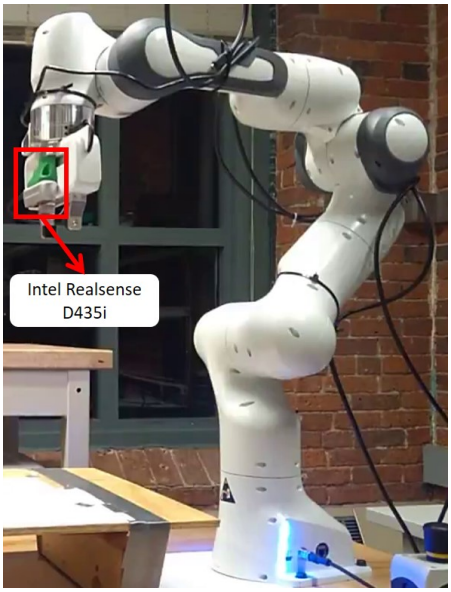
Workspace description

We consider an eye-in-hand system that allows us to move the camera to any viewpoint within the manipulator workspace. In our implementation, we use an Intel Realsense D435i (Fig. 3.1a) as the vision system on Franka Emika Panda (Fig. 3.1b) arm for our eye-in-hand system.

The camera can be moved to infinitely many configurations in this setup which would make it difficult for the active vision policies to come up with the next viewpoint to move to. To reduce the dimension of active vision algorithm's action space i.e. make the movements discrete, the camera movement is constrained to move along a viewsphere, always pointing towards and centered around the target object (a common strategy also adopted in Paletta and Pinz (2000), Arruda et al. (2016) and Calli et al. (2018a)). The radius of the viewsphere (v_r) is set based on the manipulator workspace and sensor properties. In the viewsphere, movements are discretized into individual steps with two parameters, step-size (v_s) and number of directions (v_d). Fig. 3.2 shows the workspace configuration used with $v_r = 0.4$ m, $v_s = 20°$ and $v_d = 8$ (N,NE,E,SE,S,SW,W,NW).

The target object is placed on a platform in front of the manipulator at a distance

(a) Intel Realsense D435i



(b) Franka Emika Panda

Figure 3.1: The manipulator and vision system used in the study.

of 0.45 m in front and 0.125 m above the base of the robot. These values were determined experimentally based on the manipulator's reach and its ability to grasp the object at different parts of the workspace. The dimensions of the objects to be used were restricted to be within a cylinder of height 35 cm and a radius of 25 cm. Multiple viewsphere radius (v_r) values were experimented before selecting 0.4 m. Fig. 3.3 shows a top-down view of the viewsphere regions marked as green or red, based on whether that point is within the workspace or not, for values of 0.4 m, 0.5 m, and 0.6 m. 0.4 m had the highest visibility percentage at 80. Further lower value could not be used due to the camera's minimum depth range of 17.5 cm at a resolution of 640x480 (it can be lowered to 10.5 cm but with a much lower resolution of 424x240).

With the workspace and the hardware described, in the next chapter, we will discuss the various point cloud processing steps and how the environment is modeled.

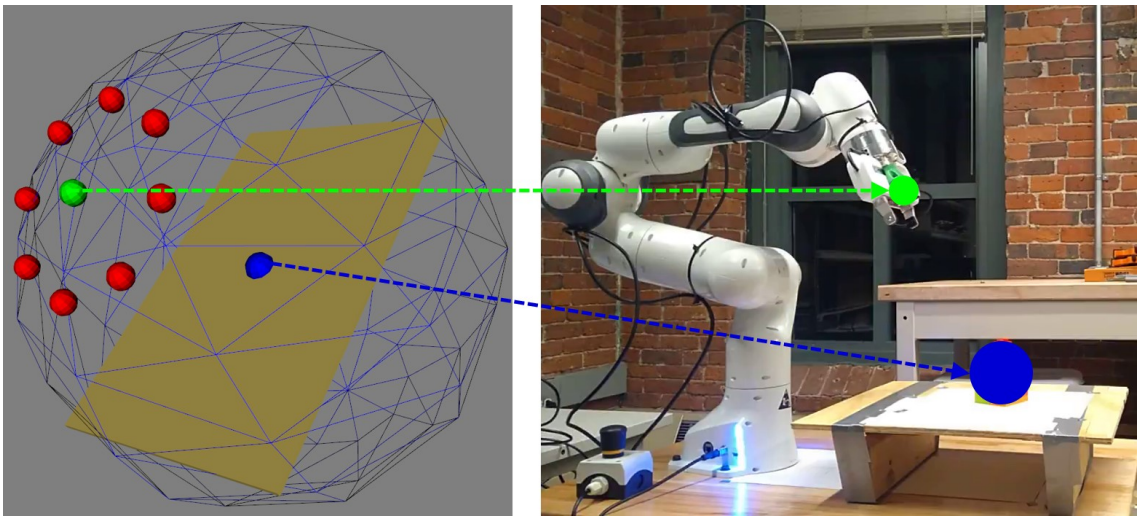


Figure 3.2: Viewsphere and its next steps with parameters $v_r = 0.4$ m, $v_s = 20^\circ$ and $v_d = 8$. The blue sphere is the expected position of the object, green sphere the current camera position and red one the next steps it can take

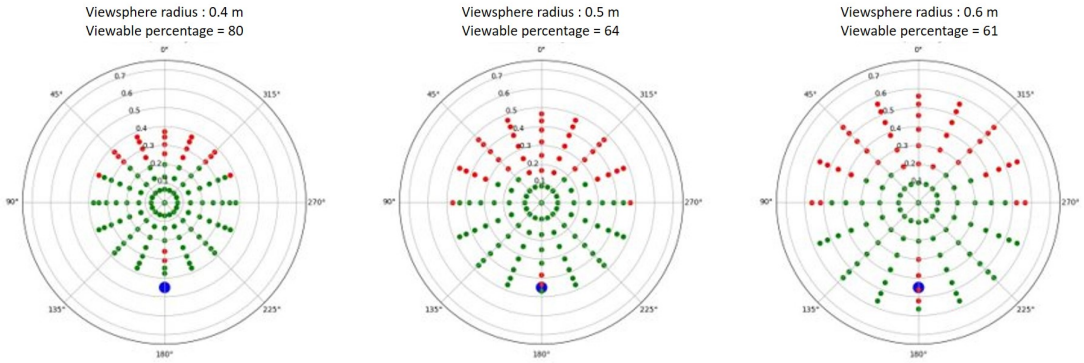


Figure 3.3: Comparison of how much information can be captured based on the viewsphere size. Blue dot: Robot Base, Centre of the circle: Object position, Green dots: Inside the workspace, Red dots: Outside the workspace

Chapter 4

Point cloud processing

The Intel Realsense D435i camera gives us point cloud data (RGB-D) of the environment. Point Cloud Library (PCL) Rusu and Cousins (2011) is used as the tool for processing the data. The point cloud data received from the camera is downsampled before further processing to reduce sensor noise and to speed up the execution time during further point cloud processing. Fig. 4.1 shows the environment as seen by the camera after downsampling.

This point cloud data which is in the camera frame is transformed to the base frame of the robot using

$$PCD^B = T_C^B * PCD^C \tag{4.1}$$

where, PCD^B is the point cloud data in robot base frame, T_C^B is the transformation matrix between the camera and the robot base frame and PCD^C is the point cloud data in camera frame i.e. the received data. T_C^B is estimated using in real-time using the manipulator's DH parameters and the joint configurations.

Sample Consensus-based plane segmentation technique is used to extract the points corresponding to the table. With table information known the points above the table are extracted and marked as object points. Fig. 4.2 shows the extracted

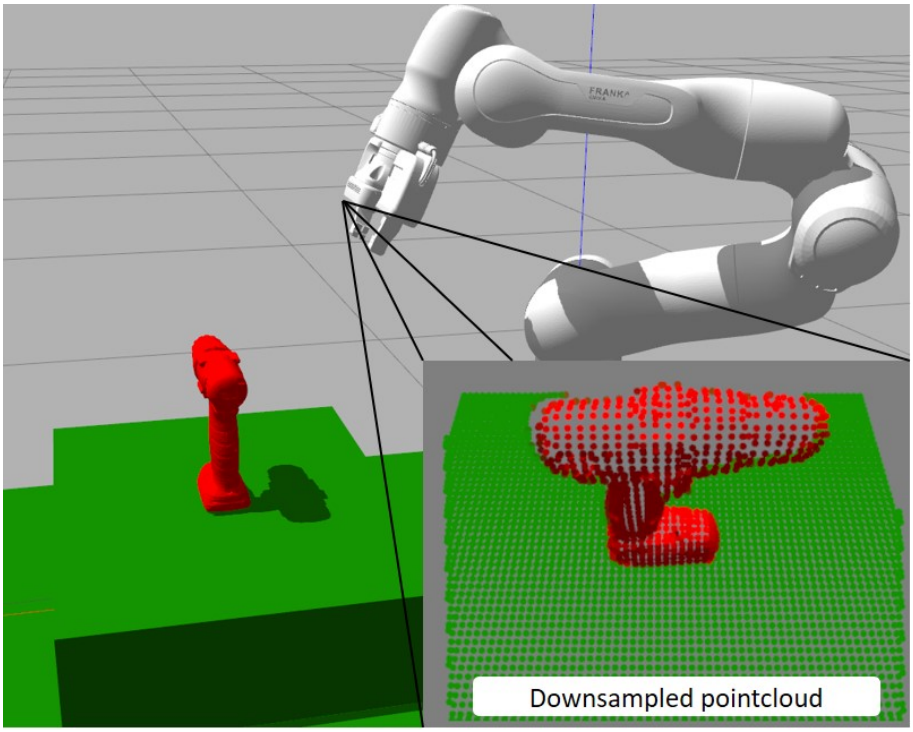


Figure 4.1: Environment as seen by the camera after downsampling in simulation.

table and object.

With every new point cloud data captured, it is fused with the previously captured point cloud data after transforming them to the robot base frame. With each fusion, the data is again downsampled to remove the duplicate points in the point cloud. The next step is to identify the unexplored regions surrounding the object which is required for the grasp synthesis algorithm as well as the active vision policies. For this purpose, the region surrounding the object is populated with an evenly spaced point cloud and then sequentially checked to determine which points are occluded. While a common visibility check approach is ray-tracing, it is a computationally intensive and time-consuming process. Instead, we take advantage of the organized nature of the point cloud data received from Intel Realsense D435i.

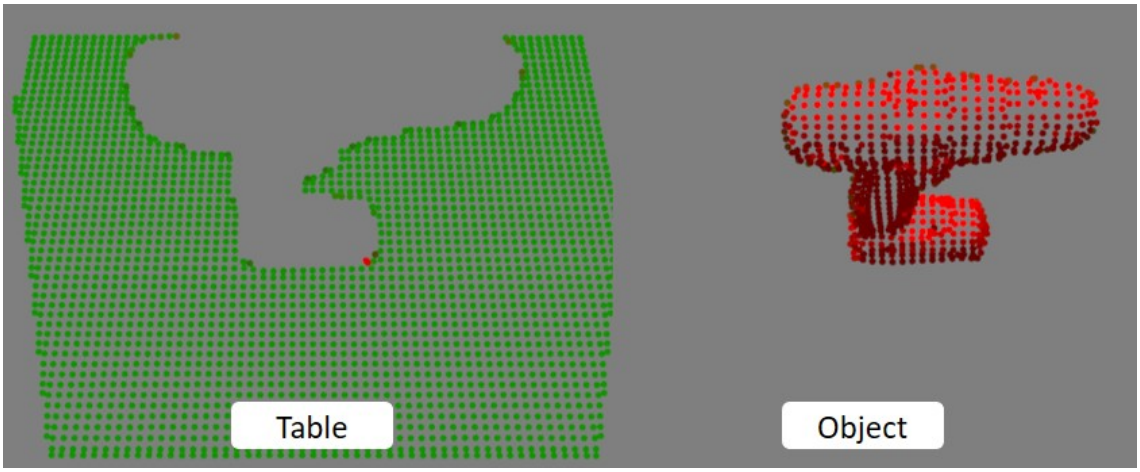


Figure 4.2: Extracted table and object

The 3D points are projected to the image plane using:

$$X_p = KX/z_0 \tag{4.2}$$

where, X_p is the projected pixel co-ordinates, X is the 3D point $\begin{pmatrix} x_0 & y_0 & z_0 \end{pmatrix}^T$, and K is the camera intrinsic matrix described by:

$$K = \begin{pmatrix} f_x & 0 & pp_x \\ 0 & f_y & pp_y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

where, f_x and f_y are the focal length, pp_x and pp_y are the principal point offset of the camera along the x-axis and y-axis respectively. The "depth value at X_p" and "z_0" are compared and if z_0 is greater than the depth at X_p, the point X is marked as occluded. This approach reduces the computation time greatly. The two images on the bottom right of Fig. 4.3 show the unexplored region generated for a drill object.

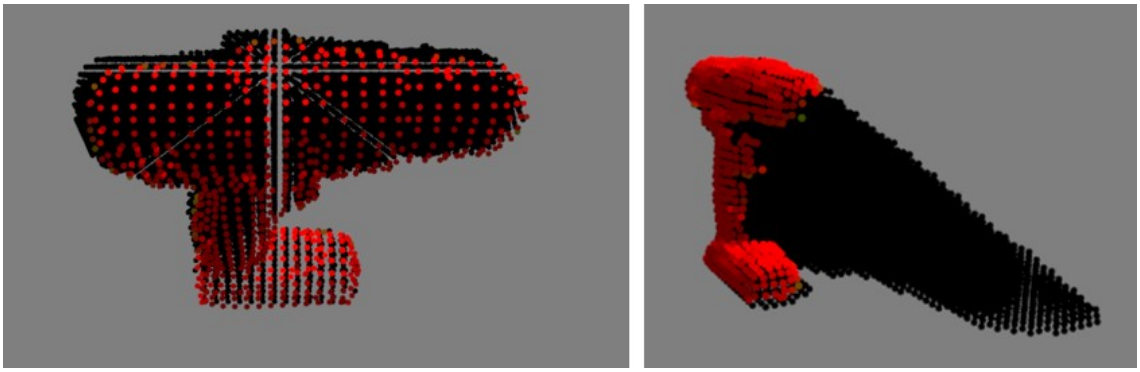


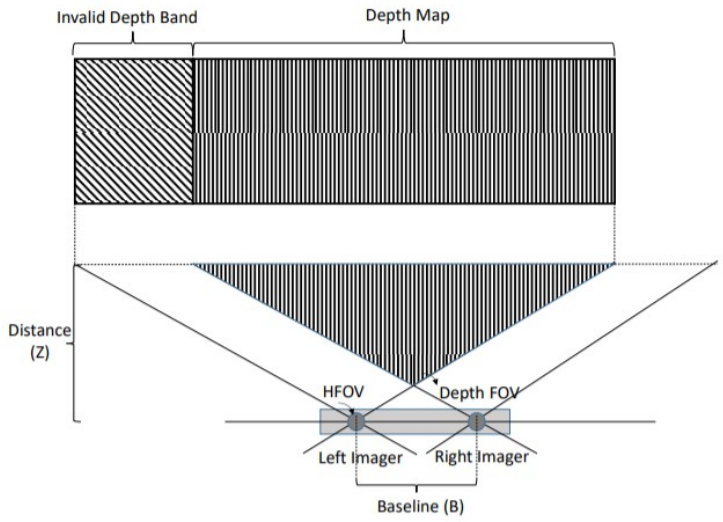
Figure 4.3: Black points show the unexplored regions in the environment for a drill object. The camera is positioned as shown in Fig. 4.1.

4.1 Additional processing for real world

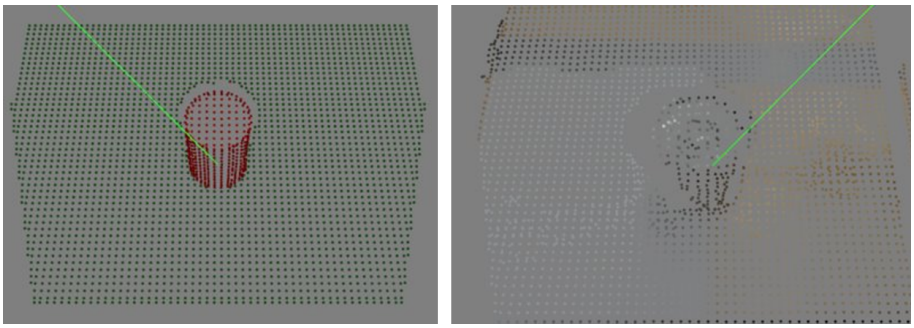
The vision system in the case of the real world and the simulation work a bit differently. To account for these changes two additional processing steps were required in this study when working in the real world.

4.1.1 Hole-Filling

In the Intel Realsense camera D435i, depth is derived primarily from solving the correspondence problem between the simultaneously captured left and right video images determining the disparity for each pixel (i.e. shift between object points in left vs right images), and calculating the depth map from disparity and triangulation. Fig. 4.4a shows the left and right imagers and the regions where depth can't be calculated. Depth also can't be calculated for surfaces that cannot be seen by both the imagers as shown in Fig. 4.4b. In the case of the simulation sensor, it just uses one imager (left) to generate the point cloud and hence can see all the surfaces in its field of view. This reduced surface visibility, especially the surfaces normal to the image plane leads to errors while updating the unexplored point cloud regions. These missing regions are considered as visible and hence unexplored regions get



(a) Depth field of view of Intel Realsense D435i camera



(b) Comparison for a can object: Simulation sensor (left), real world sensor (right). Notice the missing surfaces in real world data.

Figure 4.4: Difference between simulation and real world sensor

incorrectly classified as explored regions.

To address missing regions, we use a hole-filling approach to generate surfaces that are missing. As the newly generated surfaces will not accurately represent the object surface these generated surfaces are only used during the unexplored region update process and not considered under object points. Fig. 4.5 shows the results after hole-filling. Also, notice a large number of missing surfaces in the cuboid. We take advantage of the organized nature of the point cloud again for this hole-filling process. The 2D array for all pixels where the depth value is missing is updated using the average depth value of its 8-connected neighboring pixels. If there are

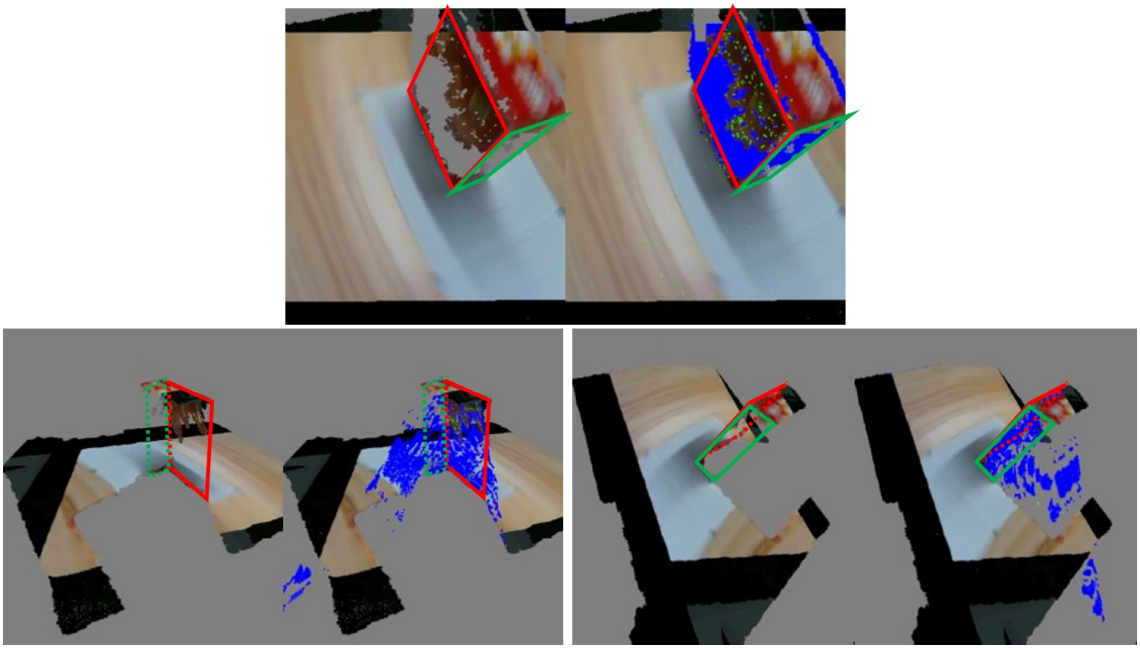


Figure 4.5: In each pair of images, the left image is the input and the right is the one after hole-filling. The blue dots represent the newly created surfaces. The red and green prisms show the two surfaces of the cuboid for comparison. The top image is from the camera perspective and the bottom ones are from different viewpoints to visualize the new surfaces added.

fewer than 4 neighbors with depth values then that pixel is not updated. Multiple passes are done to ensure larger holes are filled up. This process is done only on the central region of the image and not on the edges.

4.1.2 Camera calibration and point cloud registration

The transformation between the camera and the end-effector is required to transform the point cloud in the camera frame to the robot base frame. Using the CAD models of the manipulator, camera mount, and the camera the transformation between the end-effector and camera can be calculated. This transformation was not accurate enough in the real world setup and caused problems with point cloud fusion where the fused point cloud had duplicate surfaces. To solve the transformation error, first

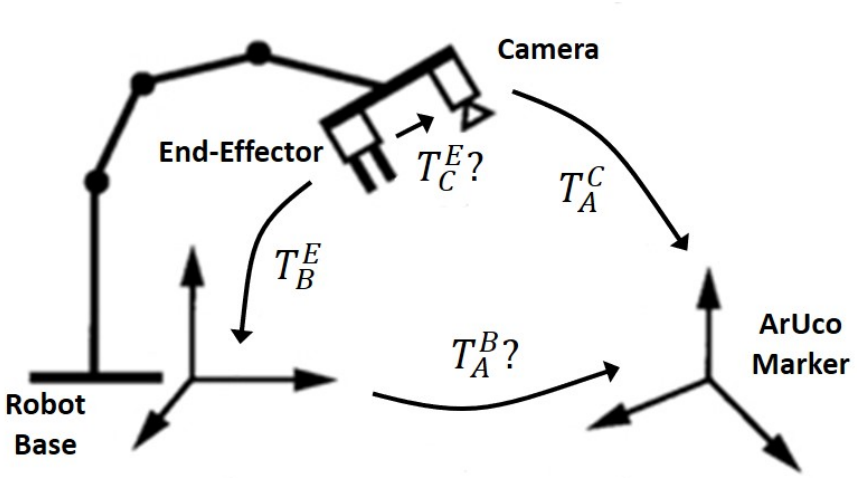


Figure 4.6: Hand-eye calibration setup. Robot Base (B), End-effector (E), Camera (C), ArUco Marker (A)

a camera calibration setup involving ArUco markers Garrido-Jurado et al. (2014) as shown in Fig. 4.6 was setup.

As shown in Fig. 4.6, we have the following four homogeneous transformation matrices:

- T_B^E: Transformation of Robot Base w.r.t End-effector.
- T_A^B: Transformation of ArUco Marker w.r.t Robot Base.
- T_C^E: Transformation of Camera w.r.t End-effector.
- T_A^C: Transformation of ArUco Marker w.r.t Camera.

The one which we are concerned about is T_C^E. These matrices can be connected by the equation:

$$T_B^E T_A^B = T_C^E T_A^C \tag{4.4}$$

T_B^E can be estimated using the manipulators' joint configuration and T_A^C can be estimated by using the ArUco marker placed on the table. The other two matrices

are constant but unknown. Equation 4.2 can be represented in the form of:

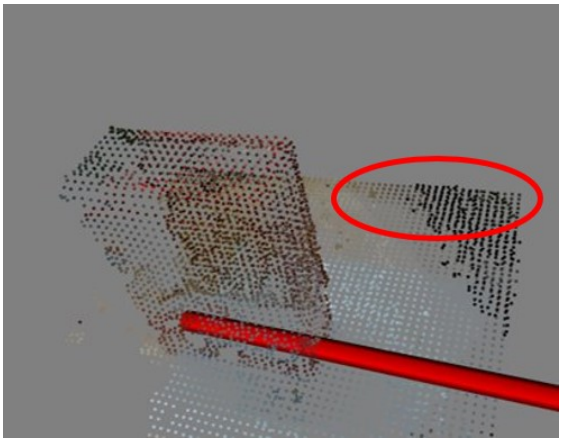
$$A\,X = Z\,B \tag{4.5}$$

where A and B are known matrices and X and Z are unknowns. As X and Y are homogeneous transformation matrices each have only 6 unknown variables (3 rotational + 3 transnational). So, we have 12 variables to solve for. If we can collect 12 or more samples by moving the camera to different locations, we will have sufficient data to solve for and find the matrices X and Z. We utilize Moveit!'s hand-eye calibration module for this (Coleman et al., 2014). We feed it with the required number of camera samples and it returns the transformation matrix T_C^E.

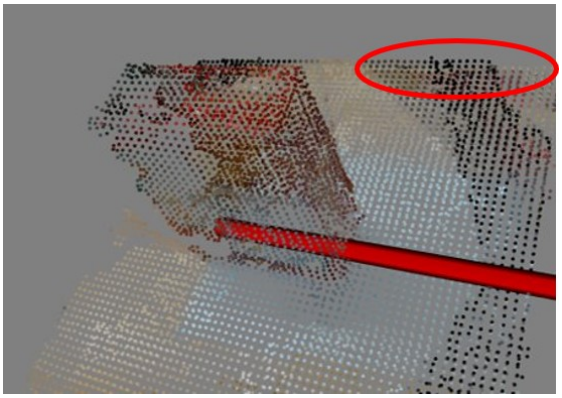
Even with the updated transformation matrix, it is not accurate enough for point cloud fusion. We use the Iterative Closest Point (ICP) based registration techniques available in PCL to finely align the point clouds before fusion. This method is most reliable when the input and target point cloud only differ in their alignment and not on their shape or structure. In our case, we need to fuse point clouds from different viewpoints which imply that they will not be the same in terms of the shape or structure. ICP based registration is used in two stages to get reliable results in this scenario.

- Extract table planes from both point clouds and do the initial alignment (6 DoF).
- Restrict the registration process to 3 DoF i.e. translation and rotation about the table plane and align the object point clouds.

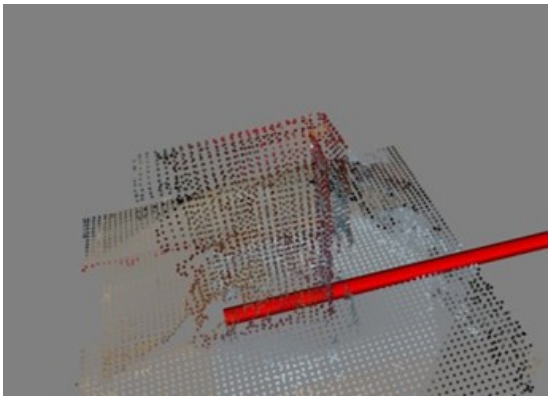
Fig. 4.7 shows the accurate alignment during fusion when accurate transformation and ICP is used. As seen in Fig. 4.7 ICP registration also doesn't work well if the



(a) Accurate transformation with ICP



(b) Accurate transformation without ICP



(c) Inaccurate transformation with ICP

Figure 4.7: Effect of using accurate transformation matrices and ICP. The red oval shows the region where misalignment is prominently visible

initial alignment is considerably off. Hence both camera calibration and ICP were required to accurately fuse the point clouds.

Chapter 5

Grasp synthesis

The goal of our pipeline is to provide enough data to an underlying grasp synthesis algorithm to find a successful grasp. As such, our methods are largely agnostic to the specific grasp synthesis algorithm used. Essentially, any gripper type i.e. parallel-jaw / multi-fingered / vacuum along with its corresponding grasp synthesis algorithm can be used in this methodology, provided the grasp synthesis algorithm can process an incomplete point cloud as input. However, these grasping algorithms are naturally preferred to be fast (since they will be run multiple times per grasp within our viewpoint optimization process), and be able to work with stitched point clouds. Most data-driven approaches in the literature are trained with single-view point clouds, and might not perform well with stitched object data.

5.1 Grasp generation

In this study, we use the Franka Emika parallel jaw gripper, which has a maximum gripper width of 8 cm and a contact surface area of 4 cm^2 as shown in Fig. 4.1. We use the term "contact point" to refer to a point in the point cloud being considered for a grasp. PCL has modules for calculating the normal vector and curvature of

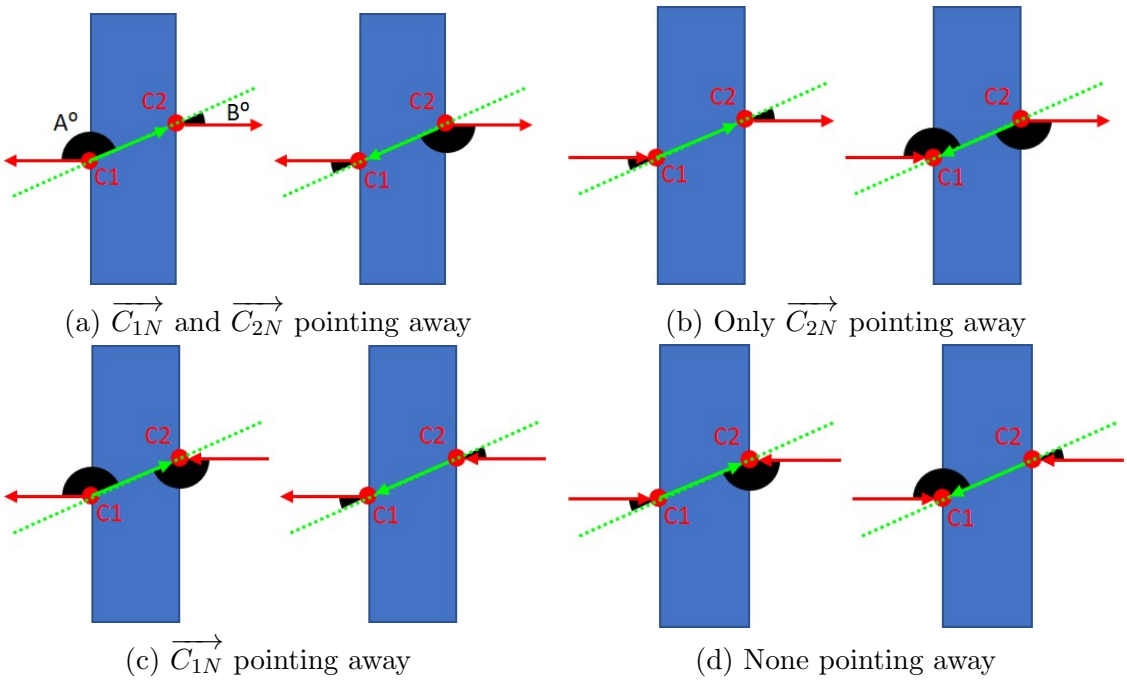
points in the point cloud which are used in the following process. As our gripper is a parallel jaw gripper, we use a force-closure-based approach similar to Calli et al. (2018a), with the following constraints:

1. Grasp quality: This quality is a value ranging from 0 to 180 and depends on the normal vectors of the two contact points being considered for the grasp and their relative position. This is calculated as:

$$GQ = 180 - (min(\angle(\overrightarrow{C_1C_2}, \overrightarrow{C_{1N}}), \angle(\overrightarrow{C_2C_1}, \overrightarrow{C_{1N}})) + min(\angle(\overrightarrow{C_1C_2}, \overrightarrow{C_{2N}}), \angle(\overrightarrow{C_2C_1}, \overrightarrow{C_{2N}})))) \tag{5.1}$$

   where GQ is the grasp quality, C_1 and C_2 are the contact points 1 and 2 respectively, C_{1N} and C_{2N} are the surface normal vectors at the contact points 1 and 2 respectively. Fig. 5.1 shows the various configurations possible for the normal vectors. For all the cases shown in Fig. 5.1, Equation 5.1 would give $180 - (min(A, 180 - A) + min(B, 180 - B)) = 180 - (180 - A + B) = A - B$. When all three vectors align with each other we will have the highest grasp quality of 180. In this study, a grasp quality requirement was set as between 150 and 180.

2. Contact patch area and curvature constraint: To grasp the object, there must be a relatively flat surface surrounding each contact point at least as large as the gripper's contact area. Based on the known gripper contact area and the points surrounding the contact point under consideration, the object contact patch area is calculated by projecting the points within a 3 cm radius of the contact point onto a plane whose normal vector is the same as that of the contact point. This projected area should be higher than a threshold for the contact point, and the curvature as calculated by PCL should be below a



(a) $\overrightarrow{C_{1N}}$ and $\overrightarrow{C_{2N}}$ pointing away (b) Only $\overrightarrow{C_{2N}}$ pointing away

(c) $\overrightarrow{C_{1N}}$ pointing away (d) None pointing away

Figure 5.1: Various possible configurations for the normal vectors for the contact points C_1 and C_2. The red arrows are the normal vectors and the green one is the vector connecting the contact points. In each configuration the left image shows $\overrightarrow{C_1C_2}$ and right one $\overrightarrow{C_2C_1}$

threshold.

3. Gripper width: The distance between the contact points should be less than the maximum gripper width i.e. 8 cm.

Each pair of contact points that satisfy the above constraints is a potential grasp. In the case of thin objects, where two contact points can't be found, each point is considered as its own pair. is A brute force search is done on all the object points to generate the possible list of grasps for the object. Algorithm 1 shows the grasp generation process.

Algorithm 1 Grasp generation process

Require: *obj* ← Object point cloud
  contact_pair ← Pair of contact points
  potential_grasps ← Array of *contact_pair* which can be a potential grasp
  for all *contact_pair* ∈ *obj* **do**
    if All constraints satisfied **then**
      Add *contact_pair* to *potential_grasps*
    end if
  end for

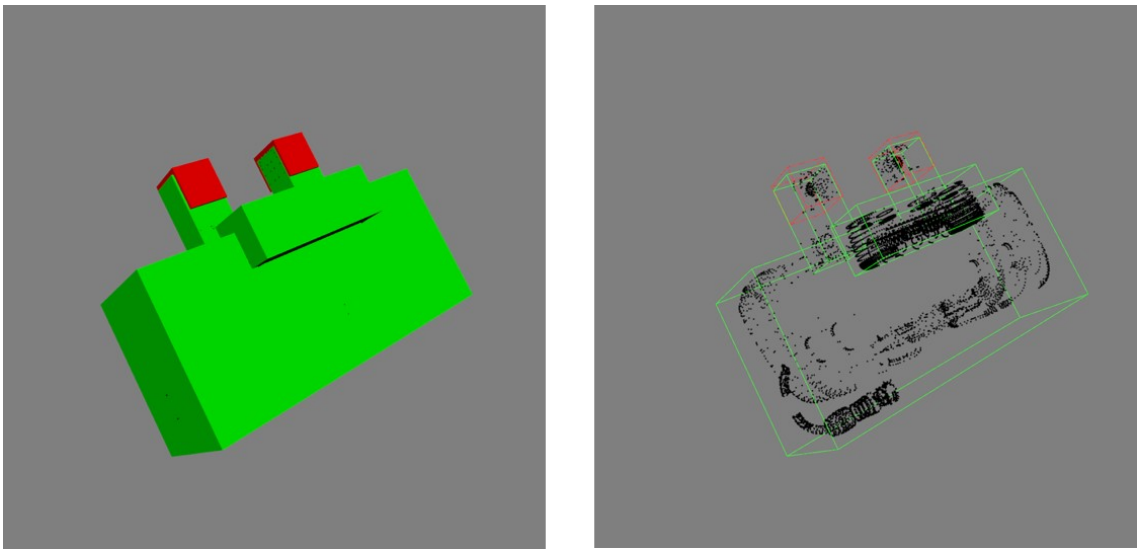


Figure 5.2: Simplified model of the gripper using cuboids for collision check.

5.2 Collision and Reachability check

From the set of potential grasps, we only need to consider the grasps which don't collide with the environment (collision check) and can be reached by the manipulator (reachability check). The collision check is done by using a simplified model of the gripper as shown in Fig. 5.2. We check for collision of the gripper model with the table and unexplored point clouds. Each grasp has infinitely many possible gripper orientations that would align with the contact points i.e. the gripper can revolve along the axis connecting the two contact points. The possible orientations have been discretized into eight evenly spaced orientations in this study. The collision

check is done in two stages to speed up the process:

- Preliminary check: If the red-colored region in Fig. 5.2 has a collision, then any orientation of the gripper will still have a collision. So, if a collision is detected at this region all further collision checks are discarded and the grasp candidate is discarded.

- Advanced check: The green-colored regions in Fig. 5.2 (fingers, hand, camera) are sequentially checked for collisions for eight different orientations, and if a collision is detected, that particular orientation of the grasp candidate is discarded.

The grasping process involves moving the manipulator to a pre-grasp pose which is 10 cm away from the grasp configuration along the negative approach vector direction. From the pre-grasp pose, it is moved along a straight line to the grasp pose to grasp the object. So, the reachability check is performed for grasp as well as the pre-grasp pose using the inbuilt functions in Moveit! (Coleman et al., 2014).

5.3 Grasp sorting and selection

The collision and reachability checks used to select a grasp from the possible grasps are computationally intensive and hence it is preferable to do as few checks as possible to select the grasp. Algorithm 2 explains the process of sorting the potential grasps using the grasp related parameters:

- Centroid: Centroid on the object point cloud.

- Euclidean distance: Euclidean distance between the contact pair's centroid and the object's centroid.

- Line distance: Euclidean distance between the contact pair's centroid and the object's line of gravity.
- Grasp quality.

The thresholds used were experimentally determined. This sorting process helps to prioritize the grasps that have a higher probability of passing the checks and which are closer to the centroid of the object.

Algorithm 2 Grasp sorting

Require: $A \leftarrow$ First *contact_pair* of a *potential_grasp*
Require: $B \leftarrow$ Second *contact_pair* of a *potential_grasp*
  if abs($A.line\_distance$ - $B.line\_distance$) $\leq$ .01 m **then**
    if abs($A.centroid.z$ - $B.centroid.z$) $\leq$ .06 m **then**
      if abs($A.euclidean\_distance$ - $B.euclidean\_distance$) $\leq$ .01 m **then**
        potential_grasp with the highest *grasp_quality* is preferred
      else
        potential_grasp with the lowest *euclidean_distance* is preferred
      end if
    else
      potential_grasp with the highest *centroid.z* is preferred
    end if
  else
    potential_grasp with the lowest *line_distance* is preferred
  end if

With the sorted list of potential grasps, the collision and reachability checks are run as described in Algorithm 3. The sorting process reduces the number of checks required to arrive at the final grasp. The first grasp orientation that passes the checks is selected.

Algorithm 3 Grasp selection process

Require: *tbl* ← Table point cloud
Require: *unexp* ← Unexplored point cloud
Require: *potential_grasps* ← Array of *contact_pair* which can be a potential grasp

```
Sort potential_grasps using Algorithm 2.
for all contact_pair ∈ potential_grasps do
  for each of the eight gripper orientations do
    if Collision and Reachability check passed then
      Select this contact_pair at this gripper orientation
      Exit both the for loops
    end if
  end for
end for
```

Chapter 6

Active vision policies

The focus of this thesis is the active vision policies, which guide the eye-in-hand system to its next viewpoints. The nature of the pipeline allows us to plug in any policy which takes point clouds as its input and returns the direction to move for the next viewpoint. The policies developed and tested have been classified into three categories as follows:

1. Baseline policies
2. Heuristic-based policies
3. Data-driven policies

Each of these sets of policies is explained below.

6.1 Baseline policies

As the name suggests these are a set of policies defined to serve as a baseline to compare the heuristic-based and data-driven policies with. The three baselines used are shown below.

6.1.1 Random Policy

Ignoring camera data, a random direction was selected for each step. No constraints were placed on the direction chosen, leaving the algorithm free even to (for instance) oscillate infinitely between the start pose and positions one step away. This movement represents the worst case for a policy not deliberately designed to perform poorly, and all methods should be expected to perform better than it in the aggregate. This policy is the standard baseline in the active vision literature.

6.1.2 Brick Policy

Named after throwing a brick on the gas pedal of a car, a consistent direction (northeast) was selected at each timestep. This direction was selected because early testing strongly favored it, but we make no claims that it is ideal. This policy represents the baseline algorithm that is naively designed and any serious policy should be expected to outperform. Any policy that performed poorly than this policy would need well-justified situational advantages to be usable.

6.1.3 Breadth-First-Search (BFS) Policy

From the starting position, an exhaustive Breadth-First-Search was performed, and an optimal path was selected. This policy represents optimal performance, as it is mathematically impossible for a discrete algorithm to produce a shorter path from the same start point. No discrete method can exceed its performance, but measuring how close each method comes to it gives us an objective measure of each method's quality in each situation.

With baselines defined, we will now discuss the other categories starting with heuristics.

6.2 Heuristic policies

The idea behind the heuristic policy is to choose the best possible direction after considering the next available viewpoints. The metric used to define the quality of each of the next viewpoints is a value proportional to the unexplored region visible from a given viewpoint.

6.2.1 2D Heuristic Policy

The viewpoint quality is calculated by transforming the point clouds to the next possible viewpoints and projecting the object and unexplored point clouds from those viewpoints onto an image plane using the camera's projection matrix. This process has the effect of making the most optimistic estimation for exploring unexplored regions; it assumes no new object points will be discovered from the new viewpoint. Since the point clouds were downsampled, their projected images were dilated to generate closed surfaces. The 2D projections are then overlapped to calculate the size of the area not occluded by the object. The direction for which the most area of the unexplored region is revealed is then selected. Fig. 6.1 shows an illustration with the dilated projected surfaces and the calculated non-occluded region. The 2D Heuristic policy is outlined in Algorithm 4.

While this heuristic is computationally efficient, it considers the 2D projected area, leading it to, at times, prefer wafer-thin slivers with a high projected area over deep blocks with low projected area. Additionally, it is agnostic to the grasping goal and only focuses on maximizing the exploration of unseen regions.

Algorithm 4 2D Heuristic policy

Require: $obj \leftarrow$ Object point cloud
Require: $unexp \leftarrow$ Unexplored point cloud
  for all $viewpoint \in$ next possible viewpoints **do**
    if viewpoint within manipulator workspace **then**
      $obj\_trf \leftarrow$ Transform obj to viewpoint
      $obj\_proj \leftarrow$ Project obj_trf onto image plane (B/W image) and dilate
      $unexp\_trf \leftarrow$ Transform $unexp$ to viewpoint
      $unexp\_proj \leftarrow$ Project $unexp\_trf$ onto image plane (B/W image) and dilate
      $non\_occ\_unexp\_proj \leftarrow unexp\_proj - obj\_proj$
    end if
    Record the number of white pixels in $non\_occ\_unexp\_proj$
  end for
  Choose the direction with maximum white pixels

6.2.2 3D Heuristic Policy

In the 3D heuristic, we focused only on the unexplored region which could lead to a potential grasp. This was done using the normal vectors of the currently visible object. Since our grasp algorithm relies on antipodal grasps, only points along the surface normals can produce grasps. These points were extracted by using a 3D bounding box with a length of twice the maximum gripper width [2*8 cm], and a width and height of 1cm. The longer axis of this box was aligned with the normal vector and the center of the box was aligned with the point in consideration. This process was done for all object points to create a 3D mask of all unexplored space that could contain a grasp. The unexplored points which were outside this mask were discarded for the next steps.

Next, like in the 2D heuristic, we transformed the point cloud to the frame of reference of the next possible viewpoints. This time, instead of projecting, we used local surface reconstruction and ray-tracing to determine all the unexplored points which will not be occluded from a given viewpoint. The direction which leads to the highest number of non-occluded unexplored points is selected. This approach

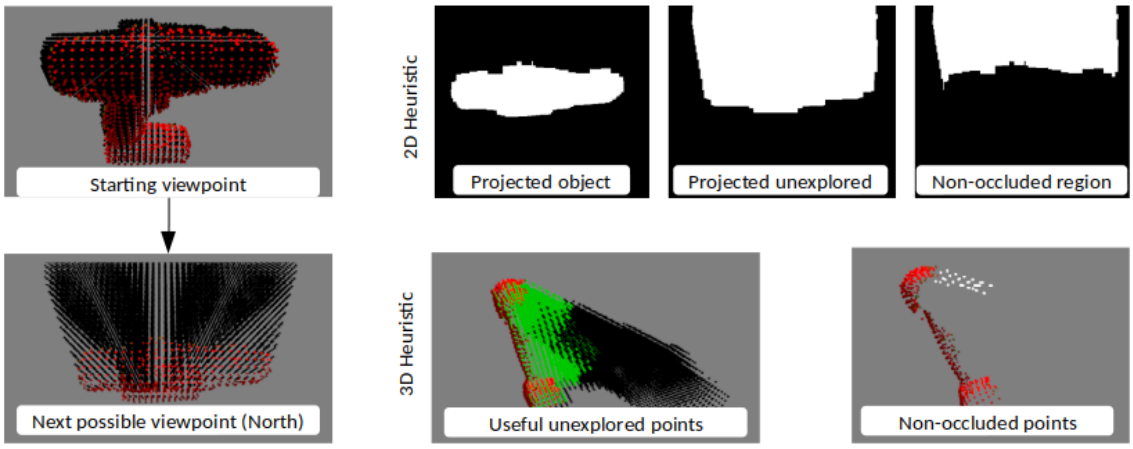


Figure 6.1: Set of images illustrating how the 2D and 3D Heuristics evaluate a proposed next step North with the drill object. The 3D Heuristic images have been shown from a different viewpoint for representation purposes.

prioritizes exploring the greatest possible region of unexplored space that, based on known information, could potentially contain a grasp. If all the viewpoints after one step have very few non-occluded points the policy looks one step ahead in the same direction for each before making the decision. Fig. 6.1 shows an illustration with the non-occluded useful unexplored region. The green points are the region of the unexplored region which is considered useful based on gripper configuration. The 3D Heuristic policy is outlined in Algorithm 5.

6.2.3 Information Gain Heuristic Policy

The closest approach to the heuristics presented in this work is provided by Arruda et al. (2016). For comparison purposes, we implemented an approximate version of their exploration policy to test our assumptions and compare it with our 3D Heuristic approach. First, we defined a set of 34 viewpoints across the viewsphere to replicate Arruda et al. (2016)'s search space. The same viewsphere setup as seen in Fig. 3.2 was used. Each viewpoint is defined by a pair of polar and azimuthal angles. Three polar angle values of 22.5° , 45° and 67.5° were used with 10, 12,

Algorithm 5 3D Heuristic policy

Require: $obj \leftarrow$ Object point cloud
Require: $unexp \leftarrow$ Unexplored point cloud
Require: $points\_threshold \leftarrow$ Minimum number of non-occluded unexplored points needed for a new viewpoint to be considered useful
$useful\_unexp\_trf \leftarrow$ Unexplored points with potential for a successful grasp
for all $viewpoint \in$ next possible viewpoints **do**
  if viewpoint within manipulator workspace **then**
    $obj\_trf \leftarrow$ Transform obj to viewpoint
    $useful\_unexp\_trf \leftarrow$ Transform $useful_unexp$ to viewpoint
    $non\_occ\_useful\_unexp \leftarrow$ Check occlusion for each $useful\_unexp\_trf$ using local surface reconstruction and ray-tracing.
  end if
  Record the number of points in $non\_occ\_useful\_unexp$
end for
$max\_points \leftarrow$ Maximum points seen across the possible viewpoints
if $max\_points \leq points\_threshold$ **then**
  Run the previous for loop with twice the step-size
end if
$max\_points \leftarrow$ Maximum points seen across the possible viewpoints
Choose the direction which has max_points

and 12 evenly distributed azimuthal angle values from 0° - 360° respectively. To calculate the information gain for each viewpoint, we modified the 3D Heuristic to consider all unexplored regions as opposed to focusing on the regions with a potential grasp. Similarly, the modified 3D Heuristic policy, instead of comparing the next v_d viewpoints, compared all 34 viewpoints and used the one with the highest information gain. A simulation study was performed to compare the camera travel distance and computation time of this algorithm to our heuristics.

6.3 Data-driven policies

Our data-driven policies utilize a fixed-size state vector as input. This state vector is obtained by modeling the object point cloud and unexplored regions point cloud with

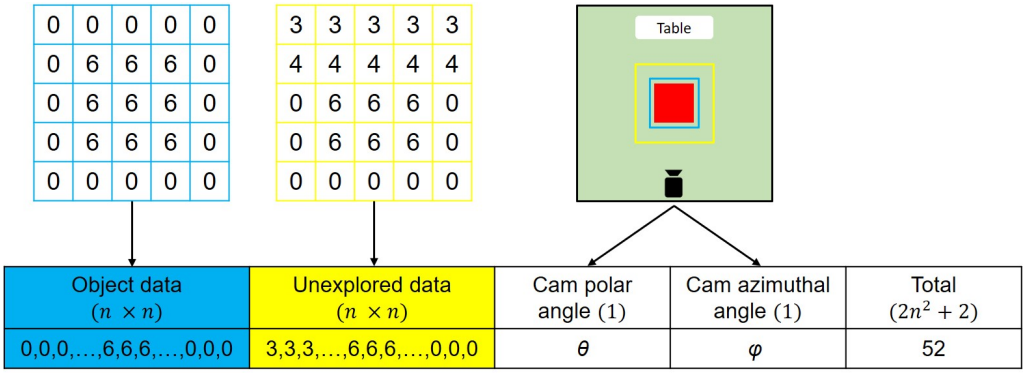


Figure 6.2: HAF-based state vector for a 6x6x6 cm cube on the table. The blue square shows the square zone considered for object data and the yellow square shows the zone considered for unexplored data. The length of the unexplored data square zone is 1.5 times that of the object square zone. The values represent the maximum height among the points within each grid. These data are flattened and merged along with camera information to generate the state vector.

Height Accumulated Features (HAF), developed by Fischinger and Vincze (2012) and used in Calli et al. (2018a) along with the camera position. We experimented with grid sizes of 5 and 7 height maps, both of which provide similar performance in our implementation, so we chose to use 5 for performance reasons. The state vector of a given view is composed of the flattened height maps of the extracted object and the unexplored point cloud and the polar and azimuthal angle of the camera in the viewsphere. The size of the state vector is $2n^2 + 2$, where n is the grid size. Fig. 6.2 shows an illustration of the HAF state vector generation process.

6.3.1 Self-supervised Learning Policy

Following the synthetic data generation procedure used by Calli et al. (2018a), we generated training data in simulation. For a given start pose, each compass direction (north, north-east, east, etc.) was explored for a grasp. If none were found, further exploration of four random steps from each compass direction was performed three times. The shortest working path was saved, along with the state vector of each

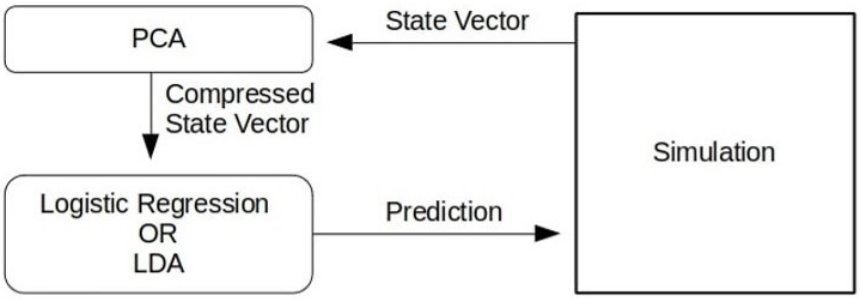


Figure 6.3: Self-Supervised Learning-based policy

camera view in the path. This process was repeated for 1,000 random initial poses each for the 10x8x4 and 20x6x5 prisms in Fig. 6.5. Further training objects were considered, but initial results generalized so well that it was not pursued. This data was used to train two self-supervised learning methods, logistic regression and LDA classification, to predict the next best viewpoint to select given the state vector of the current viewpoint. In both of the methods, we first applied PCA to each state vector to further compress it to 26 components, as shown in Fig. 6.3. All the components used in this policy were implemented using the scikit-learn library Pedregosa et al. (2011).

6.3.2 Deep Q-Learning Policy

In an attempt to improve on the self-supervised methods, a deep Q-Learning policy was also trained to predict, for a given state vector, the next best viewpoint using Keras library tools Chollet (2015). Four fully connected 128 dense layers and one 8 dense layer, connected by Relu transitions, formed the deep network that made the predictions as shown in Fig. 6.4. In training, an epsilon-random gate replaced the network's prediction with a random direction if a random value exceeded an epsilon value that decreased with training. The movement this function requested was then performed in simulation, and the resulting state vector and a binary grasp found

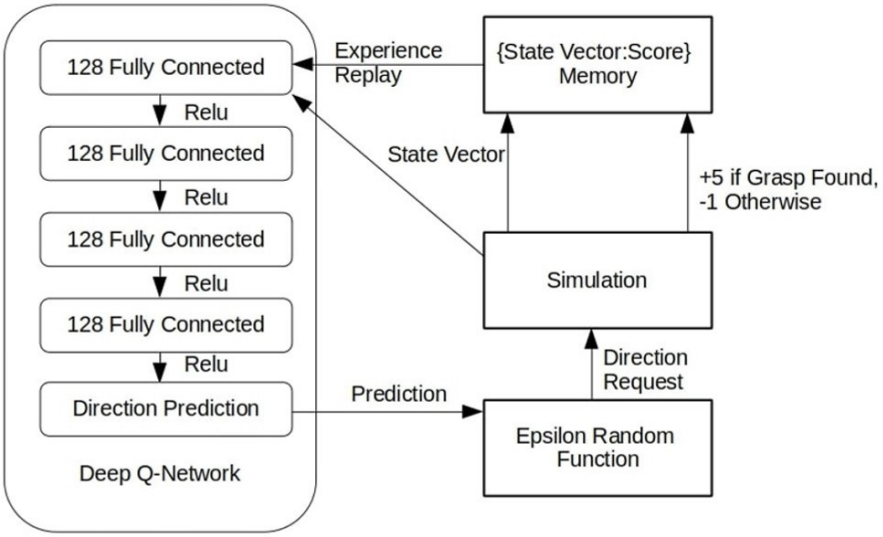


Figure 6.4: The Deep Q-Learning policy

metric were recorded. Once enough states had been captured, experience replay randomly selected from the record to train the Q-Network on a full batch of states each iteration. The Q-Learning was trained in simulation to convergence on all of the objects in Fig. 6.5, taking roughly 1,300 simulated episodes to reach convergence. We hoped that, given the relatively constrained state space and strong similarities between states, meaningful generalizations could be drawn from the training set to completely novel objects.

For all data-driven approaches, the objects used for training were never used in testing.

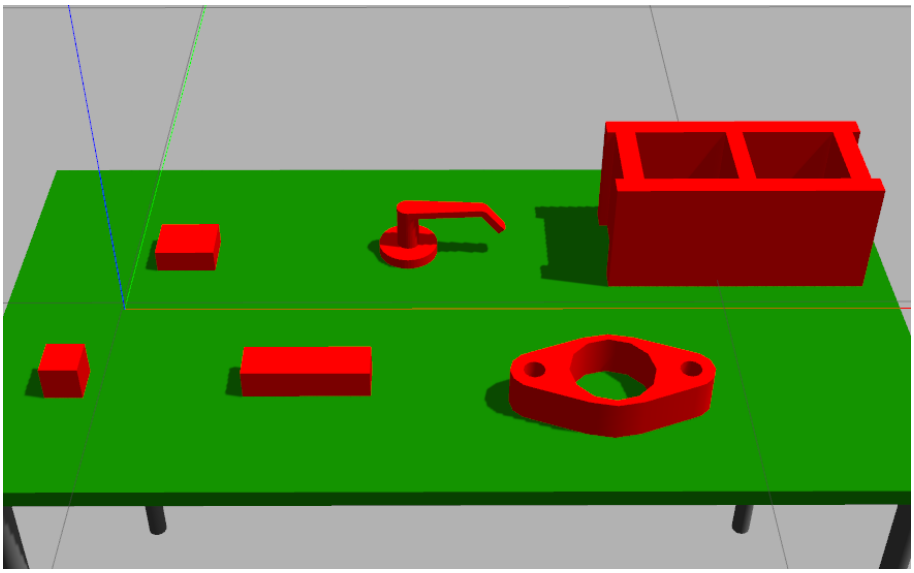


Figure 6.5: Objects used for simulation training. Left to right: prism 6x6x6 cm, prism 10x8x4 cm, prism 20x6x5 cm, handle, gasket, cinder block.

Chapter 7

Simulation and Real world Results

The methodology discussed in the above chapters was implemented and tested in simulation and the real world. The setups used for the testing are shown in Fig. 7.1. The maximum number of steps allowed before an experiment is restarted was set to 6 based on preliminary experiments with the BFS policy. The start position of the manipulator i.e. the position from which the viewpoint optimization is started is shown in Fig. 3.2.

7.1 Simulation Study

The extensive testing in simulation was done on a set of 12 objects from the YCB dataset (Calli et al., 2015) which are shown in Fig. 7.2. To ensure consistency, we applied each policy to the same set of 100 poses for each object. This approach allowed us to produce a representative sample of a large number of points without biasing the dataset by using regular increments, while still giving each policy identical conditions to work in. This sampling was done by generating a set of 100 random values between 0 and 359 before testing began. To test a given policy with a given object, the object was spawned in Gazebo in a stable pose, with 0° of rota-

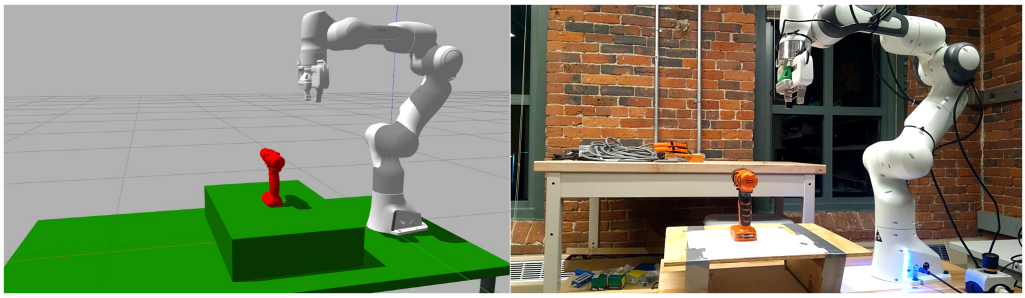


Figure 7.1: The setup as seen in simulation environment (left) and lab environment (right) with power drill in place

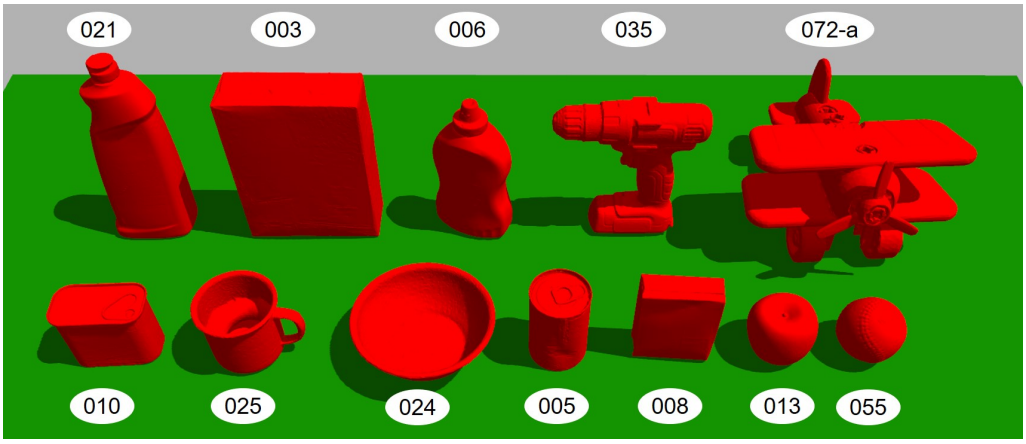


Figure 7.2: The set of object used for simulation testing. YCB object IDs : 3, 5, 7, 8, 10, 13, 21, 24, 25, 35, 55, 72-a

tion about the z-axis. The object was then rotated by the first of the random value about the z-axis, and the policy was used to search for a viable grasp. After the policy terminated, the object was reset, and rotated to the second random value, and so on.

The number of steps required to synthesize a grasp was recorded for each of the objects and its 100 poses. The use of baseline policies i.e. random for the lower limit and BFS for the upper limit helped us in classifying the objects as easy, medium, and hard in terms of how difficult is it to find a path that leads to a successful grasp. Objects are "Easy" when taking a step in almost any direction will lead to a successful grasp, and "Hard" when very specific paths are needed to find a grasp.

Two objects with similar BFS and random performance will have similar numbers of paths leading to successful grasps, and so differences in performance between the policies would be due to algorithmic failures, not an inherent difficulty. Performance of a policy Y w.r.t BFS at a given Step X is given by the ratio

$$\text{Policy performance}_{Y,X} = \frac{\text{Number of successful grasps at Step X using policy Y}}{\text{Number of successful grasps at Step X using BFS}} \tag{7.1}$$

The random policy performance at Step 2 is used for the classification of objects as easy, medium, and hard. For example, if the BFS result shows that out of 100 poses 40 poses have a successful grasp found after the second step and the random policy is only able to find a grasp at after the second step for 10 poses, the policy is considered to have performed at 25% of the optimal performance or in other words the ration= would be 0.25. Objects with the ratio at Step 2 $\leq$ 0.40 are considered hard, objects between 0.41 and 0.80 as medium, and objects with a ratio $>$ 0.80 as easy. With this criteria the test objects were classified as follows:

1. Easy: Tomato soup can (005), Bowl (024), Mug (025)
2. Medium: Apple (013), Bleach cleanser (021), Power drill (035), Baseball (055)
3. Hard: Cracker box (003), Mustard Bottle (006), Pudding box (008), Potted meat can (010), Toy airplane (072-a)

The discussion of results will be done based on this classification to aid in ease of comparison. Table 7.1 shows the average steps taken and the success percentage at step 5 for the objects tested. From Table 7.1 we can see that for Easy objects using any policy leads to a successful grasp and all policies achieve 100 success at the 5 step mark with average steps at 1.1. Moving to the medium category objects, we see that the random policy starts to fail and needs more steps than other policies to find

Table 7.1: Simulation results for applying each policy to each object in 100 pre-set poses. Success is defined as reaching a view containing a grasp. For cases where no grasp was found, the step count was considered to be 6 for average calculation. Average steps are shown along with the success percentage in square brackets. The color coding for each object shows the best policy in green and worst in red.

	Object ID	Object	Average steps [Success % at step 5]							
			Baseline Policies			Heuristic Policies		Data-driven Policies		
			Random	Brick	BFS	2D Heuristic	3D Heurisitc	Qlearning	PCA→LR	PCA→LDA
Easy	005	Tomato Soup Can	1.6 [100]	1.0 [100]	1.0 [100]	1.3 [100]	1.0 [100]	1.0 [100]	1.0 [100]	1.0 [100]
	024	Bowl	1.0 [100]	1.1 [98]	1.0 [100]	1.0 [100]	1.0 [100]	1.1 [99]	1.1 [98]	1.1 [99]
	025	Mug	1.0 [100]	1.1 [100]	0.9 [100]	1.1 [100]	1.1 [100]	1.1 [100]	1.0 [100]	1.0 [100]
		Average	**1.2 [100]**	**1.1 [99]**	**1.0 [100]**	**1.1 [100]**	**1.0 [100]**	**1.1 [100]**	**1.0 [99]**	**1.0 [100]**
Medium	013	Apple	2.9 [100]	1.4 [100]	1.3 [100]	1.4 [100]	1.8 [100]	1.4 [100]	1.9 [100]	1.8 [100]
	021	Bleach Cleanser	1.8 [96]	1.2 [100]	1.0 [100]	1.3 [100]	1.2 [100]	1.1 [100]	1.3 [100]	1.2 [100]
	035	Power Drill	2.9 [88]	2.3 [100]	1.1 [100]	2.3 [100]	1.6 [100]	1.8 [100]	1.6 [100]	1.7 [100]
	055	Baseball	2.7 [98]	2.0 [100]	1.4 [100]	1.8 [100]	1.9 [100]	2.0 [100]	1.8 [100]	1.8 [100]
		Average	**2.6 [95]**	**1.7 [100]**	**1.2 [100]**	**1.7 [100]**	**1.6 [100]**	**1.6 [100]**	**1.7 [100]**	**1.6 [100]**
Hard	003	Cracker Box	4.4 [46]	2.1 [100]	1.5 [100]	1.6 [100]	1.7 [100]	2.3 [97]	2.0 [90]	2.0 [90]
	006	Mustard Bottle	4.1 [47]	1.8 [100]	1.4 [100]	1.9 [92]	1.4 [100]	2.6 [83]	1.4 [100]	1.4 [100]
	008	Pudding Box	4.5 [50]	2.2 [98]	1.6 [100]	3.6 [58]	1.7 [100]	1.9 [98]	1.9 [99]	1.8 [98]
	010	Potted Meat Can	4.9 [37]	2.5 [100]	1.8 [100]	2.8 [88]	2.0 [100]	2.0 [98]	2.0 [99]	1.9 [100]
	072-a	Toy Airplane	5.6 [17]	5.2 [28]	3.9 [100]	5.1 [64]	3.9 [77]	5.4 [20]	5.5 [16]	5.5 [19]
		Average	**4.7 [39]**	**2.8 [85]**	**2.0 [100]**	**3.0 [81]**	**2.1 [95]**	**2.8 [79]**	**2.6 [81]**	**2.5 [81]**

a grasp. All other policies require 1.6 steps whereas the random needs 2.6. There is not much differentiating the performance of other policies. It is the hard category of objects in which we see a significant performance difference between the policies. The 3D Heuristic completely outperforms the rest with an average step of 2.1 (BFS is 2.0) and a success percentage of 95. It is followed by the self-supervised policies at 2.5. The random policy miserably fails with an average step size of 4.7 and a success percentage of 39. For the scenarios where no grasp was found at step 5, the step count was considered as 6 for the average step size calculation. Fig. 7.3, 7.4, 7.5 show the detailed plot of step-wise success rate for the objects and the policies.

The data collected can also be analyzed using the policy performance metric defined in Equation 7.1. Fig. 7.6 shows the performance of the policies at Step 1 and Step 3 using the policy performance metric Again we see a similar pattern in the results. Almost all policies perform close to the optimal BFS solution for easy objects at Step 1. At Step 3, both easy and medium objects perform as well as the

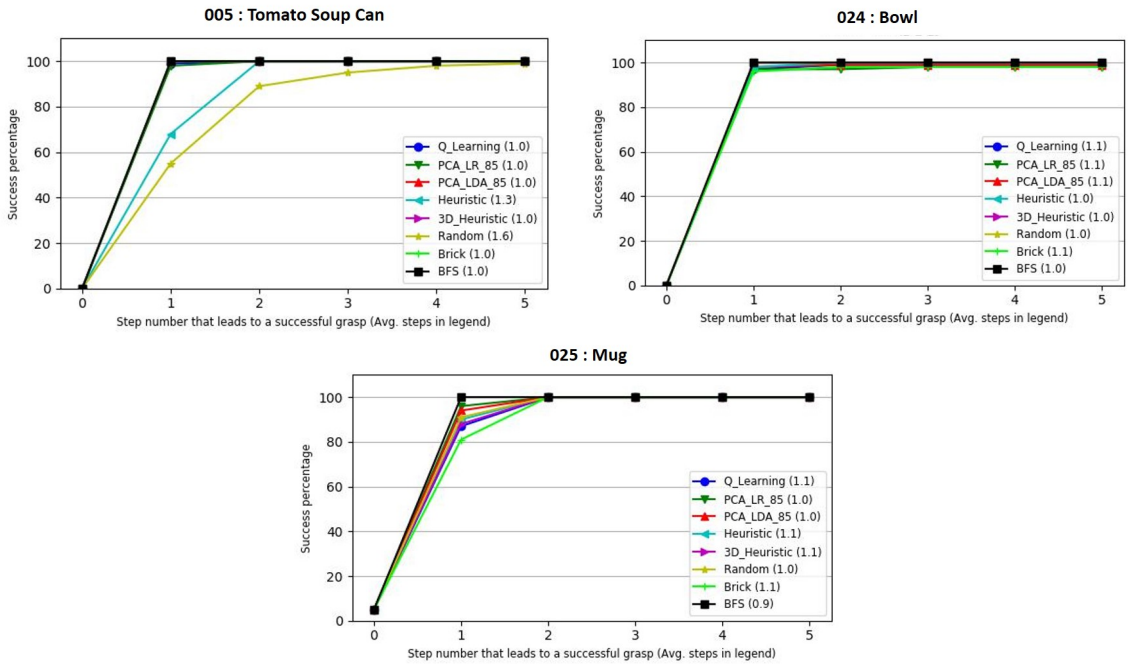


Figure 7.3: Easy classified objects : Simulation results showing the success percentage after each step

BFS solution. Overall, in simulation, the 3D Heuristic performed the best, followed by the self-supervised learning approaches, Q-Learning, and the 2D Heuristic. For half of the objects we tested, the 3D Heuristic performed best, while for objects 003, 010, 013, 021, 025, and 055 another algorithm performed better.

One reason the 3D Heuristic may be failing in some cases is that the heuristics are constrained to only considering the immediate next step. Our data-driven approaches can learn to make assumptions about several steps in the future, and so may be at an advantage on certain objects with complex paths. In addition, the optimistic estimations explained in Section 6.2.2 will not hold for all objects and cases, causing the Heuristic to waste time exploring promising-looking dead-ends.

One reason the data-driven techniques underperform for some cases may be due to the HAF representation used to compress the point cloud data, which creates a very coarse-grained representation of the objects, obliterating fine details. Addition-

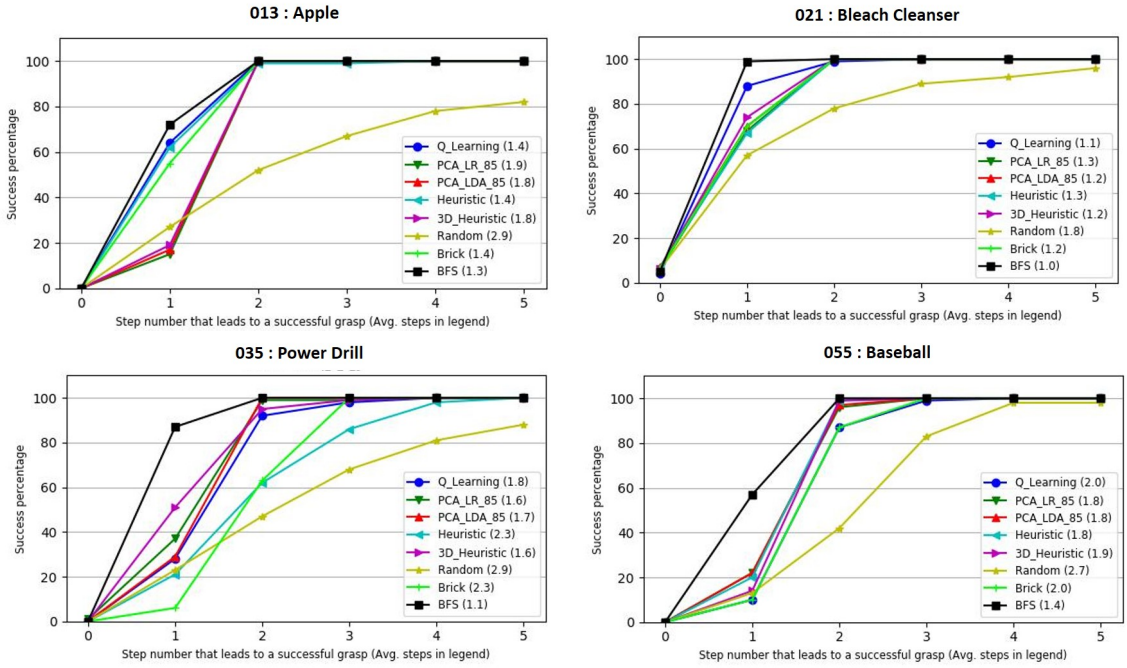


Figure 7.4: Medium classified objects: Simulation results showing the success percentage after each step

ally, HAF representations cannot represent certain types of concavities, hampering their utility for complex objects. A much finer grid size, or an alternative representation of the state vector, could help in improving the performance of the data-driven techniques.

The radar plots shown in Fig. 7.7 for three objects and policies illustrate the point that every policy takes a different path to find the grasp. An observation with data-driven policies was that they did not prefer the directions S, SW, and SE. This directional aversion can be attributed to the training set of objects used, for which grasps were found without taking steps in these directions and also explains the significant failure of data-driven policies on the toy airplane object (072-a) which requires a step to be taken in S/SW/SE to generate a grasp as done by the heuristic policy.

We found that all methods consistently outperformed random, even on objects

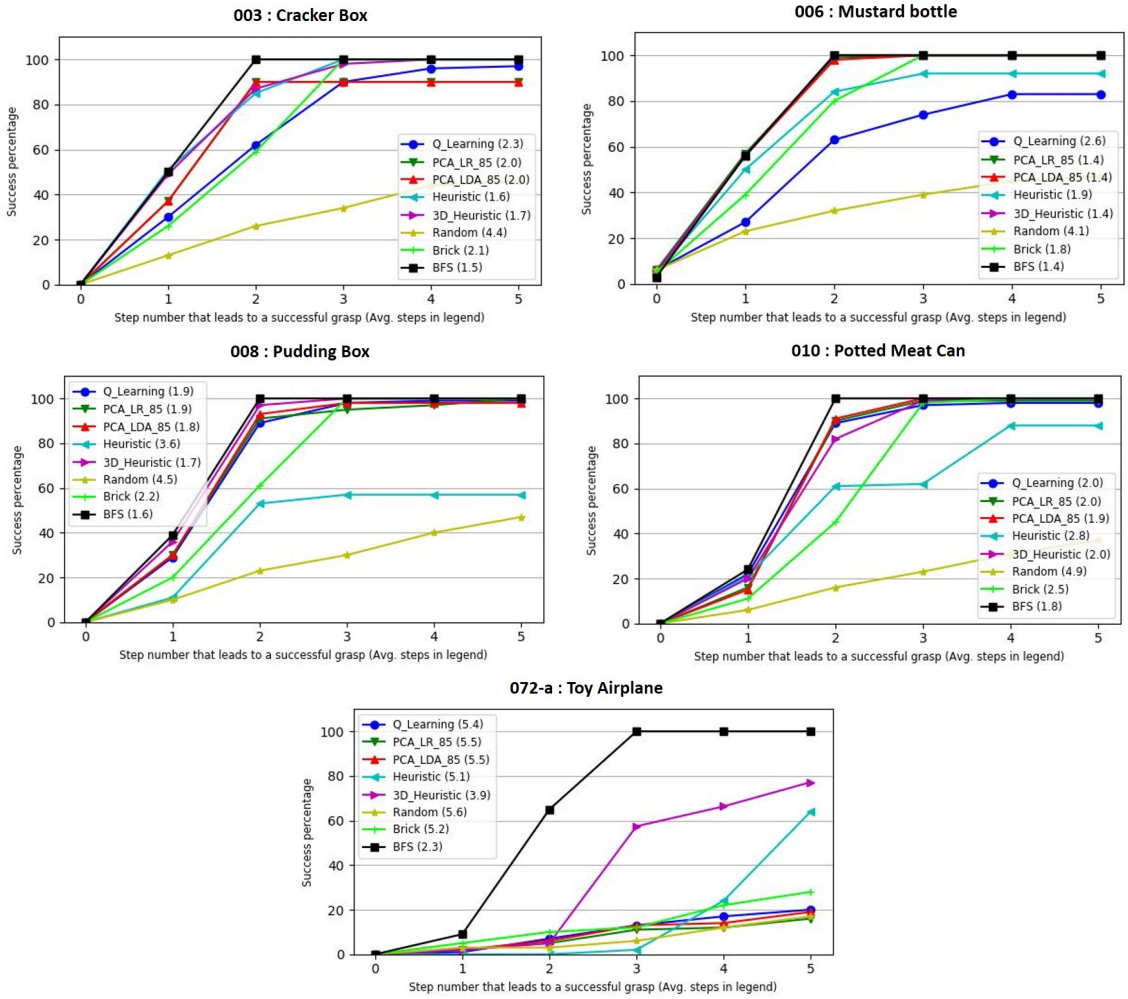


Figure 7.5: Hard classified objects: Simulation results showing the success percentage after each step

classified as hard. It is important to note that even the brick policy was able to find successful grasps for all objects except for the toy airplane object (072-a), suggesting that incorporating active vision strategies even at a very basic level can improve the grasp synthesis for an object.

The toy airplane object (072-a) deserves special attention as it was by far the hardest object in our test set. It was the only object tested for which most algorithms did not achieve at least 80% optimal performance by step 5, as well as having the lowest random to BFS ratio at step 5. We also saw (both here and in the real

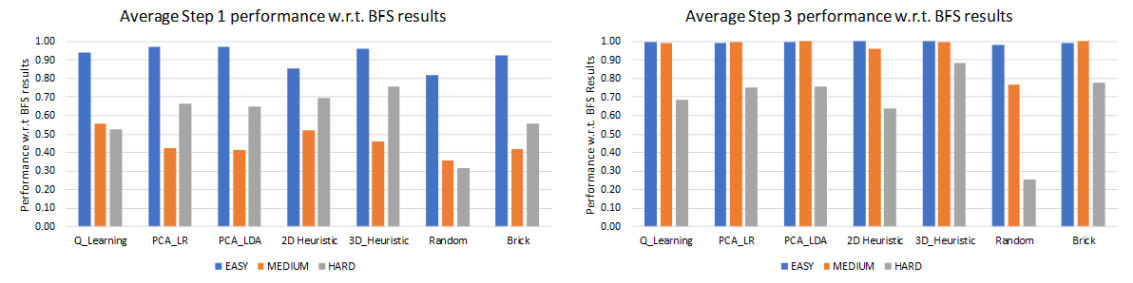


Figure 7.6: A comparison of the performance of various policies for objects categorized into easy, medium, and hard, for Step 1 and Step 3

world experiments) that heuristic approaches performed the best on this complex and unusual object, while the data-driven approaches all struggled to generalize to fit it.

Easy and Medium category objects come very close to optimal performance around step 3, as seen in Fig. 7.6. Given how small the possible gains on these simple objects can be, difficult objects should be the focus of future research.

Of the methods we examined, Heuristics (2D, 3D, and information gain) had the advantage of fast set up time, since they did not need training, but longer run time, since they performed more complicated calculations. The deep Q-learning had the disadvantage of needing extensive training time but ran quickly, and the self-supervised learning approaches (LDA and logistic regression) could be trained quickly and ran quickly, but needed a long initial data collection period.

7.2 Comparison with Information Gain Heuristic

Using the same simulation setup the Information Gain Heuristic policy was compared to the 3D heuristic policy. The comparison results are shown in Table 7.2, where the number of viewpoints required was converted to the effective number of 3D Heuristic steps for comparison. One step is the distance travelled to move to an

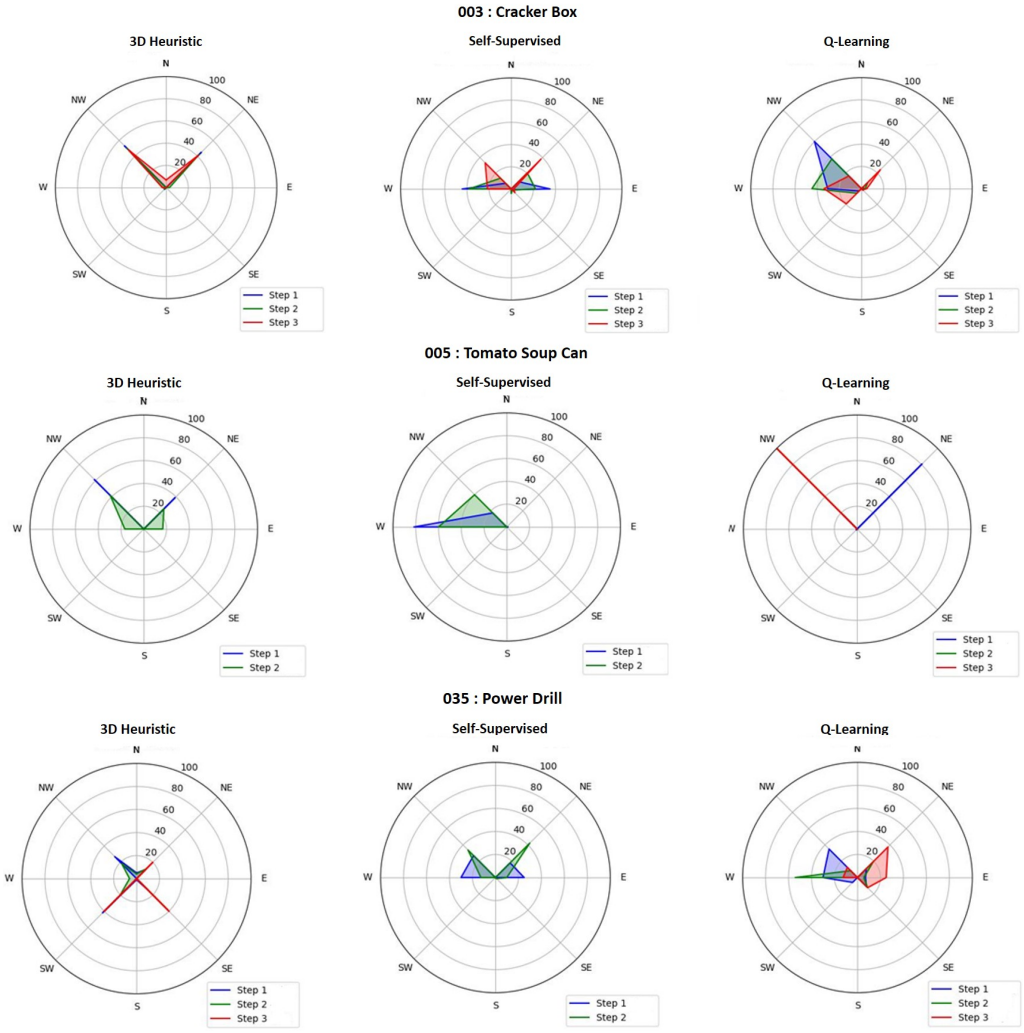


Figure 7.7: Radar plots showing the percentage of each direction taken at each step for 3 different objects and policies.

adjacent viewpoint along the viewsphere in the discretized space with $v_r = 0.4$m, $v_s = 20°$.

We see an average of 41% reduction in camera movement and with the 3D Heuristic policy, confirming our theory that only certain types of information warrants exploration and that by focusing on grasp containing regions we can achieve good grasps with much less exploration. As a side benefit, we also see a 73% reduction in processing time with the 3D Heuristic policy, as it considers far fewer views

Table 7.2: Comparison between the exploration pattern employed by the Information Gain Heuristic and the 3D Heuristic's grasp weighted exploration.

Object name	Steps Comparison				Timing Comparison (sec)		
	Information Gain Heuristic		3D Heuristic	Reduction %	Information Gain Heuristic	3D Heuristic	Reduction %
	nVPs	nEffSteps	nSteps				
Tomato Soup Can (5)	1	2.3	1	57	22	3	86
Mustard Bottle (6)	0.92	2.1	1.4	33	35	10	71
Pudding Box (8)	1	2.8	1.6	43	26	5	81
Potted Meat Can (10)	1.2	3.3	2	39	22	5.5	75
Apple (13)	1	2.3	2	13	9	4	56
Bowl (24)	1	2.9	1	66	14	3	79
Mug (25)	1	2.3	1	57	17	3	82
Power Drill (35)	1.2	2.9	2.2	24	54	19	65
Toy Airplane (72-a)	1.6	4.7	3	36	75	28	63
	Average			41	Average		73

in each step.

7.3 Real World Study

The real world testing was done on a subset of objects in simulation along with two custom objects built using lego pieces. The grasp benchmarking protocol in Bekiroglu et al. (2020) was implemented to assess the grasp quality based on the five scoring parameters specified. Another grasp benchmarking protocol focused on vision-based approaches is Kootstra et al. (2012), but it is simulation-based and needs the input to be in the form of stereo images which does not fit well with our pipeline's need for a point cloud input. Also, it lacks the shaking and rotational test metrics available in Bekiroglu et al. (2020). The 3D Heuristic and the Q-Learning policies were selected and tested with the objects. The results for the tests performed are shown in Table 7.3. A total of 18 object-pose-policy combinations were tested with 3 trials for each and the average across the trails has been reported. The objects used along with their stable poses used for testing are shown in Fig. 7.8.

In real world trials, we found that the 3D heuristic works consistently, but the Q-Learning is at times unreliable. When run in simulation, the paths Q-Learning

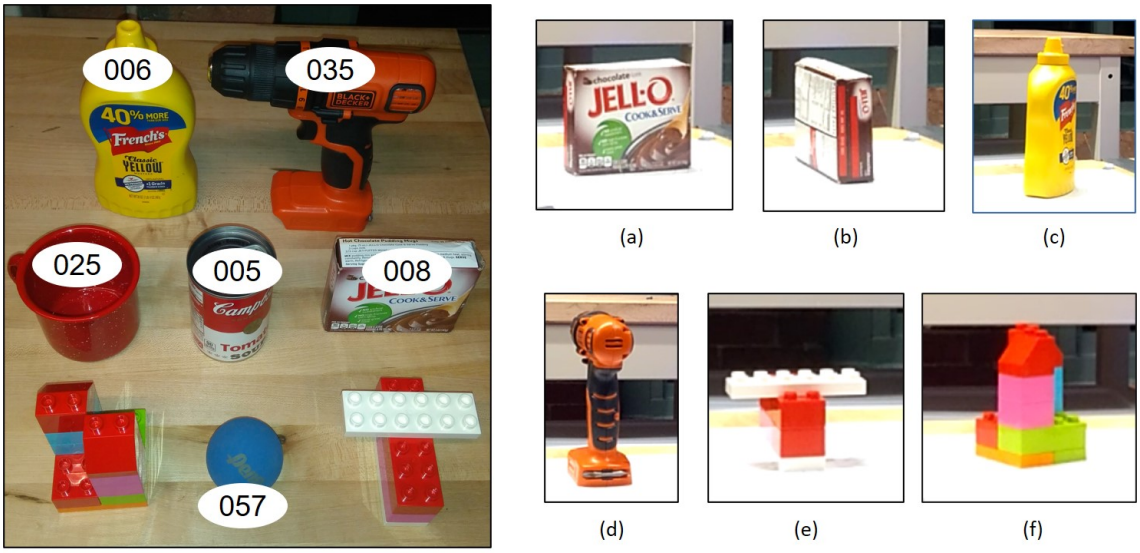


Figure 7.8: The left image shows the set of objects used for real world testing along with their YCB IDs. On the right are the stable poses used for testing (the manipulator base is towards the right). (a) [YCB ID : 008] Stable Pose #1, (b) [YCB ID : 008] Stable Pose #2, (c) [YCB ID : 006] Stable Pose #1, (d) [YCB ID : 035] Stable Pose #1, (e) [Custom Lego 1] Stable Pose #1, (f) [Custom Lego 2] Stable Pose #1. Objects with YCB IDs 005 and 057 are considered symmetrical and are used in the same orientation as shown in the left image. Likewise, object 025 was tested in only one stable pose, with the handle facing away from the robot.

picks for the real world objects produce successful grasps. The difference between our depth sensor in simulation and the depth sensor in the real world as discussed in Section 4.1.1 seems to be causing the disconnect. This sensor difference also explains why more steps were required in the real world than in simulation. The heuristic policy adapts to the missing information whereas the Q-Learning is not able to do as it has not been trained with such data. Nonetheless, the reliability of the 3D Heuristic demonstrates that simulated results can be representative of reality, although there are some differences.

Table 7.3: The list of objects tested for 3D Heuristic and Q-Learning policies along with the benchmarking results

Object name	Stable Pose	Policy	Policy Success Percentage	Avg Steps taken	Avg Grasp Quality	Benchmarking results				
						C1	C2	C3	C4	C5
Chocolate pudding box (8)	1	Q Learning	67	5.50	167	434	8	100	100	100
		3D Heuristic	100	3.00	167	205	4	100	100	100
	2	Q Learning	100	3.00	172	680	9	100	100	100
		3D Heuristic	100	3.00	171	675	12	100	100	100
Mustard container (6)	1	Q Learning	100	3.00	158	430	15	33 (F2)	0 (F3)	-
		3D Heuristic	100	2.33	161	370	6	33 (F2)	0 (F3)	-
Metal Mug (8)	1	Q Learning	100	2.00	180	125	40	100	100	100
		3D Heuristic	100	2.00	180	80	20	100	100	100
Tomato soup can (2)	1	Q Learning	0	-	-	-	-	-	-	-
		3D Heuristic	100	3.00	166	331	2	100	100	100
Racquet Ball (57)	1	Q Learning	100	4.00	150	40	2	100	100	100
		3D Heuristic	100	5.00	154	32	2	100	100	100
Power Drill (35)	1	Q Learning	100	3.00	164	351	1	100	100	100
		3D Heuristic	100	2.00	165	450	3	100	100	100
Custom Lego Object 1	1	Q Learning	0	-	-	-	-	-	-	-
		3D Heuristic	100	3.00	162	230	4	100	100	100
Custom Lego Object 2	1	Q Learning	0	-	-	-	-	-	-	-
		3D Heuristic	100	4.50	168	180	1	100	100	100

Chapter 8

Conclusions

In this thesis, heuristic-based and data-driven policies are presented to achieve viewpoint optimization to aid robotic grasping. In the simulation and real world testing, we implemented a wide variety of active vision approaches and demonstrated that, in overall performance, the 3D Heuristic outperformed both data-driven approaches and naive algorithms. We concluded that prioritizing exploration of grasp-related locations can produce both faster and more accurate heuristic policies. Also, we noticed that the data-driven policies had an edge over heuristic policies for some objects due to their inherent nature of considering multiple steps ahead as opposed to the heuristic policies which can see only one step ahead. From our optimal search, we demonstrated that for most objects tested, both types of approaches perform close to optimal. We were able to identify that the complex objects in our test set like the toy airplane and custom Lego objects are not only dissimilar to our training objects, but they are also objectively more difficult for viewpoint optimization. In the real world testing, we demonstrated that while sensor differences impacted all algorithms' performances, the heuristic-based approach was sufficiently robust to generalize well to the real world while our data-driven approaches were

more sensitive to the changes in sensor behavior.

Both types of policies, i.e. heuristic-based and data-driven, had their pros and cons. The execution times of the policies were less than 1 sec for the data-driven policies and for the heuristic ones they ranged from 0.5 sec to 5 sec based on the size of the target object. The speed difference is due to the processing of raw point cloud data in the heuristic policies as opposed to the compressed state vector used in the data-driven policies. This data compression removes potentially useful information. The nature of our state-vector makes data-driven policies less reliable as seen with the tests involving the custom Lego objects. Using different data compression methods to generate the state vectors containing more data could be used to enhance the performance of the data-driven techniques along with using more objects for training and testing.

Future research should prioritize what we have identified as difficult objects over simple ones, as it is only in the more difficult objects that gains can be made and good policies discerned from poor ones. Additionally, our work depended on discrete movements through the viewsphere. Future work should consider the possibility that continuous motion through the viewsphere may outperform discrete strategies.

Bibliography

Ammirato, P., Poirson, P., Park, E., Košecká, J., and Berg, A. C. (2017). A dataset for developing and benchmarking active vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1378–1385.

Arruda, E., Wyatt, J., and Kopicki, M. (2016). Active vision for dexterous grasping of novel objects. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2881–2888.

Bekiroglu, Y., Marturi, N., Roa, M. A., Adjigble, K. J. M., Pardi, T., Grimm, C., Balasubramanian, R., Hang, K., and Stolkin, R. (2020). Benchmarking protocol for grasp planning algorithms. *IEEE Robotics and Automation Letters*, 5(2):315–322.

Caldera, S., Rassau, A., and Chai, D. (2018). Review of deep learning methods in robotic grasp detection. *Multimodal Technologies and Interaction*, 2(3):57.

Calli, B., Caarls, W., Wisse, M., and Jonker, P. (2018a). Viewpoint optimization for aiding grasp synthesis algorithms using reinforcement learning. *Advanced Robotics*, 32(20):1077–1089.

Calli, B., Caarls, W., Wisse, M., and Jonker, P. P. (2018b). Active vision via extremum seeking for robots in unstructured environments: Applications in object

recognition and manipulation. *IEEE Transactions on Automation Science and Engineering*, 15(4):1810–1822.

Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52.

Calli, B., Wisse, M., and Jonker, P. (2011). Grasping of unknown objects via curvature maximization using active vision. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 995–1001.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Chu, F. J., Xu, R., and Vela, P. A. (2018). Real-world multiobject, multigrasp detection. *IEEE Robotics and Automation Letters*, 3(4):3355–3362.

Coleman, D., Sucan, I., Chitta, S., and Correll, N. (2014). Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. *arXiv e-prints*, page arXiv:1404.3785.

Daudelin, J. and Campbell, M. (2017). An Adaptable, Probabilistic, Next-Best View Algorithm for Reconstruction of Unknown 3-D Objects. *IEEE Robotics and Automation Letters*, 2(3):1540–1547.

de Croon, G., Sprinkhuizen-Kuyper, I., and Postma, E. (2009). Comparing active vision models. *Image and Vision Computing*, 27(4):374–384.

Du, G., Wang, K., Lian, S., and Zhao, K. (2021). Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*, 54(3):1677–1734.

Findlay, J. and Gilchrist, I. (2003). Active vision: The psychology of looking and seeing. *Journal of Neuro-ophthalmology - J NEURO-OPHTHALMOL*, 26.

Fischinger, D. and Vincze, M. (2012). Empty the basket - A shape based learning approach for grasping piles of unknown objects. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2051–2057.

Fu, X., Liu, Y., and Wang, Z. (2019). Active Learning-Based Grasp for Accurate Industrial Manipulation. *IEEE Transactions on Automation Science and Engineering*, 16(4):1610–1618.

Gallos, D. and Ferrie, F. (2019). Active vision in the era of convolutional neural networks. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pages 81–88.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Huebner, K., Welke, K., Przybylski, M., Vahrenkamp, N., Asfour, T., Kragic, D., and Dillmann, R. (2009). Grasping known objects with humanoid robots: A box-based approach. In *2009 International Conference on Advanced Robotics*, pages 1–6.

Karasev, V., Chiuso, A., and Soatto, S. (2013). Control recognition bounds for visual learning and exploration. In *2013 Information Theory and Applications Workshop (ITA)*, pages 1–8.

Khalfaoui, S., Seulin, R., Fougerolle, Y., and Fofi, D. (2012). View planning approach for automatic 3D digitization of unknown objects. In *Lecture Notes in*

Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 7585 LNCS, pages 496–505. Springer Verlag.

Kootstra, G., Popović, M., Jørgensen, J., Kragic, D., Petersen, H., and Krüger, N. (2012). Visgrab: A benchmark for vision-based grasping. *Paladyn, Journal of Behavioral Robotics*, 3(2).

Kurenkov, A., Ji, J., Garg, A., Mehta, V., Gwak, J., Choy, C., and Savarese, S. (2018). DeformNet: Free-form deformation network for 3D shape reconstruction from a single image. In *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, volume 2018-January, pages 858–866. Institute of Electrical and Electronics Engineers Inc.

Mahler, J., Matl, M., Liu, X., Li, A., Gealy, D., and Goldberg, K. (2017). Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. *arXiv preprint arXiv:1709.06670*.

Morales, A., Recatala, G., Sanz, P., and del Pobil, A. (2001). Heuristic vision-based computation of planar antipodal grasps on unknown objects. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 1, pages 583–588 vol.1.

Paletta, L. and Pinz, A. (2000). Active object recognition by view integration and reinforcement learning. *Robotics and Autonomous Systems*, 31:71–86.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn:

Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2016-June, pages 3406–3413. Institute of Electrical and Electronics Engineers Inc.

Rasolzadeh, B., Björkman, M., Huebner, K., and Kragic, D. (2010). An active vision system for detecting, fixating and manipulating objects in the real world. *The International Journal of Robotics Research*, 29:133 – 154.

Richtsfeld, M. and Vincze, M. (2008). Grasping of unknown objects from a table top. In *Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments*.

Rusu, R. B. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA)*.

Salganicoff, M., Ungar, L. H., and Bajcsy, R. (1996). Active learning for vision-based robot grasping. *Machine Learning*, 23(2-3):251–278.

Saxena, A., Wong, L., Quigley, M., and Ng, A. Y. (2010). A vision-based system for grasping novel objects in cluttered environments. *Springer Tracts in Advanced Robotics Robotics Research*, page 337–348.

Viereck, U., Pas, A., Saenko, K., and Platt, R. (2017). Learning a visuomotor controller for real world robotic grasping using simulated depth images. In Levine, S., Vanhoucke, V., and Goldberg, K., editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 291–300.

Wang, C., Zhang, X., Zang, X., Liu, Y., Ding, G., Yin, W., and Zhao, J. (2020). Feature sensing and robotic grasping of objects with uncertain information: A review. *Sensors*, 20(13):3707.

Zhang, H. and Cao, Q. (2019). Fast 6D object pose refinement in depth images. *Applied Intelligence*, 49(6):2287–2300.

Zheng, Z., Ma, Y., Zheng, H., Gu, Y., and Lin, M. (2018). Industrial part localization and grasping using a robotic arm guided by 2d monocular vision. *Industrial Robot: An International Journal*, 45(6):794–804.