UNIVERSITY OF CALIFORNIA SAN DIEGO

Grasp Viewpoint Planning Methods for Low-cost Robotic Manipulators

A Thesis submitted in partial satisfaction of the requirements for the degree Master of Science

in

Electrical Engineering

by

James Latham Smith

Committee in charge:

Professor Henrik I. Christensen, Chair
Professor Nikolay A. Atanasov, Co-Chair
Professor Michael C. Yip

2020

The Thesis of James Latham Smith is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____
Co-Chair

_____
Chair

University of California San Diego

2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Dr. Henrik I. Christensen for his guidance throughout the stages of my thesis projects. He was instrumental in guiding me through the process of going from an idea to a finished project and was there to assist me in any struggles along the way. When I first arrived in his office, I had little to no robotics experience, and now I feel confident in my skills as a roboticist due to his helping hand.

Next, I would like to thank my fellow graduate students. Priyam Parashar was always extremely helpful and whether it be spit-balling random project ideas or her guiding me back on track from a crazy idea, her insight was tremendously helpful and greatly appreciated. Carlos Nieto-Granda, whose work ethic (and working hours) are unmatched of any researcher I've met was always there to lend some advice towards my project. What was was thought extremely difficult before speaking with him became easy after our discussions. Both Shengye Wang and Ruffin White were extremely helpful with all things ROS, Docker, and Linux and always provided excellent advice on which tools could help me get the job done. Ahmed Qureshi and Anwesan Pal were both helpful in all things related to neural networks; they gave countless tips on how to finally get my training to converge and fielded any of my questions related to neural nets.

I want to thank my mother, father, and three brothers for their support and encouragement throughout my education.

Lastly, I want to thank my wife and best friend, Yihan Zi for putting up with my ups and downs and keeping me motivated throughout my degree. When the going got tough, she was always there to cheer me up and helped me to stay true to my goals. This thesis would not be possible without her by my side.

ABSTRACT OF THE THESIS

Grasp Viewpoint Planning Methods for Low-cost Robotic Manipulators

by

James Latham Smith

Master of Science in Electrical Engineering

University of California San Diego, 2020

Professor Henrik I. Christensen, Chair
Professor Nikolay A. Atanasov, Co-Chair

To enable a robotic platform that is low-cost, yet useful for manipulation tasks in household and low-volume production settings without prior calibration, two algorithms for grasp viewpoint planning are proposed. Both algorithms rely on performing motions on a virtual sphere in an object-centric workspace, utilizing only RGB images from an eye-in-hand camera to reach an optimal grasping viewpoint. The first method succeeds on grasping more than 80 % of the time on the Rethink Robotics Baxter Platform by utilizing image moments to calculate an optimal grasp pose, and the second method is capable of orienting the end effector within 5 degrees of an object on a simulated Universal Robotics UR5 platform using a Convolutional Neural

Network (CNN). From experimental results of these two algorithms, it can be shown that an RGB camera is sufficient for viewpoint planning and grasping of novel objects with simple geometries. Additionally, a viewpoint planning model can be trained for a new object to increase alignment accuracy and grasp success rates in a short period of time.

Chapter 1

Introduction

To allow robots to enter a human-centric world with large variability, new approaches are needed that are invariant to the exact object or workspace present. At the same time, the cost of robotic manipulators needs to be drastically reduced, but a reduction in their cost will likely reduce some of the precision guarantees found with more expensive systems.

This thesis proposes two algorithms to enable a robotic arm to achieve a suitable pre-grasp orientation using only an RGB eye-in-hand camera sensor. The first algorithm is an image moment based method that utilizes the principle components of the segmented image of the object to maneuver around a sphere centered on the object. The second approach utilizes supervised learning to learn a model from a dataset of sample images labeled with an optimal action to take on a virtual sphere to end at a desired grasping orientation. Both methods are designed for an uncalibrated workspace with minimal required setup, enabling them to be deployed easily on low-cost, low-precision robotic manipulators, while at the same time being robust to environmental changes that are inherent to the world we live in. An uncalibrated workspace here means that no explicit measuring of any of the workspace layout is required, no fiducial markers are used, and no prior CAD models of the object are utilized for grasping. The only calibration necessary is the intrinsic calibration of the eye-in-hand camera and for most cameras, the factory calibrated

intrinsics are sufficiently good for experiments.

Robotic manipulators have reached the point where they are no longer confined to factory floors, performing highly repetitive manufacturing tasks, but instead are commercially available. Collaborative robotic platforms such as Rethink Robotics' Baxter [8] or the Universal Robotics UR series [1] are comparatively low-cost, yet capable of enabling a new age of robotics. These platforms have the flexibility to complete a variety of tasks, whereas many traditional factory manipulators are hard-coded for repetitive, precise tasks. Also, they are safe to work around humans, by placing velocity and acceleration limitations on joints and by detecting collisions with nearby objects or operators and protectively stopping themselves to avoid damage and injury. The recent advances and cost-reduction in robotic manipulators makes them desirable for use in homes and workplaces, where previously only humans could complete repetitive tasks, but there is still work to be done to enable robots to successfully operate in the world we live in at a reasonable cost. A challenge for several of these new products is the reduced accuracy and repeatability that comes with lowering component and engineering costs.

Beyond advances in robotic manipulation, advances in camera sensors (due to their extensive integration into mobile phones) have paved the way for mounting cheap, small, lightweight cameras onto robotic manipulators to increase their perceptible space and enhance their ability to manipulate objects dynamically. In addition, computer vision has shown tremendous progress in almost all sub-disciplines which has trickled over to the robotics community to enable interesting new research and applications. Although depth cameras such as the Microsoft Kinect or Intel Realsense Series (see [11]) are low cost and easy to use to get depth information using stereo vision or time of flight, they require more power, both from a computation and energy standpoint, and are still too large to integrate well into an eye-in-hand system like those proposed in this work. These are the exact reasons that this research focused on utilizing RGB-only cameras, although this work can later be ported over to an RGBD platform later as they become more capable, smaller, and more energy and computationally efficient.

To enable the utilization of robot manipulators in less structured environments including domestic settings it is necessary to consider methods that allow closed-loop servoing to unknown or partially known objects for grasping. Ideally the method should be able to handle unknown objects using a limited amount of compute resources and pre-training necessary to be deployable on service robots in homes or in collaborative settings with humans, and should not rely on extremely precise robotic manipulators, as these are far too expensive to be widely used.

1.1 Previous Work

1.1.1 Grasping Familiar Objects

There is much previous work in the area of grasping familiar objects, due to industrial robotic manipulators performing highly precise, repetitive tasks on the same objects for manufacturing. In most industrial robotics instances, objects are placed at predetermined poses and the robots motions can be pre-planned. This simplifies the perception problem, as there is no need to find where an object is in the workspace and also simplifies the motion planning problem, as we can pre-plan trajectories and therefore no collision-checking needs to be performed at runtime. These methods have historically yielded great results with the caveat of inflexibility to changes in objects, workspace or manufacturing process as the robot work cell needs to be retooled and reprogrammed if even a simple change occurs. Also, the cost of such systems is prohibitively high. This inflexibility and cost barrier is what makes this older workflow incapable of handling settings such as a household or small manufacturing production run.

One method of grasping known objects is to use a CAD model of the object, to enable tracking it's location in space. This process can yield accurate pose estimation results, and then the arm can be servoed to a grasp pose. Work such as [6], [12] do just that, and then, a tool like GraspIt! [18] can output a good candidate grasp pose based on the CAD model of the object. Tracking the pose of objects in a workspace is generally very accurate for highly textured objects,

but such methods are limited in that they require the use of CAD models and require proper initialization and lots of computational resources.

1.1.2 Grasping Novel Objects

Commonly used methods for grasping novel objects are model-driven or model free. Model driven approaches, which develop a set of rules for interacting with objects have always been the main approach, but recent advances in simulations, computational power, and deep learning have improved model-free approaches to make them a viable solution for many robotic manipulation tasks. These two approaches will be discussed in this section for grasping novel objects.

Another approach when no 3D model exists for an object is to use a structure from motion (SFM) framework such as [25] to reconstruct the object in view. This boils down to going from a novel object to a known object by ”learning” about the object through reconstruction. Such methods don't require a CAD model, as it is generated on the fly, but explicitly reconstructing the object is a computationally expensive and error prone process and generally requires depth data, which has the downfalls discussed at the beginning of this chapter.

There is prior research in using Image Based Visual Servoing (IBVS) with image moments as features for grasping such as [5], [17] and specifically for planar objects in [23]. Also, simple parallelogram features are used in [22] to achieve an optimal grasp pose for a pick-and-place task. With these approaches, the object is known ahead of time as the operator needs to first determine the visualization of the image features from a successful grasping orientation. Beyond this required initialization, throughout the servoing scheme, the camera can lose view of the object, causing the visual servoing process to fail.

This work is based on earlier viewpoint optimization work such as [15] but adds the elements of using image moments instead of junction points and utilizes a robotic manipulator to control the camera motion, benefiting from advances in both image processing and robotic

manipulation.

Model-free Robotic Manipulation for Grasping

There is significant recent work in using either Deep Learning, Reinforcement Learning or a combination of the two to achieve excellent grasping results for unknown object. The problem with these methods has been the requirement to generate enough training data to sufficiently learn a grasping policy. One method to generate enough data to help these data-driven methods perform well was used in [14], but required large resources to run a group of robots for many hours. Also, algorithms such as DEX-NET [16] and it's many successors, enabled a robot arm to get cutting-edge results in grasping novel objects. The most recent of which, DEX-NET 4.0, has the capability of choosing to use either a suction gripper or a parallel gripper, per object on the fly. Both of these concepts are limited in that they are only capable of performing top-down grasps, because it is a simple way to limit the dimension of the problem. It is a much more tractable problem to train a neural network to output an (x,y) coordinate with a gripper orientation angle (θ) in the image frame than trying to output the full pose of an end-effector in 3D space ($x, y, z, \theta, \rho, \phi$). This makes using such algorithms in an unstructured setting for small-scale manufacturing difficult as it would require the objects to be placed in the correct orientation ahead of time. Other works such as [2] show promise for learning to grasp with limited robot hours required, but there are still few methods shown to successfully grasp completely novel objects in a way that is useful for manufacturing. Examples of generalizing to new objects on training data from previous objects are [9], [19], but they only perform well for objects similar to those previously trained on and can not generalize to a completely novel object.

Recent work in viewpoint planning such as [4] achieves promising results by utilizing RL to optimize the grasping viewpoint, but relies on a RGBD sensor and calls on other grasp synthesis algorithms to decide on an optimal grasp.

The algorithms described in this work are unique in that they use only a monocular camera

and perform the grasp optimization in a self-contained, computationally-efficient manner that, and which doesn't require a high precision robot to perform. Also, of equal importance, the resulting grasps enable the robotic platform to perform tasks with the gripped object, and can make assumptions about the orientation of the object post-grasp.

1.2 Hardware

In this section, the hardware used for experiments is discussed.

1.2.1 Robotic Platforms

Figure 1.1: The Baxter platform

Figure 1.2: The Baxter's parallel gripper with eye-in-hand camera and IR sensor

Baxter

Baxter [8] is a collaborative robot with two 7-DOF arms, produced by Rethink Robotics. Baxter is easy to setup, comes with an easy-to-use ROS Python/ C++ interface, and many safety features that make it very useful for robotics research.

Universal Robotics UR5

The Universal Robotics UR5e is a collaborative robotic arm with 6-DOF widely used in research and industry. It is priced under $ 50,000 USD, making it a great choice for a small scale manufacturing operation.

1.2.2 Other Hardware

The computer used for experiments is a PC with an Intel i7 8700K with 16GB of DDR4 RAM and a Nvidia GTX 1080 Ti. The system uses Ubuntu 18.04 (Bionic) within a Docker

container for all experiments.

A Raspberry Pi and 8MP camera were attached to Baxter's gripper (directly in between the parallel grippers as shown in 1.2) for real world experiments, and the simulated UR5 has a simulated camera attached and centered on the end-effector.

1.3 Software

All software components were written in Python [24] (version 2.7) for its ease of prototyping and ROS support. ROS [21] (Robotics Operating System) is a middleware that provides an excellent infrastructure for robotics software development. The ROS version used is Melodic. For simulating robotics interactions, the Gazebo simulator [13] was used because of its convenient interface with ROS. The simulation environment sped up development by enabling integration testing of software components at faster than real-time speed. The Open Computer Vision Library (OpenCV) [3] was utilized for all image processing. PyTorch [20] was utilized for neural network implementations in the CNN Viewpoint Planner.

Chapter 2

Methodology

The task of grasping an object with a robotic manipulator can be split into three subtasks:

- localize the object
- determining a suitable grasp
- performing the grasp

This work attempts to complete all three subtasks using two different approaches, guiding a robotic arm to grasp novel objects in an uncalibrated workspace. The two algorithms share a perception front-end that processes eye-in-hand camera images as well as control methods for traversing a virtual sphere centered on the object. The reason for traversing a virtual sphere is that it allows the camera to maintain a relatively constant distance to the object, meaning that image features should remain similar. Also, movement along the sphere can be performed with very simple geometric primitives of moving tangent to it, followed by a centering of the image frame on the object. The overlap of the two algorithms will be discussed first, followed by their differences.

2.1 Perception

The perception module processes RGB images from the eye-in-hand camera and outputs the pixel-wise location, major and minor axes lengths, and the major axis angle of detected objects in the image. Although the implemented perception pipeline utilizes HSV thresholding, which is a common method for fine-tuning thresholds in the HSV colorspace to segment an object, any image processing algorithm that can output bounding boxes of objects in an image can be used in its place [1]. The algorithm assumes that only a single object is in view of the eye-in-hand camera, although this work can be expanded to handle multiple objects in the future without too much added complexity, as it just requires the algorithm to focus on one of the n objects in view over successive frames. Also, utilizing other detection methods such as saliency detection would allow for objects of any color to be used, expanding the potential of this work even further. Figure 2.1 illustrates the full perception pipeline as it moves from HSV thresholding to object detection.

Figure 2.1: The three stages of image processing pipeline performed on a sample cuboid object. The left image shows the original image taken from the eye-in-hand camera. The center image shows the HSV segmented image The right image shows the detected blob with a rectangle representing the centroid, dimensions and angle of the major and minor axes.

After HSV thresholding the image, the binary mask is passed on to a blob detector. These values are then passed on to the viewpoint optimizer which will be discussed in the following chapters.

[1] For the first Image Moment Viewpoint Planner, another requirement is the angle of the bounding box as this is used to align camera with the object

2.1.1 HSV Thresholding

HSV thresholding involves utilizing the Hue, Saturation and Value (HSV) color space for finding thresholding cutoffs in colored objects, and subsequently filtering out pixel values that are outside of that range. To convert an image from the RGB to HSV color space we use the following procedure:

$$V = max(R,G,B) \tag{2.1}$$

$$S = \begin{cases} \frac{V-min(R,G,B)}{V} & if\ V \neq 0 \\ 0 & if\ V = 0 \end{cases} \tag{2.2}$$

$$H = \begin{cases} \frac{60(G-B}{V-min(R,G,B)} & if\ V = R \\ 120 + \frac{60(B-R)}{)}V - min(R,G,B) & if\ V = G \\ 240 + \frac{60(R-G)}{V-min(R,G,B)} & if\ V = B \end{cases} \tag{2.3}$$

The output of the HSV thresholding step is a binary mask where a pixel is set to 1 if the corresponding HSV conversion of that pixel are within the thresholds and one otherwise. The HSV thresholds were manually tuned based on the object to be grasped and the environmental conditions. Because the binary mask can be rather noisy, a final step to filter out any noise is to perform a erosion, dilation, followed by another dilation and erosion.

2.1.2 Object Detection

The object detection step takes the HSV thresholded binary mask and outputs $\bar{x}$, $\bar{y}, l_{minor}$, l_{major} and θ_{major} using Equations 2.5, 2.7, 2.8, and 2.9.

Moment Equations

The equation for the $p-q^{th}$ Moment is:

$$M_{p,q} = \sum_{i,j \in object} i^p j^q \tag{2.4}$$

To calculate the centroid $(\bar{x}, \bar{y})$ of O from the binary segmented image I_b, the following formulas are used:

$$(\bar{x}, \bar{y}) = \left(\frac{M_{1,0}}{M_{0,0}}, \frac{M_{0,1}}{M_{0,0}} \right) \tag{2.5}$$

Next, we calculate the second-order central moments using:

$$\mu'_{2,0} = \frac{M_{2,0}}{M_{0,0}} - \bar{x}^2,\ \mu'_{1,1} = \frac{M_{1,1}}{M_{0,0}} - \bar{x}\bar{y},\ \mu'_{0,2} = \frac{M_{0,2}}{M_{0,0}} - \bar{y}^2 \tag{2.6}$$

which are then used in:

$$\theta = \frac{1}{2} tan^{-1} \left(\frac{2\mu'_{1,1}}{\mu'_{2,0} - \mu'_{0,2}} \right) \tag{2.7}$$

$$l_{major} = \sqrt{8 \left(\mu'_{2,0} + \mu'_{0,2} + \sqrt{4\mu'^2_{1,1} + \left(\mu'_{2,0} - \mu'_{0.2} \right)} \right)} \tag{2.8}$$

$$l_{minor} = \sqrt{8 \left(\mu'_{2,0} + \mu'_{0,2} - \sqrt{4\mu'^2_{1,1} + \left(\mu'_{2,0} - \mu'_{0.2} \right)} \right)} \tag{2.9}$$

to calculate the angle and lengths of major and minor axes respectively.

2.2 Spherical Traversal

This section describes the control algorithm that enables a camera on an end effector to traverse a virtual sphere centered on the object's center using only an RGB camera, given the

centroid of the object in the image frame. The radius of the sphere can be maintained by making corrections using distance approximations to the object's centroid.

Spherical traversal can be achieved by alternating between traveling in small line segments of distance d, and then pitching and yawing the camera, centering it on the object of interest as seen in Figure 2.2.

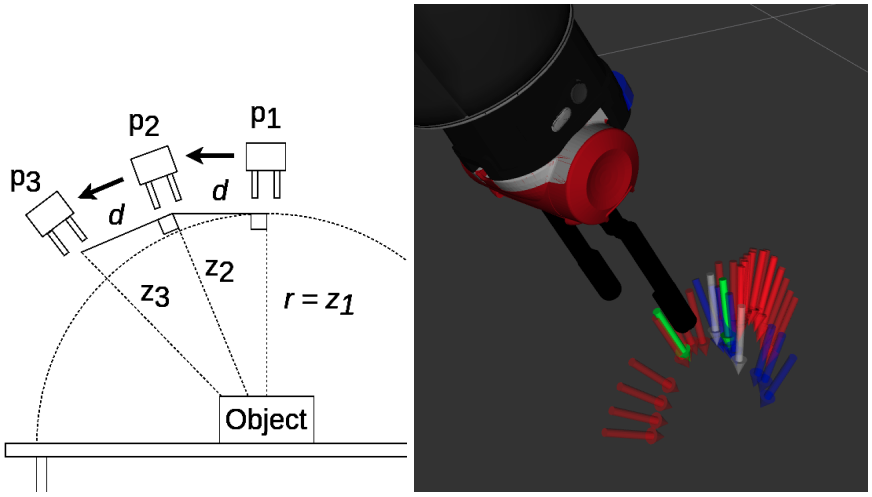


Figure 2.2: Visualization of spherical traversal. Red arrows represent the end effector pose through minor axis traversal, blue arrows represent the end effector pose though major axis traversal, the white arrow is the starting pose, and the green arrows represent the optimal poses associated with minor and major axis traversal

Planar Motions

For planar motions, only the translation of the end-effector changes while rotation remains constant. The end-effector's rotation component is represented as $\mathbf{R}_t$, a rotation matrix at time t. The translation component at time t is $\mathbf{T} = [x_t, y_t, z_t]^T$. We can calculate T_{t+1}, the translation at

time $t+1$ after moving distance d in direction θ using the following formulas:

$$x_{t+1} = d * cos(\theta)$$
$$y_{t+1} = d * sin(\theta)$$
$$z_{t+1} = 0$$
$$T_{t+1} = [x_{t+1}, y_{t+1}, z_{t+1}]^T$$
$$\mathbf{t}_{t+1} = \mathbf{R}_t \cdot t_{trans}$$

Note: The direction θ is defined in the camera frame, so 0 radians corresponds to right, $\frac{\pi}{2}$ up, π left, and $\frac{3\pi}{2}$ down.

Rotational Motions: Yaw Pitch and Roll

For yaw, pitch and roll motions, only the rotation of the end-effector changes while its translation component remains constant. With $\mathbf{R}_t$ and $\mathbf{T}_t$, the effector rotation at time $t+1$ after performing a rotation of θ about the camera's x, y or z axis (for yaw, pitch and roll respectively) can be represented as $\mathbf{R}_x$, $\mathbf{R}_y$, or $\mathbf{R}_z$.

Centering Object in Image

To center the eye-in-hand camera on the object, a proportional controller reduces the error between the centroid of the segmented object and center pixel of the image frame, by yawing and pitching the camera while maintaining the same 3D position. Pose updates are performed as follows:

The motion scalar is a scalar multiple of the desired sphere radius, which decreases as the radius decreases (a closer object will require less camera motion to center on).

Algorithm 1 Centering Algorithm

procedure CENTERONOBJECT(thresh, max attempts, motion scalar)
  Get object centroid location $(\bar{x}, \bar{y})$
  while num attempts $\leq$ max attempts **do**
    $error_x = \frac{image_w}{2} - \bar{x}$
    $error_y = \frac{image_h}{2} - \bar{y}$
    if $error_x \geq thresh$ *or* $error_y \geq thresh$ **then**
      yaw camera($\frac{error_x}{motionscalar}$)
      pitch camera($\frac{error_y}{motionscalar}$)
    else return
    end if
  end while
end procedure

Distance Adjustment

As can be seen from 2.2 that after a few translations, the camera ends up moving off from the radius. To solve this, we estimate the distance to the object of interest using A.1, and then adjust the current radius based on the current estimate. This enables a more accurate spherical traversal, when necessary.

2.3 Image Moment Viewpoint Planning Algorithm

The Image Moment Viewpoint Planning Algorithm utilizes the perception front end and spherical motions defined in the previous section to arrive at an optimal grasping viewpoint for simple objects.

The main contributions of this algorithm are:

- A method that allows an uncalibrated robotic arm system with only an RGB eye-in-hand camera to grasp simple objects regardless of their orientation without a 3D model or prior experience with these objects

2.3.1 Algorithm Description

The viewpoint optimizer's goal is to find a suitable grasp pose for a new object. It can be broken down into two sections, the traversal of the virtual sphere guided by image moments and the optimization over the observed image features at each pose in the traversal process to find the ideal grasp. It performs the traversal, followed by the optimization for both the minor and major axis of the object, and upon completion, the camera's x-axis aligns with the object's minor axis and the camera's y-axis aligns with the objects major axis, enabling a successful grasp for a cuboid or cylindrical object.

Arc Traversal

The algorithm begins by centering the eye-in-hand camera on the object in view using the centering methodology discussed in 2.2. Next, it aligns the camera's Y-axis with the object's major axis by rotating the manipulator's wrist joint until it minimizes the angle between the two axes. By moving only a single joint whose motion won't cause collisions, we don't need to call the IK solver for this alignment portion. After alignment, the camera traverses a virtual arc centered on the object, by performing the spherical traversal method described above. It moves both directions along this arc until a joint limit is reached or a collision occurs, saving the joint angles associated with each viewpoint. At the conclusion of this traversal, the algorithm utilizes the optimization function discussed in Algorithm 3.

Viewpoint Optimizer

The image features used as optimization parameters are the ratio of the minor to major axis, R. The heuristic used for finding the optimal viewpoint is to first fit a 4^{th} order polynomial to the data, where the x-axis represents the sample number (the first set of samples' numbers are reversed so that if the arm were to follow the samples, it would traverse a complete arc) and the y-axis represents the ratio R. For cuboids and cylinders, there is either a single inflection point,

which is the optimal viewpoint, or three inflection points, the middle of which is the optimal viewpoint.

Overall Alignment Procedure

After the minor axis traversal and optimization, the arm moves to the joint angles associated with the chosen inflection point, and thus the arm arrives at it's estimate of the optimal alignment with the objects minor axis. Next, the same procedure is repeated for the major axis, traversing the virtual arc once again, optimizing using the same optimization function, only this time it will arrive at the final grasping viewpoint (as we have optimized both minor and major axes' alignment). From here, we are at a canonical view of the object and different grasping methods can be performed to grasp the object.

Algorithm 2 Arc Traversal Algorithm

procedure TRAVERSEARC($axis, \Delta d, radius$)
  Save current joint angles as $q[0]$
  while joint limit is not reached **do**
    Move Δd in $+y$
    Z_{est} = ESTIMATEZ($axis$)
    Move arm $Z_{est} - radius$ in z
    Recenter camera on centroid
    Store l_{maj}, l_{min} and q_t in l, w and q
  end while
  Save current joint angles as $q[0]$
  while joint limit is not reached **do**
    Move Δd in $-y$
    Z_{est} = ESTIMATEZ($axis$)
    Move arm $Z_{est} - radius$ in z
    Recenter camera on centroid
    Store l_{maj}, l_{min} and q_t in l, w and q
  end while
  return l, w and q
end procedure

Algorithm 3 Grasp Viewpoint Optimization Process

procedure OPTIMIZEVIEWPOINT($radius$)
  l, w, q = TRAVERSEARC($minor, \Delta d, radius$)
  Calculate ratio of l/w R_{major}
  Fit R_{major} to P^4 and find middle inflection point
  Move arm to q_{min}
  l, w, q = TRAVERSEARC($major, \Delta d, radius$)
  Calculate ratio of l/w R_{minor}
  Fit R_{major} to P^4 and find middle inflection point
  Move arm to q_{min}
end procedure

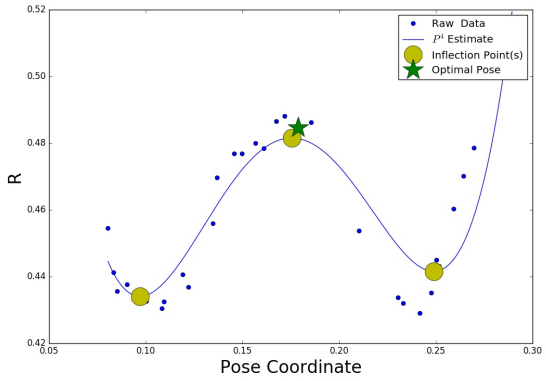


Figure 2.3: Optimization functions with 3 inflection point

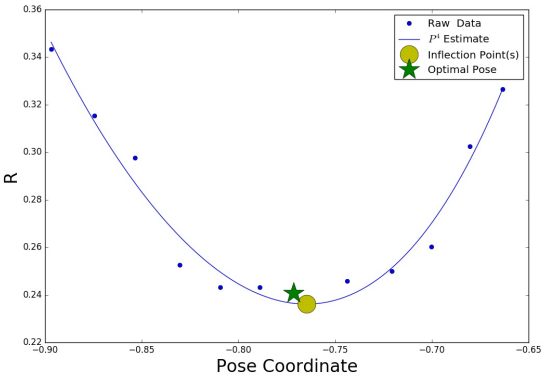


Figure 2.4: Optimization functions with 1 inflection point

Grasp Completion

After the gripper is orientated in the optimal pose for grasping, the arm moves in the direction of the lens (+Z) incrementally towards the object. During this descent, at each increment, the eye-in-hand camera is recentered on the object's centroid to ensure it remains centered between the grippers (and alignment with major axis can be performed, but from experiments, this wasn't necessary). The descent is complete when the infrared sensor mounted on the gripper detects an object within 10cm, at which point the grippers are closed and the object is lifted up from the surface, completing the grasp.

2.4 CNN Grasp Viewpoint Planner

In this section, a closed-loop viewpoint planning method for grasping is proposed that relies on a neural network to map an image to the optimal action on a sphere to arrive at a suitable grasp viewpoint. It lifts some of the restrictions of the algorithm proposed in Section 2.3, by enabling the camera to move within the image plane and by not forcing it to complete a full arc traversal. Also, because of the use of a neural network, no heuristics need to be provided and it can be trained for new objects quickly by collecting a set of sample images and their associated "optimal" actions. Lastly, by only basing actions off of the current eye-in-hand image, the algorithm becomes closed loop and can dynamically adjust to motion of the object.

This algorithm utilizes the same perception front end explained in 2.1 with the same underlying assumption that the object can be segmented from its background successfully and the same motion primitives for spherical traversal defined in 2.2. The following sections will break down the algorithm in more detail.

State Space

There are a few options for the state space representation in the spherical traversal environment. These are any combination of:

- the pose of the end effector (with or without noise): ee_t
- the most recent k images from the eye-in-hand camera: $I_t, I_{t-1}, ...I_{t-k}$
- the object of interest's pose at time t: p_{obj}

In the end, to reduce complexity and the amount of information needed to be passed to network (which must be known at runtime), only the RGB image from the end effector represents the current state.

Action Space

The action space is defined by the set: $a = [a_1\ a_2\ a_3\ a_4\ a_5]$ where a_1 is UP, a_2 is LEFT, a_3 is DOWN, a_4 is RIGHT and a_5 is NO OP, or no action. Action directions are relative to the image frame (DOWN is $+Y$ and RIGHT is $+X$).

Reward Function

The reward function used for learning was the inverse of the orientation error between the pose of the camera $\mathbf{x}$ and the object's normal $\mathbf{y}$:

$$R(\mathbf{x}, \mathbf{y}) = cos^{-1}\left(\frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}||\mathbf{y}|}\right) \tag{2.10}$$

2.4.1 Network Architecture

The network used for the policy is CNN that maps images to an action along the surface of the sphere. The network used is from the Resnet family [10] with 50 layers, and uses the pretrained weights from the ImageNet dataset [7].

2.4.2 Training Data Generation

The object to be grasped is first placed at a known position and orientation in the robot's workspace, which is necessary to label the image with the optimal action. In simulation, this data is readily available, whereas in the real world, the object needs to be placed on top of a known location (using a fiducial marker is the easiest way to do this). Next, demos are generated by taking images of the object from n viewpoints on the virtual sphere of radius r, either at random poses or at a set of pregenerated, equally spaced poses on the sphere. Figures 2.5 and 2.6 show examples of poses generated by each process. After moving to the pose on the sphere, the centering algorithm (Algorithm 1) is run to ensure that the object is centered within the image frame. Each viewpoint's image is then labeled with the action that would result in the largest reward, i.e. reducing the error between the object's normal vector and the camera's viewpoint vector per Equation 2.10.

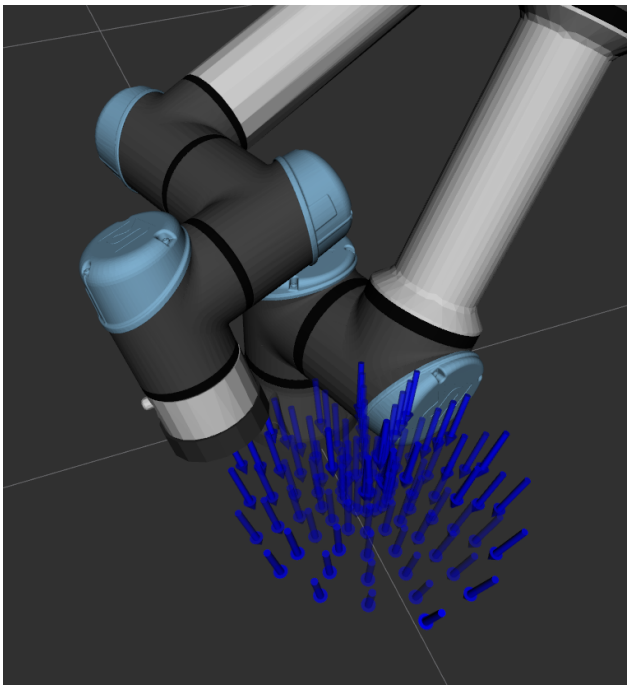


Figure 2.5: Poses Generated for "Uniform" Datasets. Each arrow represents a pose's translation, which is rotated about camera axis to generate more datapoints.

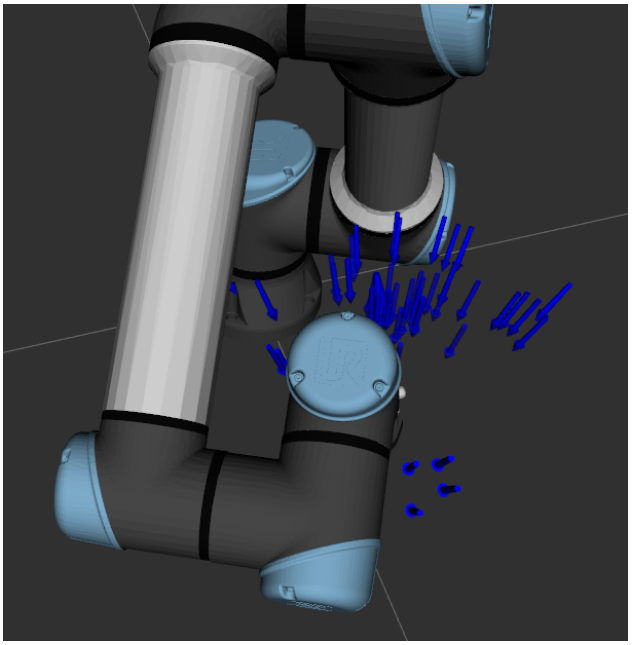


Figure 2.6: Poses Generated for "Random" Datasets. Each arrow represents a pose's translation, which is rotated about camera axis to generate more datapoints.

2.4.3 Data Pre-processing

First, the 640x480 image is center cropped to size 128x128. Then it is scaled down to a 64x64 image, shifted in both X and Y by an amount sampled randomly from the uniform distribution (with a maximum shift of 8 pixels in either direction). This encourages the network to be robust to slight errors in the centering algorithm.

2.4.4 Training Procedure

To train the network, a 70/30 training / validation split was used. The loss function used is Cross-entropy Loss (see Equation A.2, with a batch size of 32, with a starting learning rate of 0.001, decaying by a factor of $\gamma = 0.1$ every 5 epochs. The total number of epochs trained was 20.

2.4.5 Evaluation Procedure

There are two methods for evaluating the model's performance: (1) The standard method of validation set vs training set accuracy and (2) Evaluating the networks performance at improving the viewpoint of the end effector camera over a series of episodes. To perform (1), the 30 % of the dataset deemed validation images are passed into the network after each training epoch and the network's prediction accuracy is calculated as per Equation A.4. For (2), the the end-effector is initialized to a random position and orientation and the object is placed in the workspace. Next, the end-effector is moved in space to find the object of interest. Then, the network is successively passed the current eye-in-hand image, and it outputs an action prediction on the forward pass. After performing each action, the centering algorithm (see 2.2) runs. We can analyze the difference between the error in end effector position at the beginning of the episode compared to the end, as well as the number of cycles to reach a NO-OP action, with the latter implying that the model has reached a grasping viewpoint that is within rew_{thresh} of the desired pose.

Chapter 3

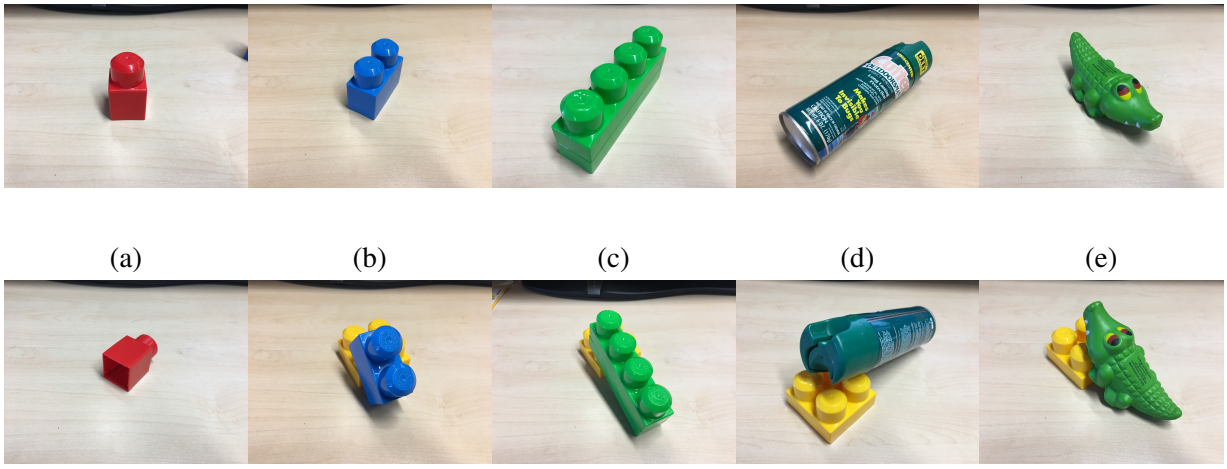
Experiments

3.1 Image Moment Viewpoint Planner

3.1.1 Experimental Setup

The experimental setup utilizes the Rethink Robotics Baxter Platform, shown in Figure 1.1. Baxter has two 7DOF arms, each embedded with an IR sensor and an eye-in-hand camera. Only one of Baxter's 7DOF arms is used for manipulation in experiments, which has been modified to include a new camera mounted directly in between the grippers in the end effector, is shown in Figure 1.2. This enables a 1-to-1 relationship between the orientation of the camera and the gripper, removing the need for eye-to-hand calibration that could result in larger error in the gripper pose. The algorithm was tested on many different starting poses of a few cuboids, a cylinder, and a small plush toy.

To perform the experiments, objects are placed at a pose that is reachable by Baxter's 7DOF manipulator, allowing it to move the eye-in-hand camera along a spherical motion around the object, centered on it's centroid. This work can later be ported to a mobile manipulator platform, which would enable the manipulator to reach this starting point quite easily (and allow for a more complete sphere traversal without reaching joint limits). Each experiment consists of



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 3.1: Sample Object Orientations. (a) and (f) show the 1x1 cuboid in upright and side orientation. (b) and (g) show the 1x2 cuboid in upright and raised orientation. (c) and (h) show the 1x4 cuboid in upright and raised orientation. (d) and (i) show the cylinder in flat and raised orientation. (e) and (j) show the alligator in flat and raised orientation.

performing the procedures defined in Section 2.3, until an optimal grasping pose is reached and then final grasping is performed.

3.1.2 Error Calculation using Homography

After the optimal grasp pose is reached the error in the yaw, pitch and roll of the gripper is calculated: Δ_ρ, Δ_θ and Δ_ψ respectively. (Note: There should be no error in roll, because after the spherical traversal is complete, the eye-in-hand camera's Y-axis is once again centered on the object centroid and aligned with the major-axis of the object). To accomplish this, using the four corners of the object in the final image we compute the homography matrix **H** that corresponds them with 4 corners of the object (measured in the real world only for error metric). From **H**, Δ_ρ, Δ_θ and Δ_ψ can be calculated and serve as a good metric for the error between the final orientation and the optimal grasp orientation. In the following sections, each object case is discussed.

3.1.3 The Cuboid Case

The cuboid case encompasses any objects that are mainly rectangular in shape which account for a large portion of common everyday objects. The objects used for testing are a few Duplo blocks, because they are easily HSV thresholded but also have some deformities, providing a good example of an imperfect cuboid and they fit between Baxter's standard parallel grippers. Also, they would be hard to track by pose detection algorithms because they are not highly textured. They also have a specular surface which creates strong reflections under certain lighting conditions so HSV thresholding ends up being effected by noise, yet the algorithm is still able to compensate. The cuboids were each placed in 2 different orientations in the workspace (as shown in Figure 3.1), and 10 grasp trials were performed for each orientation. After a grasp, the object is placed roughly back at the same orientation. The first orientation for each object is the object sitting on a table at a random orientation. This is a good base case, but for the second orientation, each sample block is stacked on top of another block, so that we can see how the algorithm handles objects resting on abnormal surfaces.

3.1.4 The Cylinder Case

The sample cylinder used was a green aerosol can. Similar to the cuboid, cylinders account for a large portion of the objects we interact with day to day. The cuboid was non-uniform in color and had a highly reflective surface, which pushed the HSV thresholding algorithm to its limits. The same process of 2 different orientations, each with 10 trials was performed for the cylinder case. Four corner points of the can were used for error calculation, compared to real-world measurements to calculate homography H. The first orientation was with the can sitting upright on the surface and the second was resting on another block, shown in Figure 3.1.

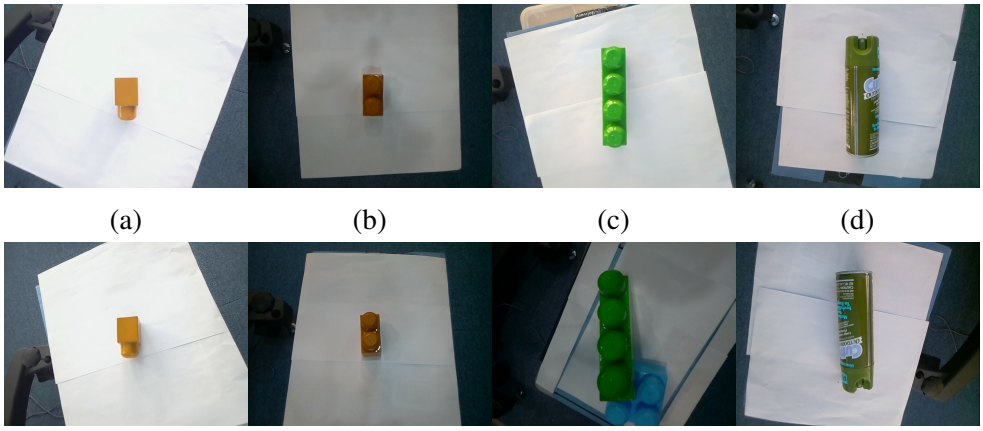
3.1.5 Alligator Case

The alligator case was used to demonstrate that this framework can perform well for objects other than simple geometric shapes such as cuboids and cylinders. The object used was a small alligator toy, seen in Figure 3.1. Similar to the cuboid and cylinder cases, 2 different orientations with 10 different grasp attempts were attempted with the alligator. Because the shape of the alligator is abnormal, we do not have orientation error estimations using the homography method used for cuboids and cylinders. The alligator orientations were lying flat on the table and on top of another sample block, shown in Figure 3.1.

3.1.6 Alignment Accuracy

Alignment accuracy was calculated using the homography method explained in Section 3.1.2 for the cuboid and cylinder objects. The results of these calculations are shown in Table 3.1. For the alligator object only grasp accuracy was recorded, because of the non-uniform shape of the alligator, which makes utilizing the homography method infeasible. Table 3.1 shows that the longer cuboids had better alignment and the cylinder case had the best alignment. This is due to the fact that the result of the optimization function for long and thin objects falls into the single inflection point case which is much easier to approximate. The 1x1 and 1x2 cuboids result in 3 inflection points, and sometimes due to noise in both the pose of the end effector and noise in the captured image, the optimization function is less well behaved and the polynomial fit can be a less than ideal approximation.

Also to note, if the optimization of the initial traversal over the object's minor axis results in a large error, this error will be propagated to the final optimal orientation. To further improve this, a smaller traversal distance Δd could be used, which will give more data to ideally find a better result from viewpoint optimization but result in a longer traversal time.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.2: Successful and unsuccessful examples of final viewpoint alignment. The top row shows good alignment results for the 1x1, 1x2, 1x4 cuboids and cylinder case and the bottom row shows the results of bad alignments.

Table 3.1: Alignment Accuracy

Case	Angle Err. Avg.	Angle Err. Std.
Cuboid 1x1	15.4°	4.6°
Cuboid 1x2	12.4°	4.2°
Cuboid 1x4	10.5°	6.3°
Cylinder	8.2°	2.1°

3.1.7 Grasping Accuracy

The results of grasp attempts for each object are shown in Table **??**. Although some of the final alignments from the viewpoint optimizer were off by up to 20 degrees, they can still provide the gripper with an orientation that results in a successful grasp. The 1x2 cuboid had the most successful grasps, with a 95% average success rate, even though the initial alignment was actually worse than the 1x4 cuboid on average. Next, the 1x4 cuboid had a 90% average success rate. For all cases, placing the object on another object and thus making it not flush with the platform did not result in significantly fewer successful grasp attempts. This shows that this method is invariant to the orientation of the object in the workplace, making it more useful in real-world scenarios in which objects are always sitting upright on a flat surface.

Most of the failure cases in all experiments were not due to proper alignment of the end effector at the end of the viewpoint optimization, but instead resulted from noisy values from the IR sensor, either causing the grippers to close too early, or not to close at all because the minimum threshold was never reached. One of the failure cases examined is when the 1x2 cuboid is placed on top of another object, part of the table is blocking the view of one side of the cuboid, which can perturb the optimization function, resulting in final grasps that align with the blocks major plane, but are not well aligned with its minor plane.

Case	Grasp Percentage
Cuboid 1x1 Orientation 1	80%
Cuboid 1x1 Orientation 2	80%
Cuboid 1x1 Overall	80%
Cuboid 1x2 Orientation 1	100%
Cuboid 1x2 Orientation 2	90%
Cuboid 1x2 Overall	95%
Cuboid 1x4 Orientation 1	90%
Cuboid 1x4 Orientation 2	90%
Cuboid 1x4 Overall	90%
Cylinder Orientation 1	90%
Cylinder Orientation 2	100%
Cylinder Overall	95%
Alligator Orientation 1	90%
Alligator Orientation 2	80%
Alligator Overall	85%

Table 3.2: Grasp Accuracy

3.2 CNN Viewpoint Planner

Experimental Setup After collecting training data per the procedure in 2.4.4, the network is evaluated according to a few different metrics: (1) the training vs evaluation dataset accuracy, (2) the final reward of a randomly initialized point on the virtual sphere, and (3) the number of time steps to reach the final reward. Each training session is run over 10 trials to ensure repeatability and all subsequent plots show the mean and covariance of the metric over the trials.

Two datasets were used for training and evaluation, each with 4096 images. The first dataset, ”Random” has 4096 random points on the virtual sphere and the second dataset, ”Uniform” has evenly spaced points on the virtual sphere (both limited to be at most 30°off-axis from the object's normal as shown in Figure 2.5). The ”Uniform” dataset takes roughly 20 minutes to collect in simulation time, which would translate to roughly 140 minutes in real-time, while the ”Random” dataset takes about 30 mins to collect in simulation time which would translate to roughly 210 minutes in real-time (using the hardware mentioned in 1.2 and Gazebo Simulator). The reason for the large difference is that the IK and Path Planner are able to quickly solve and move to nearby points, as the ”Uniform” dataset poses are calculated by incrementally increasing the ϕ and θ angle of the camera (rotating the camera about camera axis at each pose to also cover ρ), whereas the ”Random” dataset poses could end up far away from one another. The collection speed of the ”Random” dataset could be improved by first generating the set of random poses and then sorting them by ϕ θ angles.

Over 20 epochs, with each epoch covering 2867 images (70 % of the whole dataset) in the dataset, the total training time was 6 mins. For most objects, training could be early stopped after 5 epochs because training and validation accuracy stopped increasing past that point.

3.2.1 Training Results

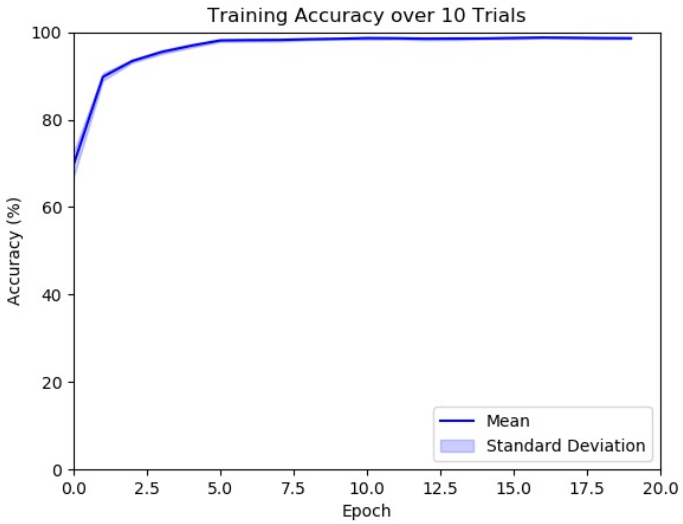


Figure 3.3: Training Plot for 1x2 Cuboid ”Random” Dataset over 10 Trials

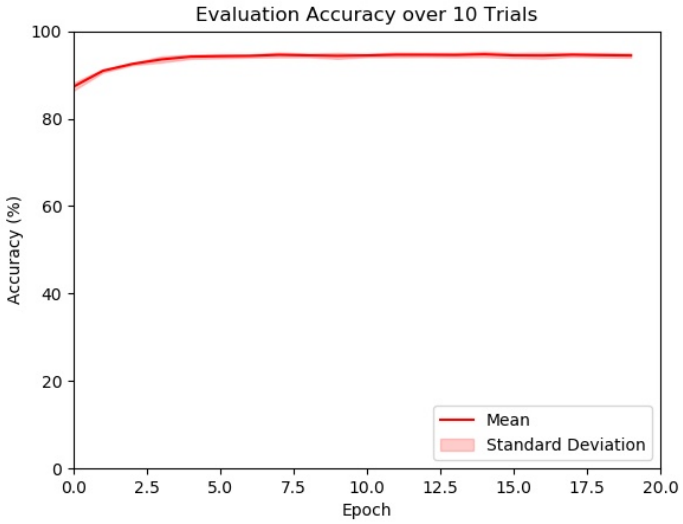


Figure 3.4: Evaluation Plot for 1x2 Cuboid ”Random” Dataset over 10 Trials

3.2.2 Evaluation on Viewpoint Planning in Simulation

Table 3.3: 1x2 Cuboid "Random" Evaluation Results in Simulation

Object	Mean Final Error	Std. Dev Final Error
Cuboid 1x2	4.5 °	2.7 °
Tomato Soup Box	8.6 °	4.8 °

3.2.3 Testing Model on Other Objects

To determine how well a trained model can generalize to a new object that is similar to the trained one, a model trained on the 1x2 cuboid was tested on 1x1 and 1x3 cuboids, as well as a simple cylinder. The results can be seen in Table 3.4. Again, the mean and standard deviation of the final rewards over 50 trials are collected for each object. The results show that the model can in fact generalize to the 1x1 and 1x3 cubioids, but doesn't generalize to the cylinder, which is as expected, because they don't have similar features.

Table 3.4: Average Final Reward and Reward Change in Simulation

Object	Mean Final Error .	Std. Dev. Final Error
Cuboid 1x1	4.3 °	2.89 °
Cuboid 1x3	6.8 °	4.3 °
Cylinder	61.5 °	12.3 °

Chapter 4

Conclusion

A simple eye-in-hand control framework for grasping novel objects in an uncalibrated workspace that only relies on RGB images was presented. We have shown a model-based approach than can work for some simple object shapes such as cuboids and cylinders as well as a non-uniform object. This method of finding the optimal end effector pose allows not only to pick up novel objects, but also allows them to be grasped in an orientation that allows for easy placement thereafter. Next, a model-free approach was presented, relying on recent developments in deep learning to train a neural network to move an end effector to an ideal grasp pose. This method has been shown to work in simulation and is ready for testing on a real robotic platform.

From the results of the image moment viewpoint planner, it is obvious that using image moments in a segmented visual servoing system can result in an grasp orientation that results in successful grasps of new objects and it doesn't require a high-precision robotic manipulator. The main limitation of this work is that is has only shown to perform well with simple objects and it takes a long time to converge to an optimal solution. Another limitation of both algorithms is that by attempting to maintain end effector poses on the virtual sphere centered on the object, the method is largely dependent on the workspace constraints of the robotic manipulator used.

Both algorithms could be made more robust by the incorporation of a holonomic base to

accommodate the workspace limits of the robotic platform. This would add degrees of freedom that would allow the spherical path to be followed more completely, allowing for successful grasps of objects in any position and orientation relative to the robot. For the CNN Viewpoint Planner, more testing on a real robotic platform is necessary, and recent data augmentation methods can allow a more robust model to be squeezed out of the same training dataset.

Appendix A

Equations

A.1 Depth Estimation

We assume a time step of t = 1, so the velocity, v of the end effector is simply Δd in the y-direction and assume there is no angular velocity component during translational motion ($\omega = 0$). This yields the following equation for the estimated depth of the object after a traversal of Δd along the camera's $y-$axis, with camera parameters f, ρ_u and ρ_v. $\bar{u}$ and $\bar{v}$ represent the distance traveled by the tracked pixel in the images x and y axes respectively.

A.2 Homography for Alignment Error

$$(J_t v)\frac{1}{Z} = \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}, \text{ where } J_t = \begin{bmatrix} -\frac{f}{\rho_u} & 0 & \bar{u} \\ 0 & -\frac{f}{\rho_v} & \bar{v} \end{bmatrix} \tag{A.1}$$

Finally, we find the least squares solution for Z in Equation A.1 yielding the estimated depth from the camera to the tracked point.

A.3 Pose on Virtual Sphere

To find the pose of an end effector mounted-camera on the surface of the sphere paramaterized by center c and radius r, and given table normal $\mathbf{n}$, we can represent the position as the x, y, and z coordinates in space. We can constrain the orientation of the camera such that he iss x-axis is parallel to the table plane (n_x). This forms the following equations:

The x-axis of this coordinate frame satisfies:

$ccam_x \cdot n = 0$

Because the camera's x-axis is constrained to be parallel to the table plane. The camera's y-axis is a free variable, while the camera's z-axis follows: $cam_z \cdot n <= 0$

meaning the camera's z-axis should always be facing the opposite direction of the table normal (ensuring the camera is focused on the object on the table, rather than away from it.

This method is used to generate random poses along the virtual sphere for generating training data for the supervised learning spherical viewpoint optimizer by varying the input point x,y,z with a normal distribution about the objects normal axis. This is done to get more data points closer to the alignment we want with the part.

A.4 Cross-entropy Loss

For a sample input I, we calculate cross-entropy loss for a classification problem with M classes $[c_1, c_2, ...c_M]$, using:

$$-\sum_{c=1}^{M} t_i log(s_i) \tag{A.2}$$

where t_i is the true class label is of the sample input (t_c is 1 if the true class label is c and all other t_i are 0) and s_i is the output score (from the model) of the sample.

A.5 Prediction Accuracy

For a classification problem with $[I_1, I_2, ..., I_n]$ samples, each from one of M potential classes, with true class labels c_i, a prediction is correct if the predicted class label is the same as the true class label, and $C(I_i, c_i, p(I_i))$ is 1, otherwise it is 0:

$$C(I_i, c_i, p) = \begin{cases} 1 & c_i = p(I_i) \\ 0 & c_i \neq p(I_i) \end{cases} \tag{A.3}$$

The overall accuracy of a model's predictions can be calculated using:

$$100\% \times \frac{\sum_{i=1}^{n} C(I_i, c_i, p(I_i))}{n} \tag{A.4}$$

Bibliography

[1] Collaborative robots from ur: Start your automation journey today. URL https://www.universal-robots.com/products/.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.

[3] G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.

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

[5] F. Chaumette. A first step toward visual servoing using image moments. In IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 1, pages 378–383. IEEE, 2002.

[6] C. Choi and H. I. Christensen. Real-time 3d model-based tracking using edge and keypoint features for robotic manipulation. In 2010 IEEE International Conference on Robotics and Automation, pages 4048–4055. IEEE, 2010.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[8] C. Fitzgerald. Developing baxter. In 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pages 1–6. IEEE, 2013.

[9] M. Gualtieri, A. ten Pas, and R. P. Jr. Category level pick and place using deep reinforcement learning. CoRR, abs/1707.05615, 2017. URL http://arxiv.org/abs/1707.05615.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[11] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik. Intel realsense stereoscopic depth cameras. CoRR, abs/1705.05548, 2017. URL http://arxiv.org/abs/1705.05548.

[12] U. Klank, D. Pangercic, R. B. Rusu, and M. Beetz. Real-time cad model matching for mobile manipulation and grasping. In 2009 9th IEEE-RAS International Conference on Humanoid Robots, pages 290–296. IEEE, 2009.

[13] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), volume 3, pages 2149–2154. IEEE.

[14] S. Levine, P. P. Sampedro, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. 2017. URL `https://drive.google.com/open?id=0B0mFoBMu8f8BaHYzOXZMdzVOalU`.

[15] C. B. Madsen and H. I. Christensen. A viewpoint planning strategy for determining true angles on polyhedral objects by camera alignment. IEEE Trans. PAMI, 19(2):158–163, Feb. 1997.

[16] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. 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.

[17] E. Malis, F. Chaumette, and S. Boudet. 2 1/2 d visual servoing. IEEE Transactions on Robotics and Automation, 15(2):238–250, 1999.

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

[19] M. Mudigonda, P. Agrawal, M. Deweese, and J. Malik. Investigating deep reinforcement learning for grasping objects with an anthropomorphic hand. 2018.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[21] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics, Kobe, Japan, May 2009.

[22] C.-L. Shih and Y. Lee. A simple robotic eye-in-hand camera positioning and alignment control method based on parallelogram features. Robotics, 7(2):31, 2018.

[23] O. Tahri and F. Chaumette. Point-based and region-based image moments for visual servoing of planar objects. IEEE Transactions on Robotics, 21(6):1116–1127, 2005.

[24] G. Van Rossum and F. L. Drake Jr. Python tutorial. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[25] A. Zakharov and A. Barinov. An algorithm for 3d-object reconstruction from video using stereo correspondences. Pattern recognition and image analysis, 25(1):117–121, 2015.