Data-Driven Statistical Models of Robotic Manipulation

Robert Paolini

CMU-RI-TR-18-25

May 2018

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Matthew T. Mason, Chair
Nancy S. Pollard
Geoffrey J. Gordon
Paul G. Backes, Jet Propulsion Lab

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Keywords: Robotic Manipulation; Data-Driven Statistical Models; Data Collection for Robotic Manipulation; Post-Grasp Manipulation; Regrasping

For my family

Abstract

Improving robotic manipulation is critical for robots to be actively useful in real-world factories and homes. While some success has been shown in simulation and controlled environments, robots are slow, clumsy, and not general or robust enough when interacting with their environment. By contrast, humans effortlessly manipulate objects. One possible reason for this discrepancy is that, starting from birth, humans have years of experience to collect data and develop good internal models of what happens when they manipulate objects. If robots could also learn models from a large amount of real data, perhaps they, too, could become more capable manipulators. In this thesis, we propose to improve robotic manipulation by solving two problems. First, we look at how robots can collect a large amount of manipulation data without human intervention. Second, we study how to build statistical models of robotic manipulation from the collected data. These data-driven models can then be used for planning more robust manipulation actions.

To solve the first problem of enabling large data collection, we perform several different robotic manipulation experiments and use these as case studies. We study bin-picking, post-grasp manipulation, pushing, tray tilting, planar grasping, and regrasping. These case studies allow us to gain insights on how robots can collect a large amount of accurate data with minimal human intervention.

To solve the second problem of statistically modeling manipulation actions, we propose models for different parts of various manipulation actions. First, we look at how to model post-grasp manipulation actions by modeling the probability distribution of where an object ends up in a robot's hand, and how this affects its success rate at various tasks such as placing or insertion. Second, we model how robots can change the pose of an object in their hand with regrasp actions. Third, we improve on the place and pick regrasp action by modeling each separately with more data. These learned data-driven models can then be used for planning more robust and accurate manipulation actions.

Acknowledgments

First of all, I'd like to thank Matt Mason for all of the fun and helpful discussions over the years. I learned to think more carefully about how things move, and am now doomed to pay attention to interesting manipulation phenomena for the rest of my life. Thanks for giving me the freedom and support to explore what I was interested in. Matt is a great advisor and an even better person, and for anyone reading this, I highly recommend working with him. Special thanks to Nancy Pollard, Geoff Gordon, and Paul Backes for all of the comments, questions, and discussions which improved this thesis.

I met so many wonderful people at the Robotics Institute who helped me along the way. Thanks to everyone I've interacted and collaborated with over the years in the MLab including Annie Holladay, Jiaji Zhou, Ankit Bhatia, Zhenzhong Jia, Zhiwei Zhang, Aaron Johnson, Reuben Aronson, Yifan Hou, Pragna Mannam, Joseph Gu, and Ariana Keeling. I enjoyed the collaborations, the lunches, and hard work. Thanks to Jean Harpley for working tirelessly to save the lab from bankruptcy, and for always having a solution to any and all of my administrative problems. Starting out a PhD can be tricky, and I want to thank Sidd Srinivasa for his invaluable guidance and collaboration early on. It was an honor and a privilege to work alongside Alberto Rodriguez in the MLab. Alberto was my first example of how a graduate student should act, and turned out to be a pretty good example to follow. I enjoyed all of our collaborations, including when I accidentally fused the robot's hand joints with Loctite the day before a deadline, as well as all of the fun conversations unrelated to work. Thanks to Breelyn Kane, Humphrey Hu, and other "old guard" RI graduate students who helped me realize there was light at the end of the tunnel.

There is life outside of grad school, so I'd like to acknowledge my friends outside of the RI as well. Thanks to Akshay Krishnamurthy for attempting to teach me theoretical machine learning including deriving convergence rates for kernel density estimation, for being an example of an ideal graduate student for me to strive to be, and for being a great friend since high school.Thanks to Hassan Khan for being my partner in crime at home and helping me feel ok about enjoying things aside from working towards my PhD every day. Watching sports, planning our survivor strategy, and playing Super Smash Bros. with you are some of my favorite memories from my time in graduate school. Thanks to the entire Ultimate Frisbee community in Pittsburgh for the years of fun exercise and friendship. Shout out to all of my teammates on CMU Mr. Yuk, Muff 'N Men, and Alloy. It was/is great playing with all of you. Thanks to my friends from high school and college including Annie Hong, Long Nguyen, Eythan Familier, Tom Gwinn, Derek Macklin, Ryan Tang, and Sue Pearring for your support and fun times over the years that made graduate school that much better. Thanks to Rachel Bowers for your encouragement as I stressfully navigated my PhD and making sure I got enough food and sleep along the way. From watching Star Wars at 6am, riding as many roller coasters as possible at Cedar Point, or taking walks through Frick park, youve been at my side and I wouldnt trade it for anything. Your constant support throughout this process is something I'm not sure

I'll be able to repay.

I am lucky to have so many family members close to Pittsburgh. Thanks to my grandmother Nancy for her wonderful home cooked meals, my grandfather Robert for his expert advice on hand design among other things, and my entire extended family near Pittsburgh who gave me a home away from home, and much needed breaks from graduate school. Thanks to my sister Cathryn for dealing with me for all of these years, and for teaching me how to be a kinder, more helpful person. Thanks to my dad Steve for all of the fun conversations and advice, unsolicited or not, and for continuing to be the person I look up to the most in this world. Finally, thanks to my mother Julie for somehow surviving the challenge of raising me to adulthood, and for her unconditional love and support. I would not be where I am today without you.

Contents

List of Figures

Chapter 1

Introduction

Humans are experts at manipulation. They can grasp and use unknown objects in their environment with ease and precision. One reason for this is that they have years of experience manipulating different objects and collecting data on how they can affect the world with their actions. With this experience, before manipulating an object, a human already has an expectation for what is going to happen. This expectation is normally correct, leading to smooth and efficient motions. By contrast, a robot's expectation of how the world will change when it interacts with it is often incorrect, leading to unreliable and failed manipulation. This is because its models of manipulation are often quite wrong. In this thesis, we improve robotic manipulation performance by building accurate manipulation models.

Models of robotic manipulation are often inaccurate because the real world is a complex place. Simple physics models can only take us so far, and even high-fidelity simulations have a difficult time modeling multiple contacts, non-Coulomb friction, stiction, deformation, and changing contact modes. Moreover, even if simulators had the capability to model the world perfectly, they might still produce incorrect models because of not using the correct parameters to match what's actually happening in the real world. The amount of dust settling on one side of the object, the amount of oil on a surface, a slight difference in weight distribution of an object, and inconsistent friction inside of a robot gripper are examples of things that would be difficult to match within a simulator. Indeed, a deterministic model of any sort seems impossible to use for modeling robotic manipulation.

Instead, in this thesis, we choose to model robotic manipulation using data-driven statistical models. By recording data from the real-world, we can measure and encapsulate any uncertainty. If dust is settling on the side of an object and causing it to slip sometimes, or if sensing inaccuracy is causing the object to tilt in different directions when grasped, we would like the robot to have a probabilistic model to account for this. By using a large amount of real robotic manipulation data, we can improve the accuracy of our models, because they are based on what the robot is actually seeing with its own sensors.

To enable data-driven statistical models we must solve two problems, both of which will be the main contributions of this thesis. First, how can we get robots to collect enough manipulation data to build useful models? Second, how can we build statistical models from this data that accurately represent the world?

Collecting a large amount of robotic manipulation data is challenging, and is often avoided by researchers. However, if we hope to achieve accurate models of manipulation actions, the robot needs to be able to collect data on its own. In this thesis, we present several different robotic manipulation experiments where data was collected, and attempt to find insights into what made the data collection successful and unsuccessful.

Once the robot has the data in hand, it can now learn statistical models with this data. Unfortunately this is also challenging, as manipulation is the iconic non-Gaussian problem. Flipping a coin, grasping an object with multiple crevices, or rotating an object through the robots hand all have multiple modalities, each with different distributions. We present solutions for statistically modeling different aspects of manipulation actions. First, we show how to predict the probability distribution of the pose of a grasped object from sensor readings. Next, we show how to predict the probability of successfully executing a post-grasp action such as placing or insertion given a certain amount of error in the predicted in-hand pose. By combining these probability functions together, we can estimate the probability of succeeding at a task given sensor readings. In cases where this probability is too low, it may be advantageous to adjust the pose of an object in the robot's hand. Thus, we also show how to model these regrasp actions by predicting the probability that the robot will maintain its grasp after an action, and also predicting the probability distribution of the resulting in-hand object pose. We also break this regrasp down into placing and picking actions to enable the robot to position the object not just in its hand, but in the world as well. All of our models are non-parametric, and improve their accuracy with more data. Armed with data-driven statistical models, robots should be able to plan and execute manipulation actions more effectively.

The rest of this thesis is organized as follows. In Chapter 2 we present the related work to this thesis. In Chapter 3 we present several robotic manipulation experiments where a large amount of data was collected. In Chapter 4 we detail the machine learning used to statistically model manipulation actions. In Chapter 5 we briefly show how our models can be used for planning. In Chapter 6 we conclude and discuss future work.

Here we summarize the contributions of this thesis:

1. **Data Collection of Robotic Manipulation** - We look at several case studies on how to collect a large amount of robot manipulation data. We then use these case studies to create a list of insights on how robots can collect manipulation data with minimal human intervention.
2. **Data-Driven Statistical Models for Robotic Manipulation** - We present methods for modeling manipulation statistically using collected data sets. We present the following statistical models:
    - Sensing capabilities of a robot hand
    - Task requirements of post-grasp manipulation actions
    - Combining the above models to predict the probability of succeeding at a task given sensor readings
    - Probability of maintaining a grasp of an object after a regrasp, place, or pick action
    - Predicting the probability distribution of where we expect the object to end up after a regrasp, place, or pick action

Finally, we validate our data-driven models by having the robot create and execute end to end plans for a series of actions to position an object in a robot's hand, or in the environment.

Chapter 2

Related Work

Our proposed work looks at how to statistically model robotic manipulation actions from collected data. First, we discuss how to model uncertainty in manipulation actions, and then we cover selected other topics relevant to this thesis including post-grasp manipulation, regrasping, and pushing. Finally we look at examples of researchers collecting data from robotic manipulation experiments.

2.1 Modeling Uncertainty in Manipulation Actions

Much of this section is published in our previous work [90, 91]. The importance of uncertainty in manipulation has long been recognized. Incorporating stochastic models into modeling, perception and control was attempted even in the 1970's, for example using Kalman filters in industrial assembly [112, 118]. For more recent work using either Kalman filters or particle filters see [42].

Numerous early experiments illustrated the necessity of modeling uncertainty. Most notable was Inoue's peg insertion work [58] which inspired the *pre-image backchaining* approach [33, 70]. Preimage-backchaining adopted a *possibilistic* approach, representing the robot's belief state by a set of possible configurations. Later work extended the approach to probabilistic models [66].

The 1980's and early 1990's saw several projects exploring grasping and manipulation under uncertainty, using both possibilistic and probabilistic models. [14, 15, 16, 21, 22, 23, 34, 43, 44, 80, 94, 117] Among these, the closest to the present work are probably [16, 23, 44], which develop Bayesian decision-theoretic techniques, applied to planar grasping problems and the problem of an object sliding in a tilting tray. Dogar and Srinivasa [30] applied similar ideas to clutter and uncertainty in the context of push-grasping. Kang and Goldberg [60] used a random sequence of parallel-jaw grasps to classify grasped objects using a Bayesian process.

Goldfeder and Allen [45] approached the problem of grasp planning from a data-driven perspective. Given partial shape information of a novel object, they use a grasp data base to choose a grasp for the novel object.

There is a substantial literature on statistical frameworks to model uncertainty. POMDPs [17] (Partially Observable Markov Decision Processes) are a general framework that describes the current problem well. Hsiao et al. [57] used a POMDP framework to track the belief of the pose of an object and tactile exploration to localize it by planning among grasping and information-gathering trajectories. PSRs [10, 119] (Predictive State Representation) are also introduced as a general framework to learn compact models directly from sequences of action-observation pairs without the need for a hand-selected state representation. Lavalle and Hutchinson [66] advocate information-spaces to formalize the process of propagating uncertainty along motion strategies. [48] explored the application of optimal control policies in information space, derived from changes in observable modes of interaction. Platt has worked on Markov Decision Process planning, with actions expressed relative to contact locations, and on compliant hand motion [98]. Petrovskaya and others have worked on belief state estimation for uncertain manipulation task geometry [95].

Mahler et al. [73] used Gaussian Process implicit surfaces to model grasping uncertainty. Schaal and Atkeson [108] used locally weighted regression and a shifting setpoint exploration algorithm to learn a dynamic model for Devil Sticking. Christopoulous et al. [25] explicitly model object pose and shape uncertainty from vision to select more robust parallel grasps. They specify shape uncertainty using affine transforms of spline models. Mahler et al. learn grasping models in simulation with DexNet 1.0 [74] and 2.0 [75]. In DexNet 1.0, they estimate the probability of force closure by assessing similarity to previous objects in a database for which grasp qualities have been computed. In DexNet 2.0, they synthetically generate point clouds, grasps, and compute grasp metrics, and use this dataset to train a convolutional neural network to predict grasp quality for real grasps using an ABB Yumi. Stulp et al. [115] learned motion primitives to optimize the chance of grasping an object with Gaussian uncertainty on its location. Cayla [18] suspended a sphere from a string to determine grasping probabilities in a grid of positions with respect to a hand.

Some work has been done on analyzing the grasp outcome as well. Morales et al. [82] used real grasps on a collection of objects to predict the reliability of the grasp process. Balasubramanian et al. [5] noted that different tasks lead humans to different initial grasps, and Faria et al. [35] were able to estimate the best part of an object to grasp based on the task using human trials. To model manipulation actions, researchers often use simulation[45][61], imitation learning[115], or models learned with collected robot data [82][90, 91]

The most similar work to ours comes from Kopicki et. al. [65]. They use regression to learn the resulting motions of real robotic push actions. They also fit multi-modal probability distributions to their data and show improvement over regression. Our work focuses on learning both the probability of maintaining a grasp after a regrasp and the resulting probability distributions of robotic regrasp actions, along with paying closer attention on how to collect a large amount of robot manipulation data.

2.2 Post-Grasp Manipulation

In the context of post-grasp manipulation, Jiang et al. [59] looked at scenes to determine good locations to place objects. However, they did not study how robust the final process of actually placing an object is, which is the subject of our work. Fu et al. [40] addressed the problem of batting an object to a goal in the presence of uncertainty. They first maximized information gain in an observation step, and then chose the action most likely to succeed. Holladay et al. [53] used inverse motion planning to determine the optimal placement of a robot's other hand to increase the probability of successfully placing objects.

2.3 Regrasping

Regrasping has been studied for a long time, starting with Paul [93], Tournassoud et al. [116], Fearing [37], and Brock [13]. Early regrasping work assumed a known world model with deterministic actions. Most regrasping work falls under three categories: pick and place [51, 113, 116], closed-loop dynamic regrasping [26][109][41], or what is generally referred to as dexterous manipulation or finger gaiting [37, 107][55][110][36][20][106]. Chavan Dafle et al. [19] present work on "extrinsic dexterity", which uses gravity, inertia, and external contacts to vary the pose of the object within the hand.

2.4 Pushing

Mason [79, 80] first developed the theory of quasi-static pushing. Later, Lynch and Mason [72] further developed the theory to include controllability and planning. Akella and Mason [3] used this theory to orient polygonal objects in a plane. The limit surface, a compact way to represent planar sliding was introduced by Goyal et al. [46] . Howe and Cutkosky [56] introduce an ellipsoid approximation to this limit surface. The Goyal's limit surface was then used by Zhou et al. [125] as a backbone for learning with data to fit an accurate sliding model with a minimal amount of data. Kopicki [65] tried to predict pushing motions by collecting data on the pose of relevant elements. Omcren [88] and [38] present other data-driven approaches to modeling pushing, generally using visual data with random push trials and standard machine learning approaches.

2.5 Data Collection for Robotic Manipulation

Some examples of data collection for robotic manipulation are presented in [8, 9, 12, 32]. Detry has shown success with data-driven affordance learning [27, 28, 29]. Pastor et al. [92] adapts the movement of a robot to succeed more often at a task through a series of trials. Stulp et al.[115] adapt the movement of a robot to handle additional uncertainty through learning. Yu et al. [123] collect a large pushing data set. Google has recently collected large RGB-D data sets of grasping and pushing [67][38]. Brost [16] collected thousands of trials of grasping a gear automatically in order to learn a success model for grasping. Agrawal et al [2] poked objects in a bin with sloped walls over 100K times for over 400 hours. Pinto and Gupta [96] grasped objects on a table over 50K times for over 700 hours, and used this to predict the best SE(2) grasp location for each object.

Chapter 3

Collecting Robotic Manipulation Data

3.1 Introduction

Humans are experts at manipulation in part because of their wealth of manipulation experience. Babies play with toys, first struggling to even pick up objects, but after hours and hours of trial and error, become more and more capable, graduating to building block towers, and perhaps even frisbee players. The intricacy of the real world necessitates a large number of trials for humans to become capable manipulators. In order for robots to manipulate objects successfully, they too must perform a large number of trials.

While it is tempting to use a simulation to give the robot the experience it needs to be successful, this has not yet been shown to be successful. There are many phenomena in the real world that are difficult to model in simulation. Non-Coulomb friction, deformation, stiction, dirt, moisture, inaccuracies in object shape or density, and sensing inaccuracies are some examples of difficult phenomena that cause simulators and the real world to diverge. Even if robots work well at manipulating objects in a simulator, this generally does not translate to success in the real world. Some researchers have tried to take results from simulation and use transfer learning to reduce the number of trials needed for robots in the real world. However, this has only been shown to work in very constrained or contrived cases, and often does not significantly reduce the number of experiments needed.

Thus, we are left with the task of trying to collect a large amount of real manipulation data with robots. Unfortunately, collecting real robot manipulation data is full of challenges, which is why many researchers expend a lot of effort avoiding it. When interacting with the real world, the robot can crash, it can drop objects, its hardware can break, and it can operate with an incorrect estimate of the current state of the world leading to an infinite loop or fatal error, to name a few.

For the rest of this chapter, we will look at different case studies of manipulation data collection. First, we look at picking highlighter markers out of a bin and trying to place, drop or insert them. Second, we look at pushing an object around on a table. Third, we look at modeling planar grasping. Next, we look at tilting an object in a tray to try to localize it. Then, we look at placing and picking a cube to regrasp it, and also look at an improved data collection setup of that. Finally, we conclude with some common themes for successfully conducting manipulation experiments.

3.2 Case Study 1: Bin Picking and Post-Grasp Manipulation with Highlighter Markers

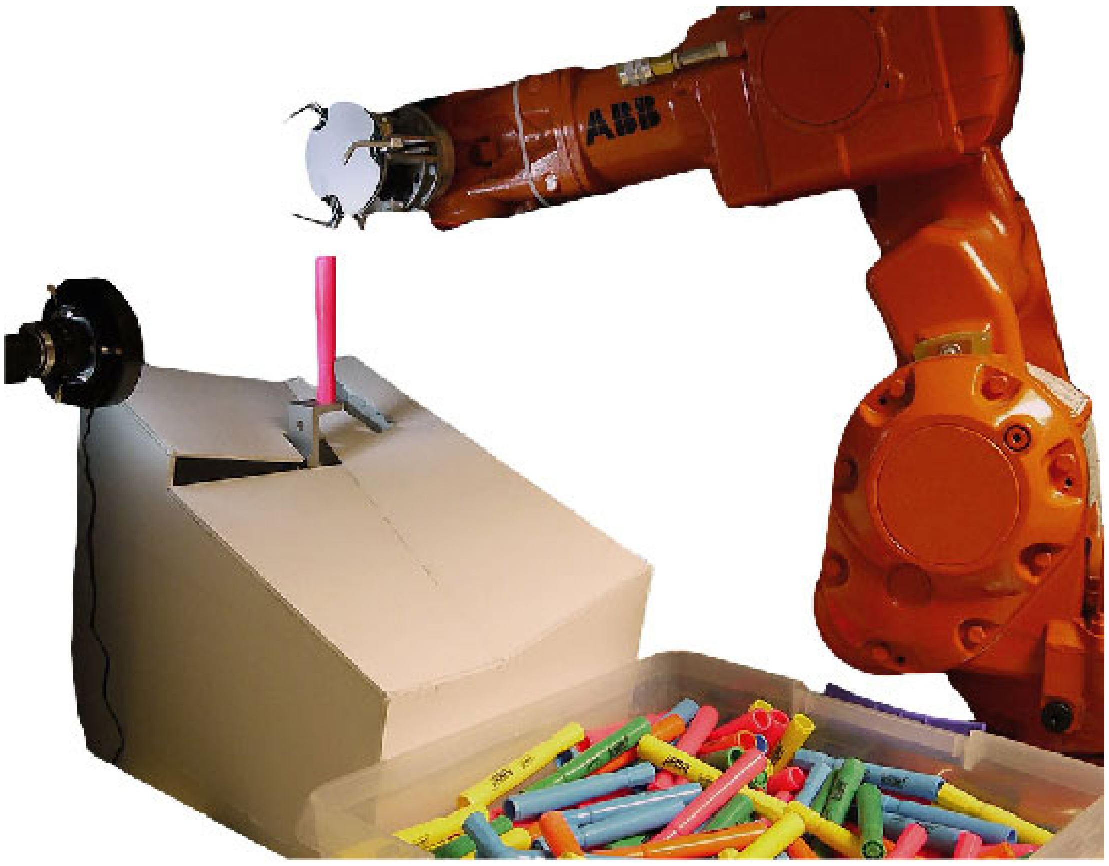


Figure 3.1: Data collection setup for bin-picking highlighter markers and placing them on a platform. The robot would reach into the bin and squeeze while twisting to grasp a marker. Then, it would attempt to balance the marker vertically on a small platform. If unsuccessful, the marker would fall and roll down the ramp back into the bin. If successful, the robot would purposely knock the marker over so it would also roll back into the bin.

In this experiment, the object was to grasp a highlighter marker from a bin and place it on a platform, drop it into a hole, or insert it into a smaller hole. Figure 3.1 shows our experimental setup for bin-picking and placing.

We used an ABB IRB 140 industrial robot arm with an MLab Simple Hand. The MLab simple hand has a single actuator which moves 3 compliantly linked fingers to close around the object. The rotation of each finger is captured by a 12-bit absolute encoder, giving us a sensing accuracy of 0.1 degrees for each finger.

Designing a robust system to collect all the necessary data was both important and challenging. The amount of data required to learn probability functions forced us to carefully design experimental setups requiring minimal human intervention. For the three post-grasp manipulation tasks we had to focus on three different aspects: object acquisition, task execution, and post-task reset. Having a human in the loop to hand objects to the robot was not an option, since we needed to execute thousands of experiments and it would also possibly introduce bias in the process.

We solved the object acquisition problem by having a large bin of objects and training an open-loop grasping strategy that singulates a marker out of the bin approximately 40% of the time. For task execution, it was important to make sure that the hand did not collide with the environment, regardless of the pose of the marker or action chosen by the robot.

Finally, for resetting the system after task execution, different strategies were used depending on the task. For placing, the object was placed at the top of a ramp and knocked back into the bin. For dropping, after some constraining moves, the object was grasped out of the hole and dropped back into the bin. For insertion, the object was held the entire time and then dropped back in. It is important to note that a fair amount of time was spent in designing robust experiments, and this should not be overlooked when trying to collect data in a real setting.

3.2.1 Vision System

We needed two separate vision systems for this experiment. One to find the whether the hand was holding a single highlighter marker, along with the in-hand pose of that highlighter marker, and one to determine whether the post-grasp manipulation task was successful. We discuss both of these vision systems here.

3.2.1.1 In-Hand Pose Estimation

We use a single RGB camera, with the pipeline shown in Figure 3.2. The most important factors here were robustness and accuracy. We needed to be sure that the hand was holding exactly one marker, and if so, for our models to be accurate, we needed to know as accurately as possible where the marker was in the hand. By making a specialized vision system for these specific highlighter markers, we were able to accomplish both of our goals.

The vision system works as follows:

1. The hand is moved in front of the camera and a background image is saved. Now, when the hand has grasped in the bin and moves in front of the camera, the images can be subtracted to eliminate random colors present in other parts of the image.
2. There are five marker colors. Using color thresholds in HSV space on the background-subtracted image, we can get five binary images that show us where each marker color might be in the image.
3. By counting the number of pixels in each binary image, we can quickly determine whether the hand is holding 0, 1, or more than 1 marker. If none of the images have enough pixels, then the hand is holding no markers. If more than 1 binary image has enough pixels, or if 1 binary image has too many pixels, then the hand is holding more than 1 object. Otherwise, the hand is holding exactly one object.
4. Next, by fitting a 2D Gaussian to the relevant binary image, an initial guess for the position and orientation of the marker in the hand can be computed.

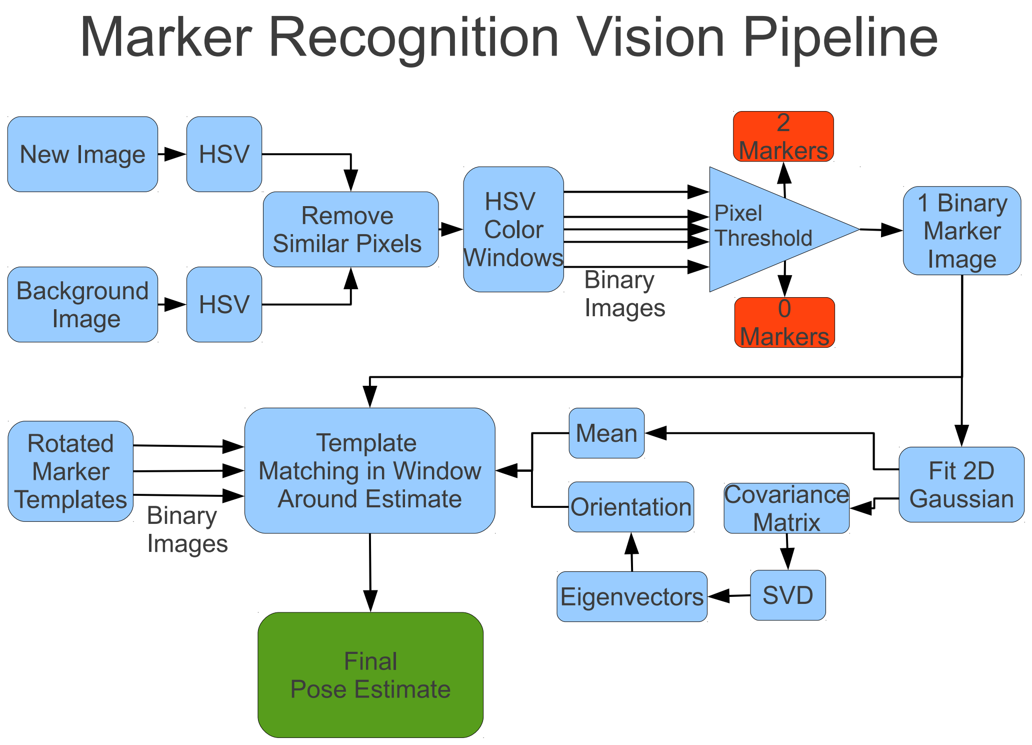


Figure 3.2: Vision pipeline for calculating the pose of a highlighter marker. First, a new image and previously taken background image are converted into the HSV color space and subtracted. Then, colors thresholds for each of the five marker colors were used to get binary images for each color. At this point, by counting pixels, one can very robustly decide if there is 0, 1, or more than 1 marker. If there is exactly 1 marker, then a 2D gaussian is fit to the data to get an initial guess for a position and orientation. Then template matching is used to find a final marker position that is robust to occlusion.

5. Because of occlusion by the hand's fingers, this estimate is generally off, and so by using a rectangular template for the highlighter marker and sliding it around the image, we can perfectly match the pose.

This system gave us sub-millimeter accuracy of marker pose. Also, because it was mostly done with binary images, the computation time was minimized. No vision system failures were observed in the 10,000 grasps performed by the robot.

3.2.1.2 Task Success Prediction

To determine whether or not the task succeeded, we used another RGB camera and a similar vision system to the one above. For a task to be successful, there will be a highlighter marker present either above the placing platform, above the dropping hole, or inside of the insertion hole. After doing background subtracting and color thresholding, if there is more than a certain number of pixels in the relevant regions, we declare task success. Again, no vision system failures were observed in the 10,000 grasps performed by the robot. This is again an example of a highly specialized but very robust and accurate system.

3.3 Case Study 2: Pushing an Object on a Table to Identify Properties

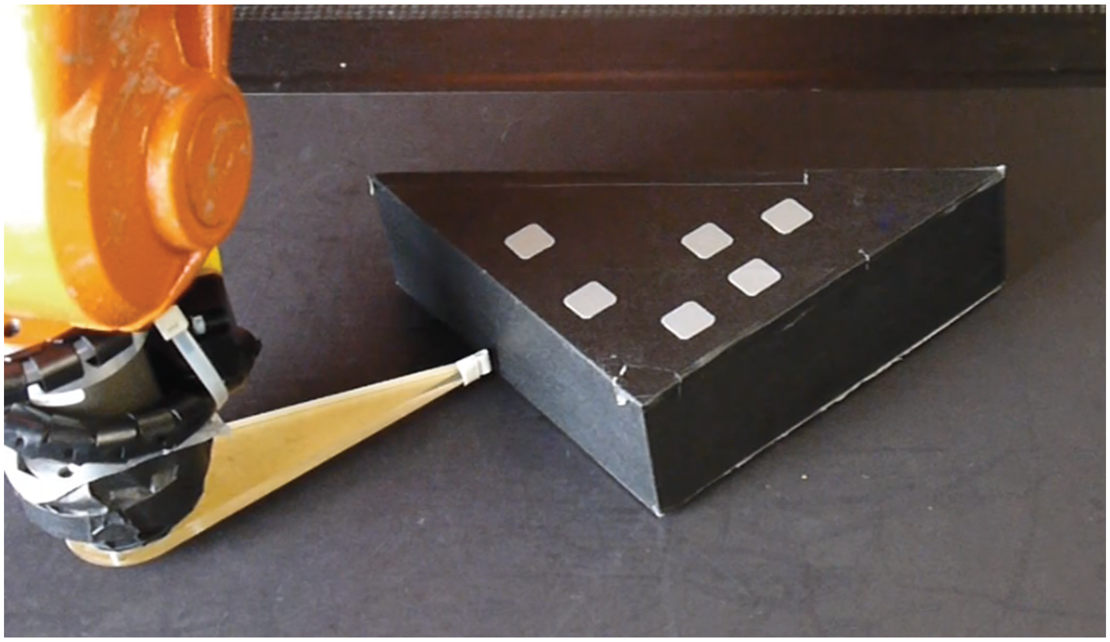


Figure 3.3: Data collection setup for determining planar sliding properties of an object. The robot was equipped with a pusher connected to a force-torque sensor, and the object was tracked using a motion capture system. The robot would pick a random contact point and random push direction, and then push the object and record data so a model could be fit. If the object got too close to the edge of the robot's workspace, the robot would drag the object back to the center.

The purpose of this next experiment was to identify planar sliding properties of an object. The results of this work are discussed in Zhou et al. [125], and our experimental setup is shown in Figure 3.3. Here, we were interested in predicting how an object would move when it was gently pushed. By recording the contact location and direction of a straight push, the resulting motion of the object, and the forces felt by the pusher while the object was moving, we could fit a model of how the object would move. The experimental setup would be varied by switching the contact pressure distribution between the object and table along with the frictional properties of the table.

While the subject of our work was to identify this model in as few pushes as possible, we still wished to have a system that could perform a large number of pushes for future experiments. First, to vary the pressure distribution, we drilled and tapped a number of holes on the underside of the object (Figure 3.4). By adjusting the position of the screws, we could experiment with different pressure distributions. The block was machined to sub-millimeter precision, so the pressure distribution relative to the edge of the object was known to high accuracy. Second, to vary the frictional properties of the table, different materials were simply fastened to the table.

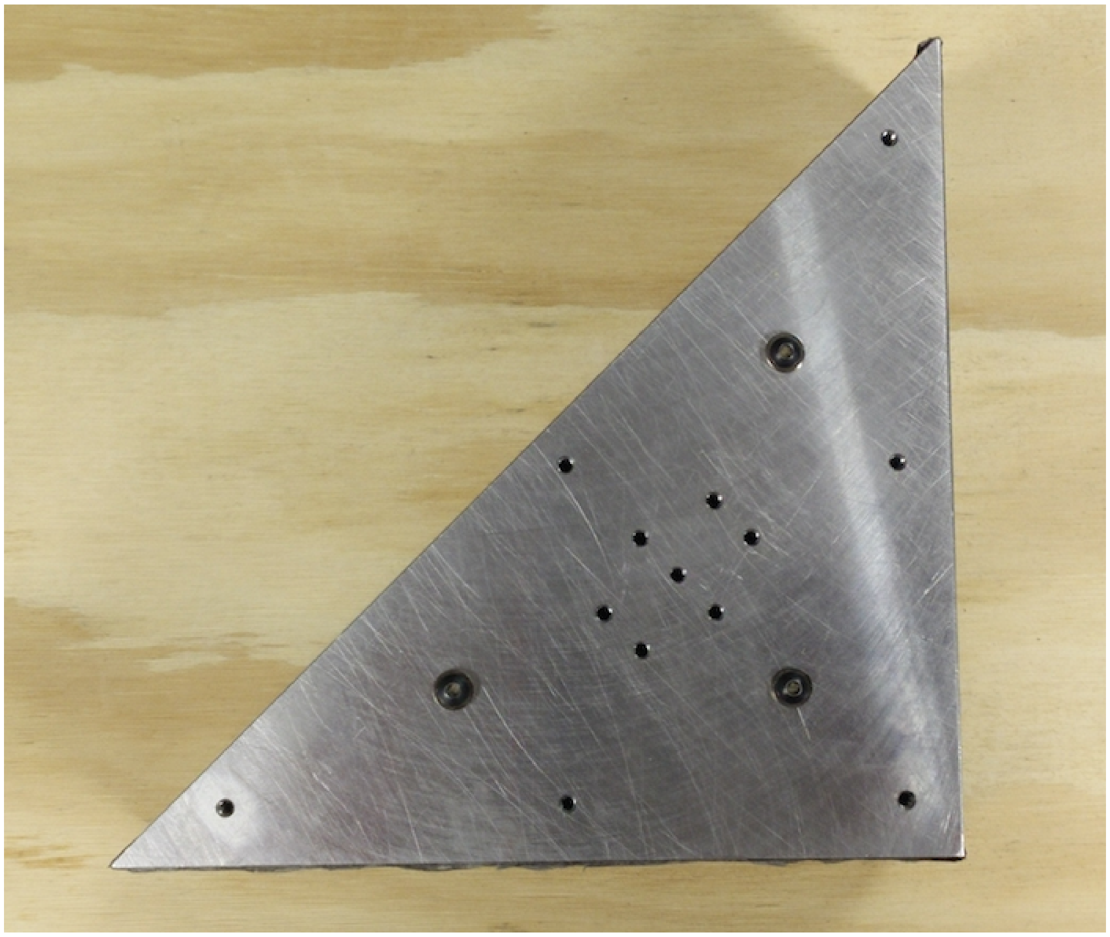


Figure 3.4: The support points of the object could be changed by adjusting the position of the screws.

To collect accurate data for our models, our sensing and actuation needed to be accurate as well. By using an ABB IRB 140 industrial robot, our actuation accuracy was 0.03mm in positional repeatability. We used an ATI Mini-40 force-torque sensor with an SI-20-1 calibration, meaning the measurable force resolution in the planar direction was 1/200 Newtons. Moreover, by increasing the mass of the block to 1.5 kg, we were able to proportionally increase the forces felt by the force-torque sensor, effectively increasing the resolution further. Finally to sense the position of the block, we used 6 OptiTrack Flex:V100 motion capture cameras. By placing a number of IR markers on the block, we could sense both the position and orientation of the block relative to the robot.

For our vision system to attain high accuracy, several careful calibration steps were needed. First, the cameras had to be calibrated with respect to each other, which was done with an off-the shelf OptiTrack procedure which involved waving a wand with known marker positions around the work space. Second, the cameras needed to be calibrated with respect to the robot, which was done by having the robot grasp an IR marker and move around in the workspace. The transform between the robot's base frame and camera frame, and the mounted position of the marker relative to the tool flange of the robot were simultaneously calculated, with the latter being discarded. Finally, the transform between the IR markers on the object and the actual object frame was computed by measuring the position of some markers by hand relative to some fixed frame, and then placing the object on that fixed frame. By looking at the position of the fixed IR markers relative to the object's IR markers, a transform could be computed. After all of these calibrations, our effective sensing accuracy was 6mm in practice. This error combines errors in each calibration step with the motion capture system error.

While sensing and actuation accuracy are critical for model accuracy, the amount of data collected is also critical. To improve this, we needed to solve two problems. First, the robot's hardware robustness had to be high enough to not break down during the experiment. The reliability of the industrial arm and force torque sensor was high, however special attention was paid to the cables connected to the force-torque sensor. The robot rotates its end effector around, and it was critical for the cables to not interfere with the object in any way, as this would corrupt our data. This was accomplished by wrapping the cable around the force-torque sensor, creating a path for the cable to slide along the robot arm, and attaching some tension to the end of the cable to pull any excess cable out of the way of the work area. This took time to get right, and had to be fixed several times after running the robot for some time revealed other flaws in the cable routing.

The second problem to solve was the object reset problem. The robot selected a random contact point and random push direction, so it was possible that the robot could push the object out of its workspace. To prevent this, if the object got close to the edge of the workspace, the robot would turn its pusher sideways and drag the object back towards the center. This was a simple way to increase the number of experiments performed without human intervention.

3.4 Case Study 3: Planar Grasping

In this experiment, we wanted to understand how an object moves as it is being grasped on a plane. Not only were we interested in whether the object would be successfully grasped, but we were interested in where the object would end up after the robot hand interacted with it. More motivation can be read in Zhou et al. [126].

The experimental setup is shown in Figure 3.5. An ABB IRB 120 robot with a Robotiq C-85 parallel jaw gripper attempts to grasp objects on a transparent acrylic table. A wide-angle upward facing webcam tracks the pose of an AprilTag [87] attached to the underneath side of the object. The frame is made of one inch 80/20, which also form raised walls that prevent the object from falling off of the table.

The camera's intrinsics are calibrated using the standard ROS camera calibration tool, and the camera's frame is calibrated to the robot by having the robot move a larger AprilTag around. Then, a further calibration is done by having the robot grasp the object and move it around on the surface of the table. This allows us to obtain a 0.3mm object pose estimation accuracy near the center of the table, and a 1.5mm accuracy near the edge of the table.

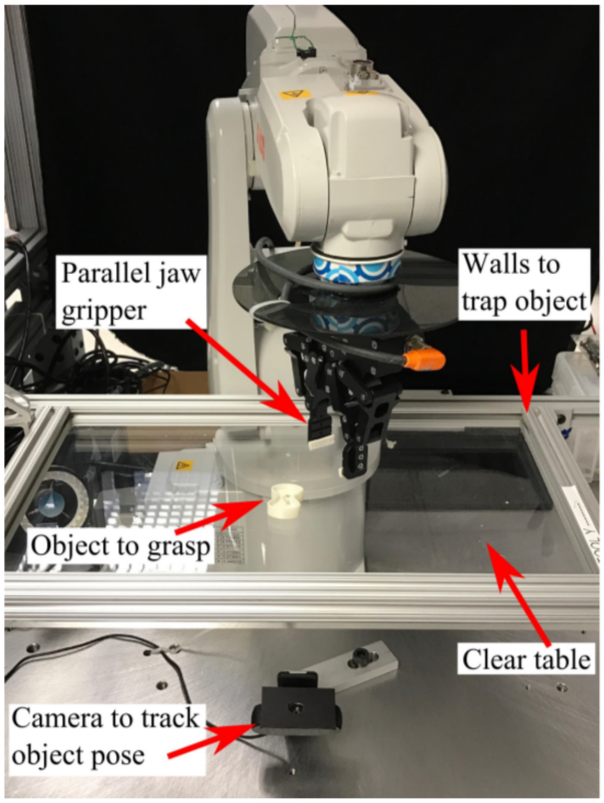


Figure 3.5: Planar grasping experimental setup. An object with an AprilTag on the underneath side is grasped on a transparent acrylic table top. The camera underneath tracks the object's pose, while the robot interacts with the object above. The walls are raised on the sides to prevent the object from falling off the table.

If the object or robot during a grasp would be too close to the edge of the table, the robot performs a reset action by dragging the object back to the center before starting its next grasp. To collect data, a random initial object pose relative to the hand is chosen, and then a grasp is performed, with the initial and resulting hand and object poses recorded. The robot is able to autonomously collect grasping trials overnight, and collects data on approximately 300 grasp trials an hour.

A few problems that were encountered during data collection were the hand losing calibration, vision system instability, and unexpected object motion. Over many experiments, the hand would start closing to a different width, so this was solved by recalibrating the hand every so often. For the vision system, we noticed that sometimes there would be multiple or zero AprilTag detections. Multiple detections were solved by querying the vision system again, and zero detections were solved by using larger AprilTags and eliminating light reflecting off of the bottom of the acrylic. Sometimes, when grasping the object, a dynamic motion would occur because of squeezing. Every so often, the object would flip up into the air, and land on its side rather than on the AprilTag. This would cause the experiment to stop, and we would reset it ourselves. As the data requirement was not that high, this was not a huge issue, but something to consider in the future.

3.5 Case Study 4: Tray Tilting

In this experiment, we were interested in expanding on Erdmann and Mason's sensorless manipulation work [34]. In their work, they showed that regardless of the initial position of an allen key in a tray, by tilting the tray in a specific sequence of directions, the allen key would always end up in the same final pose. Instead of explicitly planning a sequence of tilts, we randomly select a long sequence of tilts, and see if the phenomenon still holds.

Our experimental setup is shown in Figure 3.6. We use an ABB IRB 120 robot with an aluminum box attached to it. An AprilTag [87] is attached to an allen key, and a downward facing camera finds the pose of the object. There is an acrylic top to the box which prevents the object from flipping over or falling out of the box.

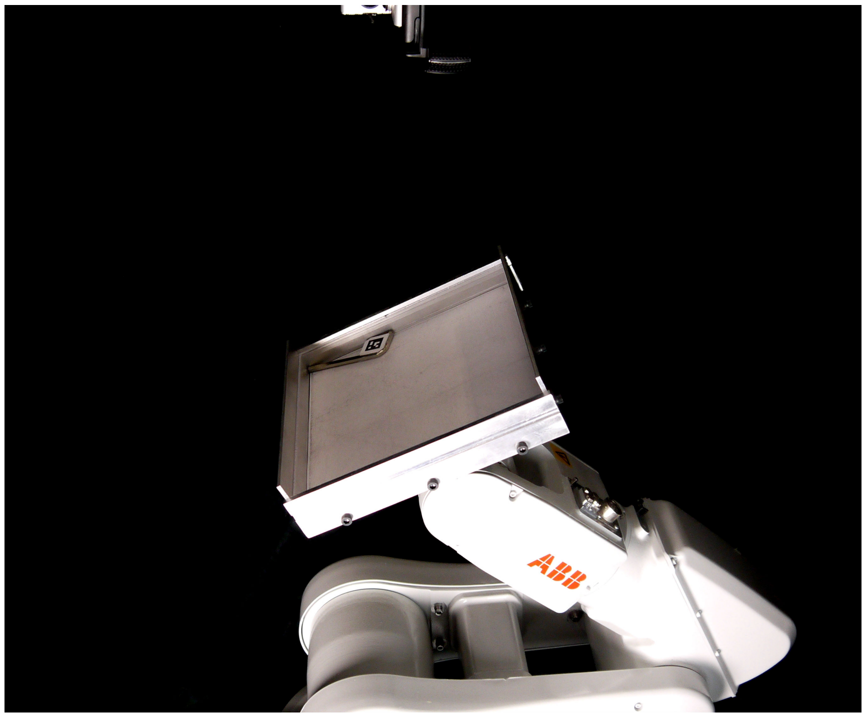


Figure 3.6: Tray tilting experimental setup. The robot tilts the aluminum tray in various directions, and the camera records the resulting pose of the object inside.

We performed over 500 trials of 50 tilts, which is over 25000 total tilts. For each trial, the robot performed an initial randomization move, where the robot quickly shook the box along several directions. The object would bounce off of the walls and arrive at a pseudo random pose for the next trial.

Several issues were encountered over the course of the experiments. First, sometimes there would be multiple or zero AprilTag detections, because of light reflections on the acrylic or other things in the camera's field of view. This problem was mitigated by simply checking that there was exactly one detection before proceeding. Second, we noticed stiction effects between the allen key and the smooth aluminum bottom of the box. We fixed this by putting a piece of paper on the bottom. The biggest problem however was that the sides of the box wore down with the allen key sliding along the sides. This led to high friction and completely changed the physics of the box over the course of the trials. This was mitigated by replacing the sides of the box as they wore down, but this remained the largest obstacle for long term experiments.

3.6 Case Study 5: Place and Pick Regrasp of a Wooden Cube

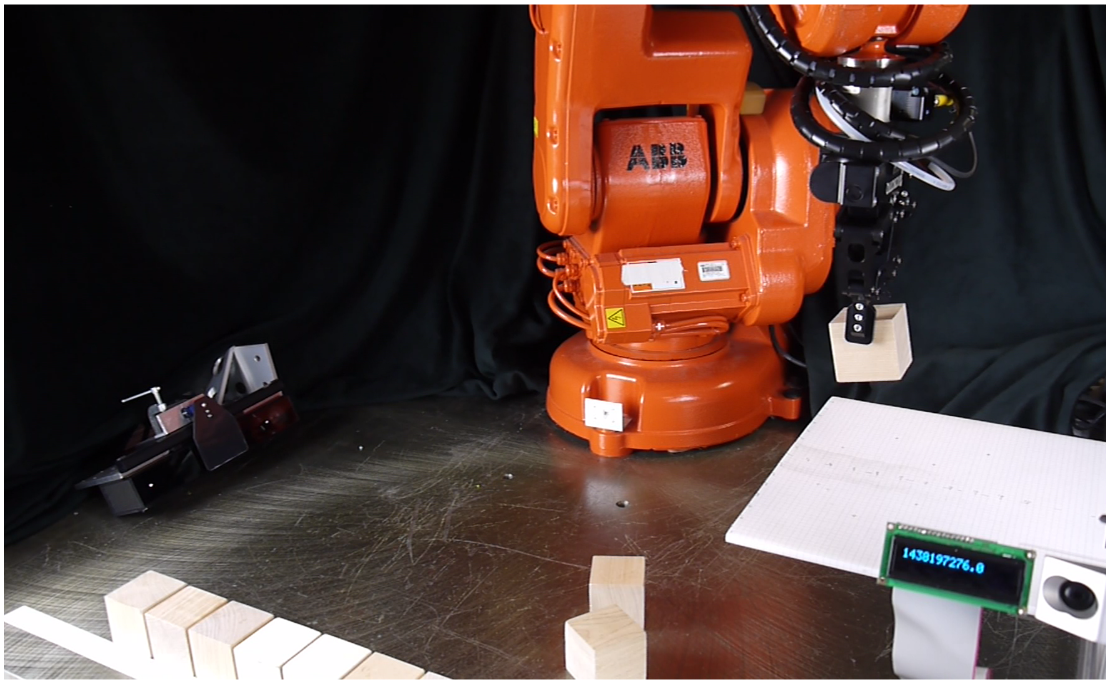
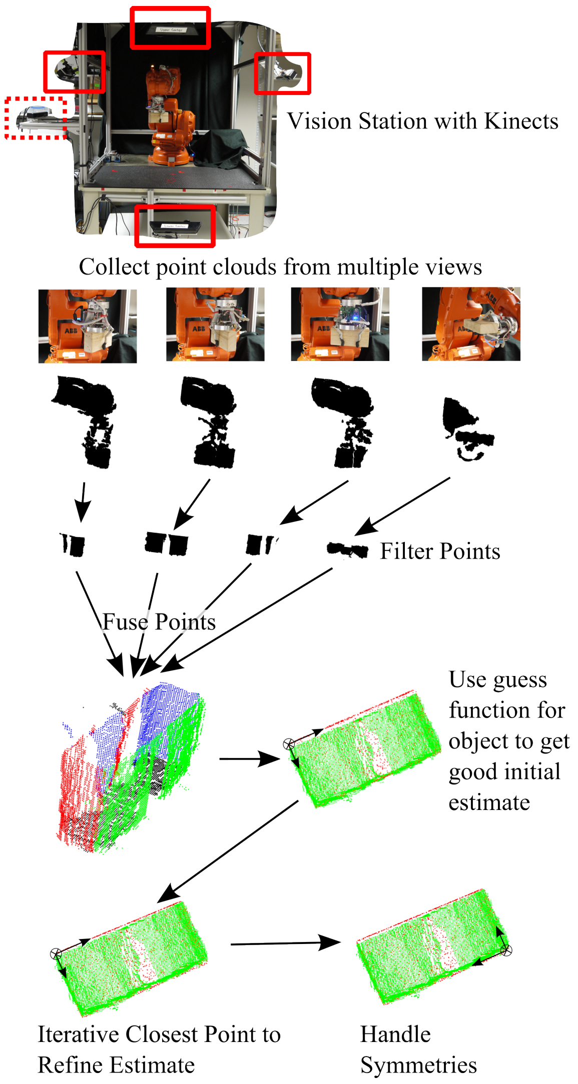


Figure 3.7: Place and pick regrasp data collection setup. We use an industrial arm with a parallel jaw gripper and place an pick objects. A three dimensional vision system based on point clouds is used to record the object position before and after a regrasp. In the event of a failure, the robot either picks up the dropped block from on top of the platform or retrieves a new block from a stack of fresh blocks. With this setup, the robot performed over 3000 regrasp experiments. We use this data to fit models for predicting the probability of success, and estimating the final pose of the block after a regrasp.

The purpose of this experiment was to model a place and pick regrasp of a cube. We wished to learn two probability functions: 1) Given the in-hand pose of an object, what is the probability that the object is still in the robot's hand after a regrasp, and 2) Given the in-hand pose of an object, what is the probability distribution of the resulting in-hand pose of the object after a regrasp? We needed to collect a large amount of accurate data to learn these models.

Our data collection setup is shown in Figure 3.7. For our experiments, we use an ABB IRB 140 industrial robot arm and a Robotiq C-85 2-fingered gripper that place and pick an object



Vision Station with Kinects
Collect point clouds from multiple views
Filter Points
Fuse Points
Use guess function for object to get good initial estimate
Iterative Closest Point to Refine Estimate
Handle Symmetries

Figure 3.8: An overview of the object pose estimation method. First, we collect point clouds from multiple views of the object to get as much coverage as possible. We can collect point clouds from each of the solid-box cameras, or collect multiple views of the object from the dotted-box camera using the robot. We then filter the points by removing known obstacles such as the robot's hand. We then fuse the point clouds together using well calibrated camera extrinsics. Next, we use a user-selected function for the specific object to initialize our pose estimate. We then use Iterative Closest Point to refine our estimate. Finally, we take into account object symmetries to return consistent pose estimates which are useful when building models for robotic manipulation.

from a metal platform. We use a $50\,\mathrm{mm}$ wooden cube as our object. Initially, the block is resting on the platform and the robot locates it and picks it up. The vision system records the initial state $\boldsymbol{s}$. Then, the robot places the cube and picks it up again using an action $\boldsymbol{a}$. The $\boldsymbol{a}$ parameters $[d, z, \alpha]$ (see Section 4.3) are sampled uniformly at random and cover the entire range of actions we wish to model for this regrasp. The vision system first checks whether or not the cube is in the robot's hand and then records the final state $\boldsymbol{s}'$. If the object is grasped, it repeats the process with a new action $\boldsymbol{a}$. If the object is not grasped, it enters a recovery procedure and then runs a new regrasp experiment. In this way, we collect a series of $D = (\boldsymbol{s}, \boldsymbol{a}, \text{grasped}, \boldsymbol{s}')$ data points. We collected 3304 data points with our experiments. The robot successfully maintained its grasp of the object after a regrasp 2642 times and failed 662 times.

Figure 3.8 shows an overview of our vision system. The first step is to acquire and fuse point clouds together. Second, we use a simple two-stage filtering process by first only keeping points inside of a boundary box, followed by ignoring any points inside of "ignore regions". All boundary boxes and ignore regions are rectangular prisms translated and rotated in 3 dimensional space. After these first two steps, almost all of the remaining points are assumed to belong to the object in question. Our third step is to run these points through a "guess" function specific to the object in question. Fourth, we use our initial guess as a seed to Iterative Closest Point (ICP) [7] to get our final pose estimate, and then check to make sure the solution is correct by looking at the percentage of matched points. Finally, if our object is symmetric in some way, we use a symmetry handler to find the closest homogeneous transform to some desired object orientation. The average positioning error of this system is $5\,\mathrm{mm}$.

The recovery procedure reduces the need for human intervention during data collection, and is split into 2 parts. If the object is not in the robot's hand, it is either resting on the platform, or has fallen off the platform. If it is resting on the platform, we command the robot to pick up the object and continue with the next experiment. If the object has fallen off of the platform, we consider the object lost, and grasp a new block from a queue of identical blocks resting on the table. The queue is 11 blocks long, and the robot takes approximately 150 trials to exhaust the entire queue and require human intervention.

3.7 Case Study 6: Improved Place and Pick Regrasping

There were two main issues with the previous regrasping experiment. First, the 5mm accuracy of the vision system put a limit on how accurate our models could be. Second, once the robot ran out of extra blocks to grasp, a human would have to intervene for the experiment to continue. This limited the total amount of data we could collect.

In order to improve the quality, quantity, and richness of the data set, we designed another data collection setup to model both placing and picking actions of a rectangular prism. This experiment shares a lot of its design with the previous experiment, but makes key improvements to the vision system, adds a reset mechanism to allow the robot to automatically recover from failure, and adds a state machine to improve software robustness. We discuss each of these elements in detail in the sections below.

Our instrumented manipulation playpen is shown in Figure 3.9 and consists of the following components:

- An ABB IRB 140 6 degree of freedom robot with 0.03mm repeatability.
- A C-85 2-Fingered Robotiq parallel jaw gripper.
- An ATI SR-48 collision sensor, which acts as an automatically resettable breakaway wrist
- 4 Intel SR300 depth cameras mounted surrounding the workspace to accurately estimate object pose (Section 3.7.1)

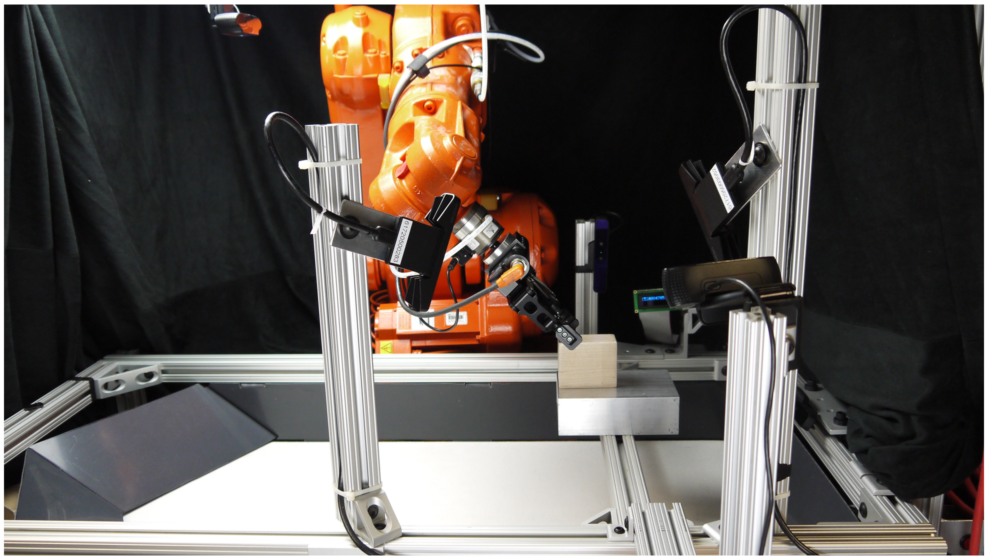


Figure 3.9: The instrumented manipulation playpen. This setup can be used to collect a large amount of accurate robotic manipulation data with minimal human intervention. For actuation, there is an industrial robot arm, collision sensor, and parallel jaw gripper. Multiple depth cameras that are accurately calibrated give us robust and accurate pose estimation. Below the experiment area lies a reset mechanism, which enables the robot to recover from failures. This setup was used to collect over 10000 pick and place operations.

- A reset mechanism below the workspace designed to retrieve an object if the robot drops it (Section 3.7.2)
- Finite state machine logic to control the experiment, enabling the system to recover from unexpected events without human intervention (Section 3.7.3)

We collect over 10000 object picks and placements, and visualize the data in Section 3.7.4

3.7.1 Depth-Based Multi-Camera Vision System

We need a vision system that can accurately estimate the pose of an object that is being manipulated. The object may be occluded, and not necessarily textured. It is also critical that when recognizing the object, our false positive rate is as low as possible. That is, when we think there is an object, there definitely is one. In practice, one of the main reasons that robots fail to manipulate objects is because they think an object is at a certain pose when it is not.

Luckily for us, there is plenty of prior work on finding the pose of objects in scenes [31] [4]. We will leverage this work, but also leverage the nature of our setup to improve the robustness and accuracy. Prior work is often more interested in recognizing many objects at once. Accuracies are often reported in pixels. As we will be manipulating objects, we need to know how accurate our vision system is in mm and degrees.

Our main insight is that the more sides of the object can be seen, the higher the the accuracy and robustness of the system. If only one plane of the object can be seen, errors of the object sliding along the plane are hard to control. However, if more than one side of the object can be seen, the pose of the object becomes ”locked” in place, and much higher accuracy and robustness can be obtained.

Our second insight is that the calibration of the cameras, both intrinsic and extrinsic, have a large effect on the overall accuracy of the system. Regardless of how advanced or precise of a vision algorithm is used, the algorithm is at the mercy of the camera calibration.

We choose to not use color as an input signal. In our experience, especially in manipulation, shadows from the robot hand or changing object poses often negatively affect the performance of vision systems that use color. In addition, often objects robots are trying to manipulate are untextured, which means using visual features other than the color of the object itself becomes quite unhelpful. Finally, because our environment is well known, as we will see it is quite straightforward to find the pixels that relate to the object in question, which is often what color is used for.

We will use multiple consumer-grade depth cameras that allow maximum coverage of the scene to generate a point cloud of the scene, and then find how a point cloud model of the object fits within that.

3.7.1.1 Depth Camera Calibration

The accuracy of our vision system depends almost entirely on the accuracy of our camera calibration. The point cloud generated by each camera must be as close to reality as possible, and the pose with respect to the world of each of the cameras generating these point clouds must also be as close to reality as possible. We detail our calibration methods for both intrinsic and extrinsic calibration below.

Intrinsic Calibration As shown by Basso et al [6], consumer-grade depth cameras have a fundamentally different error profile than color cameras, and must be calibrated accordingly. Generally, absolute depth camera error is quadratic as a function of distance, and their RMS error is also quadratic as a function of distance. Basso developed a non-parametric calibration method to correct for both of these errors, and we use their method to calibrate each of our cameras. We briefly describe their method here.

The method takes as input depth images from the camera of a plane from various poses. For an accurate calibration, the pose of the depth camera with respect to the plane must be known. In Basso’s paper, the pose of the camera with respect to the plane is calculated by placing a known checkerboard on the plane. By using a calibrated color camera, the transform between the color camera and depth camera is jointly estimated with the pose of the depth camera with respect to each plane.

There are two parts to Basso’s method. First, the depth images are undistorted using a local undistortion map, which adjusts the depth map so that planes seen in the image that previously were distorted look flat. Second, the depth images are globally modified so that the distances of the planes seen in the image represent reality.

Extrinsic Calibration Now that each of our depth cameras have metrically correct point clouds, we must calculate the pose of each depth camera with respect to the world. Previous methods have used a color camera at a known transform from the depth camera looking at a pattern of some kind to estimate the depth camera pose [84] or the IR image of the depth camera. However, using a color camera or the IR image introduces an extra step of error into the calibration, as there will be some error in the transform. We choose instead to directly use the point cloud from the cameras to calibrate them, as these point clouds will be what are used when estimating the object pose.

Ideally, we'd like a target that we can move around in the work space and calibrate all of the cameras at the same time. To do this, we attach a sphere on a stick to the robot, and move it around. As we do not know exactly where the sphere is relative to the robot's tool flange, and we don't know where each camera is, we'll have to solve a hand-eye calibration problem. We follow Wu et al's algorithm [120], which first computes a closed-form approximate solution using Kronecker Product, and then iteratively computes a more accurate solution using Gauss-Newton and $so(3)$ Lie-Algebra for rotation. We will refer to this algorithm as $Wu2017$.

After collecting a large number of samples, we run the following algorithm:

- For each camera, remove the background by finding which voxels don't change occupancy across most of the clouds
- Use RANSAC to find the position of a sphere of radius r in each point cloud
- Use RANSAC together with $Wu2017$ with a low acceptance rate to find the pose of each camera and sphere position
- For each point cloud, only keep points where the sphere should be, according to our previous calibration
- For each stripped down point cloud, use RANSAC to find the position of the sphere
- Use RANSAC with $Wu2017$ with a high acceptance rate to find the pose of each camera
- Use a slightly modified version of $Wu2017$ to find the pose of all cameras and one sphere position.

3.7.1.2 Finding the Object in the Point Cloud

With our calibration complete, we can combine the point clouds from all of the cameras into one point cloud in the world frame. Our task now is to figure out whether an object exists in that point cloud, and if so, where. We will do this in four steps: preprocessing, global matching, local optimization, and verification. We explain each below:

Preprocessing Often, much of the environment will be known to the user ahead of time when doing controlled experiments. The vision system's robustness and speed can be improved by removing points that are for sure not the object. To do this, the user can set a region of interest, and ignore regions. All regions are specified as rectangular prisms transformed to an arbitrary SE(3) pose. Checking if a point is inside or outside a rectangular prism amounts to checking its distance from 3 planes, which is fast. In our case, when finding the pose of an object on a platform, we set a region of interest in the vertical space above the platform. When finding the in-hand pose of an object, we set the region of interest to be a region in the grasp area of the hand, and each of the fingers and base of the hand as ignore regions. In practice, this reduces the number of points from 1000000 to 10000.

Global Matching and Local Optimization We use the $surface\_matching$ package from OpenCV, which is an implementation of Drost et al. [31]. This algorithm downsamples the point cloud and creates global model descriptors using orientated point pair features, and matches them locally using a voting scheme. It finds good pose estimates, which are then refined using a pyramidal Iterative Closest Point approach.

3.7.1.3 Verification

Once final pose estimates from each candidate have been computed, we then check how many model points have a neighbor within some distance ϵ. Whichever candidate has the highest proportion p of matches is our selection. If that proportion is lower than some threshold δ, we determine that there is no object in the image, otherwise we return the object pose, along with p as a confidence value. The more of the object seen by the cameras, the more certain we are that the object has been found.

3.7.1.4 Symmetry Handler

The final step of our algorithm is to handle any symmetries. Robots often handle symmetric objects. Bottles, boxes, and balls are all examples of objects that have some form of 3-dimensional symmetry, meaning they are indistinguishable for a finite or infinite set of possible transformations. If we are to model a manipulation action, we need to return consistent results of where the object is in the robot's hand to get consistent performance. In addition, the robot needs to know when 2 object poses are indistinguishable, so if it encounters a pose it has never seen before, it knows how to relate it to past poses. Thus our two goals are: 1) Determine how *close* two object poses are to each other taking into account symmetry, and 2) Given an object pose, return the *closest* pose to some desired pose.

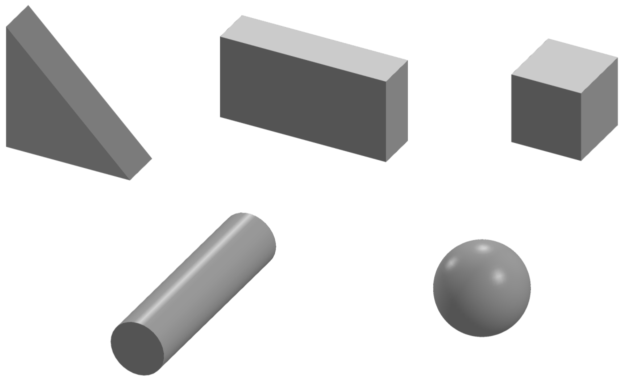


Figure 3.10: Many objects are symmetric, which can make it difficult for vision systems to report consistent pose estimates. The top row of objects each have order symmetries. The triangular prism has 2 possible configurations, the rectangular prism has 4, and the cube has 24. In the second row, the cylinder has an infinite symmetry about its axis, along with an order symmetry (flipping it 180 degrees). The sphere can take on an arbitrary orientation.

We define the *closeness* of 2 poses to be the magnitude of the difference quaternion $||q - p||$ where q and p are the quaternion representing the orientation of each pose. Thus, given an object orientation p, the *closest* pose would be one whose orientation q minimizes the expression $||q-p||$ where q is in the set of symmetric orientations.

Below, we briefly outline how we handle different types of object symmetries to answer the above 2 questions. For a full discussion of handling symmetry in 3 dimensions, please see the technical report [89].

Order Symmetry The top row of Figure 3.10 shows different order symmetries that can occur. An order symmetry occurs when an object can be rotated a fractional amount around an axis and look the same. Objects can have multiple order symmetries. An isosceles triangular prism has 1 order symmetry, a rectangular prism has 2 order symmetries, and a cube has 3 order symmetries. As the number of symmetries is finite, we can list all of the possible poses an object can take on, and pick the symmetry that minimizes $||q - p||$.

Cylindrical Symmetry In the case of cylindrical symmetry, such as the object shown in Figure 3.10, there are an infinite number of possible orientations the object can take on and still be indistinguishable, so it is impossible to list them out. Instead, it is possible to show that after transforming into the symmetry frame where the axis of symmetry is the z-axis, if our desired orientation is $p = [p_0, p_x, p_y, p_z]$, then the closest orientation to that attainable via cylindrical symmetry is $q = [p_0/\sqrt{p_0^2 + p_z^2}, 0, 0, p_z/\sqrt{p_0^2 + p_z^2}]$.

Note that sometimes, an object can exhibit both cylindrical and order symmetry (for instance, a cylinder or hour glass). In this case, we perform the above minimization twice, one for each side of the cylinder.

Spherical Symmetry A final option is for an object to have spherical symmetry, namely the object is a sphere. In this case, its orientation can be anything, and thus we have the condition $||q - p|| = 0$. If our 2 spheres have identical centers, then they are indistinguishable from one another.

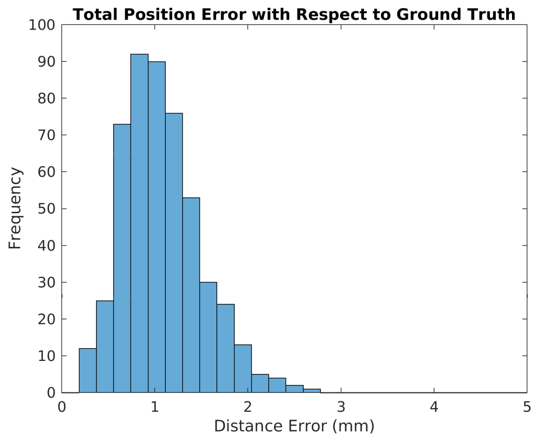
3.7.1.5 Evaluation

The accuracy of our manipulation models directly depend on the accuracy of our vision system. The SE(3) pose of the object must be accurately estimated both in the world, and with respect to the robot's hand. Assuming equal coverage by cameras, the pose estimate of the object with respect to the robot's hand will be strictly worse than in the world. This is because the error in robot motion will be compounded onto the object pose error. Thus, we evaluate the in-hand object pose as it represents a worse case analysis of the accuracy of our vision system. Also note that rather than using reprojection error or pixel wise error which is standard in many computer vision papers, we specifically care about the pose estimation error, and so this is what we will evaluate.

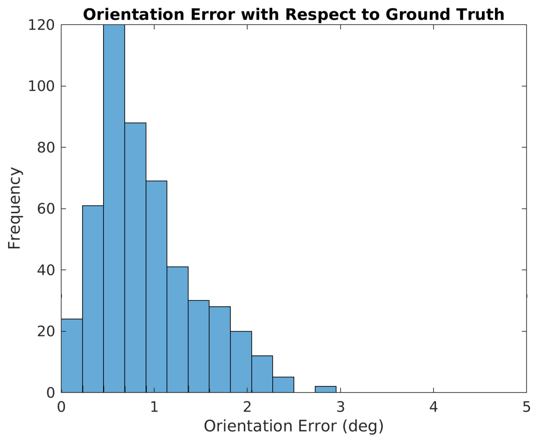
To evaluate the accuracy, we have the robot grasp a rectangular prism of known size at a known pose. Then, we move the object around to various places in the work space, and query the vision system about the object's pose. Because the pose of the robot is known, we can calculate the pose of the object with respect to the robot's hand at each of these positions. In the ideal case, all of the calculate in-hand object poses will be identical. This will of course not be the case, and we can use the deviation in object poses to estimate our accuracy.

The results of the experiment are shown in Figure 3.11. The object was moved around to 500 different poses in the work space. The error in position and orientation from the ground truth pose are shown in the two histograms. The position error is computed simply as the distance between object origins. The orientation error is computed by finding the rotation matrix that rotates one frame to another, and then computing the magnitude of the angle when the rotation matrix is formulated as rotating about an axis. Our average position error is 0.6mm and our average orientation error is 0.5 degrees. The standard deviation is 0.4mm and 0.5 degrees. Note that if the ground truth pose we used was slightly off, these errors could possibly be even lower.

This evaluation proves to us that even with off the shelf hardware, with careful calibration, the accuracy of our vision system can be quite high.



(a)



(b)

Figure 3.11: Vision system accuracy evaluation. A block held in the hand in a known pose is moved around the workspace, and a histogram of errors with respect to the SE(3) position and orientation of the block are shown. Our average accuracy is 0.6mm and 0.5 degrees, with standard deviations of 0.4mm and 0.5 degrees.

3.7.2 Reset Mechanism

While in other avenues of robotics, such as computer vision or self-driving cars, it is relatively easy to collect a large amount of data, it is more difficult in robotic manipulation because of two fundamental problems. First, generally when robots interact with the world, the end of the robot's task looks different than the beginning. If the robot is picking up an object from a table and placing it on a shelf, the object is now on the shelf instead of the table. If the robot wanted to repeat this experiment, it would first need to grab the object from the shelf and put it back on the table before proceeding. A second problem is that when robots fail while performing an

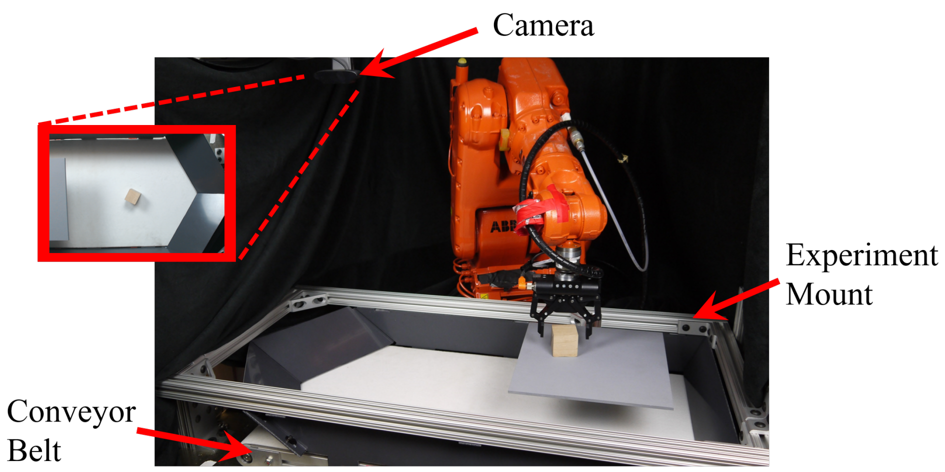


Figure 3.12: Our reset solution. In the event of a failure, the object falls somewhere onto the conveyor belt, which then moves the object into a clear area. The vision system then localizes the object, and the robot reacquires the object into its hand. See Figure 3.13 and Figure 3.14 for more details.

experiment, it may be impossible to recover. For instance, in the table to shelf example, if the robot accidentally drops the object off of the table, it may not have long enough arms to reach the object that has fallen on the floor. Or even if it could reach it, perhaps the object is stuck underneath the table in such a way that it couldn't grasp it with its current gripper. To enable robots to collect a large amount of manipulation data on their own, we must solve both of these "reset" problems to proceed.

Many researchers have overcome reset problems and collected datasets of robotic manipulation data. The standard approach is simply to use human intervention and collect as much data as possible. Yahya et al [122] get a robot to learn how to open a door after numerous experiments, with humans closing any doors the robots manage to open. Mülling et al [83] teach a robot to play ping pong, and use humans to help with imitation learning and reloading the ball machine. Another approach is to construct the robot experiment such that all states are viable. Christiansen et al [24] experimented with tray-tilting, where every position of an allen key or rectangular block inside of a tray was a valid state. The object could not escape the tray, so the robot could perform the experiment without the need for human intervention. Liarokapis et al [68] used a string to suspend an object in the air while learning how to dextrously manipulate them. If the object ever slipped out of the robot's hand, it could simply try again without any need for human intervention. Another approach is to use disposable objects. Paolini and Mason [90] studied place and pick regrasping on a platform. Any time the object fell off a platform, the robot would simply grasp a new, identical object from a queue. A human could reload the queue every so often, thus requiring minimal intervention. Another related field is parts feeding, which is often used in factories to present humans or robots with a known part oriented in a specific way. While most parts feeding solutions such as bowl feeders [111] are designed for a specific part, there has been some work with conveyor belts [71], and vibration [69].

For planar manipulations such as two-dimensional grasping [16, 126] or pushing [123], the object is kept on a table and is dragged back to the center when it gets close to the edge of the workspace. Dragging an object back to the center allows the object to be grasped or pushed from any direction. This enables the robot to collect thousands of experiments without the need for human intervention. Another data collection technique is one of placing numerous objects on a table, and having a robot attempt to pick up and set down these objects over and over again [28, 62, 76, 86, 96, 114]. If the table and robot's workspace are large, this experiment can continue for some time until human intervention is necessary to pick up any fallen objects and recenter them. Some researchers have used bins as a catch-all for failures. Levine et al [67] have robots repeatedly grasp objects from a sloped bin and drop them back into the bin after any successes. Paolini et al [91] place markers on top of a platform and then knock them back into a bin via a ramp after each experiment. At the Carnegie Science Center in Pittsburgh, PA, the basketball shooting robot ”Hoops” uses a slightly slanted floor, a sweeping motion, and a depression in the floor to recollect the basketball after it shoots to try again [1].

The paragraphs above highlight several solutions for enabling long-term robot manipulation experiments. As long as dragging or tilting a tray to move the object back to the center is an option, uncluttered planar manipulation problems are largely solved. For more complicated two and three-dimensional manipulation problems, the solution is less clear. While the setups mentioned above allowed researchers to collect the data they needed, the experiments were set up in a very specific way in order to collect the data. If researchers had to design new reset procedures every time they designed a new experiment, it would greatly increase the setup time for robotic manipulation experiments, and slow the growth of the field. In this section, we look at a more general way to solve this reset problem, such that minimal effort is needed to run a different experiment.

Not all robotic manipulation reset problems are going to be solved by a single mechanism or process. We are interested in three-dimensional robotic manipulation experiments, including both grasping and post-grasp manipulation actions such as regrasping, placing, insertion, tossing, and tool manipulation. Within this task set, we choose to solve a specific class of reset problems which have two large assumptions. First, we will focus on the case where there is only one object in the workspace. This is a major assumption, as most of the world is inhabited by more than a single object that a robot can interact with. However, there is still much to learn about how a robot can manipulate a single object, and many researchers are still focused on solving this problem. If researchers had thousands of real robotic manipulation trials to validate their robot and single object interaction models, we could be more successful at interacting with multiple objects in the world. Moreover, designing a general reset system that could reset multiple objects to predefined poses is an intractable task.

The second major assumption is that we will only worry about resetting the experiment when the object is no longer in the intended experiment area. We define the intended experiment area as the part of the robot's workspace where the object is expected to be in if the experiment is going well. If the robot is moving an object from a platform to a shelf, the intended experiment area is the platform and shelf. If the robot is throwing and catching a ball, the intended experiment area is the area above the robot's arm. It is impossible to design a reset procedure that is general enough to reset the majority of robotic manipulation experiments, simply because they are so different. Resetting an experiment when a robot is placing an object on a shelf requires an entirely different procedure than resetting an experiment when a robot is flipping a pancake. The one thing that most robotic manipulation experiments have in common is that the robot requires human help if the object disappears from the intended experiment area. This could be because the robot dropped the object off of a table, it pushed the object too far away for its arm to reach, or an object rolled underneath the experiment platform. In addition, note that the researcher can still hand-design robot actions to purposely remove the object from the intended experiment area. This could be done if the object is in an undesireable or unknown location in the experiment area, and resetting the object will enable to robot to continue on with its experiment.

3.7.2.1 Design Requirements

The goal of our reset mechanism is to enable researchers to exert minimal effort when designing a new manipulation experiment. They should be able to use different objects over a variety of manipulation tasks, without needing to redesign the way in which the robot recovers from failures. For a reset mechanism to be useful for researchers, it must have the following characteristics:

- **Versatile:** It must work across a variety of experiments and a variety of objects. Ideally, minimal or no changes should need to be made across different objects of a specific size (in our case, larger than a 1cm box but smaller than a 10cm box). In addition, the reset mechanism must accomodate many different experimental setups.
- **Unobtrusive:** It does not prevent certain experiments because of size. Ideally, the reset mechanism could be added to a variety of robotic manipulation experiments with minimal changes to the environment. Things to consider include robot workspace, lighting, power, mounting points, or netting that can get caught.
- **Fast:** To maximize the number of experiments a robot can perform, the reset process should be as fast as possible. Within reason, the time it takes to recognize that a reset is needed along with the reset procedure itself should be minimized to maximize data collection time.
- **Robust:** Since this design will be replacing human intervention, it should fail as infrequently as possible. Even if all of the above criteria are satisfied, if the reset process itself is prone to failure, this defeats its purpose. Our designs should favor simpler mechanisms and actions which are less likely to fail or need maintenance over time.

3.7.2.2 Possible Designs

In the sections below, we highlight six solutions out of dozens of possibilities that were all ultimately rejected in favor of our final design. In addition to the solutions we cited above in the introduction, it is our hope that researchers with specific needs will find this collection of possibilities useful.

Vacuum: The vacuum solution consists of a hose vacuum suspended above the robot workspace which could be used to recover an ungraspable object. The robot could drag the hose around the workspace until the object is sucked up. While this solution does not obscure the workspace, the vaccuum nozzle may need to be modified for certain objects, and depending on the experiment, the hose may not reach all necessary locations. In addition, if the workspace is large, it may take a long time to move the hose over the entire workspace.

Movable Walls: A movable walls solution would involve walls which could be translated or rotated for object recovery. This solution has the potential to be fast, versatile, and unobtrusive, but poses some concerns about reliability when used with small or flat objects. In addition, depending on the experimental setup, the walls might be blocked by part of the experiment, reducing their effectiveness.

Magnets: This solution involves an elevated platform with a line of magnets around the edge. These magnets would catch and hold a dropped object, which could then be recovered by the robot. However, this would require a sensor in order to find the object, and limits the robot to manipulating ferrous materials. This fails our unobtrusive and versatile requirements, as objects would have to be modified in order to be magnetic.

Air Jets: A solution involving air jets would function in a similar manner to an air hockey table, with air valves used to push objects away from walls and into the robots reach. The solution will not obscure the workspace, but it limits the weight of the objects the robot can manipulate and would require a sensor to locate the object, potentially reducing its versatility. In addition, certain experiment setups may be prone to objects getting stuck in unrecoverable locations after the air jets have blown.

Cloth Walls: This solution involves an elevated platform with loose cloth walls attached on all sides. The cloth acts as a net, and by lifting the cloth the robot can return the dropped object to the center of the cloth. This solution is fast and versatile, but poses concerns regarding its reliability and unobtrusiveness. Depending on the experiment setup, the cloth wall could get stuck on experiment features. Also, the cloth would obscure large amounts of the workspace, making it cumbersome to use.

Collection Area: A collection area solution involves a raised platform mounted over a bin with sloped walls. The bottom of the bin is a small area where the object location is relatively known. This solution was pursued for several weeks before it was determined via prototype that the minimum height needed for an object to slide down the walls exceeded the height above which the robot could not access all of its workspace. Thus, while it passes the versatility, speed, and robustness criteria, because of its size, it fails the unobtrusiveness criteria and was ultimately rejected. A second concern was that an object falling to the bottom may end up near one of the walls, which would make it difficult to grasp and reduce robustness.

3.7.2.3 Final Design

Overview Our reset mechanism consists of an experimental area that is mounted above a conveyor belt, surrounded by walls. We begin our description of the reset process when the robot has lost track of the object in its intended experiment area. At this point, we either assume the object has fallen from the experiment area, or the robot will perform some actions to guarantee the object is no longer in the experiment area. These actions could include the robot dragging its gripper along the surface of the platform to push the object off of the platform, blindly grasping and dropping the object on the conveyor, blowing air over the experiment, etc. The difficulty of this depends on the specific experimental setup, but all the robot has to guarantee is that the object has fallen off of the experiment area.

Now the main portion of our reset procedure begins. First, the conveyor belt is moved forward for a certain amount of time so the object moves to the front, and eventually to the front center

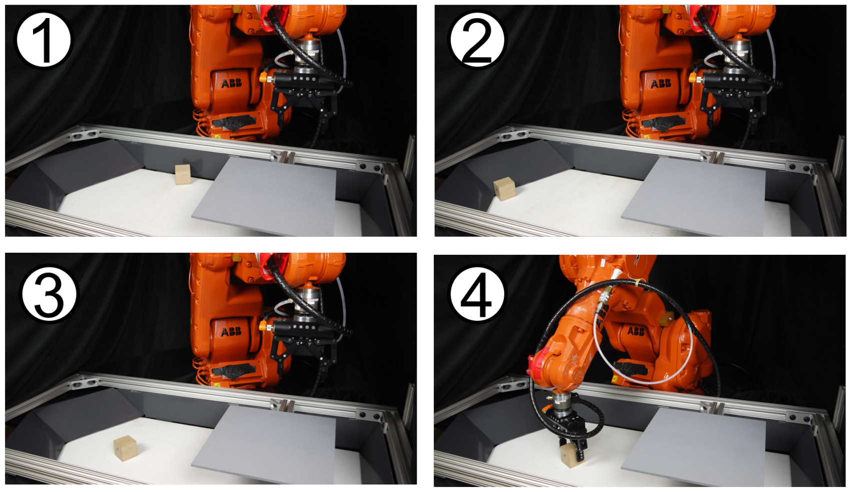


Figure 3.13: The reset process. First, after a failure, the object falls or is purposely knocked off of the experiment area and onto the conveyor. Next, the conveyor moves the object towards the front. When the object encounters the front sloped walls, it is directed towards the center. After some time the conveyor reverses direction and leaves the object in an open area for the robot to grasp. The vision system locates the object, and the robot moves to that location and picks up the object, ready for its next experiment.

because of the angled walls. The conveyor belt moves for long enough to reasonably guarantee any object will be in the front center, which in our case was 6 seconds. Second, the conveyor belt reverses direction for a short amount of time so the object is now in a clear space. Third, the overhead camera locates the object and chooses where to grasp. The robot then moves and grasps the object. Any extra steps necessary such as regrasping the object in a specific way can be performed in a reliable way in the experiment area. This completes the reset procedure, and the next experiment can begin.

This design is versatile because most objects can be centered using the conveyor, and many different experimental setups can be mounted above the conveyor. It is unobtrusive, because the walls only need to be as high as the longest side length of an object being manipulated. It is fast, and could be even faster if a faster conveyor belt was used. Finally, it is robust because there is only one moving part, and the end result is an object in an open flat area which is easy to detect and grasp reliably. In the sections below we discuss our design in more detail.

Mechanical Design The conveyor belt's width and length are chosen to be slightly larger than the robot's workspace in the experiment area. A large movable floor underneath the robot enables the user to design various experiments and still capture objects that have fallen from the experiment area. In our case, we selected a 50cm wide, 95cm long area for the conveyor. The height of the walls were chosen to be taller than the majority of objects the robot would interact with, in our case 10cm. The height of the conveyor belt is 7cm, making the total height of the reset mechanism 17cm. Thus, the reset mechanism is able to capture falling objects from the robot's entire workspace without being too obtrusive.

The conveyor is connected via a timing belt to a stepper motor. The timing belt pulleys have 14 teeth and 28 teeth giving us a 2:1 gear reduction to increase the amount of torque we can apply to the belt. The stepper motor has a current limit of 4 Amps, and we can move the conveyor at 0.8 meters / second while being able to apply reasonable force and avoid resonance/stall.

1.5 inch 80/20 channel lines the top of the reset mechanism, making it easy to mount various experimental platforms and setups. The walls are made with 1/8 inch PVC sheets that are cut and bent, and attached to 80/20 and the conveyor belt with 5/16" 80/20 hardware.

One issue we spent some time resolving had to do with round objects. In prototyping experiments, round objects such as balls or pencils would roll around even after the conveyor belt had stopped. This was a problem because either the object would end up in a pose that was ungraspable by the robot, or it would move after the vision system localized it, which would cause the robot to miss the object when grasping. This problem was resolved by finding a conveyor belt material with a much larger rolling resistance, and in our case we used a belt made of polyester felt. When an object lies on this surface, it sinks in to the felt, which means in order for it to move, it has to overcome the hole it finds itself in. Our round objects now rolled much less, and by playing with the acceleration and decceleration of the belt, we were able to reduce rolling even further.

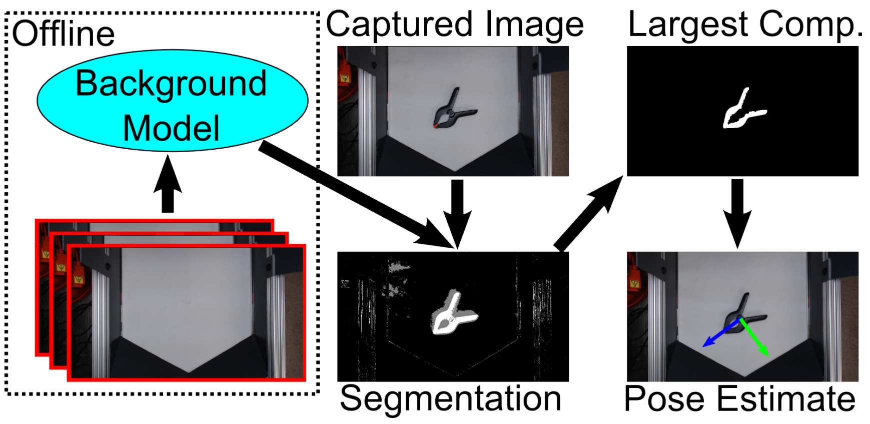


Figure 3.14: The vision system used to estimate the object pose on the conveyor. First, a Gaussian mixture background model is created with many empty conveyor belt images. Then, when an object is moved into position, the background and shadows are segmented out, leaving the foreground. An erosion and dilation step removes some noise, and then the largest connected component is found. We then look at the mean and covariance of this connected component to estimate the pose.

Vision System For the vision system, we use a Logitech c920 webcam pointed downwards at the conveyor belt. Its intrinsics are calibrated using AprilCal [101], and its extrinsics are calibrated by having the robot move an AprilTag [87] around the workspace and doing a hand-eye calibration [120].

Our vision pipeline is shown in Figure 3.14. First, we create a Gaussian mixture background model of the conveyor belt by moving it and taking lots of images [127]. Now, when an image with an object is shown, we get an initial segmentation of background and foreground. We then do an erosion and dilation step to remove some noise, and then find the largest connected component, and assume this is our object. We can then find the mean and dominant direction of the object. This mean and direction can then be converted to an object pose in the robot's workspace by using the camera's extrinsics and known height of the conveyor belt. Note that if the largest connected component is small, we can assume the object is still in the experiment area.

Many other vision pipelines could be used to trade off simplicity, generality, and accuracy, however we chose this pipeline as a first step.

Grasping the Object Once the pose of the object is known, all that remains is for the robot to grasp the object. There has been much research on how to pick up objects off of a surface, and any of these methods could be implemented here. We have again chosen a very simple approach, which is to use a parallel jaw gripper to grasp the object perpendicular to its major axis. Objects that have a clear major axis or circular symmetry can be grasped easily, whereas cubes or weirdly shaped objects are more challenging. Again, the user can decide how to trade off robustness versus generality with the grasping method they choose to use.

Robustness Note that the grasping and vision systems do not need to be perfect, because if the robot fails to grasp the object, the reset process can simply be repeated, and the robot can try again. As long as the combination of the vision and grasping systems are accurate enough to not require too many tries, we can satisfy our goal of having the reset process be fast. One of the only ways this reset mechanism will require human intervention is if the object manages to fall outside of the conveyor belt. If the conveyor belt is chosen to be large enough, and the angled walls above the 80/20 platform are high enough, this is unlikely.

3.7.2.4 Experimental Validation

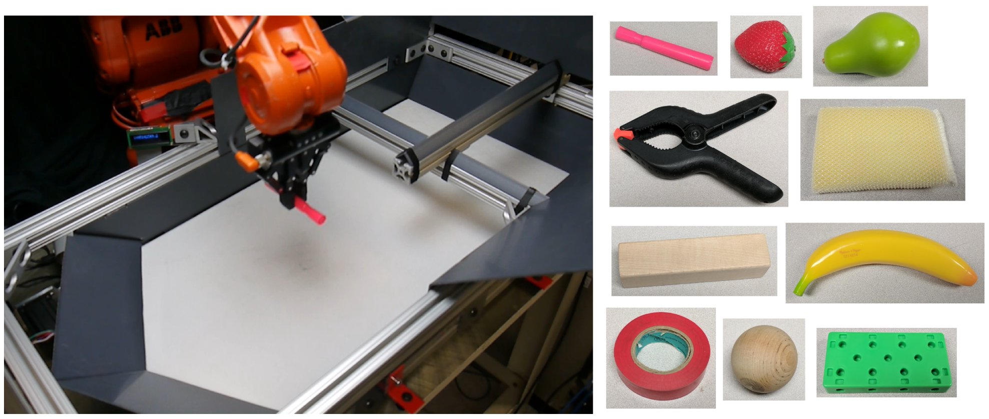


Figure 3.15: Our experimental validation setup is shown on the left. The robot drops an object somewhere in the experiment area, and then the reset process tries to get the object back in the robot's hand. The objects used in our experiments are shown on the right. Successes and types of failures are recorded, and results are shown in Table 3.1.

To validate our design, we dropped different objects 500 times each from random locations around our experimental area (Figure 3.15). In order to test the generality of our system, we did not change a single line of code across all experiments. We set up the experimental area such that when an object was dropped, its pose would be randomized because of impact with the experiment setup itself. Note that for many experiments it is possible or even likely that the object remains in the experiment area. The user would then have to design robot actions to help reset the object in that case. These actions are often difficult to generalize and are generally task specific. Please see Section 3.7.2.5 for further discussion. We are interested in whether the robot can successfully recover when an object is dropped off of the experiment area.

Object	Success	Missed Grasp	Vision Failure	Catastrophic Failure
Highlighter	497 (99.4%)	0 (0%)	3 (0.6%)	0 (0%)
Big Clamp	469 (93.8%)	18 (3.6%)	13 (2.6%)	0 (0%)
Small Sphere	500 (100%)	0 (0%)	0 (0%)	0 (0%)
Long Block	499 (99.8%)	0 (0%)	1 (0.2%)	0 (0%)
Plastic Strawberry	478 (95.6%)	6 (1.2%)	16 (3.2%)	0 (0%)
Electrical Tape	334 (66.8%)	47 (9.4%)	119 (23.8%)	0 (0%)
Plastic Pear	498 (99.6%)	0 (0%)	2 (0.4%)	0 (0%)
Sponge	491 (98.2%)	4 (0.8%)	5 (1.0%)	0 (0%)
Green Block	498 (99.6%)	0 (0%)	2 (0.4%)	0 (0%)
Plastic Banana	500 (100%)	0 (0%)	0 (0%)	0 (0%)

Table 3.1: Success Rates for Different Objects

Table 3.1 shows results for different objects. There are four outcomes that we keep track of. The reset is considered a success if the robot is able to grasp the object again after it has dropped it. A catastrophic failure means the object ends up in an unrecoverable position such that human intervention is needed. A vision failure means the vision system was unable to locate the object, or the determined pose of the object was in an ungraspable position, such as being too close to a wall. Finally, a missed grasp means when the robot went to grasp the object, it missed. Note that in the case of grasp or vision failure, we simply run the reset procedure again and retry for our next trial.

On the whole, our system performs quite well. The first thing to notice is that none of the objects suffered any catastrophic failures. This means that at no point during our experiments was any human intervention necessary. Since we retry after a vision or grasping failure, the reset process may take longer, but the robot is always able to reacquire the object into its hand eventually. Most of the objects had a very high success rate, however the big clamp, plastic strawberry, and electrical tape were slightly less successful. There were two different reasons for failures. In the case of the big clamp and the strawberry, the object is oddly shaped, so when it is being directed towards the front center, it would sometimes rock or rotate instead of sliding towards the center. This led to the object ending up too close to the edge and in an ungraspable position. In the case of the electrical tape, as the experiments went on the tape deformed, and the friction between the conveyor belt and object was too small in comparison with tape and side wall. This led to the object not sliding towards the center and being too close to the edge in an ungraspable position. Note that both of these problems could be solved by designing more specific conveyor belt motions for these actions. See Section 3.7.2.5 for more discussion.

Our experimental validation shows our system works well for a variety of objects, and is a viable system for researchers to use moving forward.

3.7.2.5 Discussion

In this section, we will discuss a set of general principles and potential pitfalls when designing reset pipelines for robotic manipulation experiments.

Generality vs Robustness One of the main issues we repeatably encountered during our design process was the tradeoff between generality and robustness. The reliability of the reset system can be improved if it is tailored specifically towards a certain experimental platform or specific object. One example of this is the vision system. Currently, we are simply doing background subtraction and finding the dominant direction of the largest remaining connected component to pick our grasping pose. We simply move to a specified height above this pose, and close our fingers. While this method works well for many objects, objects without a clear dominant direction like cubes, or objects which are oddly shaped and have a specific reliable grasp pose will not be grasped as reliably using this method. If the vision system were tweaked to have a cube detector or a detector which uses image features or deep learning to localize the object, our grasp success rate would increase, at the cost of generality. Alternatively, the grasping strategy could be modified to improve the success rate, such as making uncertainty reducing grasps or actions first before making a final grasp. Depending on the users application, they can tweak the vision system or grasping strategy to give them the level of robustness required. Note that currently, in the event of a grasp failure, we simply run the reset process again and try to grasp again. As long as our grasp success rate is reasonably high, we dont need too many trials for success. In this way, we are able to still maintain our generality at the cost of some efficiency.

Another example of generality vs robustness is the conveyor belt motion. For some objects, such as the electrical tape or clamp, if we moved the conveyor belt back and forth like a Coulomb pump [100], the object would end up in the center much more often. For other objects, this additional movement is not necessary, and would slow down the reset procedure. Again we are faced with a choice between making the reset procedure more specific for an object to improve efficiency, but sacrificing generality in the process.

A third example involves different reset strategies depending on the experimental area. There are certain motions or actions a robot can take that can knock objects off of an experimental area and onto the conveyor belt. However, these motions are specific to the experiment being done. While a sweeping motion may work well in the case of simple platform, if another experiment has posts, the robot will have to avoid those posts when trying to push the object onto the conveyor belt. If the user wishes to change no code between experiments, they can either very carefully engineer the experiment environment so the object never gets stuck anywhere undesireable, or accept the fact that human intervention will be necessary to continue on with the experiment.

The user will have to find the right point to hit on the generality versus robustness scale depending on their application. If an experiment is going to be run for many months, it makes sense to invest extra time in making the reset procedure more specific and robust to the experiment being performed. If instead the experiment is a quick test or only a thousand trials are needed, it may be more time effective to stick with a more general reset mechanism, and have humans standing by to assist.

Expecting the Unexpected When performing thousands of robotic manipulation experiments, low probability events will occur. We must design our reset process so that it can handle the unexpected. While we cannot prepare for all possible events, the more robust our system is to unexpected events, the less often human intervention will be necessary. One example of this is objects bouncing or rolling in unpredictable ways. Since our conveyor belt is larger than our robot's workspace, one might expect the sloped capture walls to be unnecessary. However, depending on the object, drop location, and experimental setup, the object could roll or bounce along and fall out of our reset mechanism. We minimized this chance by adding sloped capture walls. These walls do not eliminate the possibility of objects escaping, but they reduce the chance significantly.

In one of our trials, a long wooden block fell in such a way that rather than falling all the way to the conveyor belt, it rested up against a cross beam of the experiment platform. When the conveyor belt moved, it wedged itself between the conveyor belt and the beam. This represents the one failure seen for the wooden block in Table 3.1. As it turned out, because the conveyor belt reverses direction during our reset process, the block became unwedged, and on the next reset attempt, it was retrieved by the robot without trouble. What we can learn from this is that one way to be robust to unexpected events is to retry the reset process again. Moreover, in the event of multiple failures, other randomizing actions could prove to be successful, such as randomly moving the conveyor belt back and forth, or moving the robots arm around in the free space around the experimental area.

Unexpected events will occur in robotic manipulation experiments. The key for decreasing the need for human intervention is to build in as many safe guards and random recovery options as possible. When the need does arise, being able to recognize it and alert the user is also important for enabling large scale data collection.

Objects Getting Stuck in the Experiment Area Regardless of reset strategy, if the object gets stuck in the experiment area itself, this can cause large headaches for the user and robot. The object could get wedged between posts, stuck in the back corner of a shelf, or land on cross beams that support an experimental platform. Many of these issues can be solved with a combination of hardware and software. On the hardware side, eliminating flat surfaces or adding roofs or ramps to prevent objects from getting stuck in various places can be quite effective. Sometimes, simply using the robot to knock objects out of undesireable configurations may be enough. While many of these strategies and fixes can be thought of in advance, it is sometimes more effective to start running experiments, and fix issues as they arise. As the experiment continues, the mean time between human interventions should continue to increase, until the robot can run for long stretches without needing humans present.

Friction Friction tends to get in the way when trying to reset objects. Ramps often need to be steeper than expected, and objects can get wedged or stuck because of stiction or other non-Coloumbic frictional effects. Care should be taken to minimize the effect that varying frictional properties can have on the reset procedure. One example of frictional issues was the rejected ”Collection Area” design which used ramps to funnel objects to a central holding area. If an object was dropped with a lot of energy onto the ramps, it would easily bounce down to the central area. However, if the object fell gently, it would experience stiction and get stuck on the ramp underneath the experiment area. This required us to make the ramps much steeper than the coefficient of friction would indicate, and eventually led us to reject the design because it took up too much of the robots workspace to be useful. Another example was when we saw unexpected frictional properties of a squished electrical tape roll. The inner cardboard roll of the tape dug into the conveyor belt, and the high friction interaction between the tape and PVC wall was not enough to move the tape roll towards the center. Sometimes objects roll or tumble towards the center, and other times they slide. We currently do not fully understand all of the frictional factors at play between objects, conveyor belt, and PVC wall, and this lack of understanding limits the robustness of our system.

Complexity Because of the number of different outcomes of robotic manipulation experiments, the more complex the reset pipeline or system is, the more prone to failure it is. Objects can fall and get stuck in weird places that could jam machinery, lighting may be unreliable because of changing experiments or object properties, and the actual reset procedure could fail in unexpected ways. The less moving parts and less complex the system is, the less points of failure the system has, which increases reliability. One example of this is the moveable walls design described in Section 3.7.2.2. This had several moving parts that interacted with each other, and we rejected this idea in part because of the worry of objects getting wedged or stuck between wall parts. Care should be taken to design a system that is as simple as possible, as this will reduce the need for specialized actions or hardware modifications in the future.

3.7.3 State Machine for Manipulation Experiments

Over the course of a robotic manipulation experiment, the robot is sure to encounter numerous unexpected situations. While sometimes these situations simply result in incorrect data being collected, they often result in the robot being unable to continue. We have already addressed some of these concerns with the reset mechanism in Section 3.7.2, however we have found that often, unexpected events cause the experiment software to have undefined behavior, resulting in a failure.

To make our software more resilient to unexpected events, we will use a finite state machine. Finite state machines are often used in flight software for NASA missions because of their reliability [52]. If we can enumerate all possible events, and can define behaviors for how the system should react to each event while in a given state, our software will be more robust. The programmer is forced to think about how the robot should respond, rather than a series of *if* statements doomed for failure. This makes our software more reliable, and less likely to require human intervention.

In addition, by encapsulating the robot’s actions into individual states, our code becomes more modular by avoiding nested if-statements that may make the code unmanageable. Debugging also becomes easier, because individual states can be activated and tested separately. The state machine’s modularity can also enable code reuse between experiments. For example, the

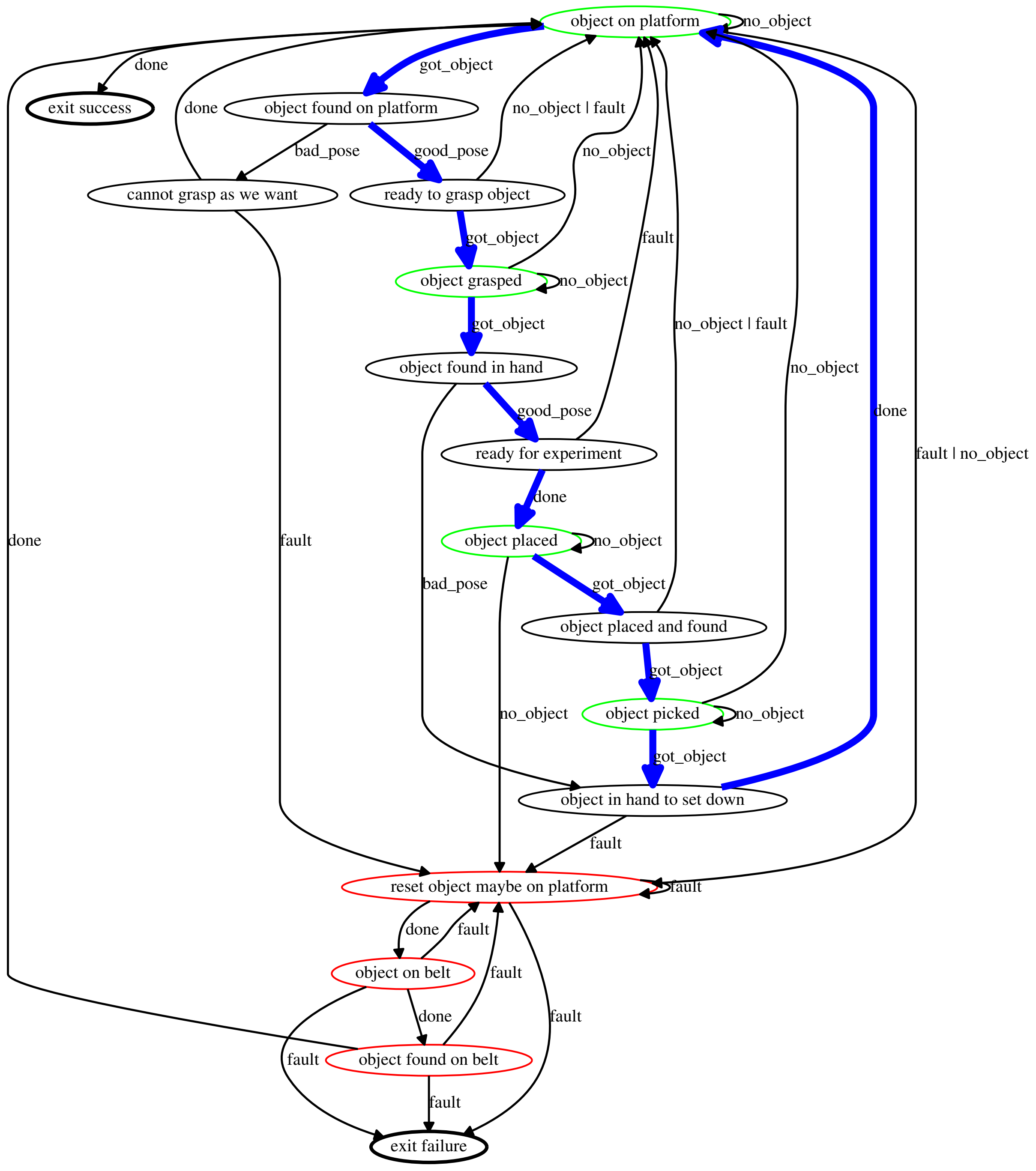


Figure 3.16: State machine used for place and pick regrasp experiments

states used for recovering from a failure using the reset mechanism will be similar from one experiment to the next.

We implemented a state machine for data collection of a place and pick regrasping action. We're interested in learning a data-driven statistical model of what happens when a robot places a block on a platform from a given initial configuration, and picks it up from the platform at a given hand pose. The outline for the experiment is as follows: 1) acquire the object off of the platform at a desired initial pose, 2) place the object, 3) grasp the object using a desired action, 4) record results, and set the object down on the table for the next experiment. While this is the nominal path, many things can go wrong, such as the block falling off the platform, the robot missing the block, or the block is unable to be grasped in the desired initial pose.

Our state machine design for this experiment is shown in Figure 3.16. Our states can respond to the following events:

- **done**: Objective successfully completed
- **fault**: Robot collision, or another unexpected error occurred
- **got_object**: Object found, or object was grasped
- **no_object**: Object cannot be found, or object wasn't grasped
- **good_pose**: Object in good pose, either in hand or on platform
- **bad_pose**: Object in bad pose, either in hand or on platform

Every state encapsulates what the current condition is at the beginning of that state. For example, in the "object found on platform" state, the object has been located by the vision system. Within this state, the goal will be to determine whether the object in its current pose can be grasped in the desired way. The bold blue arrows in Figure 3.16 indicate the nominal path that would be taken if everything goes as expected. The robot picks the object off the table in some desired initial pose, places it down, moves its hand to a desired pose and closes its fingers, confirms it has reacquired the object, and then sets it on the table to begin the next experiment. The states highlighted in green are where the vision system is used to localize the object pose relative to the robot.

When an unexpected event occurs, the program either transitions directly to the object being on the table, or through one or more intermediate states before the object is returned to the table. The part of the state machine that involves the reset mechanism is shown in red.

3.7.4 Collected Data

We collect data from 13943 experiments with 10472 successful initial grasps, 9246 successful object placements, and 4141 successful final grasps. Figure 3.17 shows which grasps succeeded and failed when picking the block up off of the platform. The 2 plots show the rectangular block lying on the platform in a "vertical" or "horizontal" configuration, and the grasp attempted for each are shown. The robot opens its fingers and slides in from the left in the figures shown. The C-shape is the end pose of the parallel jaw gripper, and the successful grasps are shown in blue, and the failed grasps are shown in red. Note that we have two different failure modes. First is when the robot tries to grasp the block near the edge, and it either misses the block entirely, or the block slips out of the robots hand. The second failure mode can be seen by the curve of red failed grasps near the far edge of the block. This is from one of the other links from the hand coming into to contact with the edge of the block leading to an edge contact instead of a surface contact, which leads to the block slipping out of the robot's hand.

Figure 3.18 shows which grasps succeeded and failed when placed. Note the visible curves

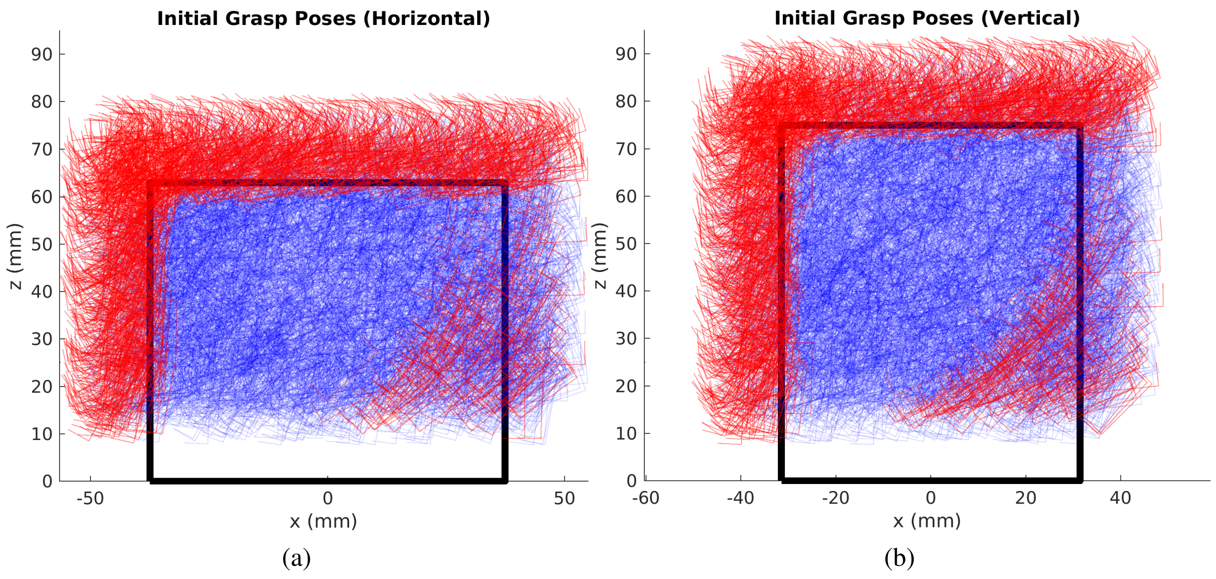


Figure 3.17: Successful and unsuccessful grasps of an object off of a platform. The thick black line is the block lying either in a (a) horizontal or (b) vertical position on the platform. The C-shapes are the end of the robot's finger tips. Red represents a failed grasp and blue represents success. Note that we have numerous failures along the edge of the block. The curve of failures is from one of the other links in the hand hitting the edge of the block which causes an edge contact instead of a surface contact which leads to the block slipping out of the robot's hand.

in the data for the successful grasps, which occur because the block is in contact with the palm or other links on those manifolds. The failed placements occur either because of a collision between the block-robot system and the table, or because the block falls off of the platform when it is released from the gripper.

Figure 3.19 shows the resulting platform poses of the rectangular prism after being placed. The two plots are for horizontal and vertical configurations of the block on the platform. Note that they are mostly in a line, which makes sense since the blocks are always released from the same gripper position.

Figure 3.20 shows what pick actions succeeded and failed. Again the robot moves in horizontally from the left side of the figure. Note that here we do not pay attention to where the object was when selecting our actions, so we have many unnecessary grasps, but we also have some successful grasps at certain angles far to the right of the block. When the gripper is flat enough, rather than having an extra link from the hand get caught on the block, that extra link pushes the block forward, resulting in a surface contact and stable grasp. If the angle of the gripper is too high, it simply passes right over the block and misses it.

3.7.5 Failure Modes

While we collected a lot of data with this data collection setup, there were still a few failure modes. First, ROS nodelets that were running the cameras were crashing silently after about

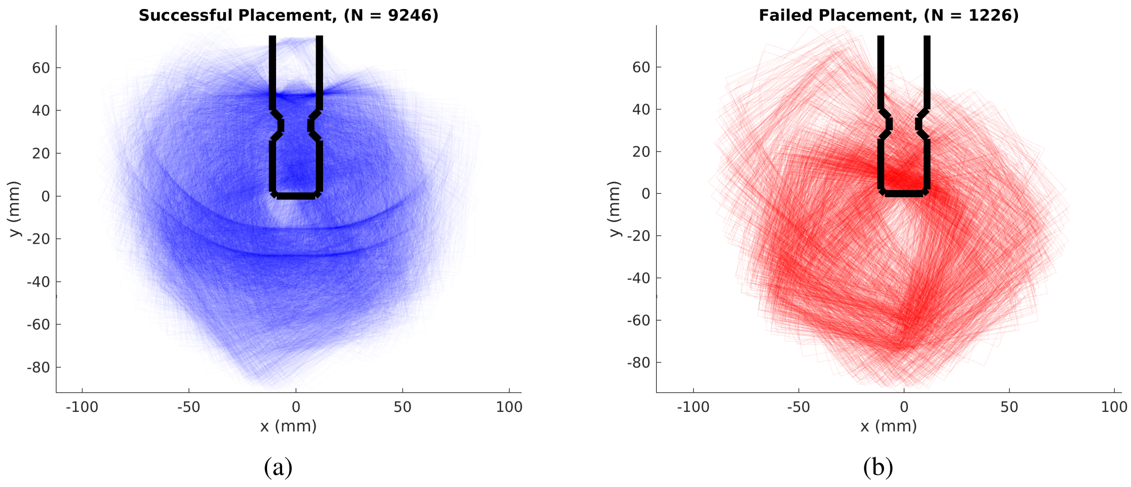


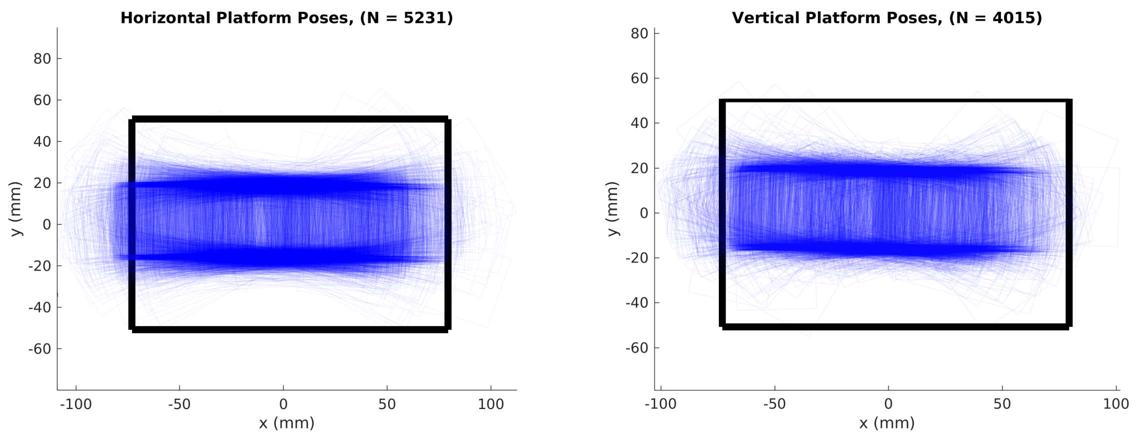
Figure 3.18: Successful and unsuccessful in-hand poses for placing. The thick black rectangle is the pose of the fingers. (a) shows the in-hand object poses that were successfully placed, and (b) shows the in-hand poses that failed to be placed, either because of a collision or the object falling off of the platform.

24 hours of operation. The only way this was solved was by writing a python script that would detect the nodelets crashing, and relaunching the vision system automatically, and having the state machine detect the crash and wait for a restart. There were also a few minor issues with the reset process. First, the path the robot took to knock off an object from the platform had to change because the block was getting stuck in an unexpected place. Second, after a night of resets, small changes in lighting caused part of the conveyor belt to be detected as the object, and the robot gave up because this was too close to the edge. This was fixed by only paying attention to pixels in a graspable area, and recalibrating the lighting every 500 experiments.

Another large problem was the degradation of the Robotiq hand we were using over time. Figure 3.21 shows the position of the grasped block along the grasp axis over time. Note that the pose of the block gradually moves about 3mm. This is because a part inside of the hand was wearing down over time. We attempted to fix the gripper a few times, shown by the black lines, but eventually it wore down again. While this drift will not considerably affect the models we're trying to learn, it highlights the need for robust hardware when doing long term manipulation experiments.

3.8 Common Themes

In the above sections, we saw six instances of robots collecting manipulation data. While the experiments were different, some common themes emerged. We discuss them below. While many researchers have no doubt experienced many of these issues before, it is important to keep these ideas in mind when designing robot manipulation experiments.



(a) (b)

Figure 3.19: An eagle eye view of the resulting object poses on the platform after a placement. All of the poses when the object is in a horizontal configuration are shown in (a) and the vertical configuration are shown in (b). The platform is shown in solid black.

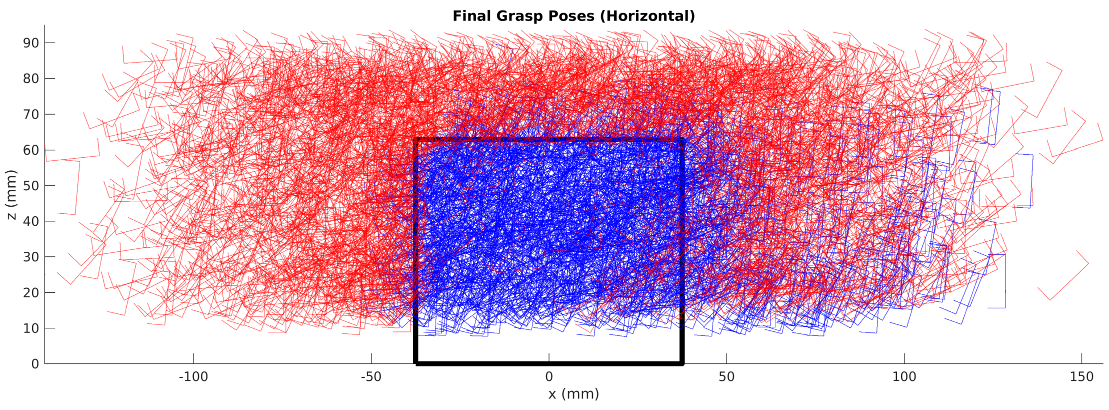
3.8.1 Accurate Sensing and Actuation

To have accurate models of manipulation data, accurate sensing and actuation is necessary. The model can only be as accurate as the sensors and actuation used. In each experiment, careful care was taken to select actuators and sensors which were accurate enough. The industrial robot arm and grippers used in our experiments give us sub-millimeter resolution. High resolution cameras were used to maximize accuracy. The mass of the pushed block was increased to obtain higher relative accuracy with the force-torque sensor. The motion capture system that we were using during the pushing experiments only had cameras with 640x480 resolution, which limits the overall accuracy of that system.

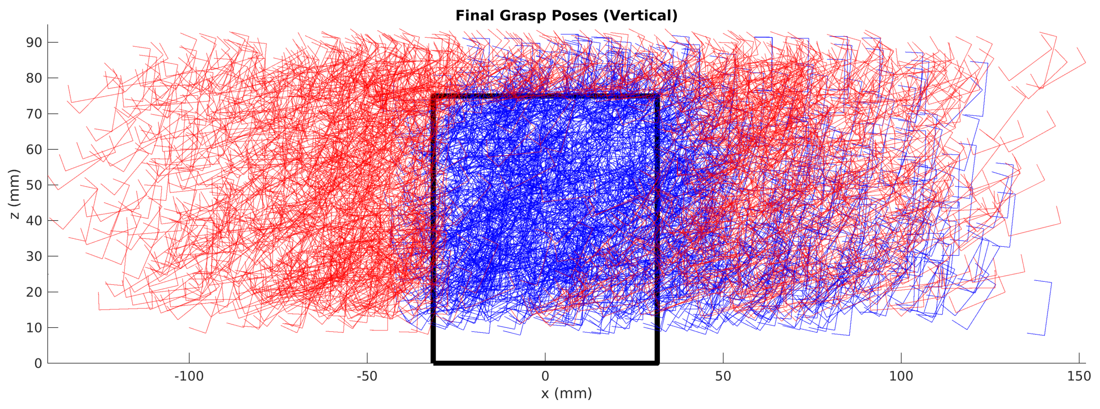
3.8.2 Calibration

In each experiment, a large amount of time was spent calibrating the system. There are many factors at play:

- Given the robot arm's joint angles, how accurately is the end effector position known? In our case, we used an industrial robot, which is calibrated at the factory. However, it is still critical to know the transform between the end effector and robot tool flange. In the case of pushing, the pusher was laser-cut with known dimensions and attached to a force-torque sensor, also with known dimensions. The hands used in the other experiments also have known dimensions which can be used by the system.
- How well are the intrinsics of the sensors used calibrated? For the ATI force-torque sensor, this is again calibrated at the factory. For the cameras, these can be calibrated using standard camera calibration techniques.
- What is the transform between the robot and sensor? (extrinsic calibration) In the case of



(a)



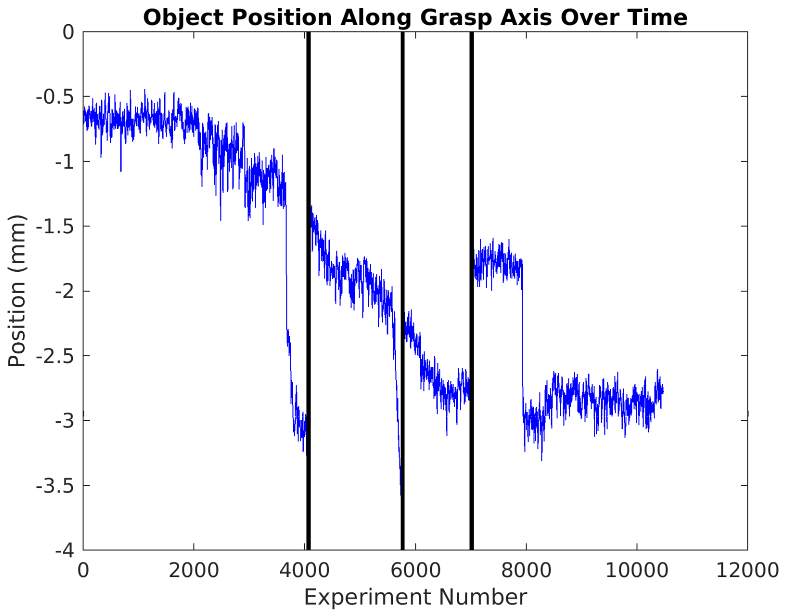
(b)

Figure 3.20: Successful and unsuccessful picking actions. The unsuccessful actions are shown in red and successful actions are shown in blue.

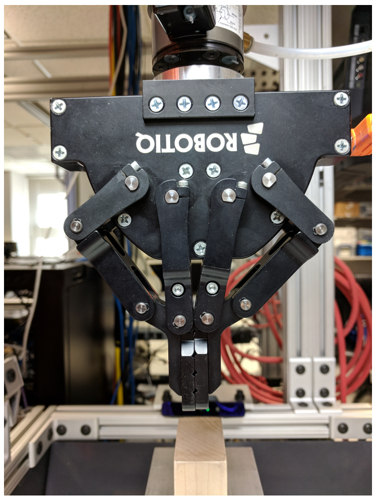
the force-torque sensor, it is rigidly attached to the robot, so it's transform is known. In the case of cameras, some specialized method must be used. For the highlighter markers, by rotating a grasped marker around and detecting the pose each time, the center of the hand in the camera frame can be found. For motion capture, moving the robot around with an IR marker allowed us to calculate the transform. For planar grasping and tray tilting, an AprilTag attached to the robot was moved around. For regrasping, by moving depth calibration patterns or a sphere around, we can correlate the robot's pose with the camera positions.

- How accurate is the object itself? If we are to precisely localize the object, its own dimensions and weight play a factor. The wooden cubes in the regrasp experiment are not perfect cubes, and have varied friction. The object being pushed had motion capture markers, but how accurately are the marker positions known with respect to the object itself?

Sources of error in each of these calibration factors compound to make our final data noisier. Special care must be taken to eliminate the largest sources of this calibration error for effective data collection.



(a) Object pose over time



(b) Gripper drifting to the left

Figure 3.21: Over the course of our data collection, internal parts of the gripper wore down over time, leading it to drift slightly to the left. This caused our data to change over time. The black lines in the left figure represent when the gripper was repaired.

3.8.3 Not Losing the Object

Special care must be taken that the objects being used in the experiment do not leave the robot's effective workspace. Anytime this happens is a time a human must step in and return the object to the robot. For the highlighter markers, the objects were in a bin, so if the robot drops an object while grasping, the object falls back in the bin and is not lost. When trying to place, drop, or insert the object, a ramp underneath the action area directed fallen markers back into the bin. In the case of a successful drop, place, or insertion, the object was knocked or grasped and dropped back into the bin. For pushing, any time the object got close to the edge of the workspace, the robot would drag the object back to the center. For planar grasping, there were walls that prevented the object from escaping, but the object would sometimes flip on its side and result in it getting lost. For tray tilting, the object was confined within the box, and the acrylic top prevented it from flipping over. For the first regrasping experiment, if the object was dropped on the platform, the robot would just pick it up again, and if the object dropped off the platform, the robot would grasp a new object from a queue. In our final experiment, we showcased our design for a general reset mechanism, which enabled a much longer run of data collection between human interventions. In all cases, special care was taken to extend the life of experiments by enabling some amount of ”reset” capability by the robot.

3.8.4 Hardware / Software Robustness

Both the hardware and software used during the experiment must be able to handle unexpected events. During manipulation actions, robots can collide with their environment, a grasped object could collide with the environment, and objects could fall onto important components. The hardware used must be robust enough for the robot to continue experimenting even if an unexpected collision occurs. Alternatively, software can be used to reduce the chance of unexpected forces. Although we attempted to use robust hardware, things still wore down. The sides of the aluminum box wore down during tray tilting, and the Robotiq hand used in regrasping broke and had to be repaired over time.

It sounds simple, but the number of experiment failures as a result of bad cable management that we experienced was quite high. In all of the experiments except tray-tilting (which didn't have any cables), cables got in the way time and time again. For the highlighter markers, sometimes the cable would droop down into the bin and get grasped, corrupting the data. Also, the cable got hooked on the placing platform, leading to system failure. In the pushing experiments, the cable got stuck inside of the robot's sixth joint, causing the robot to jam. It also was sometimes too loose and would touch the object while the robot was attempting to push it. In the planar grasping experiment, the cable would get caught in the sixth joint because of large rotations. In the regrasping experiment, the cable would sometimes get in the way of the vision system, and care was taken to make sure the cables did not get hooked on cameras.

On the software side, it is also important to plan for unexpected events. In each of the experiments, error recovery states were written in so that in the event that an object is lost or a collision occurs, the robot does the safe, correct thing to continue. For example, in the regrasping experiment, sometimes the vision system would not see an object the first time. Before proceeding, we would check again, just to make sure, and then look for the object somewhere else, before finally resorting to picking up a new block.

Chapter 4

Data-Driven Statistical Models of Manipulation

4.1 Introduction

For robots to be better at manipulation, they need to have accurate models of how their actions will affect the world. In the previous chapter, we showed how robots can collect a large amount of manipulation data. In this chapter, we look at how to use this data to learn models for manipulation. We choose to use statistical models instead of deterministic models to capture the inherent uncertainty in the real world. Even if we were to believe that the world was deterministic, we feel it is impossible to know every parameter needed to determine the outcome of an action perfectly. By using statistical models, we can not only model the predicted outcome of a task, but we can also model the uncertainty. This enables the robot to take more informed actions.

For the rest of this chapter, we present statistical models that can be used at different steps along the manipulation process. In the first section, we present results on predicting the probability distribution of where an object is in a robot's hand after it has been grasped as a function of its sensors. We also show how to predict the probability function of how likely a robot will succeed at a manipulation task as a function of its error in its estimate of object pose. We show how to combine these probability functions so a robot can compute the likelihood of succeeding at a manipulation task given its sensor readings.

In the second section, we present statistical models for a place-and-pick regrasp action. The robot can change the pose of an object in its hand by setting it down and picking it up again, and we show how to model two probability functions. First, we present a model for predicting the probability that the object will still be in the robot's hand after the regrasp. Second, we show how to predict the probability distribution of the resulting in-hand pose of the object after the regrasp.

In the third section we present an improved approach for statistically modeling placing and picking an object. We enlarge the state space by using a rectangular prism instead of a cube. We first show how to use calibrated classifiers to accurately predict the probability of picking and placing an object. Then we show how to use a k-NN approximation of conditional kernel density estimation to compute probabilistic transition models for both picking and placing.

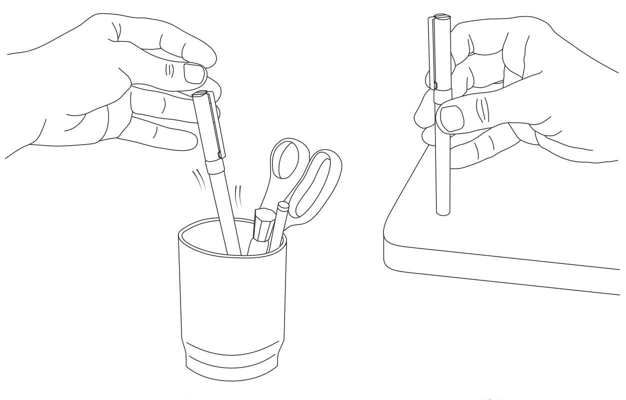


Figure 4.1: Two manipulation tasks (a) dropping and (b) balancing, with different required accuracy of the pose of the manipulated object.

4.2 Statistical Framework for Post-Grasp Manipulation

The work in this section was published in [91].

4.2.1 Introduction

We study the problem of post-grasp manipulation, where a robotic manipulator performs a task with a grasped object, like setting down a mug, inserting a key into a hole, or flipping a pancake with a spatula. In each of these examples, knowing the pose of the object in the robot's hand, the *grasp state*, is often critical. Intuitively, harder tasks demand a more accurate estimate of the grasp state than simpler ones. For example, in Figure 4.1, balancing a pen on a table requires more accuracy than dropping it into a container.

More generally, consider a manipulator, an object to manipulate, a task, and a parametrized set of actions designed to accomplish the task. We build a data-driven framework to automate the process of deciding whether the task is solvable with the available hardware and set of actions, and find the action most likely to succeed.

The proposed statistical framework is suited to model post-grasp manipulation tasks like the ones described above. We model these tasks by breaking them into two independent steps. First, estimate the state of the grasp with available sensor information, and second, model the accuracy requirements that the particular task imposes on our state estimation. This separation yields the benefit that we can use the same model of state estimation for different tasks, and the same model of task requirements for different manipulators. Using this framework, each sensor reading generates a probability function in task action space, enabling us not only to find the optimal action, but to understand just how likely that action is to succeed.

Figure 4.2 illustrates the process for the task of placing an object. First, we use sensors in the hand to estimate the probability distribution of the pose of the grasped object, which will be referred to as the belief state. Second, we predict the probability of succeeding at a task given the pose uncertainty. Both of these are computed based on data-driven models. Finally, we combine these probability functions to predict the probability of success of each available action, and choose the action most likely to succeed.

We test the framework with three different manipulation tasks: placing an object, dropping it into a hole, and inserting it, all three described in Section 4.2.1.2. The experimental setup in Figure 4.2 consists of a simple gripper [81, 102] mounted on a robotic arm that iteratively grasps an object from a bin, estimates the distribution of the pose of the object, computes the probability of success for all available actions, chooses the optimal one, and executes it.

4.2.1.1 Motivational Simple Example

Let us first look at the simple example in Figure 4.3. A two-fingered planar gripper holds a rectangular object to insert it into a hole. We assume the object always makes face contact with the palm of the gripper, so that once it is in the hand, it has only one degree of freedom, which we call grasp state x. One of the fingers of the gripper has a noisy angle sensor z. The angle z informs us of the position of the rectangular block. The question we study is, given a sensor reading z, where should we move the gripper (how do we choose action a) to maximize the probability of success of inserting the object, and how likely are we to succeed. Notice there are two main factors that influence the probability of success:

- How accurately can we estimate the pose of the object? We will refer to this factor as the *sensing capabilities* of the hand.
- What is our margin for error when inserting the object? We will refer to this factor as the *task requirements*.

Both are empirically modeled with the output of tailored experiments.

If there is no noise in the sensor reading z, and the geometries of both hand and object are known, we know with zero uncertainty the state of the grasp x, i.e., the location of the object. While not true in general, for the simple example in Figure 4.3 there is a one to one mapping between z and x, and we can therefore recover x perfectly. However, with the addition of noise, the estimate of object pose becomes a distribution $P(x|z)$, rather than an isolated configuration. Figure 4.4 shows how that distribution changes for three different sensor noise models $P(z|x)$. Section 4.2.2.1 details the process to derive the belief state $P(x|z)$ from the observation model $P(z|x)$.

The second factor to consider is the margin of error the task allows. Notice that we can effectively change that margin of error by varying the shape of the hole. Figure 4.5a shows three differently shaped holes that induce three different task requirements. The larger the hole, the less accurately we need to know the object location. In this case, the action is parameterized by a, the gripper position we choose. Figure 4.5b shows that the probability of successful insertion varies with the error ϵ in the estimate of the object pose.

We can now combine the probabilistic models for the sensing capabilities of the gripper (Figure 4.4b) and task requirements (Figure 4.5b) to find the optimal placement a of the gripper to maximize the probability of successful insertion. Figure 4.6 shows the result for the three sensor noise models in Figure 4.4a and the three holes in Figure 4.5a. Note that even when the estimation of the pose is uncertain, if the hole is wide the chance of success remains high. Likewise, if the hole is a perfect fit for the part, even the smallest amount of noise in the sensor

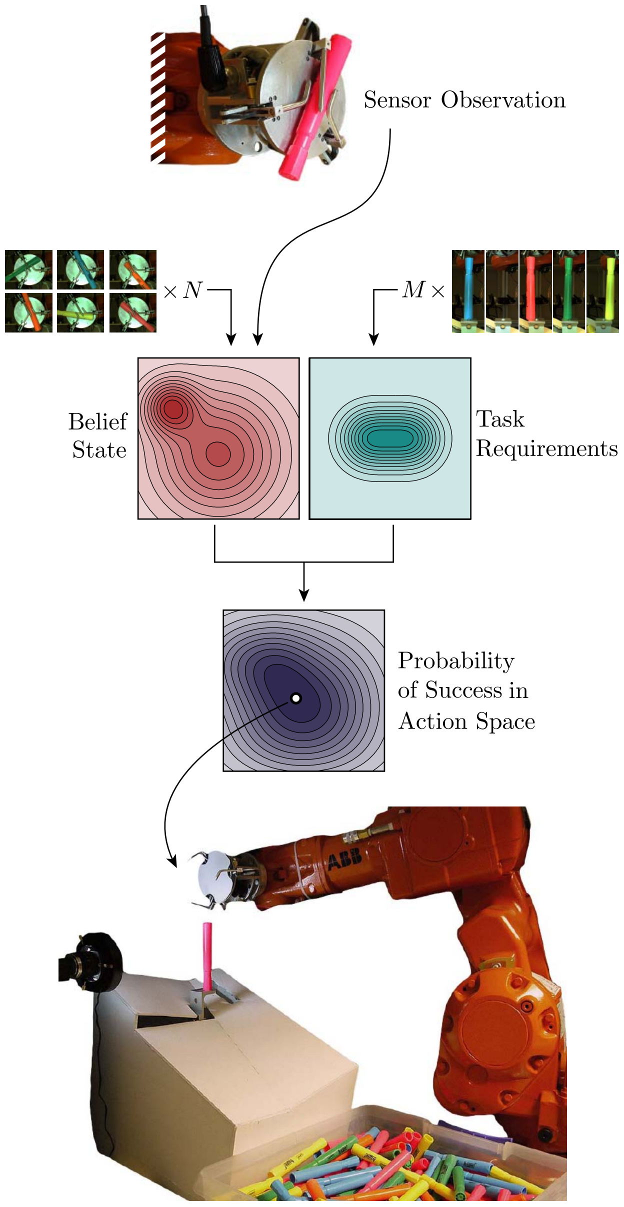


Figure 4.2: Procedure to choose the optimal action to accomplish a manipulation task. First, we estimate the belief state of the grasp, that is, a probability distribution of grasp state from sensor readings. Second, we learn how robust our task is to state uncertainty. Finally, we combine them to estimate the probability of success of all available actions and choose the best one.

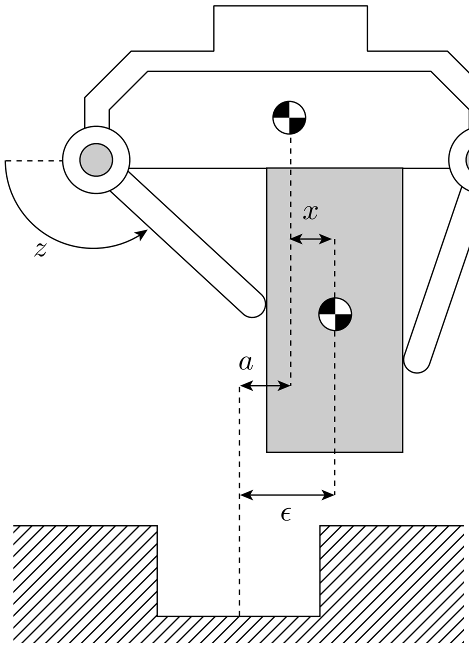


Figure 4.3: Motivational simple example in Section 4.2.1.1. A two-fingered planar gripper grasps a rectangular object with the goal of inserting it into a hole. The horizontal freedom of the part is parametrized by x. One of the two fingers has a noisy angle sensor z, which we use to estimate the location of the object in the hand. We note with action a the horizontal position of the gripper that we choose to vertically insert the object in the hole. Finally, ϵ represents the error in a with respect to the optimal value that would align the centers of object and hole. The goal is to estimate the likelihood of successful insertion using action a given the sensor reading z.

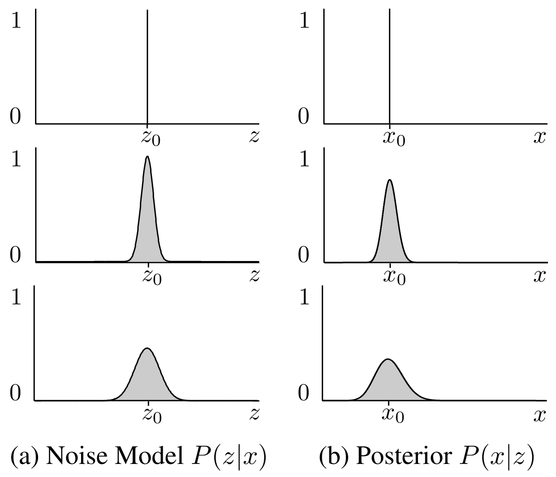


Figure 4.4: Sensor model for the gripper in the motivational example in Figure 4.3. a) The observation or noise model $P(z|x)$ is the distribution of possible values z we read from the sensor, assuming the object is at x. The level of noise in the sensor determines the sharpness of the distribution $P(z|x)$. b) The posterior distribution $P(x|z)$ of the pose of the object x is obtained by inverting the observation model. If the sensor has no noise (top row), for the simple gripper in Figure 4.3 which has no sensor aliasing, we recover the position of the object with zero uncertainty. The higher the level of noise, the less certain we are about the location of object in the hand.

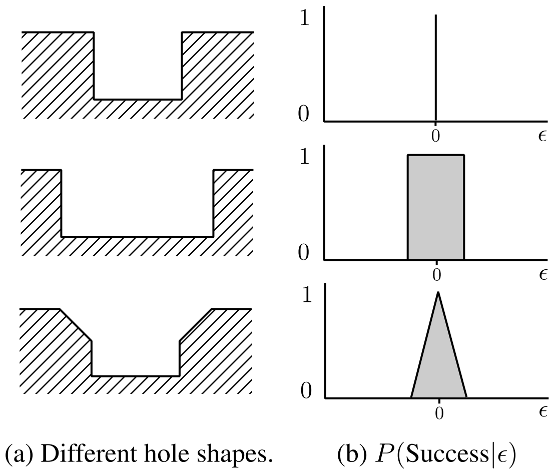


Figure 4.5: (a) Three different hole shapes. (b) Precision required in the estimation of the pose of the object for a successful insertion, induced by the three holes. $P(\text{Success}|\epsilon)$ is a model of the probability of successful insertion as a function of the error ϵ in that estimate. The top hole has the exact same size as the part, which requires perfect accuracy of object position, the middle hole is wider, and requires less accuracy. The corners of the bottom hole are chamfered, which also allows for some deviation where, for the purpose of this example, we assume a continuous degradation of the probability of success as the error increases.

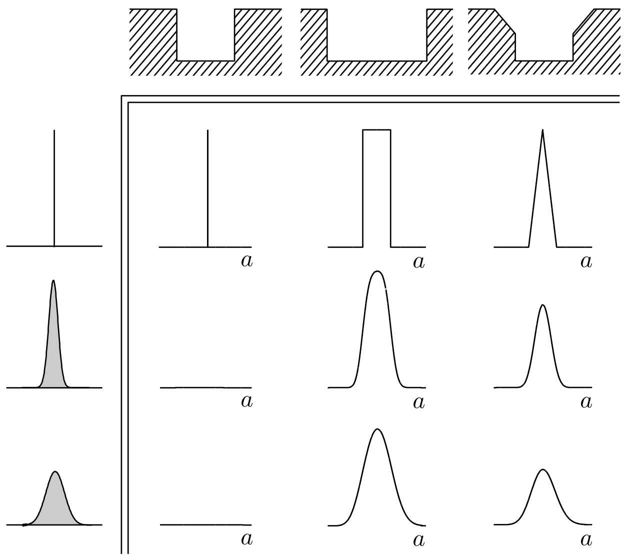


Figure 4.6: Probability of successful execution of the simple task in Figure 4.3 as a function of the chosen action a. The matrix shows the probability of success over choice of action for different combinations of noise models (rows) and hole shapes (columns). As expected, both the *sensing capabilities* of the hand, and the *task requirements* affect the predicted probability of success. Accurate sensors (top row) allow us to reliably execute difficult tasks, and simple tasks (central column) allow for noisier sensors.

reading will make it impossible to insert the part in the hole.

The example is a simple model to illustrate the principles. Real scenarios are more complex and assumptions are often violated, making it difficult to solve analyticalyl or through simulation. We leverage analytical models to define the structure of the statistical framework, but the actual models are learned directly from observed data. Our goal is for a robot to learn these probabilistic models on its own so it can predict the likelihood of success for a given post-grasp manipulation task.

4.2.1.2 Experimental Manipulation Tasks

We evaluate the proposed framework on three different real manipulation tasks: placing, dropping, and inserting an object. In all three cases the object is a highlighter marker, like the one in Figure 4.7a. The gripper used is prototype 3 of the MLab Hand [81, 102, 104], the simple gripper in Figure 4.7b. It has three fingers all compliantly connected to a single actuator. When the hand grasps an object, the fingers "fall where they may" around the object. Both the fingers and the actuator have encoders, and by looking at those sensor values, we estimate the pose of the grasped object. We use the encoders in the three fingers as the input to the learning system.

The placing task consists of picking a marker from a bin full of markers, and balancing it vertically on top of a platform. As illustrated in Figure 4.8, we use an intermediate step where we push the marker against the platform, to make sure that there is enough clearance to execute the placing action.

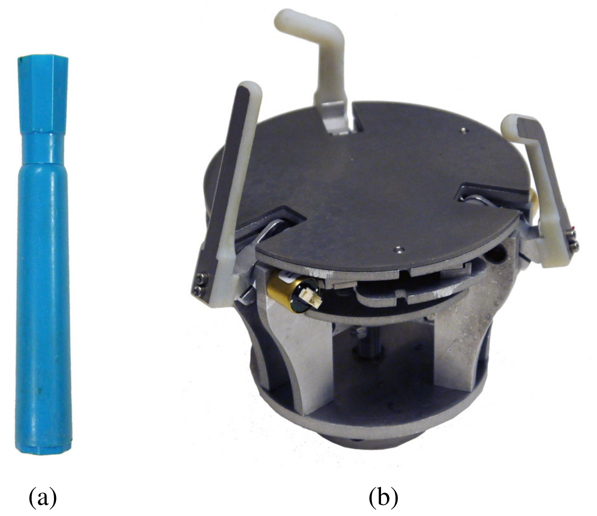


Figure 4.7: (a) A highlighter marker used as an object in the experiments. (b) Prototype 3 of the MLab Hand.

The second experimental task is to drop a marker inside a relatively large hole, as illustrated in Figure 4.9. We will see that the experiments corroborate the intuition that dropping is simpler than placing, in the sense that it can get away with a more noisy estimation of the pose of the marker.

Finally, insertion is a horizontal version of the peg-in-hole problem, where the hand tries to insert the grasped marker into a relatively small hole, as shown in Figure 4.10. Again, experiments will corroborate that insertion is a more demanding manipulation task than either placing or dropping.

Note that, in the three cases, the goal is to execute the task and accurately predict the probability of success using only feedback from in-hand sensors. For our experiments we assume that the highlighter marker always lies flat against the palm of the hand.

The configuration space of a cylindrical object lying flat on the palm is four dimensional: the two polar coordinates of the axis of the cylinder, the location of the cylinder along the axis, and the rotation with respect to that same axis.

However, we only consider the first two dimensions, since the last two dimensions are unobservable to the sensors, and not relevant for the execution of the three tasks. Note that for placing and insertion we included an intermediate pushing step to explicitly reduce the uncertainty of the cylinder along its axis (see Figure 4.8 and Figure 4.10). This reduction has computational and data-requirement benefits, as later discussed in Section 4.2.5.3, and is beneficial for visualization purposes and clarity of exposition.

The chosen state representation is then, that of a symmetrical cylinder in the plane as parametrized by the polar coordinates $x = (r, \theta)$ of its axis, as in Figure 4.11. That is $x \in X = \mathbb{R} \times \mathbb{SO}(2)$.

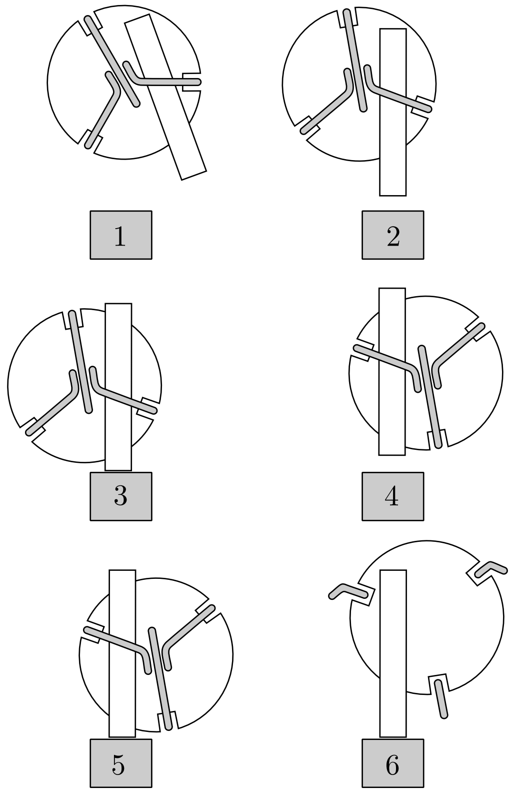


Figure 4.8: Diagram of the strategy used to place a highlighter marker. 1) The hand picks a marker out of the bin, and estimates its location. The rest of the strategy is open loop. 2) Reorientation of the hand so that the marker is perpendicular to and centered with the placing platform. 3) Push against the platform to center the location of the marker along its axis. 4) Rotation of 180 degrees and re-centering with respect to the platform. 5) Push again against the platform. 6) Release marker.

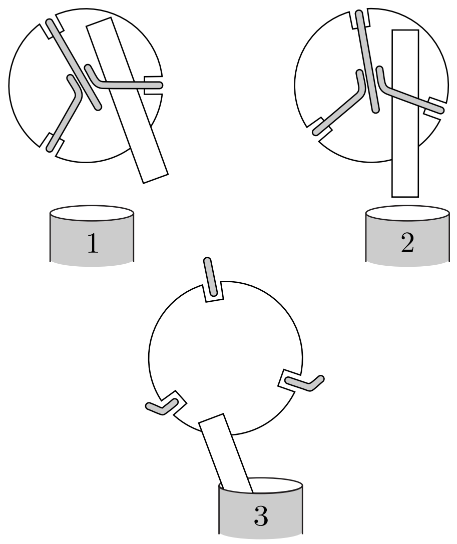


Figure 4.9: Diagram of the strategy used to drop a highlighter marker into a hole. 1) The hand picks a marker out of the bin, and estimates its location. The rest of the strategy is open loop. 2) Reorientation of the hand so that the marker is centered with the hole. 3) Release marker.

4.2.2 Statistical Framework

Given a manipulation task and a sensor observation $z \in Z$ of the state of the task, our goal is to find the action a from a set of available actions $\mathcal{A}$ that maximizes the expected performance of accomplishing the task. The following diagram illustrates three different strategies to approach the problem:

$$\begin{tikzcd}
 & & P(a|z) \\
\begin{matrix} Z \times \mathcal{A} \\ (z,a) \end{matrix} \arrow[rru, bend left] \arrow[r] \arrow[rd] & \begin{matrix} X \times \mathcal{A} \\ (x,a) \end{matrix} \arrow[r] & P(a|x) \\
 & \begin{matrix} \mathrm{Bel}(X) \times \mathcal{A} \\ (P(x|z),a) \end{matrix} \arrow[r] & P(a|P(x|z))
\end{tikzcd}$$

The first and most straightforward strategy is to model the performance of an action directly as a function of sensor observations. The decision on what action to execute and how likely it is to succeed is based upon the history of sensor readings. It makes the least assumptions about the system but also uses the least knowledge about the structure of the problem. It is also the most difficult to implement, since the complexity of the model depends strongly on the dimension of both the sensor and action space, which might be large.

The second strategy introduces an intermediate step where sensor inputs z are first projected into a more compact representation of state, noted here by x, and all information not captured by that representation is assumed to be irrelevant for planning optimal actions. In this work, we chose x to be the pose of the grasped object. The probability of success of an action is then modeled as a function of the most likely pose of the object x rather than the sensor observations

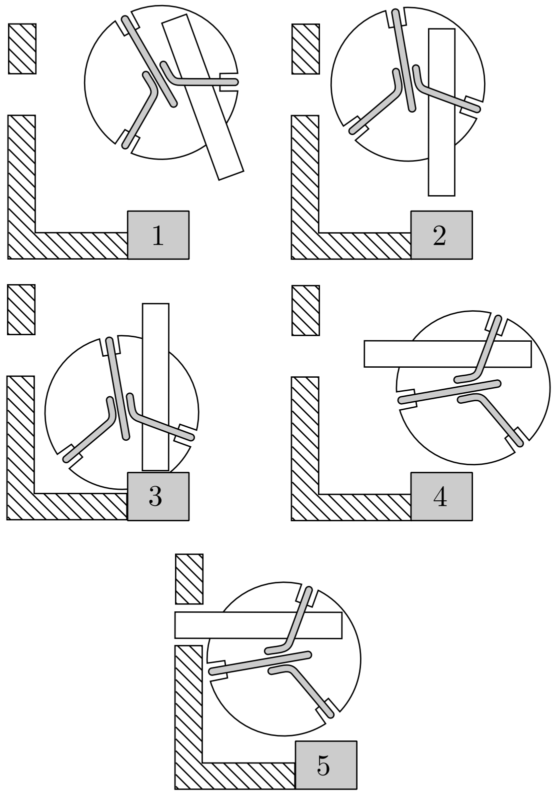


Figure 4.10: Diagram of the strategy used to insert a highlighter marker into a hole. 1) The hand picks a marker out of the bin, and estimates its location. The rest of the strategy is open loop. 2) Reorientation of the hand so that the marker is perpendicular to and centered with the placing platform. 3) Push against the platform to center the location of the marker along its axis. 4) Alignment of the axis of the marker with the axis of the hole. 5) Insert marker.

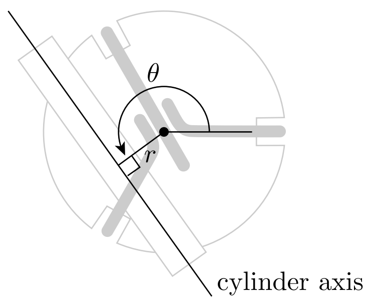


Figure 4.11: Parametrization of the pose of the cylindrical marker in the hand. We assume the marker is always flat against the palm, and parametrize its location by the polar coordinates (r, θ) of its axis. We will ignore the exact location of the marker along the axis, which is unobserved both by the parametrization and the sensors in the hand.

z directly. The intermediate representation x potentially reduces the model complexity, since the dimension of state space is generally smaller than that of sensor space. On the other hand, it introduces the possibility of information loss or lack of observability. It also fails to address uncertainty in the system induced by noisy sensors.

We implement a third approach, which encapsulates uncertainty by representing the system by its belief state $P(x|z)$ rather than just by its most likely value x. By explicitly considering uncertainty in the state of the task we can make a more informed and accurate prediction on the probability of success of a given action.

The dimension of the space of belief distributions $\mathrm{Bel}(X)$ is too large to model the probability of success of an action $P(a|z)$ directly as a function of the belief $P(x|z)$. We can alleviate this problem by marginalizing the probability of success of an action $P(a|z)$ with respect to the true state of the system x:

$$\begin{aligned} P(a|z) &= \int_X P(a|z,x) \cdot P(x|z) \mathrm{d}x \\ &= \int_X P(a|x) \cdot P(x|z) \mathrm{d}x \end{aligned} \tag{4.1}$$

where, in the last step, we make the assumption that the state representation x is informative enough that the output of an action is conditionally independent of sensor observations z, given the true state x.

This assumption enables the computation of the probability of success $P(a|z)$. Note, however, that for some tasks the pose of an object is not always fully representative of the grasp state. For example, in a compliantly actuated gripper, the state of the actuators also contains information on how stiff the grasp is, which is not captured by the pose of the object and might be relevant to determine the outcome of an action.

It is key to note that (4.1) divides the problem of modeling the performance of an action $P(a|z)$ into two simpler ones: modeling the distributions $P(x|z)$ and $P(a|x)$. Respectively, these represent the *sensing capabilities* of the gripper and the *task requirements* for a successful task execution. The following subsections detail the approach to model them, as well as the process to combine them to give an accurate estimate of $P(a|z)$.

4.2.2.1 Sensing Capabilities

The shape of the belief state $P(x|z)$ depends on several factors, including the geometries of the manipulator and object, the location and type of sensors, and the type of grasp. Assuming fixed geometries for the manipulator, object, and sensors, we will see that different grasps yield differently shaped beliefs.

We will pay special attention to the sharpness of the belief as an indicator of the confidence we get on the pose of the object. As illustrated in Figure 4.12 the choice of grasp has an important effect on that confidence. We will say that *some grasps are more informative than others*.

In this section, we describe the process to model $P(x|z)$ from experimental data. Learning $P(x|z)$ directly is usually data-intensive, since it can be arbitrarily shaped and the complexity of the model depends on the dimension of sensor space. To simplify the process, we use Bayes rule to flip the conditioning in $P(x|z)$ to $P(z|x)$, the likelihood or observation model of the system.

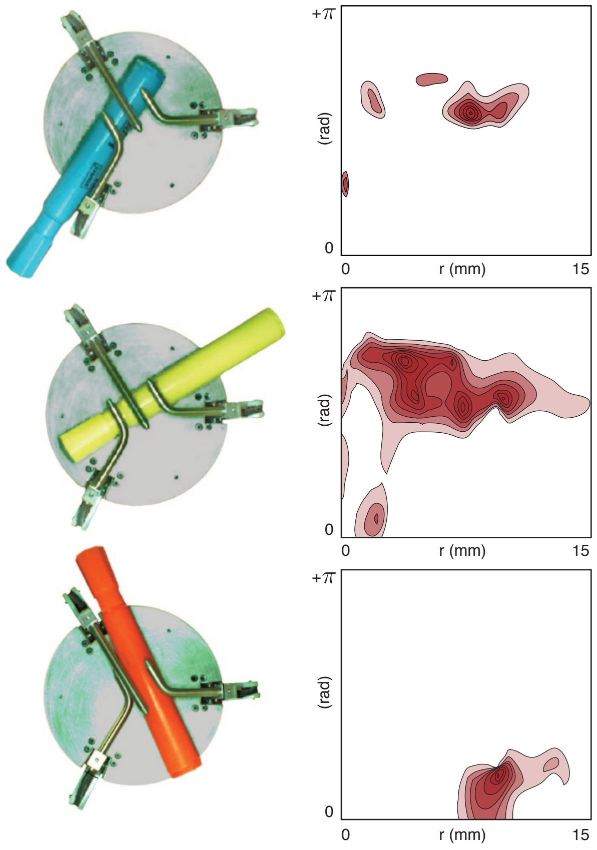


Figure 4.12: Three grasps of a highlighter marker, and the corresponding estimated beliefs of the pose of the object. Note that grasps where the object is localized or constrained by geometric features of the hand yield sharper beliefs. These tend to correspond to more stable grasps.

$P(z|x)$ is the distribution of sensor readings given the true state of the system. Unlike the posterior distribution $P(x|z)$, which can be arbitrarily complex due to possible lack of observability or sensor aliasing, the likelihood $P(z|x)$ tends to be simpler and we assume here to follow a Gaussian distribution $P(z|x) \sim \mathcal{N}(z; \mu(x), \sigma^2(x))$. In order to make the learning feasible, we also assume independence between sensors. This leads us to the following equation for the posterior distribution:

$$\begin{aligned} P(x|z) &= \frac{P(z|x)P(x)}{P(z)} \\ &\simeq \mathcal{N}(z; \mu(x), \sigma^2(x)) \cdot \frac{P(x)}{P(z)} \\ &\simeq \frac{P(x)}{P(z)} \cdot \prod_{k=1}^{L} \mathcal{N}(z_k; \mu_k(x), \sigma_k^2(x)) \\ &\sim P(x) \cdot \prod_{k=1}^{L} \mathcal{N}(z_k; \mu_k(x), \sigma_k^2(x)) \end{aligned} \tag{4.2}$$

where $P(x)$ is the state distribution prior to any sensor observations, both μ_k and σ_k are functions of the true state of the system x, and L is the number of sensor dimensions. Since $P(z)$ is independent of x, we omit it and normalize $P(x|z)$ a posteriori. Whenever we use the expression $\mathcal{N}(z; \mu(x), \sigma^2(x))$, we will refer to the decomposition induced by the independence between sensors assumed in (4.2).

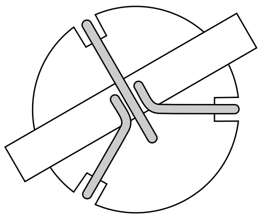
We now detail the process of estimating the prior distribution $P(x)$ and the observation model $P(z|x)$ from a collected dataset $C_1 = \{(z^i, x^i)\}_i$ of pose/sensor readings pairs. Figure 4.14 shows data of the 2000 grasps collected for C_1 (see the dataset in multimedia Extension 1).

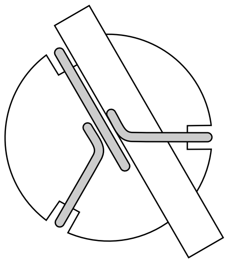
Learning the Prior Distribution $P(x)$ The prior distribution $P(x)$ is the distribution of the state of the system before considering sensor information. In our case, it is a reflection of the distribution of stable grasps yielded by the combined geometries of object and gripper. Figure 4.13 illustrates the three most stable configurations or grasp types for the hand and object used. The expectation is that the prior distribution $P(x)$ will cluster around those three grasp types.

We regress $P(x)$ by estimating the density of the pose of the object in state space. We use Kernel Density Estimation to model $P(x)$ as a sum of kernels:

$$P(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x^i}{h}\right) \tag{4.3}$$

where K is a Gaussian kernel, h is the bandwidth parameter and x^i are the state points in the dataset C_1. The bandwidth parameter is chosen automatically to minimize the mean integrated squared error following the algorithm in Botev et. al [11]. Figure 4.14 illustrates the learned prior distribution. As expected, it shows three clusters corresponding to grasp types I, II and III in Figure 4.13.


(a) Grasp type I


(b) Grasp type II

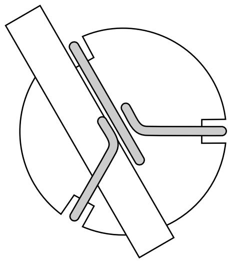
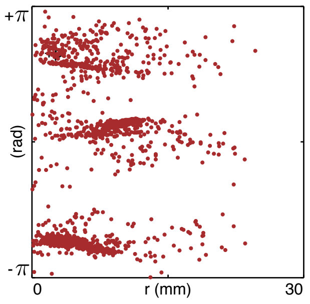
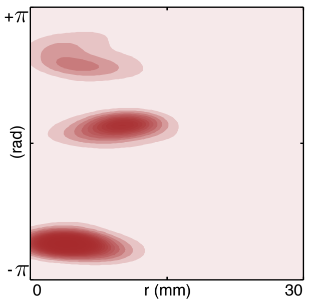

(c) Grasp type III

Figure 4.13: The three most stable configurations of the object/gripper pair used in our experiments.



(a)



(b)

Figure 4.14: (a) Distribution of grasp states in dataset C_1. Each dot corresponds to the final pose of the marker after an experimental grasp. (b) Corresponding prior distribution $P(x)$ estimated with Kernel Density Estimation. The three clusters in the distributions correspond to the three expected grasp types in Figure 4.13.

Learning the Observation Model $P(z|x)$ Equation (4.2) yields an approximation of the observation model and expresses it in terms of functions $\mu_k(x)$ and $\sigma_k(x)$:

$$P(z|x) \approx \prod_{k=1}^{L} \mathcal{N}(z_k; \mu_k(x), \sigma_k^2(x)) \tag{4.4}$$

We use Gaussian Processes (GP) [99] to regress functions $\mu_k(x)$ and $\sigma_k(x)$ for each sensor. For that we again use the dataset C_1. The process is detailed in the following steps:

1. Use a GP on half of the data points in C_1 to estimate the mean of the observation model $P(z|x)$, $\mu_k : X \longrightarrow \mathcal{Z}_k$. This implies training one independent GP for every sensory input, as a function of r and θ which computes the most likely sensor values for every possible state of the system x. Note that we will get a better estimate of the observation model for the regions of the state space that are most often observed, since those areas will be more populated with the collected data.
2. Complement the other half of the dataset C_1 with the sensor readings $\bar{z}^i = \mu(x^i)$ predicted by the learned observation model, and the squared error yielded by that prediction $\Delta^2 z^i = (z^i - \bar{z}^i)^2$, $C_1^+ = \{(z^i, x^i, \bar{z}^i, \Delta^2 z^i)\}_i$.
3. Use GPR on C_1^+ to regress the variance of the observation model $\sigma_k^2 : X \longrightarrow \Delta^2\mathcal{Z}_k$. Again, this implies training one independent GP for every sensor in the system.

By following these steps, we can now estimate $P(z|x)$ as in (4.4) (see Multimedia Extension 2 for example code). Figure 4.15b and Figure 4.15c illustrate the estimated observation model and posterior distribution $P(x|z)$ for the example grasp in Figure 4.15a.

4.2.2.2 Task Requirements

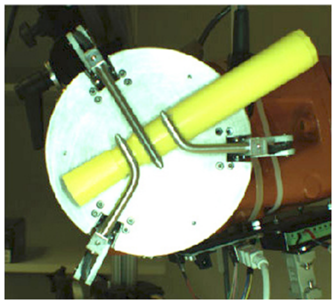
We now model the probability of success of an action, $P(a|x)$. This will tell us how accurate the estimation of the state of the grasp must be for an action to successfully execute the task.

While not required in general, we choose to state parameterize the set of actions. For example, for the task of placing a cylindrical object, we design an action a so that given the true state of the grasp first turns the cylinder so it is upright with respect to the ground, and then sets it down. We will note an action parameterized by state as a_p, where p indicates the state of the system for which the action was designed.

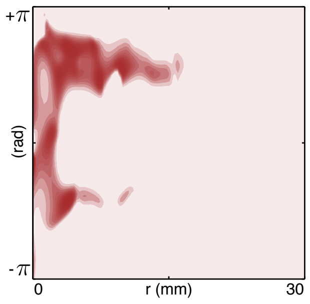
In general, the success of an action depends both on the specific action a_p itself and the true state of the grasp x. However, in order to reduce the complexity of the process, we will assume that, if the true state of the system is x, the probability of success of action a_p only depends on $(x - p)$, the difference between the real and assumed state of the grasp. For example, when placing a cylinder whose estimated axis is 1 degree off from its true state, we are more likely to succeed than if we try to place an object several degrees off.

We model the outcome of an action a_p as a Bernoulli random variable of parameter ϕ_{a_p} that depends on the true state x, so that:

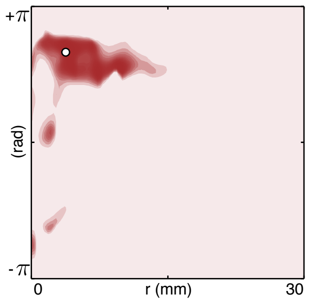
$$P(a_p = 1|x) = \phi_{a_p}(x) = \phi(x - p) \tag{4.5}$$



(a)



(b)



(c)

Figure 4.15: (a) Example grasp with its corresponding estimated (b) likelihood $P(z|x)$ and (c) posterior distribution $P(x|z)$. The dot corresponds to the most likely pose of the object.

The use of state parameterized actions also allows us to introduce controlled noise in the space of mismatches $\epsilon = (x - p)$. To learn $\phi(\epsilon)$, during each task execution, if the true state of the grasp is x, instead of choosing the action a_x, we execute action a_p with $p = x + \epsilon$, where ϵ is a uniformly distributed error in the space of system states.

We now detail the process of estimating the task requirements model $\phi(\epsilon)$ from a dataset $C_2 = \{(z^i, x^i, \epsilon^i, y^i)\}_i$, where ϵ^i is the error in system state and y^i is the success/failure output of the trial (see dataset in Multimedia Extension 3). For each task, we uniformly sample ϵ from $\mathcal{E} = [-\Delta r_{\text{max}}, \Delta r_{\text{max}}] \times [-\Delta\theta_{\text{max}}, \Delta\theta_{\text{max}}]$. We choose Δr_{max} and $\Delta\theta_{\text{max}}$ to be large enough to cover the range of errors we care about, and assume everything falling outside that range to be a failure.

To learn the model, we regress the Bernoulli parameter ϕ from dataset C_2 containing the outcome of more than 1000 executions for three manipulation tasks: placing, dropping, and insertion (see Multimedia Extension 4 for example code). The results are illustrated in Figure 4.16. As expected, when $|x - p|$ increases, the likelihood of task success decreases. Note that for the dropping task the probability decreases much slower than for placing or insertion. This indicates that dropping a marker into a hole is easier than balancing it on a platform or inserting into a small hole. The task tolerates more error.

The task requirements for insertion resemble the shape of an X. This can be explained by noticing that if we incorrectly try to insert the marker too high, but also tilt it downward, the end of the marker still manages to fit in the hole. Besides enabling accurate estimates of the probability of successful task execution, computing these task requirement distributions also give us insight in to what types of errors the task execution is robust against.

In general, the more data we use the more accurate the regressed distributions of task requirements are. The magnitude of the variance returned by the Gaussian Process Regression can be used to define a stopping criteria. In our case, we use the average standard deviation to assess how certain we are about the learned distribution. The bottom graphs in Figure 4.16 shows how the average standard deviation changes with the number of experiments for each task.

4.2.2.3 Matching Task Requirements with Sensing Capabilities

Here we combine the models of $P(x|z)$ and $P(a|x)$ to estimate the probability of success of an action a_p. For that, we extend (4.1) as:

$$\begin{aligned} &P(a_p = 1|z) = \\ &= \int_X P(a_p = 1|x)P(x|z)\mathrm{d}x \\ &= \int_X \phi(x-p)\mathcal{N}(z;\mu(x),\sigma^2(x))\frac{P(x)}{P(z)}\mathrm{d}x \\ &\simeq \int_{\mathcal{E}} \phi(\epsilon)\mathcal{N}(z;\mu(p+\epsilon),\sigma^2(p+\epsilon))\frac{P(p+\epsilon)}{P(z)}\mathrm{d}\epsilon \end{aligned} \tag{4.6}$$

where we apply the change of variables $\epsilon = x - p$, and $\mathcal{N}(z;\mu(x),\sigma^2(x))$ represents the decomposition in (4.2).

In the experiments we approximate the integral numerically (see Multimedia Extension 5 for example code). We grid the space of mismatches between real and estimated states into $N_r \times N_\theta$. Letting ϵ_{ij} be the corresponding error, we can approximate the integral in (4.6) as:

$$P(a_p = 1|z) \simeq \sum_{i=1}^{N_r}\sum_{j=1}^{N_\theta} \phi(\epsilon_{ij})\mathcal{N}(z;\mu(p+\epsilon_{ij}),\sigma^2(p+\epsilon_{ij}))\frac{P(p+\epsilon_{ij})}{P(z)}\Delta A \tag{4.7}$$

where ΔA is:

$$\Delta A = \frac{4\Delta r_{\max}\Delta\theta_{\max}}{(N_\theta - 1)(N_r - 1)}$$

Once we compute $P(a|z)$ we find its maximum in action space to choose the optimal action to execute. Depending on that maximum probability we can decide either to execute the task with the optimal action or to abort the execution. Figure 4.17 shows the complete process for an example grasp.

The presented framework decouples the learning of the sensing capabilities of the hand from the learning of the requirements of the task. For a given hand and object, we only need to compute its sensing capabilities once. For any given new task, we only have to compute the task requirements, and then follow the described procedure to estimate the overall probability.

Another possible scenario where the decoupling between both models is useful is in sharing models of task requirements between different robots. For example, an industrial robot could learn in a room for days at a time, and a mobile manipulator could reuse those learned models.

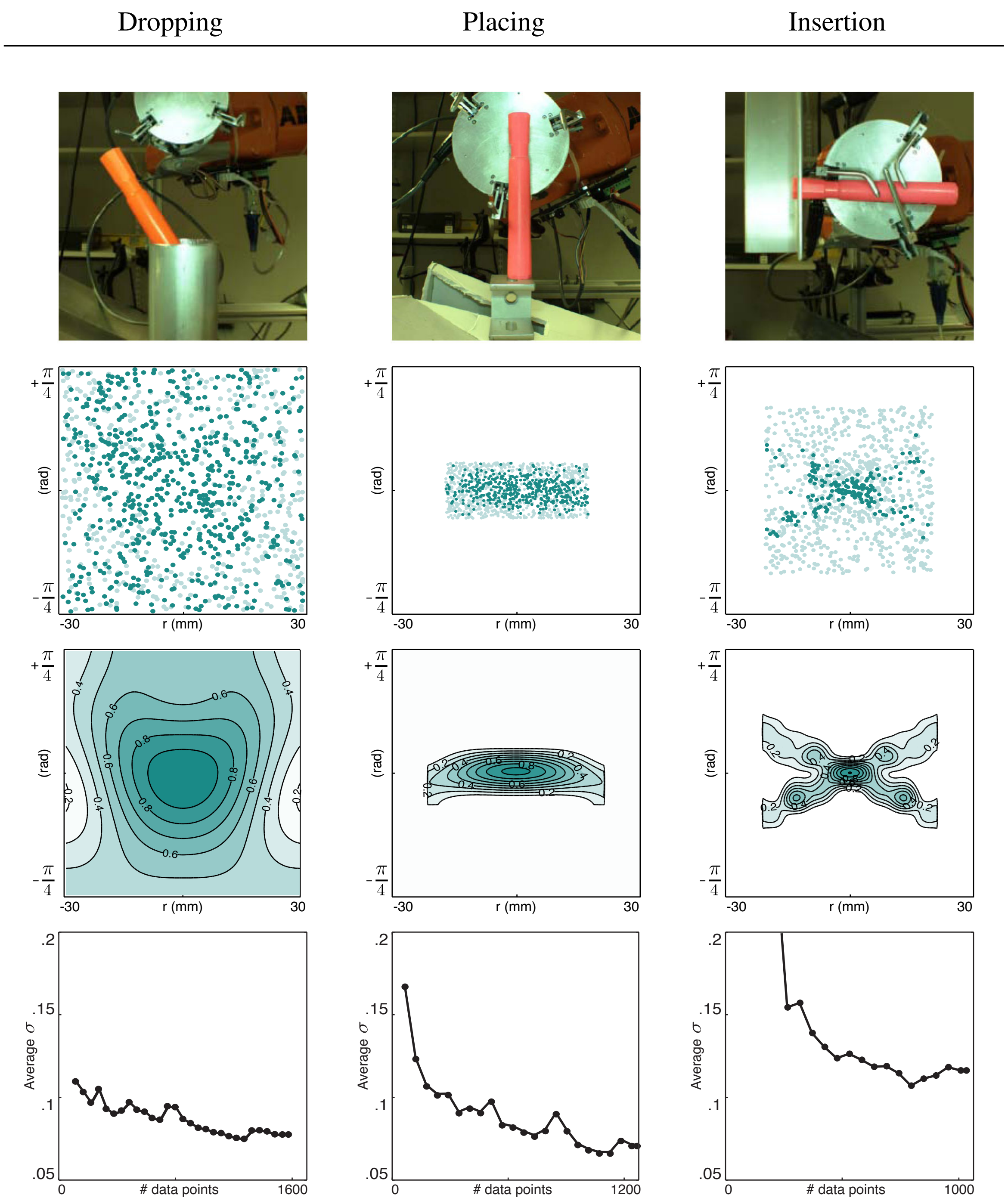


Figure 4.16: Learned task requirements for three manipulation tasks: dropping, placing and insertion. [2nd Row] Dataset C_2 of task execution with perturbed states. Dark points are successes and light ones are failures. Notice that the range of perturbations is different for each task. [3rd Row] Distribution of task requirements $P(a_p = 1|x)$ as a function of the error in state estimation. [4th Row] Average standard deviation of the regression of the Bernoulli parameter of $P(a_p = 1|x)$ obtained with a GP. This is used as a rough estimate of the convergence of the algorithm and stoping criteria.

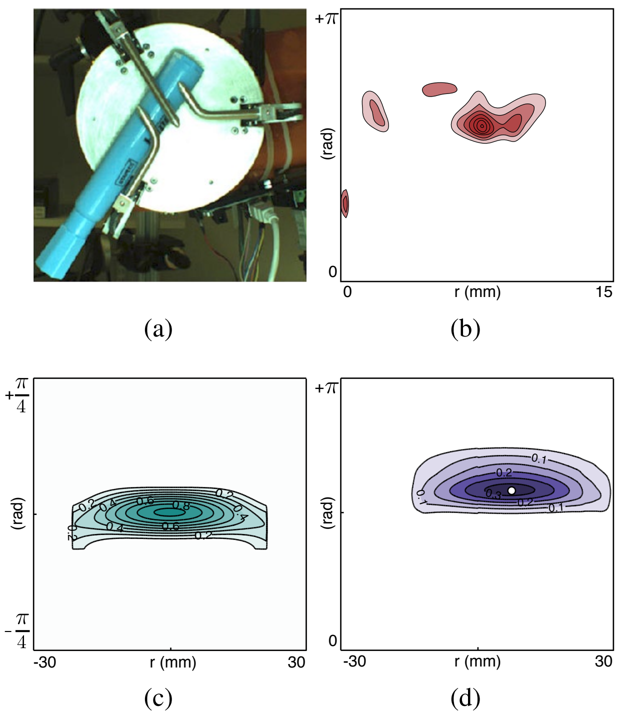


Figure 4.17: Complete process to compute the probability of success at placing a marker. (a) Example grasp. (b) Learned belief $P(x|z)$ of the pose of the object. (c) Learned task requirements, $P(a|x)$, for the placing task. (d) Estimated probability of success for the parametrized set of placing actions. The white dot corresponds to the optimal action to execute.

4.2.3 Experimental Validation

To validate the proposed framework, we use the hand and object in Figure 4.7 to complete three different manipulation tasks. This requires one training set for the *sensing capabilities* of the hand, $P(x|z)$, and three training sets to estimate the *task requirements*, $P(a|x)$, one for each of the tasks. After learning these functions, for any new grasp, we can predict the action most likely to successfully execute a task and its expected probability of success.

We are interested in evaluating the accuracy of the predicted probability of success. For that we execute each task 500 times according to the action most likely to succeed as predicted by the learned models. After each execution, we note down both the predicted probability of success and the actual outcome of the experiment (see data in Multimedia Extension 6).

To test the validity of the predictions, we group grasps by their predicted task success probability and compare it with their correspondent experimental success rate. For example, if we take all grasps that were predicted to succeed at an action around 60% of the time, the average experimental success rate for those grasps should ideally be around that same 60%.

Figure 4.18 compares the experimental success ratios with the predictions by the learned models. We see that for the three tasks, the experimental probability follows the predicted probability, supporting the validity of the framework.

Both the quality of the in-hand sensor feedback and the difficulty of the manipulation task determine the range of values of the probability of success. From Figure 4.18, we see that dropping is the easiest of the three tasks, followed by placing, and lastly insertion. This could have been predicted by looking at the task requirements and noting that dropping has the widest distribution of success in the presence of pose error. The plots in Figure 4.18 indicate that the proposed framework successfully predicts the probability of success of actions independently of the complexity of the task.

Predicting the probability of success of an action allows us to make an informed decision on what action to execute and improve the overall system performance. The bottom row of Figure 4.18 shows the precision-recall curves of the three tasks. They reveal that we can effectively increase the success rate in task execution if we decide to abort executions whose predicted probability of success are below a certain threshold. By changing that threshold, we can move along the precision-recall curve, and achieve a desired performance. Dropping increases from 80% to near 100% success, placing increases from 40% to 60%, and insertion increases from 50% to 60%.

4.2.4 Conclusion

We introduced a general statistical framework to model the problem of post-grasp manipulation. The framework is composed of the following three steps:

- **Off-line learning of sensing capabilities:** We learn a model to estimate the belief state of the grasp from in-hand sensor information. The training process creates two models: first, a prior distribution of the expected final grasp states, and second, an observation model of the hand/object pair that relates grasp states to expected sensor readings. Both models are constructed from experimental data, and the combination allow us to estimate the distribution of the state of the grasp $P(x|z)$ from sensor readings.

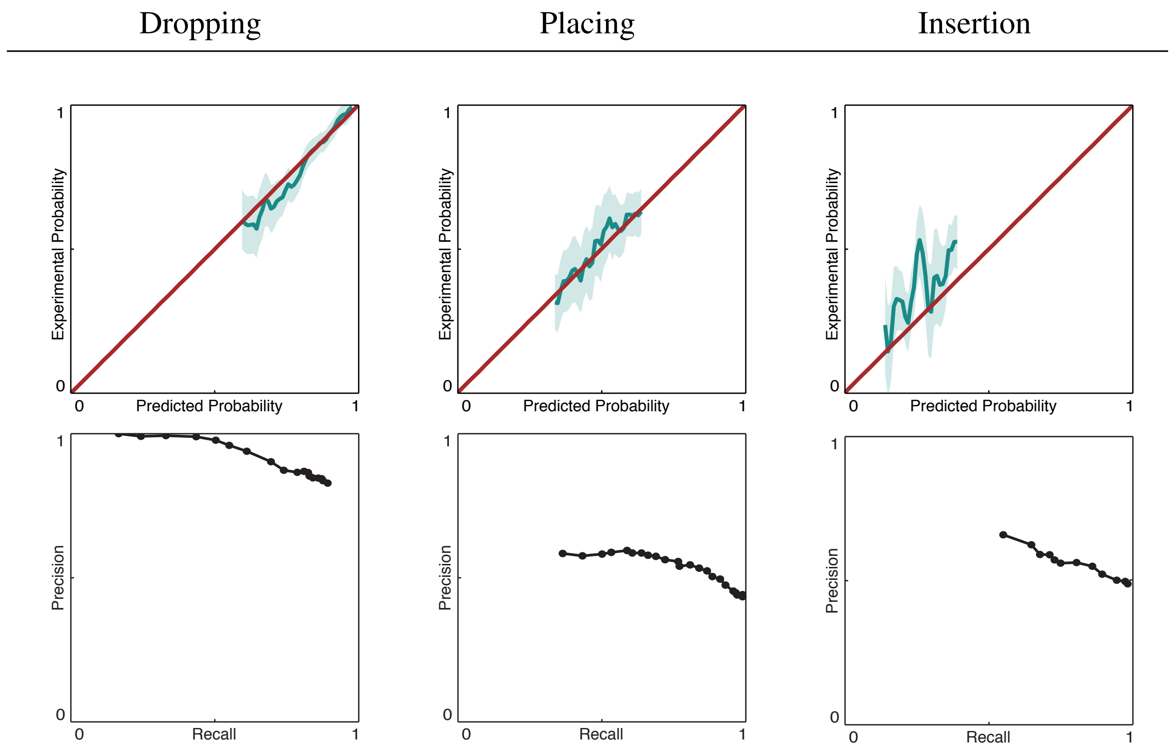


Figure 4.18: [Top] Comparison between the experimental and predicted probability of success. The shaded region is a 95% confidence interval of the estimation of the Bernoulli parameter, according to a binomial distribution. The plots show that the predictions follow the experimental observations quite well. [Bottom] Precision-recall curves of the success in task execution formed by only considering task executions whose predicted probabilities were above certain thresholds. The plot shows that we can increase the success rate in task execution by rejecting low probability grasps in the three tasks.

- **Off-line learning of task requirements:** We also learn a model of the object pose accuracy required to execute a specific post-grasp manipulation task. The model captures how the probability of successful task execution degrades as a function of object pose error. We train this model by systematically introducing controlled perturbations in the state of the grasp, and recording the relationship between the noise introduced and the success/failure outcome.
- **On-line estimation of the probability of success:** During execution time, we use the off-line learned models for the sensing capabilities of the hand and the task requirements to make accurate predictions of the probability of success. That prediction can be used, for example, to choose the optimal action to execute from a set of actions or to improve the overall performance of the system by deciding to abort runs when the predicted probability is too low.

We implemented the framework and tested it on three different post-grasp manipulation tasks: dropping, placing, and insertion. To validate the framework, the robot performed over 8000 real grasps, and over 1000 trials each of placing, dropping, and insertion.

4.2.5 Discussion

4.2.5.1 Insights

The proposed statistical framework is general, and is designed to be implemented on a real system. The unpredictability of the real world, especially when contact and physical interaction are involved, strengthens the case for a statistical framework.

The sensing capabilities of a hand/object pair help us understand which grasps are informative and which are not. This can inform both the design of hands, as well as the design of grasp policies, by being aware of what it means to be a good grasp from the point of view of grasp observability. Different hands and strategies can then be tested to see exactly which configurations give us the most certainty in object state, which would improve post-grasp manipulation tasks down the road.

Learning the accuracy requirements of a task by artificially adding noise is an excellent tool to identify weaknesses in task execution. We can discover, for example, that placing tends to fail when the object is in a certain pose, so adding a move to avoid that pose could improve the overall task success rate.

We can combine these two models, *sensing capabilities* and *task requirements*, to predict the probability of successful task execution. The separation of the two models allows us to mix and match different tasks and hands. If we reuse the same hand, for each new task, we only need to learn its task requirements. Conversely, if we are executing the same task with different hands, for each new hand, we only need to learn its sensing capabilities.

Estimating the probability of success is powerful, since it allows the robot to increase its overall task success rate by not taking unnecessary risks. We can, for example, find an optimal policy to minimize mean time to success in an abort and retry scheme, similar to [103]. In addition, since we estimate the probability of success for the complete set of actions, we could consider optimization with different cost functions, or in the presence of constraints.

4.2.5.2 Assumptions

While the proposed framework is general, we make a few simplifying assumptions to implement it on a real system. The goal of these assumptions is to reduce the total amount of data required to learn, with minimal sacrifices to functionality. Here is a summary of those assumptions:

- We assume that the probability of success of an action is conditionally independent of sensor readings given the true state of the system, $P(a|x, z) = P(a|x)$. This is reasonable when the chosen state is a good representation; however, if it is incomplete, the estimate of $P(a|x)$ may be suboptimal. For example, the location of the center of mass of the marker relative to the center of the hand seemed to have an effect on the probability of success of the dropping task. However, the state representation (r, θ) we choose does not capture it, and consequently it is not observable. This is treated as noise in the process, and although we are still able to give accurate predictions of the probability of dropping success, they could be better.
- We assume that the observation model $P(z|x)$ for an object/hand pair is unimodal and normally distributed. If the grasp is at a fixed and known state x, the distribution of readings z that we get from the sensors in the hand is induced by the sensor noise, which we assume here to be Gaussian. Note that the framework still holds without the Gaussian assumption. It is however a common convenience to reduce the amount of data needed to learn the observation model.
- We assume independence between in-hand sensors. This is generally not true. Still it is a common simplification to reduce the complexity of the distribution to learn. It effectively restricts the type of distributions we can learn, since instead of learning a distribution in the joint space of all sensors, we learn a single dimensional distribution for each sensor and multiply them.
- Finally, we assume that the set of actions designed to execute a task is state parameterized. This is often true, but again it is a convenience that allows us to reduce the dimension of the model of task requirements. The framework still holds for sets of actions not parametrized by state. However, to estimate $P(a|x)$, we would need to sample the product space of actions and states, rather than just the space of mismatches between action and state.

4.2.5.3 Scalability

As this work is based on data, we briefly discuss the scalability of this framework for varying dimensionality in sensor space, state space, and action space.

- **Sensor Dimension:** If L is the number of sensors in the hand, we learn $2L$ Gaussian Processes to compute the sensing capabilities of the hand, corresponding to the mean and variance of $P(z|x)$ for each sensor. Note that when we add additional sensors to our hand, while the number of GPs increases linearly, the amount of data used to train each GP is still the same, so it does not affect the accuracy of our estimates. If for example we remove a sensor, the belief is now much less peaked, because we would indeed know less about the state of the object. While the overall success rate may change based on the absence or addition of sensors, the *accuracy of predictions* should not change.

- **State Dimension:** Increasing the dimension of the state space on the other hand can have an effect on the accuracy. First, the sensing capabilities would now need to handle an additional dimension when computing the mean and variance of $P(z|x)$. It has been shown that the number of data points required to achieve the same accuracy for a GP grows exponentialy with dimension [49]. In the case of sensing capabilities we can do slightly better, as often the state is clustered in small areas of the space, or has an underlying lower dimensional representation. If the state is truly uniform, then data needs grow exponentially to maintain the same level of accuracy.

  In the case of task requirements, the data requirement also grows exponentially with dimension, since we are uniformly sampling errors in each dimension.
- **Action Dimension:** If we assume actions are state parameterized, then the action dimension increases with the state dimension, and all the statements above hold true. If instead, actions are not state parameterized, then increasing the dimension of the action space will again cause the data required to learn our task requirements to a given accuracy to grow exponentially. Note that increasing the action dimension will not affect sensing capabilities.
- **Computational Dependence on Dimension:** In addition to the data requirement, another factor that cannot be ignored is the computational complexity of calculating these probability functions. As the number of data points increases, the computation time for the Gaussian Processes used to compute the sensing capabilities and task requirements suffer as $O(n^3)$. Gaussian Processes are much more dependent on the number of data points compared with the dimension of each data point, so both sensor and state dimension do not have too much of an impact.

  Conversely, when trying to combine task requirements and sensing capabilities together, that integral is directly dependent on the state dimension. Currently, we are gridding up the state space finely in each dimension and numerically approximating the integral. This is exponential with dimension, and puts a large requirement on time and memory (if things are being precomputed). Given that we are computing the integrals of probability distributions, we should be able to lessen this requirement by using particle and monte-carlo based approaches. This will enable us to sample only in relevant regions and increase efficiency.

4.2.5.4 Future Directions

One area we would like to explore in the future is understanding the statistical significance of the distributions the robot learns. During experiments, we had no analytical means of knowing when we had *enough* data. While we were able to show that our probability estimates improved with more data, more analysis is needed to quantify how data density affects performance.

Carrying statistical significance through all of the distributions would enable us to compute a confidence bound on our final probability distribution in action space. This would expand the usefulness of our framework and help us understand how collecting more data affects our estimates. Another direction that we are interested in exploring is using active learning to selectively sample so as to reduce overall data requirements.

Finally, when the robot encounters a situation where the probability of success is not accept-

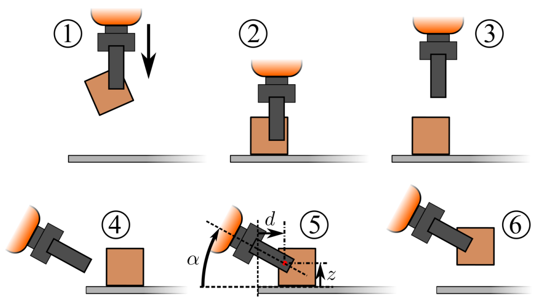


Figure 4.19: The place and pick regrasp action studied. Initially, the robot is holding a block between its fingers. Then, it moves downwards to a specific pose above the platform, where the block may move to conform to the new contact. It then opens its fingers and repositions itself at a certain position and orientation with respect to the edge of the platform. It then closes its fingers and moves upwards, completing the regrasp. There are several scenarios where the robot will not be holding the block in step 6. In step 1, the object could slip out of the hand. In step 2, the object-hand system could collide with the table. In step 5, the robot could miss the block and fail to pick it up.

able, the options to increase it are to either abort and retry, design better actions, or use better hardware. Instead we could consider a scenario where, based on the computed probability distributions, the robot decides to execute extra actions aimed at improving the expected probability of success rather than aimed directly at solving the task.

In particular, we are interested in developing robots with regrasping capabilities. Inserting a key into a lock is nearly impossible if you are holding it by its teeth rather than its head. Being able to regrasp objects, either to reduce state uncertainty or change configuration, would greatly increase the capabilities of current autonomous robotic systems.

4.3 Data-Driven Statistical Modeling of a Cube Regrasp

The work in this section was published in [90].

4.3.1 Introduction

Humans are experts at reorienting objects in their hands. They use this skill to adjust their grip of a pencil to write with it, or to change their grasp of a key from its teeth to its head to unlock a door. By contrast, once a robot has picked up an object, it generally maintains the same grasp as long as the object is in contact with the hand. If a robot does adjust an object grip, it is generally a predetermined operation with deterministic results, and only applicable for a constrained set of initial and desired final grasp poses. In contrast, humans can adapt to different objects with arbitrary initial and desired final poses. One possible explanation of this discrepancy is that humans have better models of how the object pose changes as a function of their actions. We show how robots can build better models of regrasp actions.

By a regrasp action, we mean any sequence of movements that results in a change of the

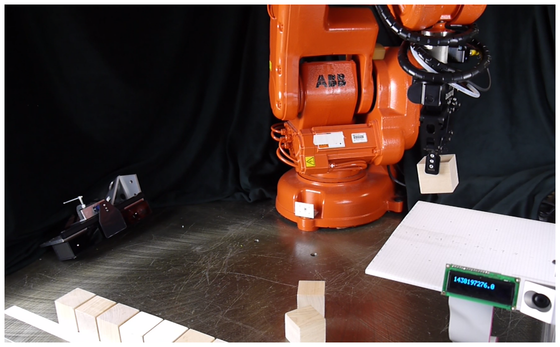


Figure 4.20: Place and pick regrasp data collection setup. We use an industrial arm with a parallel jaw gripper and place an pick objects. A three dimensional vision system based on point clouds is used to record the object position before and after a regrasp. In the event of a failure, the robot either picks up the dropped block from on top of the platform or retrieves a new block from a stack of fresh blocks. With this setup, the robot performed over 3000 regrasp experiments. We use this data to fit models for predicting the probability of not dropping the object, and estimating the final pose of the block after a regrasp.

object pose with respect to the hand. Often, the final pose of an object is critical to a task. If the initial pose of an object is arbitrary, then the robot must use models to determine what regrasp actions to take to move the object to the desired pose. Physics-based models can only take us so far. Modeling multiple contacts along with impact, friction, and uncertainty in object size, mass, finger shape, dirt, etc can make it difficult to compute models a priori. We encapsulate this real-world noise and uncertainty in the model as a probability distribution. A simple Gaussian is not a good model of the probability distribution for manipulation. Flipping a coin, or the difference in object pose based on whether or not contact occurs cannot be represented as a unimodal Gaussian distribution. We use kernel density estimation to estimate a multi-modal probability distribution from collected data to model regrasps.

We study a place and pick regrasp of a cube (Figure 4.19) as a first step to understand the challenges involved in statistical modeling of regrasp actions. We collected data from 3300 robot regrasps, and used this data to learn two models: 1) Given an initial object pose and regrasp action, how likely is it that the object remains grasped? and 2) Given an initial pose and action, where do we expect the object to end up? We show that our learned model, despite not having any prior knowledge of the task, achieves comparable accuracy to a physics simulator and a geometric model.

4.3.2 Method

4.3.2.1 Task Description

The place-and-pick regrasp action we will learn is shown in Figure 4.19. The robot moves down vertically to a fixed height above a platform, releases the object, and then attempts to grasp it again at a specified position in the workspace. Note that in step 2, the object pivots and slides in the fingertips when it comes into contact with the platform, which we expect to be difficult for physics-based models to capture. Our regrasp action $\boldsymbol{a}$ is parameterized by three continuous variables, d, z, and α, which represent the pose of the hand frame with respect to the edge of the platform. The parameters of action $\boldsymbol{a}$ are not relative to the object pose on the platform, because this data may be unavailable to the robot while it is performing a regrasp.

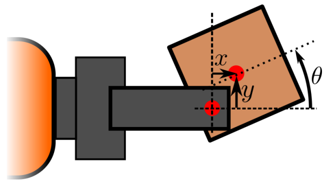


Figure 4.21: The state space used. While the world is six-dimensional, because we are grasping a cube with a parallel jaw gripper, we can reduce the state space to three dimensions. Note that as the cube is symmetric, we restrict θ to be between $-\pi/4$ and $\pi/4$.

Our state $\boldsymbol{s}$ is represented by three continuous variables, x, y, and θ, corresponding to the relative pose of the cube with respect to the hand, as shown in Figure 4.21. Note that this is a planar state space; we will not consider out of plane rotations or grasps. Any grasps of this kind will be considered "not grasped" for the purposes of this research.

In order for the robot to successfully model this regrasp action, we must learn two probability functions:

1. The probability that the object is in the robot's hand after a regrasp action, $P(\text{grasped}|\boldsymbol{s}, \boldsymbol{a})$
2. The probability distribution of the final state given that the object is still grasped, $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}, \text{grasped})$

Note that if the object is not grasped after a regrasp action, its final state $\boldsymbol{s}'$ does not exist, since $\boldsymbol{s}'$ is the in-hand pose of the object. Learning these two probability distributions enables us to solve planning problems such as: 1) what action maximizes the chance of maintaining a grasp? or 2) what action maximizes the chance of the center of the object being at most $1\,\text{cm}$ away from the center of the fingers? For this work, we focus solely on learning the above distributions from data, and leave planning with these models as future work.

4.3.2.2 Predicting the Probability of Maintaining a Grasp

To estimate the probability that the object is in the robot's hand after a regrasp action, we use kernel density estimation with Bayes discriminant rule [78, 105]. We estimate the probability of retaining and not retaining the object using kernel density estimation, and then, for a query point, determine which of the two probabilities are greater. That is, we would like to calculate:

$$P(\pi_i|\boldsymbol{s}, \boldsymbol{a}) = \frac{p_i P(\boldsymbol{s}, \boldsymbol{a}|\pi_i)}{\sum_j^g p_j P(\boldsymbol{s}, \boldsymbol{a}, \pi_j)}$$

where $p_i = P(\pi_i)$ is the prior probability of a randomly selected observation being in class π_i, g is the total number of classes, and $P(\boldsymbol{s}, \boldsymbol{a}|\pi_i)$ is the conditional probability density of an observation given that it is in class π_i. In our case, we have two classes, grasped and not grasped, so we will learn two probability densities using kernel density estimation:

$$P(\boldsymbol{s},\boldsymbol{a}|\pi_i) = \frac{1}{N_i}\sum_{j}^{N_i} K^s_{h_1}(\boldsymbol{s},\boldsymbol{s}_j)K^a_{h_2}(\boldsymbol{a},\boldsymbol{a}_j)$$

where N_i is the number of training observations belonging to class π_i. Note that we set $p_i = N_i/\sum_j^g N_j$ as our prior probabilities.

We define our kernel functions $K^s_{h_1}$ and $K^a_{h_2}$ by first expressing distances in state and action space, and then use a Gaussian kernel over these distance functions:

$$\begin{aligned}
D_a(\boldsymbol{a}_1,\boldsymbol{a}_2) &= \left\| \begin{bmatrix} d_1 \\ z_1 \\ \rho\alpha_1 \end{bmatrix} - \begin{bmatrix} d_2 \\ z_2 \\ \rho\alpha_2 \end{bmatrix} \right\|^2 \\
D_s(\boldsymbol{s}_1,\boldsymbol{s}_2) &= \left\| \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right\|^2 + 2\rho^2(1-\cos(\theta_1-\theta_2)) \\
K^s_{h_1}(\boldsymbol{s},\boldsymbol{s}_j) &= \frac{1}{\eta_s(\rho,h_1)}\exp\left(-\frac{1}{2h_1^2}D_s(\boldsymbol{s},\boldsymbol{s}_j)\right) \\
K^a_{h_2}(\boldsymbol{a},\boldsymbol{a}_j) &= \frac{1}{\eta_a(\rho,h_2)}\exp\left(-\frac{1}{2h_2^2}D_a(\boldsymbol{a},\boldsymbol{a}_j)\right)
\end{aligned}$$

where ρ is the radius of gyration for the object, which allows us to properly trade off distance and angle, while η_s and η_a normalize the kernels so they represent probability distributions. Note that for the state distance function, we use a cosine function to handle angle wrap-around for object pose (i.e. $-\pi = \pi$). Both of the distance functions D_a and D_s represent squared distance in action and state space. The units for d, z, x, and y are mm, while α and θ are in radians. Thus, our distance functions have units of mm^2, and our bandwidths h_1 and h_2 have units of mm.

We choose the values of bandwidths h_1 and h_2 that minimize the cross-validated negative log likelihood of the observed data:

$$\begin{aligned}
NLL(h_1,h_2) &= -\frac{1}{N_i}\sum_{j}^{N_i}\hat{P}_{-j}(\boldsymbol{s}_j,\boldsymbol{a}_j|\pi_i) \\
\hat{P}_{-j}(\boldsymbol{s},\boldsymbol{a}|\pi_i) &= \sum_{k\neq j}^{N_i} K^s_{h_1}(\boldsymbol{s},\boldsymbol{s}_k)K^a_{h_2}(\boldsymbol{a},\boldsymbol{a}_k)
\end{aligned}$$

where $\hat{P}_{-j}(\boldsymbol{s},\boldsymbol{a}|\pi_i)$ is the estimator of the conditional probability density with observation j removed.

4.3.2.3 Predicting the Final Object Pose

To predict the resulting probability distribution of the cube after a regrasp action, we will use kernel *conditional* density estimation. We formulate our conditional density estimate using our kernels from above and roughly following Hall, Racine and Li[50]:

$$\begin{aligned}
P(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a},\text{gra}) &= \frac{P(\boldsymbol{s}',\boldsymbol{s},\boldsymbol{a}|\text{grasped})}{P(\boldsymbol{s},\boldsymbol{a}|\text{grasped})}\\
P(\boldsymbol{s}',\boldsymbol{s},\boldsymbol{a}|\text{gra}) &= \frac{1}{m}\sum_i^m K^s_{h_3}(\boldsymbol{s}',\boldsymbol{s}'_i)K^s_{h_4}(\boldsymbol{s},\boldsymbol{s}_i)K^a_{h_5}(\boldsymbol{a},\boldsymbol{a}_i)\\
P(\boldsymbol{s},\boldsymbol{a}|\text{gra}) &= \frac{1}{m}\sum_i^m K^s_{h_4}(\boldsymbol{s},\boldsymbol{s}_i)K^a_{h_5}(\boldsymbol{a},\boldsymbol{a}_i)
\end{aligned}$$

where m is the number of experiments where the object was in the robot's hand after a regrasp.

We will choose values for h_3, h_4 and h_5 that minimize the integrated squared error, again using cross validation (see [50] for more details):

$$ISE(h_3,h_4,h_5) = \int \left(\hat{P}_{ssa} - P_{ssa}\right)^2 P_{sa} d\boldsymbol{s}\, d\boldsymbol{a} d\boldsymbol{s}'$$

where

$$\begin{aligned}
P_{ssa} &= P(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a},\text{grasped})\\
P_{sa} &= P(\boldsymbol{s},\boldsymbol{a}|\text{grasped})
\end{aligned}$$

Note that for both $P(\text{grasped}|\boldsymbol{s},\boldsymbol{a})$ and $P(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a},\text{grasped})$, we could have chosen more complex kernels or used different learning algorithms. However, for this work, we select simple models to understand the viability of a data-driven framework for modeling regrasps. In our future work, we plan to evaluate different non-parametric methods for estimating these probability functions.

4.3.3 Validation

We collect a large data set D, explained in Section 3.6. We now compare our learned model with an off-the-shelf simulator and a rudimentary geometric model using our data set D. We randomly select a hold-out test set of 1000 data points which we use to compare all three methods. We describe the geometric and simulation models below.

4.3.3.1 Geometric Model

The challenging part of this regrasp to model is what happens during the initial block placement (Step 2 of Figure 4.19), as there are many possible contact modes including no contact followed by an impact and settling, sliding or pivoting in finger tips, and sliding or rotating against the platform. For simplicity, we assume that during this step, once an object corner contacts the platform, the object rotates about the contact point until it lies flat on the platform. Given an initial object pose $s = [x, y, \theta]$, if c is the distance from the center of the hand to the edge of the platform when placing and w is the width of the block, we can calculate the distance from the edge of the platform to the center of the block q as

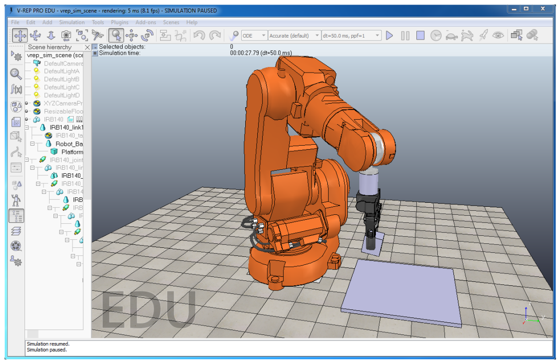
$$q = \begin{cases} \theta \geq 0, & c + y + \frac{w}{2}(\sin(\theta) - \cos(\theta) + 1) \\ \theta < 0, & c + y + \frac{w}{2}(\sin(\theta) + \cos(\theta) - 1) \end{cases}.$$

Now, given an action $a = [d, z, \alpha]$, if g is the maximum horizontal distance away from the center of the block that the robot can still grasp the object without missing it, then we will successfully grasp the block if $|d - q| \leq g$, and that final pose will be

$$\boldsymbol{s}' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} (q-d)\cos(\alpha) + (z - w/2)\sin(\alpha) \\ (q-d)\sin(\alpha) - (z - w/2)\cos(\alpha) \\ \gamma \end{bmatrix}$$

$$\text{with} \quad \gamma = \begin{cases} 0 \leq \alpha \leq \pi/4, & \alpha \\ \pi/4 \leq \alpha \leq \pi/2, & (\alpha - \pi/2) \end{cases}.$$

Creating probability distributions from this kinematic model is difficult, as we do not know the distribution of errors on our parameters. Note that even if we did, even for this rudimentary model, the probability distributions would be multimodal and non-Gaussian.



V-REP PRO EDU - vrep_sim_scene - rendering: 5 ms (8.1 fps) - SIMULATION PAUSED
Scene hierarchy
Selected objects:
Simulation time:
EDU

Simulation resumed.
Simulation paused.

Figure 4.22: The simulation environment used . Using VREP, we have modeled the same industrial robot arm and gripper used in our physical experiments, and placed the platform in the same relative location. We place the block in the simulated robot's hand in the same initial pose as our real trials, and record whether or not the object is still grasped after the regrasp in simulation. If so, we record the final pose of the block.

4.3.3.2 VREP with ODE as a Simulation Model

Using the simulation environment VREP [39], we have modeled an ABB IRB 140 robot with a Robotiq C-85 two-fingered gripper just as in our physical experimental setup. The simulation environment is shown in Figure 4.22. The platform is placed in the same location, and we use a $50\,\mathrm{mm}$ cube with the same density and frictional properties as our real wooden cube. We can now place the object into the simulated robot's hand at a given initial state $\boldsymbol{s}$, ask the robot to perform the regrasp action $\boldsymbol{a}$, and then observe whether the object was grasped. If so, we record the final state $\boldsymbol{s}'$.

Note that setting up the simulator was a challenge in and of itself. Even the well-tuned ODE in VREP still cannot handle parallel grasping well, and once the block is also made to slide against the table and in the hand, it is difficult to get stable results. The two most difficult phenomena to model in simulation with physics are 1) how the contact patch between the parallel jaws and the cube changes as the hand slightly loosens its grip on the object, and 2) what happens to the cube at the onset of contact with the platform.

Like the geometric model, creating probability distributions using a simulator is difficult as the simulation is deterministic. We could vary initial parameters slightly and fit the resulting data using the probabilistic model described in this work. However, it is unclear which parameters to vary and how much to perturb them by in order to get plausible results. Moreover, if the underlying physics model is wrong, even this may not give a realistic distribution.

Table 4.1: Classification Accuracy of Predicting whether the Object Remains Grasped

Setting	Geometric	Simulation	Data-Driven
In Hand	74.8 %	72.8 %	76.2 %
Platform	89.3 %	-	90.7 %

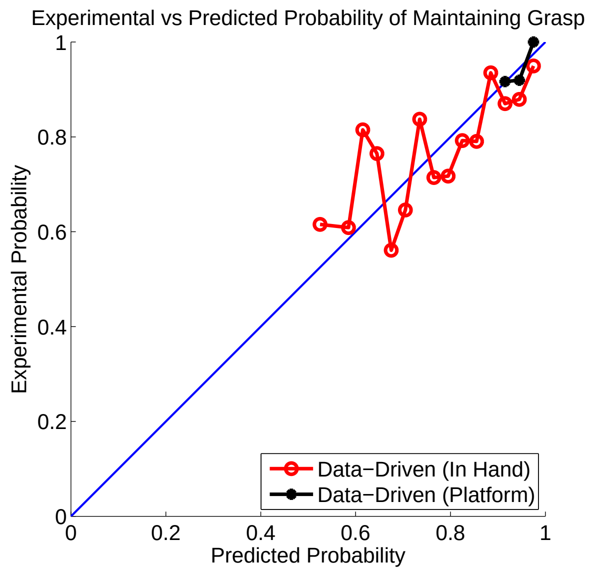


Figure 4.23: Comparison between the experimental and predicted probability of maintaining the object after a regrasp. Each data point is the average experimental grasp probability for 5%-wide bins of predicted probability.

4.3.3.3 Validation Results: Predicting if the Object Remains Grasped

Table 4.1 shows our results for predicting if the object is still grasped after a regrasp action. We compared the classification accuracy of the three models for two separate conditions. First, we

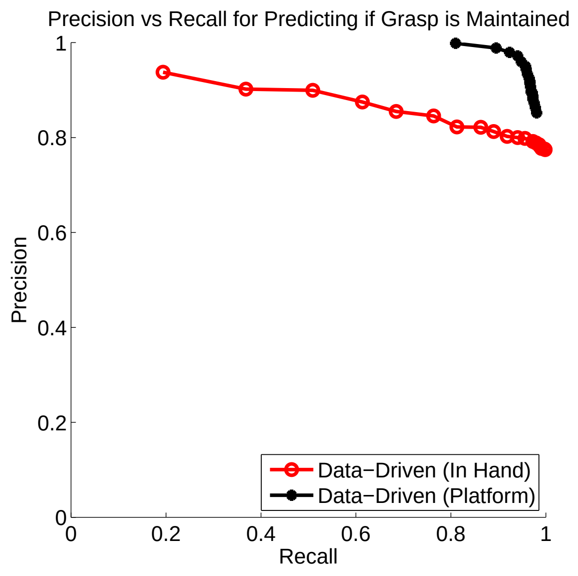


Figure 4.24: Precision-recall curve for predicting whether the robot is still holding the object after a regrasp

consider the condition where we are given the pose of the object in the robot's hand and the parameterized regrasp action to perform. Second, we consider the condition where we know the pose of the object on the platform and predict the probability that the robot will be able to successfully pick it up. All three models perform comparably, even though our data-driven model is given no prior information about the task.

In Figure 4.23, we binned our predicted probability in 5% increments, looked at the percentage of those points where the object was still grasped, and plotted the results. If our predictions are good, the mean of the true grasp probability should follow the straight line. Our predicted probabilities for the platform condition match better than the in-hand condition, which is expected.

If we can predict the probability of maintaining the object after a regrasp, this means we can adjust the decision boundary to achieve different precision and recall values. This is plotted in Figure 4.24. Note that the platform case gives us a much better precision-recall curve, and that these precision-recall curves are not easily achievable without a data-driven model.

Table 4.2: Mean Pose Estimation Accuracy (mm)

Setting	Geometric	Simulation	Data-Driven
In Hand	11.7	10.8	13.0
Platform	5.7	-	6.3

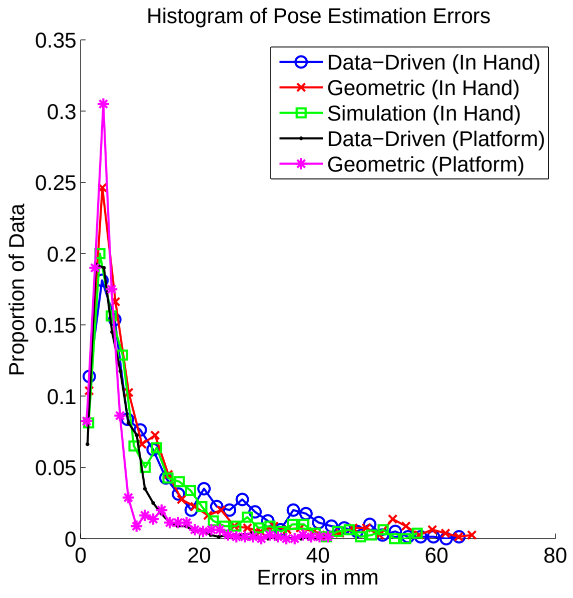


Figure 4.25: Histogram of prediction errors. Note that most of the error is less than 10mm, and the data-driven approach achieves comparable accuracy to the other approaches.

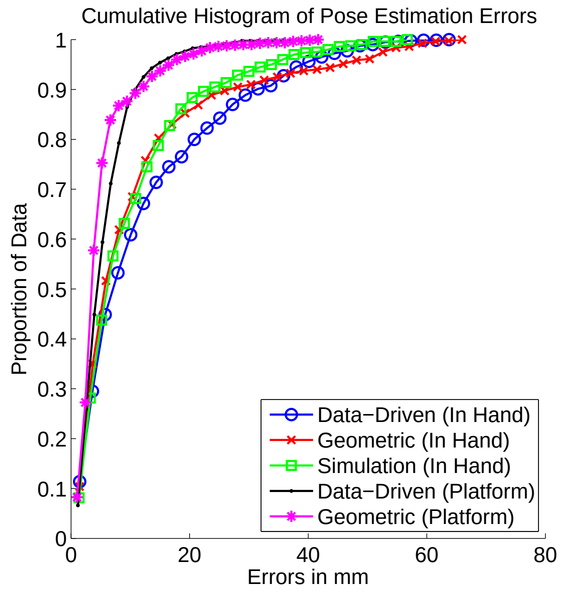


Figure 4.26: Cumulative histogram of prediction errors. Over 80% of the data has an error of less than 10mm for the platform case. The data-driven method achieves comparable accuracy to the other approaches.

4.3.3.4 Validation Results: Final Pose Estimation

To evaluate the predictive power of our pose estimation models, we looked at the mean pose estimation accuracy. We used the square root of our distance function $D_s(\boldsymbol{s}_1, \boldsymbol{s}_2)$ as a measure

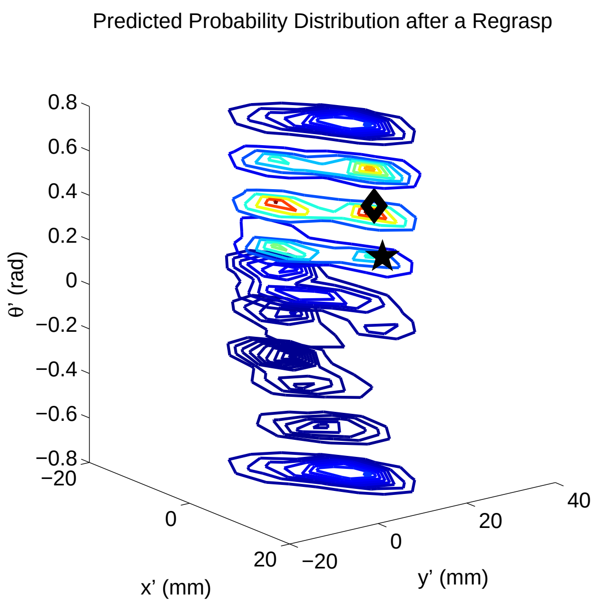


Figure 4.27: Predicted probability distribution of a final pose after a regrasp. The most likely predicted pose is the diamond and the true pose is the star. Note the multi-modal nature of the distribution.

of accuracy. Note that if the distribution is multi-modal, this measure does not reward capturing that multi-modality. However, since we do not have the true underlying distribution, we use the mean pose estimation accuracy as a baseline. Our results are shown in Table 4.2, Figure 4.25 and Figure 4.26. Again, our data-driven model achieves comparable accuracy with no prior information.

With our data-driven model, we can also calculate the entire resulting probability distribution in pose space, which is shown in Figure 4.27. Note the multi-modal nature of the distribution.

4.3.4 Conclusions

In this work, we introduced a way to model robotic regrasping using a large amount of real data. First, we briefly discussed how we collected the real robot manipulation data needed for our models. We then showed how to predict the probability of maintaining the grasp of an object given an initial position and robot regrasp action using this data. In addition, we showed how to estimate the probability distribution of where the object will end up in the robot's hand given an initial pose and a robotic regrasp action. We compared our models with a simulator and a rudimentary physics model and showed that our data-driven models have comparable performance even with no prior knowledge of the task.

In the future, we are interested in extending these models to other objects, regrasp actions, and hands. We are especially interested in extending our models to SE(3) space to handle three dimensional rigid body transformations. We are also interested in improving the accuracy of our vision system to improve model accuracy. Finally, we are interested in exploring other non-parametric models to achieve higher fidelity.

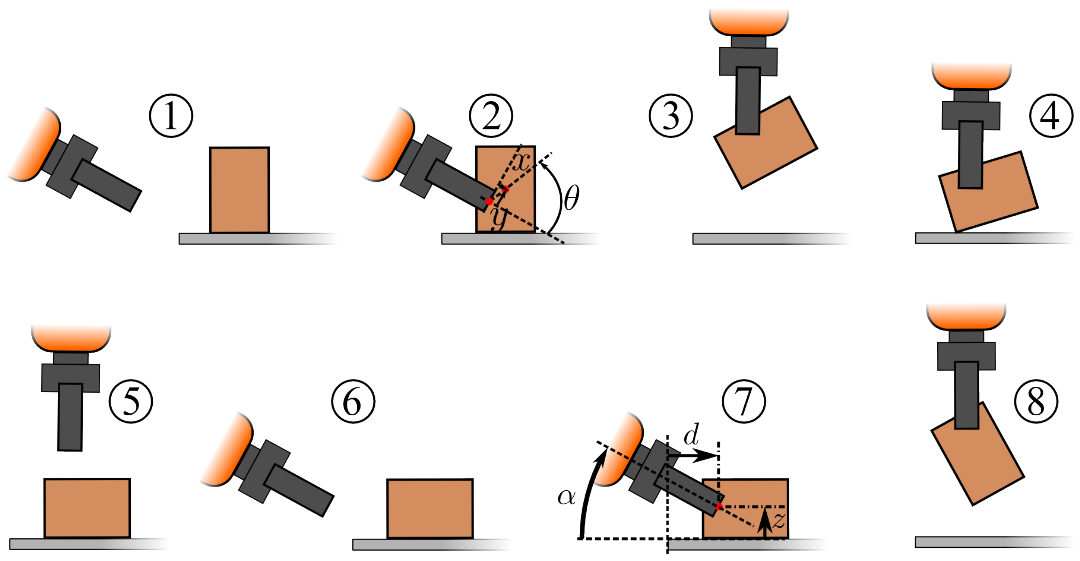


Figure 4.28: Data collection procedure for place and pick regrasp studied. A desired object pose relative to the hand is chosen, and the rectangular prism is placed in the correct orientation on the table. Then, the robot moves to the desired pose and attempts to grasp the object. If the object is grasped, its pose is recorded and then the robot attempts to place the object. The object may end up in a vertical or horizontal position on the table after the placement, or it may collide or fall off the table. The robot then moves to a desired pose relative to the the edge of the platform and attempts to grasp the object. If successful, the resulting object pose is recorded again. The block is then gently placed on the table for the next experiment to begin (not shown).

4.4 Improved Place and Pick Models

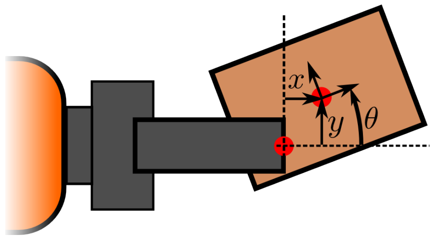
We now present an improved approach to statistically modeling manipulation actions, specifically placing and picking an object on a platform. While in the previous section we attempted to model an entire place and pick action with one model, in this section we break the model into smaller pieces, and combine them into the full model when necessary.

We will first show how to accurately model the probability of successfully picking, placing, and predicting what side an object will land on when being placed. We will then present a slightly modified kernel conditional density estimation technique which improves speed and robustness for predicting the transition model of placing and picking and object.

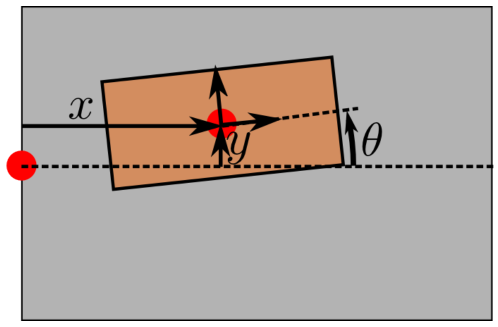
4.4.1 Task Description

The robot performs the set of actions shown in Figure 4.28. Before the experiment, a desired initial object pose and picking action are chosen. The desired initial pose is specified according to Figure 4.29. As the object is a rectangle, it has 2-fold symmetry, and we restrict θ to be in the interval $[-\pi/2, \pi/2]$. The picking action is parameterized by d, z, and α, which parametrize the pose of the hand with respect to the platform, similar to the previous section.

To begin the experiment, the object is placed on the platform either horizontally or vertically,



(a)



(b)

Figure 4.29: (a) In hand state space of rectangular prism. $\theta = 0$ corresponds to the long side of the block perpendicular to the fingers. (b) Platform state space of object, using an eagle eye view of the platform. $\theta = 0$ corresponds to the long side of the block aligned with the long side of the platform.

so that the desired initial grasp pose $\boldsymbol{s}_d$ can be achieved. The robot moves in horizontally, grasps the object, and then moves up and to a vertical position. The in-hand pose $\boldsymbol{s}_h^i$ is recorded. The robot then moves down to a specified height, opens its gripper, and moves out of the way. The object's platform class C_p (vertical or horizontal) and pose $\boldsymbol{s}_p$ are recorded. The robot then moves to the desired picking action $action$ and attempts to grasp the object. The final in-hand pose of the object $\boldsymbol{s}_h^f$ is also recorded. In the event of a failure at any placing or picking step (collision, dropped/missed object), the experiment is aborted and the next experiment starts. We record how much of each experiment was completed as a number S. Thus, the following data set is collected:

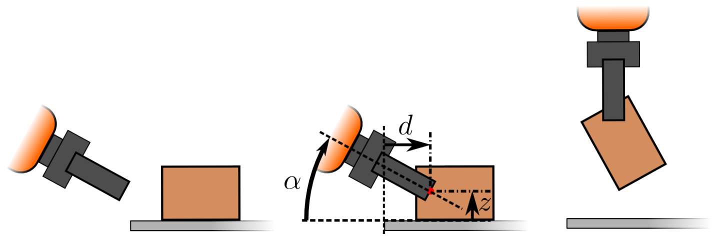
$$D_k = [\boldsymbol{s}_d, \boldsymbol{s}_h^i, C_p, \boldsymbol{s}_p, \boldsymbol{a}, \boldsymbol{s}_h^f, S]_k$$

Using this data, we will learn the following models:

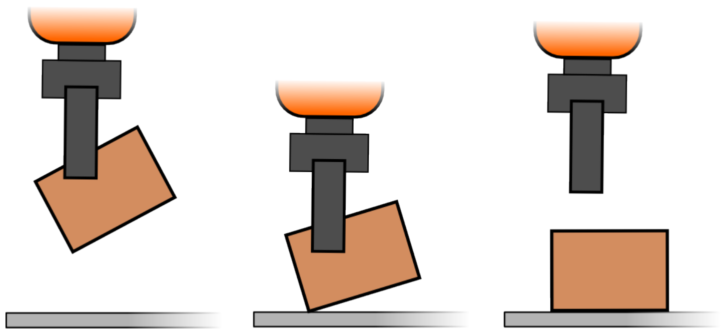
1. $P(\text{placed}|\boldsymbol{s}_h)$, the probability of placing a block given an in hand pose
2. $P(C_p|\boldsymbol{s}_h, \text{placed})$, the probability of a platform class given a successful placement and an in hand pose
3. $P(\boldsymbol{s}_p|\boldsymbol{s}_h, \text{placed}, C_p)$, when placing an object, the probability distribution of an object pose on the platform given an initial pose, the platform class, and that the placement was successful
4. $P(\text{grasped}|\boldsymbol{s}_p, C_p, \boldsymbol{a})$, the probability of grasping a block given a platform class and pose, and a picking action
5. $P(\boldsymbol{s}_h|\boldsymbol{s}_p, C_p, \boldsymbol{a}, \text{grasped})$, when picking a block, the probability distribution of the resulting in-hand pose given a platform class and pose, and a picking action, and that the object was successful

With these models, we can forward simulate the following two actions, shown in Figure 4.30, which can then be used for planning:

1. **Picking:** Given an object pose on a platform and a picking action, did we grasp it, and if



(a)



(b)

Figure 4.30: (a) The picking action we would like to model. The robot moves in from the left to a SE(2) pose relative to the edge of the platform, closes its fingers, and moves upwards. This action has three parameters. (b) The placing action we would like to model. The robot moves to a set pose, opens its fingers, and retracts. Note that this action has no parameters, because the robot always moves to the same place, regardless of where the object is.

so, where is it? $P(\text{grasped}, \boldsymbol{s}_h | \boldsymbol{s}_p, C_p, \boldsymbol{a})$

2. **Placing:** Given an in-hand object pose, do we successfully place the object, and if so, where is it? $P(\text{placed}, C_p, \boldsymbol{s}_p | \boldsymbol{s}_h)$

To learn the 5 models listed above, we must be able to learn models of the following form:

1. $P(1|x)$, classification with accurate probability estimates
2. $P(y|x)$, conditional density estimation

Note that these models are not necessarily Gaussian or unimodal, and so we will again use a non-parametric approach to learn both types of models.

For all of our models, we will use a Gaussian kernel. All of our models have as input some $SE(2)$ pose $[x, y, \theta]^T$. We scale the angle θ by the object's radius of gyration ρ, and then divide the entire vector by a scalar to keep the data around $(-1, 1)$, which gives us a scaled input vector $v = (1/s)[x, y, \rho\theta]^T$. Our kernel is then simply:

$$K_\gamma(v_1, v_2) = (2\pi\gamma^2)^{-3/2} \exp\left(-\|v_1 - v_2\|^2/(2\gamma^2)\right)$$

For in-hand pose, $\theta \in [-\pi/2, \pi/2]$, and so to deal with angle wrap around, we augment our dataset with copies of any data near the angle boundary. That is, any data $\theta_i \in [-\pi/2, -\pi/2 + \epsilon]$ is copied with new angle $\theta_i + \pi$, and any data $\theta_i \in [\pi/2 - \epsilon, \pi/2]$ is copied with new angle $\theta_i - \pi$.

ϵ is chosen such that K_γ can be accurately computed near either angle boundary. Although this increases the size of our dataset, we find this is the fastest and easiest way to handle angle wrap around correctly.

4.4.2 Accurately Predicting Class Probabilities

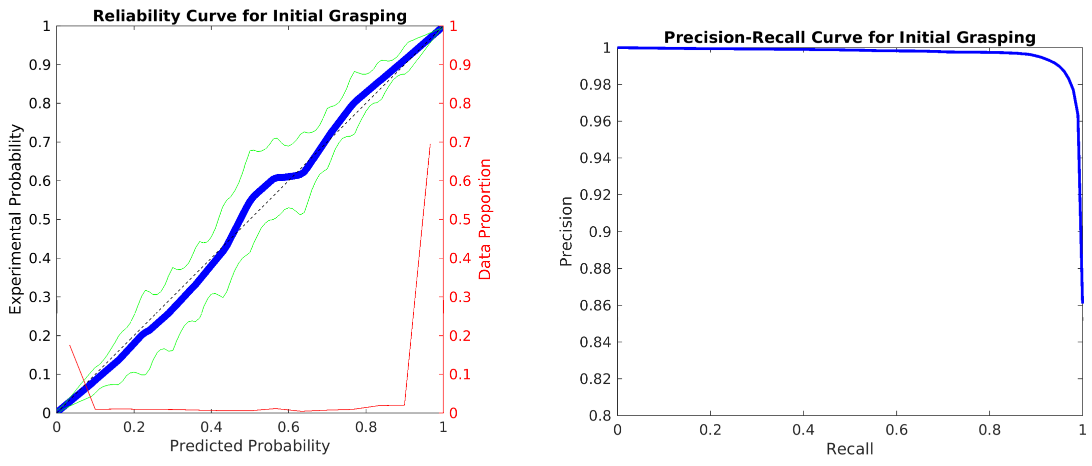
4.4.2.1 Theory

If we are to forward simulate manipulation models, it is imperative that we accurately predict the probability of success. The robot can use this predicted probability in planning to accurately predict the overall likelihood of success and adjust according to the desired amount of risk.

Classifiers generally have no guarantee on their probability predictions, and often only output a decision function. Also, even if a classifier does compute a log-loss based on probability, the accuracy of those probability estimates are still at the mercy of any assumptions made in the model. As an example, the Naive-Bayes classifier assumes that each feature is conditionally independent of every other feature. If this is not valid in our underlying distribution, the probability estimates from the Naive-Bayes classifier will be incorrect.

We can correct any errors in our probability estimates by calibrating our classifier [85, 97, 124]. The basic idea is to hold out some of the data, use the rest of the data to train a classifier as usual, and then use the held out data to adjust or "calibrate" the output of the classifier so it reflects the true probability. This can be made more data efficient by using K-fold cross validation rather than a single hold out set.

There are two main methods for calibration: sigmoid [97] and isotonic [124]. We will use isotonic calibration, which requires more data than the sigmoid method, but is more general. Suppose the output of our classifier (decision function or probability estimate) for a data point v_i



(a) (b)

Figure 4.31: Reliability curve for predicting whether or not we will grasp an object. Our predicted probabilities match the experimental probabilities well. This predicted probability can be used to move along a precision-recall curve to allow the robot to trade off risk.

is $f(v_i)$, and the true target is y_i. Isotonic regression assumes:

$$y_i = m(f(v_i)) + \epsilon$$

where m is a monotonically increasing function. We find the optimal function $\hat{m}$ such that the overall error is minimized:

$$\hat{m} = \arg\min_m \sum (y_i - m(f(v_i)))^2$$

Note that in order to have an unbiased estimate of $\hat{m}$, it is imperative that a separate set of data that was not used to train our classifier is used during the calibration.

For all of our prediction models, we will use a Gaussian kernel support vector machine, followed by an isotonic calibration. The bandwidth for our kernel γ and SVM soft margin parameter C are found via 3-fold cross validation on a randomly sampled set of (γ, C). Then, using those parameters, we split the data into 5 folds, and train 5 SVMs trained on 4 folds and the isotonic function trained on the remaining fold. The predicted probability is the average output of each of our SVM+isotonic classifier. The training step is implemented with the *CalibratedClassifierCV* function from the python package *scikit-learn*.

For predicting the probability of successfully grasping a block, there are only two classes: success or failure. For predicting the probability of placing a block however, there are three: block lands horizontally, block lands vertically, or failure. To do this multi-class classification, we train two classifiers in a hierarchical fashion. First, we learn $P(\text{placed}|\boldsymbol{s}_h)$, which is the probability of successfully placing the block. Then, using the data points where placing succeeded, we learn $P(C_p|\boldsymbol{s}_h, \text{placed})$, which is the platform class probabilities. Multiplying these probability functions together enables us to predict the probability of each class.

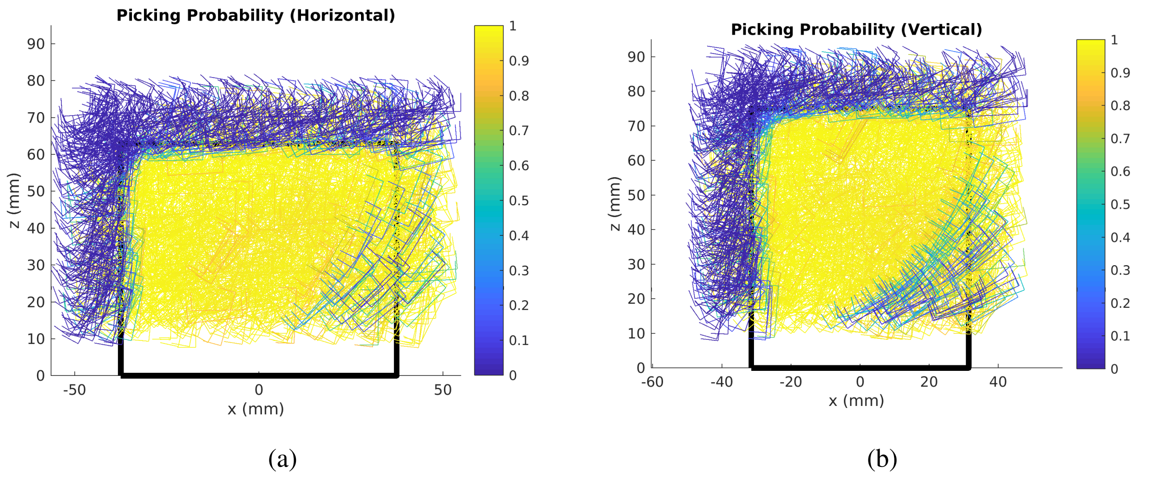
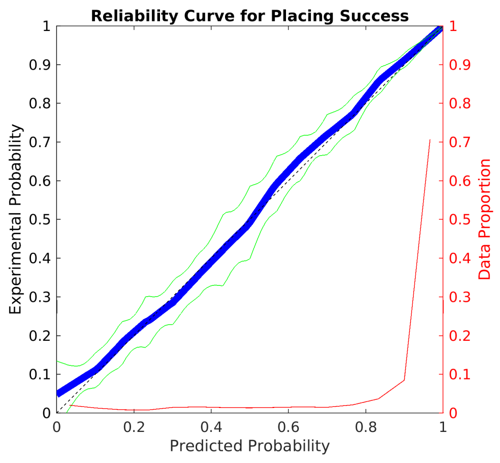


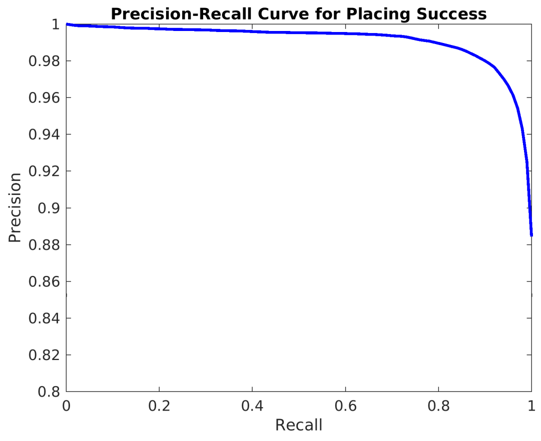
Figure 4.32: Probability of grasping an object from random initial poses according to our model. Note the similarity between our predicted probabilities and Figure 3.17.

4.4.2.2 Results

We will now show how well our resulting models perform with regards to probability prediction. In Figure 4.31, we show results for predicting the probability of successfully grasping an object. 3000 grasps were held out as a validation set and the model was trained on the rest. The x-axis is the probability of success the classifier predicts for each of the held out data points. The held out data points are binned, and the proportion of data that succeeds is the experimental probability. Ideally, the predicted probabilities match the experimental probability, and the data would follow the dotted line $y = x$. We randomly sampled from our dataset 30 times, and show the aggregated results in our plot. The thick blue line is the mean "reliability" of our classifier. The green lines are one standard deviation above and below the mean and represent the uncertainty. The red curve is the proportion of the data that falls in each of the predicted probability bins. Note that almost all of the data is predicted to be either near 100% or 0%, which makes sense, as most of our grasps either had full overlap between the gripper and block or no overlap. We see that for bins well represented by the data, our probability predictions are nearly perfect, with almost no uncertainty. Once we are able to predict the probability, by adjusting our cutoff point, we can trade off precision and recall, shown in Figure 4.31b. Knowing this probability can give our robots the power to make more informed decisions. If we select $p = 0.5$ as our threshold, our classification accuracy for grasping is 96.8%. In Figure 4.32 we randomly sample a set of possible grasps and visualize the probability. Note that both the edge failures and curved failures seen in Figure 3.17 are also represented in our model, as expected.



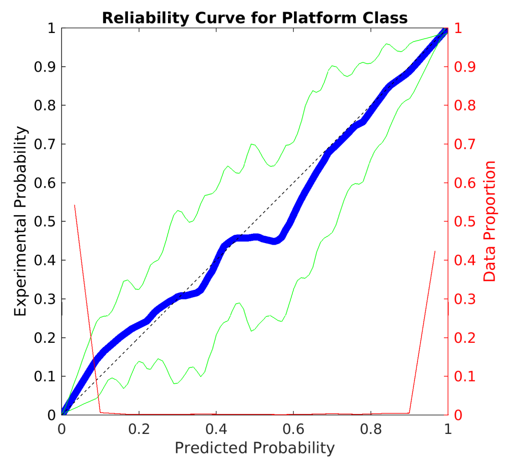
(a)



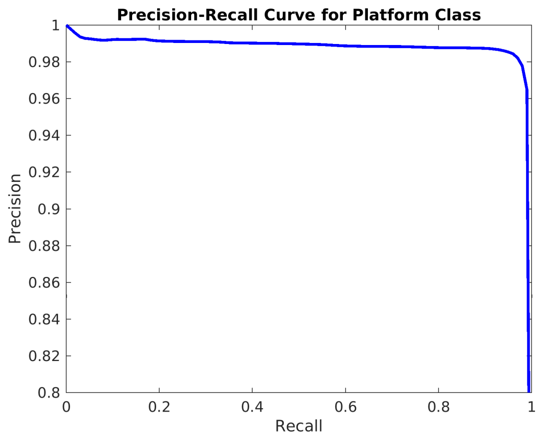
(b)

Figure 4.33: Reliability curve for predicting place success. Again our predicted probabilities match the experimental probabilities well, and that predicted probability can be used to trade off precision and recall.

In Figure 4.33 we show the probability of successfully placing a block on the platform without any collisions or the block falling off of the platform. Setting the threshold at $p = 0.5$ our classification accuracy is 93.2%. Figure 4.34 shows the probability of whether the block lands horizontally or vertically on the platform after a placement. Again, setting the threshold



(a)



(b)

Figure 4.34: Reliability curve for predicting whether the object will end up horizontally or vertically. Again our predicted probabilities match the experimental probabilities well, and that predicted probability can be used to trade off precision and recall.

at $p = 0.5$, our classification accuracy is 98.2%. In all cases, we see our classifiers predict probabilities well.

4.4.3 Accurately Predicting the Resulting Object Pose

4.4.3.1 Theory

We would like to predict the resulting probability distribution after an action. For the placing action, because the robot always moves to the same height before releasing the object, our input is SE(2), and our output is the SE(2) platform pose that the object ends up in. For the picking action, although there is the SE(2) platform pose and action $\boldsymbol{a} = [d, h, \alpha]^T$ as input, we can reduce our dimensionality by projecting the object's platform pose to a single value, x, which is where it is horizontally with respect to the edge of the platform. Then, for a given action $\boldsymbol{a}$, we can compute the relative hand pose with respect to object pose x. This again gives us an SE(2) input and SE(2) output. Thus, for both placing and picking, we can model the transition model as:

$$P(v_f|v_i), \qquad \text{where} \quad v_i = (1/s_i)\begin{bmatrix} x_i \\ y_i \\ \rho\theta_i \end{bmatrix}, \quad v_f = (1/s_f)\begin{bmatrix} x_f \\ y_f \\ \rho\theta_f \end{bmatrix}$$

As in Section 4.3.2.3, we will use kernel conditional density estimation:

$$P(v_f|v_i) = \frac{\sum_j^m K_{\gamma_1}(v_i, v_i^j) K_{\gamma_2}(v_f, v_f^j)}{\sum_j^m K_{\gamma_1}(v_i, v_i^j)}$$

Let the n-th nearest neighbor of v in our dataset be expressed as the function

$$v^n = q(v, n)$$

By choosing numbers $k_1 \ll m$ and $k_2 \ll m$, we can approximate our kernel conditional density estimate as follows:

$$\tilde{P}(v_f|v_i) = \frac{\sum_j^{k_2} \sum_t^{k_2} \mathbb{1}(j = t) K_{\gamma_1}(v_i, q(v_i, j)) K_{\gamma_2}(v_f, q(v_f, t))}{\sum_j^{k_1} K_{\gamma_1}(v_i, q(v_i, j))}$$

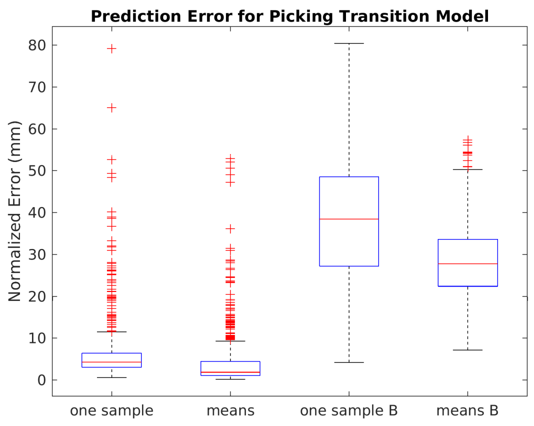
where $\mathbb{1}()$ is the indicator function. Note that by choosing k_1 such that $K_\gamma(v, q(v, k_1 + 1)) \ll K_\gamma(v, q(v, 1))$, and k_2 such that there is a sufficient amount of commonality between the v_i and v_f data, $\tilde{P}(v_f|v_i) \simeq P(v_f|v_i)$. k_1 and k_2 can be adjusted to trade off accuracy versus speed.In our experiments, we chose $k_1 = 100$ and $k_2 = 500$ which both offer significant speedups compared with $m \approx 8000$. Note that we use a Kd-tree implementation for fast nearest-neighbor lookup.

Now that we have our new formulation for kernel conditional density estimation, all that's left to do is find the kernel bandwidths γ_1 and γ_2. If the bandwidths are too small, we will overfit to the data and not be able to generalize well between data points. If the bandwidths are too large, we will overly smooth our system and have poor model accuracy. See Figure 4.37 for a visualization of this. To optimize these bandwidths, we should be minimizing the integrated squared error (ISE) discussed in Section 4.3.2.3. However we have found that reasonable results are obtained by finding the bandwidths that minimize the negative log likelihood of the joint probability $P(v_f, v_i)$, as suggested by Holmes et al. [54]. That is:

$$\gamma_1^*, \gamma_2^* = \arg\max_{\gamma_1, \gamma_2} \frac{1}{M} \sum_s^m \log\left(P_{-s}(v_f|v_i) P_{-s}(v_i)\right)$$

$$P_{-s}(v_f|v_i) P_{-s}(v_i) = \sum_j^m \mathbb{1}(j \neq s) K_{\gamma_1}(v_i, v_i^j) K_{\gamma_2}(v_f, v_f^j)$$

A problem that can occur during likelihood cross-validation is extreme observations or heavy-tailed distributions significantly affecting the outcome. In order to mitigate these effects, we



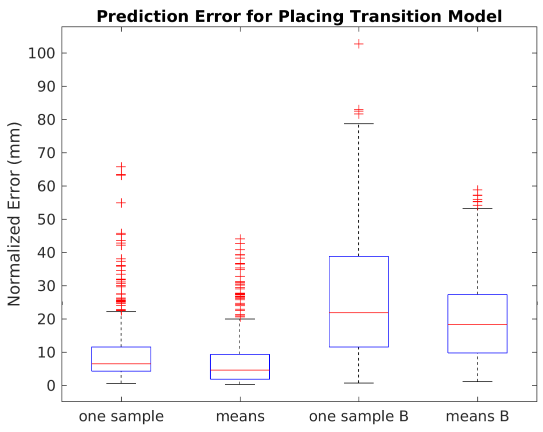


Figure 4.35: Transition model evaluation. Note that we outperform our baseline both when we take a single sample from our distribution or when we average many samples. Our median pose estimation error is 1.9mm for picking and 4.6mm for placing.

follow Wu [121], and replace the natural log in the equation above with the following robust function:

$$\log^{\star}(x; a) = \begin{cases} \log(x) & x \geq a \\ \log(a) - 1 + x/a & x < a \end{cases}$$

a is chosen to balance robustness to outliers and accuracy. We have found in practice that the bandwidths found using robust likelihood cross validation are reasonable, and this significantly reduces the training time for our model compared with optimizing ISE.

4.4.3.2 Results

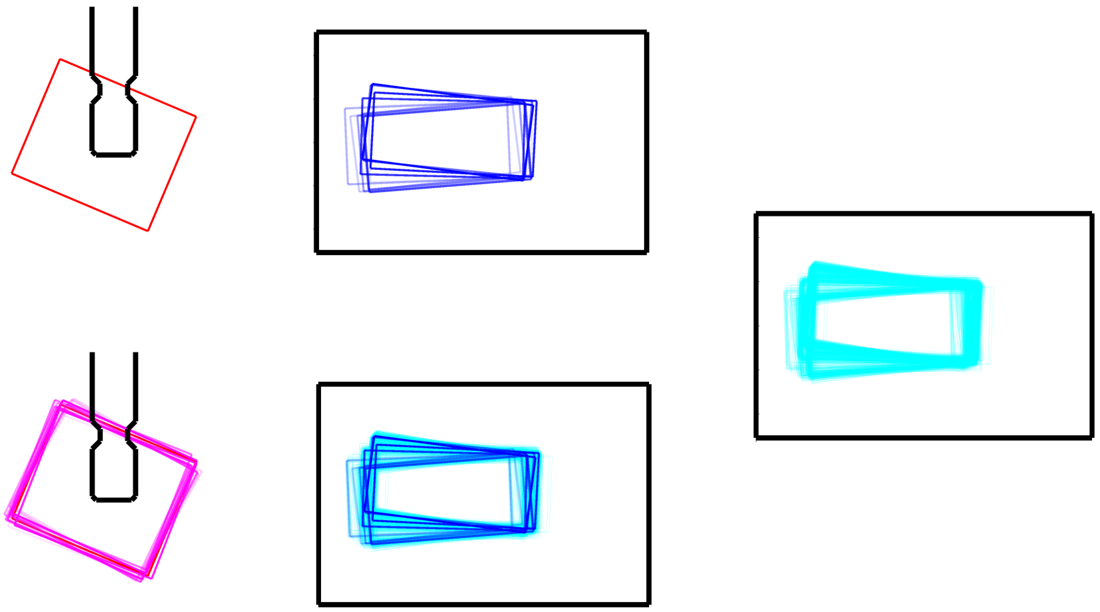


Figure 4.36: Intuition for kernel conditional density estimation. The object is shown in red grasped by the black robot figers. The pink poses are the grasped poses that are close in our dataset. The blue blocks are where the objects ended up when those pink poses were placed on the platform. A Gaussian blur can be added around these poses, which gives us our final probability distribution shown in cyan.

We will now assess how well our transition models do at predicting the resulting distribution. To evaluate our models, we hold out 500 data points and see how our model's prediction of the data compares with the true outcome. Note that because our model outputs a probability distribution of where the object will end up, and we do not have ground truth for this distribution, it can be tricky to evaluate this correctly. We look at three metrics and compare each against a baseline. For the first test, we will sample a single pose from our resulting distribution for each test point, and compare this with the true outcome. Note that since we're simply taking a single sample, this could be a very poor match with the true pose. The second test will take the mean of 1000 output samples from our model and compare that with the true outcome of each data point. Note that if the true output distribution is multi-modal, this mean could be arbitrarily bad,

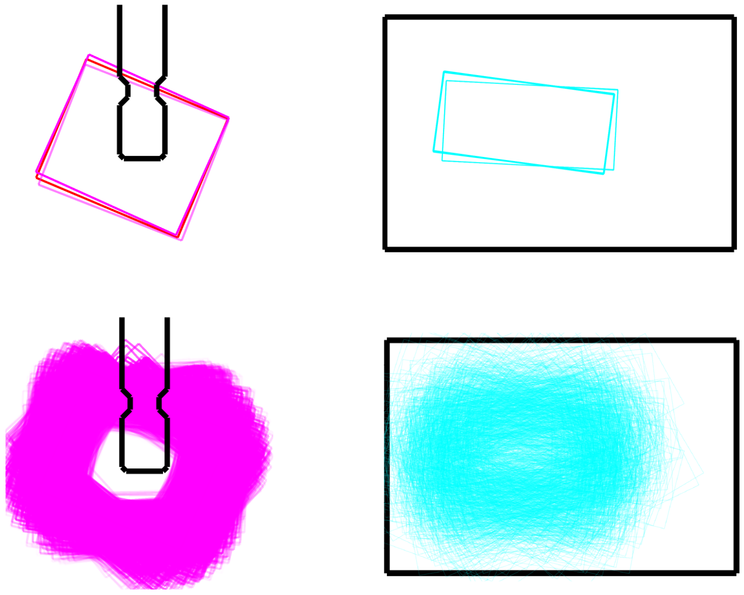


Figure 4.37: Importance of bandwidth selection for KCDE. If the bandwidths are too small, as shown in the upper figures, the model will overfit to data nearby and not have good generalization. If the bandwidths are too large, as shown in the bottom figure, the model will smooth over too much of the data and not learn anything. It is important to select the right bandwidths by finding the balance between these two extremes by minimizing the integrated squared error.

as it could predict a pose in between two modes of the distribution. Our third test computes the negative log likelihood of each resulting output point. Since $\int_V p(\boldsymbol{s}_f|\boldsymbol{s}_i)dV = 1$, the likelihood $\log p(\boldsymbol{s}_f|\boldsymbol{s}_i)$ will depend on the total volume of the space, and so this value is hard to interpret. For each of these tests, we will compare our models with a baseline, which will randomly sample a pose from the output training set, and use that as a prediction.

Figure 4.35a shows a box and whisker plot of the prediction errors for estimating object pose after a pick action for our first 2 metrics. As expected, our model outperforms the baseline significantly. The error is computed as $\|[x_1, y_1, \rho\theta_1] - [x_2, y_2, \rho\theta_2]\|$, which is a normalized distance that trades off position and orientation in a reasonable way. The median error for the single sample output is 4.3mm and 1.9mm for the case where 1000 model samples are averaged. This implies that most of our output distrbutions can perhaps be well represented by a unimodal distribution. Note however that if we are using samples from these models for planning, our formulation will work regardless of whether the distribution is unimodal or multimodal. For the third metric, our model outperforms the baseline with an average negative log likelihood of -1.84 compared with the baseline's 4.88.

Figure 4.35b shows model prediction errors when predicting placing poses for both the horizontal and vertical configurations. Again our model outperforms the baseline. The median error for the single sample output is 6.5mm and 4.6mm for the case where 1000 model samples are averaged. Note that these errors are larger because the placing action is more random than a picking action, as the object is sometimes dropped on the platform. For the third metric, our model outperforms the baseline with an average negative log likelihood of -0.66 compared with the baseline's 5.97.

Another way to evaluate our pose estimation models is to use a Kernel Two-Sample Test [47]. This test takes samples from two distributions, and computes a test statistic by finding the largest difference in expectation over reproducing kernel Hilbert space (RKHS) functions, called the maximum mean discrepancy (MMD). To evaluate our models, we can compute the average probability distribution across a number of hold out input poses in a neighborhood around key points. The output poses from those held out input poses should be coming from the average probability distribution computed from our models. Across various key points, with a significance level of $\alpha = 0.05$, we accepted the null hypothesis that our output data came from this modeled probability distribution.

To gain additional intuition for these models, we show a sample place model result in Figure 4.36. The object is initially held by the black outline of the gripper and is shown in red. The pink poses show the closest poses in our data set in input space. The blue poses are the resulting placement locations where each of the pink poses ended up. We can then add a Gaussian blur to these blue poses, shown in cyan. The cyan blocks on the platform is our resulting predicted probability distribution of what we expect to occur when the robot places the block.

4.4.4 Forward Simulation

In order to use these models in planning, we need to be able to forward simulate both our success and transition models in an accurate and computationally efficient way.

For simulating success, we use Algorithm 1, which can be vectorized in Matlab. The bottleneck in speed is the computation of the kernel SVM, which requires computing a Gram matrix with the data points and support vectors. In practice, we can forward simulate 100,000 points in 4 seconds on a standard laptop.

Algorithm 1 Forward simulate a success model

```
Input: input particles $\{\boldsymbol{s}_i\}$
Output: remaining particles $\{\boldsymbol{s}_f\}$
  function ForwardSimulateSuccess($\{\boldsymbol{s}_i\}$)
      $\{v_i\} \leftarrow scale\_data(\{\boldsymbol{s}_i\})$
      $\{\boldsymbol{s}_f\} \leftarrow \{\}$
      for $j = 1 : N$ do
          $p^j \leftarrow P(1|v_i^j)$
          if $p^j > rand(0,1)$ then
              $\{\boldsymbol{s}_f\} \leftarrow \{\boldsymbol{s}_f\} + \boldsymbol{s}_i^j$
          end if
      end for
      return $\{\boldsymbol{s}_f\}$
  end function
```

For forward simulating transition models, we use Algorithm 2. Note that we are selecting a point in our data set probabilistically according to how close each point v_i^k is to each input point. Then, we use the selected input points' corresponding output point as the mean and $\gamma_2 I(3)$ as the covariance to a multivariate normal distribution, which we sample our output point from. This procedure approximates $P(v_f|v_i)$. This procedure can also be vectorized in Matlab, and in practice we can forward simulate 100,000 points in 1 second on a standard laptop.

Algorithm 2 Forward simulate a transition model

```
Input: input particles $\{\boldsymbol{s}_i\}$
Output: resulting particles $\{\boldsymbol{s}_f\}$
  function FORWARDSIMULATETRANSITIONMODEL($\{\boldsymbol{s}_i\}$)
      $\{v_i\} \leftarrow scale\_data(\{\boldsymbol{s}_i\})$
      $\{\boldsymbol{s}_f\} \leftarrow \{\}$
      for $j = 1 : N$ do
          $\{p_x\} \leftarrow \{\}$
          $p_T \leftarrow 0$
          for $k = 1 : M$ do
              $p_x^k \leftarrow K_{\gamma_1}(v_i^j, v_i^k)$
              $p_T \leftarrow p_T + p_x^k$
          end for
          pick index $t$ randomly from histogram $\{p_x\}/p_T$
          $v_f^j \leftarrow rand\_norm(v_f^t, \gamma_2)$
          $\boldsymbol{s}_f^j \leftarrow scale\_data\_inverse(v_f^j)$
          $\{\boldsymbol{s}_f\} \leftarrow \{\boldsymbol{s}_f\} + \boldsymbol{s}_f^j$
      end for
      return $\{\boldsymbol{s}_f\}$
  end function
```

Chapter 5

Planning

5.1 Planning Sequences of Manipulation Actions

With statistical models of robot manipulation actions in hand, a natural next step is to plan with these actions. We attempt to answer the following question:

Given an initial pose of an object in a robot's hand or on a platform, find a sequence of actions to move the object to a desired final pose within the robot's hand or on the platform.

Suppose we have an objective function $J(\boldsymbol{s}, \boldsymbol{s}_g)$, which represents how close we are to a desired final pose $\boldsymbol{s}_g$. An example objective function for $\boldsymbol{s} = [x, y, \theta]$ could be:

$$J(\boldsymbol{s}, \boldsymbol{s}_g) = \exp\left(-D_s(\boldsymbol{s}, \boldsymbol{s}_g)/(2\sigma^2)\right)$$
$$D_s(\boldsymbol{s}_1, \boldsymbol{s}_2) = \left\| \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right\|^2 + 2\rho^2(1 - \cos(\theta_1 - \theta_2))$$

however, J can be any function we wish to optimize.

Suppose our initial object pose is $P(\boldsymbol{s}_0)$, and we wish to find a sequence of actions $S = \{a_1, ..., a_M\}$ such that we end up at our goal. After each action a_i, we will end up at state $\boldsymbol{s}_i$. So, we wish to find the sequence of actions that maximizes the expected value of our objective:

$$S = \{a_1, ..., a_M\}$$
$$S^* = \arg\max_S \mathbf{E}\left[J(\boldsymbol{s}_M, \boldsymbol{s}_g)\right]$$

which we can estimate using Monte-Carlo Integration:

$$S^* = \arg\max_S \frac{1}{N} \sum_k^N J(\boldsymbol{s}_M, \boldsymbol{s}_g) P(\boldsymbol{s}_M | \boldsymbol{s}_0^k, S), \quad \boldsymbol{s}_0^k \sim P(\boldsymbol{s}_0)$$

$$P(\boldsymbol{s}_M | \boldsymbol{s}_0, S) = \prod_{i=1}^M P(\boldsymbol{s}_i | \boldsymbol{s}_{i-1}, a_i)$$

where $P(\boldsymbol{s}_i|\boldsymbol{s}_{i-1}, a_i)$ can be forward simulated from our learned success and transition models. Note that for post-grasp manipulation actions in Section 4.2, there is only one action step, and so J can be maximized via a simple grid search or random sampling in that action space.

We apply the framework above to sequences of placing and picking actions. With the forward simulation algorithms shown in Section 4.4.4 we can quickly simulate many particles. The only challenge that remains is how to choose what sequences of actions to use. While there are many algorithms for planning on Markov Decision Processes with continuous actions and states [63, 64, 77], we will initially use brute force search, and then sample around actions that are succeeding more often. With our current set of actions, our plans will have very short horizons, on the order of two to four steps, which means the branching factor is not too high. In addition, given that we generate our plans offline, we are less concerned with computational constraints. The purpose of this section is to validate our models and show how they can be used for planning. We hope others will use more efficient planning methods with our models in the future.

5.2 Results

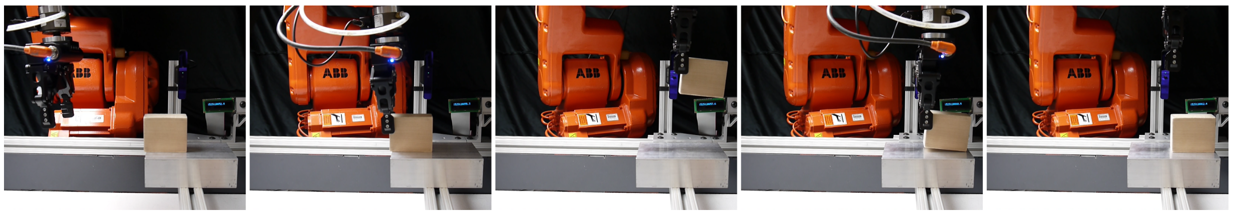


Figure 5.1: Executed plan for moving an object 8cm to the right.

We select several scenarios to test our placing and picking plans. Note that all of these plans will be executed open loop, with no sensing in between. The robustness of these plans could be improved by using sensing, or closed-loop control, however the purpose of this section is to validate our statistical models.

- **Platform Translation**: Starting with an object on the platform, move the object 8cm to the right.
- **Platform Rotation**: Starting with the object on the platform, rotate the object 90 degrees and move it 4cm to the left.
- **Handling Uncertainty**: Starting from an in-hand pose such that the object sometimes lands in a horizontal configuration and sometimes in a vertical configuration, find the sequence of actions that maximizes the probability that the object comes to rest horizontally on the platform.

For each plan, the number of steps was set ahead of time. A place action is one step, and a pick action is another step. Our planner automatically chooses to use a place or pick action depending on whether the object is in the hand or on the platform. Note that for the placing action, there are no action parameters, as the robot always moves to a specified height and releases its gripper. For the picking action, we have a three dimensional action space $\boldsymbol{a} = [d, h, \alpha]$, which is the pose

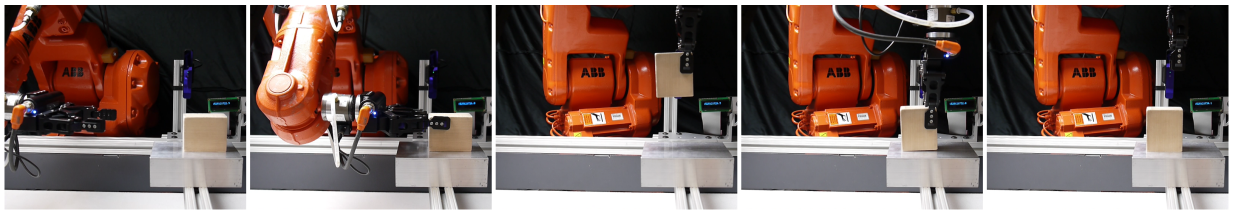


Figure 5.2: Executed plan for rotating an object 90 degrees and moving the center of mass 4cm to the left.

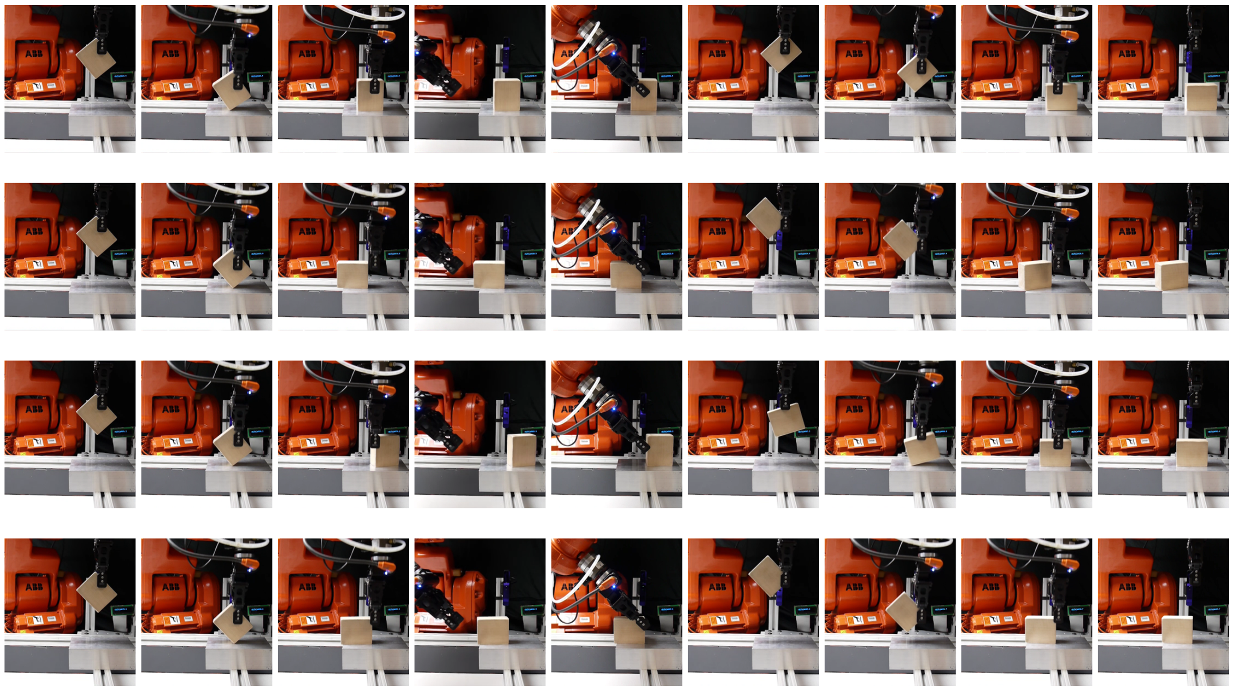


Figure 5.3: Plan for making sure the object ends up in a horizontal configuration on the platform, regardless of an uncertain initial placement

of the gripper with respect to the platform. We initially start with 100000 particles and randomly select action parameters when needed. Then, we forward simulate 100000 particles again, this time sampled from the sequences of actions that were successful. Finally, we bin our action sequences, select the top 5, and rerun our simulation to find the best action sequence.

We executed the resulting plans for the three scenarios discussed above. Figure 5.1 shows the robot's plan for moving the block horizontally 8cm to the right. Note that because the robot always places the block with the same hand pose, the robot correctly realizes that it needs to grasp the block as far left as possible in order to move it 8cm.

Figure 5.2 shows the robot's plan for rotating and moving the block as desired. Note that our planner realizes the robot needs to grasp the block horizontally, and from a high enough position for both the 90 degree rotation and 4cm translation requirement to be satisfied.

Figure 5.3 shows four runs of the robot attempting to place the object on the platform horizontally. Because of the initial grasped pose, the object can end up either in a horizontal or vertical configuration the first time around. However, through forward simulation, our planner finds a grasp pose that is able to capture both the horizontal and vertical poses by moving to the correct horizontal position, and then selects the right angle so that when the object is placed on the platform the second time, the object is likely to end up in a horizontal configuration. This plan showcases the power of modeling the entire resulting probability distribution of an action rather than just its most likely pose.

In this short chapter we have formulated a simple framework for planning with data-driven statistical models, and executed plans for three separate scenarios. Our plans were executed on a real robot, and moved the object to a desired pose in each case. In the future, we look forward to using more computationally efficient planning models which are also probabilistically complete.

Chapter 6

Summary and Future Work

6.1 Contributions

As discussed in the introduction, this thesis makes the following contributions:

1. **Data Collection of Robotic Manipulation** - We look at several case studies on how to collect a large amount of robot manipulation data. We then use these case studies to create a list of insights on how robots can collect manipulation data with minimal human intervention.
2. **Data-Driven Statistical Models for Robotic Manipulation** - We present methods for modeling manipulation statistically using collected data sets. We present the following statistical models:
    - Sensing capabilities of a robot hand
    - Task requirements of post-grasp manipulation actions
    - Combining the above models to predict the probability of succeeding at a task given sensor readings
    - Probability of maintaining a grasp of an object after a regrasp, place, or pick action
    - Predicting the probability distribution of where we expect the object to end up after a regrasp, place, or pick action

    Finally, we validate our data-driven models by having the robot create and execute end to end plans for a series of actions to position an object in a robot's hand, or in the environment.

6.2 Applications

6.2.1 Factory Robot Learning to Manipulate a New Object

Suppose a new product needs to be assembled in a factory. A robot could be given a part that needs to be assembled, and play with it for several days on its own. At the end of this session, the robot has good statistical models of how to manipulate this object. It knows how the object will move when it grasps, places, pushes, or regrasps it, to name a few. Now, the engineers specify how this part needs to be assembled. The robot knows how to grasp the object in the desired way, and can assemble the part. Error recovery can also be automatically handled by the robot as it can use its learned models to plan correcting motions. Rather than engineers spending days carefully designing specialized fixtures and robot motions to assemble this part, the robot has learned how to do it on its own. This could lead to faster and cheaper production.

6.2.2 Anomaly Detection in Factories

As robots are working to build products in a factory, sensor data could be collected to build statistical models of the task at hand. After many successful runs, a model of what the nominal operating condition of the manipulation action can be learned. Now, each time an action is performed, the sensor readings can be compared against the nominal model. If the probability of a certain combination of sensor readings is low according to the nominal model, there is a high likelihood of an anomaly. Engineers could be alerted to a potential problem in the factory, improving efficiency and potentially decreasing manufacturing costs.

6.2.3 Improving Personal Robotics via the Cloud

A robot operating in the home faces numerous challenges because of the sheer number of different objects it needs to manipulate. It also faces a harsher environment than a controlled laboratory setting, with both higher sensing and actuation uncertainty. One way in which home service robots might be improved is if they all pooled their resources as they manipulate objects in the home via the cloud. The combined experiences of all of the robots could be used to build statistical models which all robots could use. In addition, there could be workcells of similar robots in more controlled settings where better sensors are available. The statistical models learned by these robots could then be integrated with the higher sensor uncertainty of the in-home robots to improve their probabilistic predictions.

6.3 Future Work

6.3.1 Generalizing Across Objects

Currently, the models we learn are tied to a specific object and a specific action. While this enables us to plan a sequence of motions for that object, what would be more useful is if this could extend across different objects. An example solution might be to train models on spheres of different sizes, which would perhaps enable us to generalize to spheres we have not seen before. Whether this is possible is unclear however, as manipulation is not continuous. For example, a grasp that is stable for a large sphere might not even be a grasp for a smaller sphere if the sphere is smaller than the distance between fingers. A parameter cannot just be tuned. The physics are fundamentally different.

6.3.2 Combining Physics and Simulators with Data

Our models are learned entirely from data. While this enables us to precisely model what happens freed from physics assumptions that may be incorrect, it also greatly increases the amount of data we need. On the other hand, using physics enables us to generalize relatively well to new situations with little or no data. By combining physics and learning together, we should be able to generalize well and have more precise models. There are two ways in which using physics or simulators might be combined with data. First, physics could be used as constraints in our learning framework, such as not allowing objects to penetrate the table or hand. Care should be taken here however, as modeling inaccuracies such as deformation or objects and environment dimensions not being known precisely might affect where the true constraint is. Sensing inaccuracies will also affect where these true constraints are, so it may be that soft constraints perform better than hard constraints.

A second way to use physics or simulators would be to use them as priors. We could generate thousands of data samples from physics models or simulators, and give this data high weight in regions where relatively little true data has been collected. Over time, as more real samples are collected, the influence of the physics prior could be diminished to pave the way for the more realistic samples collected from the robot itself.

6.3.3 Improving Data Efficiency

Another important problem to solve is how to be more efficient in our data collection. Currently, we just sample actions uniformly in regions which may result in successful actions. This is wasteful in two ways. First, a large number of our samples may result in failures, as seen in Figure 3.20. Second, certain actions may be “boring”, where small changes do not result in large differences in physics. This implies that these regions can be sampled more coarsely, whereas regions near edges in the distribution should be sampled more finely.

If we learned models online as we collected data, we could sample from regions where our models are performing poorly. Performance could be measured as predicting success probabilities well or accurately predicting the resulting pose after an action. Collecting data points where our performance is low should increase the overall accuracy of our models for an equivalent number of data points.

A second point to consider is when certain portions of the state space are “rare events”. Suppose we have an object with a slot just slightly wider than the width of the robot’s fingers. If one of our actions caused large forces to be applied to the object, we may only be able to resist this force if the robot’s fingers are in this slot. If we sampled our grasps uniformly, the robot would almost never encounter this firm grasp, and would assume the second action which applies large forces to the object always results in a failure. Thus, in addition to learning models online, it may also be important to do planning online, so the robot can focus its effort on getting to regions in its state space it is unfamiliar with.

6.3.4 Closing the Loop

The models we have of manipulation actions can only be so accurate. This could be because our hardware is changing over time, we have not yet collected enough data, the object we are using is different, etc. If our models are somewhat close, wrapping a feedback loop around this model could enable us to improve robustness, accuracy, and generalization. How accurate the models need to be, what type of sensing is important (vision, force, tactile), and required closed loop frequency are all interesting open questions.

6.4 Concluding Remarks

In this thesis, we have proposed to advance the field of robotic manipulation by modeling manipulation actions more accurately by leveraging real data. We showed how to statistically model robotic manipulation with data, and how to collect enough real robotic manipulation data to make the models accurate. In addition, we showed how robots can use these models to automatically plan a series of actions to change where the object is from any initial pose to any final pose, either in the hand or environment. It is our hope that this thesis will show the power of real data in increasing the manipulation capabilities of robots both in factories and the home.

Bibliography

[1] Carnegie science center, ”hoops”. `http://www.carnegiesciencecenter.org/exhibits/roboworld-meet-robots-hoops/`. Accessed: 2017-06-28. 3.7.2

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016. 2.5

[3] Srinivas Akella and Matthew T. Mason. Posing polygonal objects in the plane by pushing. *IJRR*, 17(1):70–88, January 1998. 2.4

[4] Aitor Aldoma, Zoltan-Csaba Marton, Federico Tombari, Walter Wohlkinger, Christian Potthast, Bernhard Zeisl, Radu Bogdan Rusu, Suat Gedikli, and Markus Vincze. Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation. *IEEE Robotics & Automation Magazine*, 19(3):80–91, 2012. 3.7.1

[5] R. Balasubramanian, Ling Xu, P.D. Brook, J.R. Smith, and Y. Matsuoka. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *Robotics, IEEE Transactions on*, 28(4):899–910, 2012. 2.1

[6] Filippo Basso, Emanuele Menegatti, and Alberto Pretto. Robust intrinsic and extrinsic calibration of rgb-d cameras. *arXiv preprint arXiv:1701.05748*, 2017. 3.7.1.1

[7] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb 1992. 3.6

[8] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesisa survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014. 2.5

[9] Pasu Boonvisut and M Cenk Çavuşoğlu. Estimation of soft tissue mechanical parameters from robotic manipulation data. *IEEE/ASME Transactions on Mechatronics*, 18(5):1602–1611, 2013. 2.5

[10] Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. Closing the Learning-Planning Loop with Predictive State Representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011. 2.1

[11] Z. I. Botev, J. F. Grotowski, and D. P. Kroese. Kernel density estimation via diffusion. *The Annals of Statistics*, 38(5):2916–2957, October 2010. 4.2.2.1

[12] Abdeslam Boularias, James Andrew Bagnell, and Anthony Stentz. Learning to manipulate unknown objects in clutter by reinforcement. 2015. 2.5

[13] David L Brock. Enhancing the dexterity of a robot hand using controlled slip. In *Robotics*

and Automation, 1988. Proceedings., 1988 IEEE International Conference on, pages 249–251. IEEE, 1988. 2.3

[14] Randy C. Brost. Automatic grasp planning in the presence of uncertainty. *The International Journal of Robotics Research*, 7(1):3–17, 1988. 2.1

[15] Randy C. Brost. *Analysis and Planning of Planar Manipulation Tasks*. PhD thesis, Carnegie Mellon University School of Computer Science, 1991. 2.1

[16] Randy C Brost and Alan D Christiansen. Probabilistic analysis of manipulation tasks: A conceptual framework. *The International journal of robotics research*, 15(1):1–23, 1996. 2.1, 2.5, 3.7.2

[17] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting Optimally in Partial Observable Stochastic Domains. *AAAI*, 1994. 2.1

[18] Alexandre Cayla. *One Sweet Robot: A Compliant 3D-Printed Robotic Hand*. B.S. Thesis, The University of Queensland, 2016. 2.1

[19] Nikhil Chavan-Dafle, Alberto Rodriguez, Robert Paolini, Bowei Tang, Siddhartha Srinivasa, Michael Erdmann , Matthew T. Mason , Ivan Lundberg, Harald Staab, and Thomas Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2014. 2.3

[20] Moez Cherif and Kamal K Gupta. Planning quasi-static fingertip manipulations for reconfiguring objects. *Robotics and Automation, IEEE Transactions on*, 15(5):837–848, 1999. 2.3

[21] Alan D. Christiansen. Manipulation planning from empirical backprojections. In *International Conference on Robotics and Automation*, pages 762–768, Sacramento, CA, 1990. 2.1

[22] Alan D Christiansen. *Automatic acquisition of task theories for robotic manipulation*. Cmu-cs-92-111, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. 2.1

[23] Alan D Christiansen and Kenneth Y Goldberg. Comparing two algorithms for automatic planning by robots in stochastic environments. *Robotica*, 13(06):565–573, 1995. 2.1

[24] Alan D Christiansen, Matthew T Mason, and Tom M Mitchell. Learning reliable manipulation strategies without initial physical models. *Robotics and Autonomous Systems*, 8 (1-2):7–18, 1991. 3.7.2

[25] Vassilios N Christopoulos and Paul Schrater. Handling shape and contact location uncertainty in grasping two-dimensional planar objects. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1557–1563. IEEE, 2007. 2.1

[26] Arlene Cole, Ping Hsu, and S Shankar Sastry. Dynamic control of sliding by robot hands for regrasping. *Robotics and Automation, IEEE Transactions on*, 8(1):42–52, 1992. 2.3

[27] Renaud Detry, Emre Baseski, Mila Popovic, Younes Touati, N Kruger, Oliver Kroemer, Jan Peters, and Justus Piater. Learning object-specific grasp affordance densities. In *2009 IEEE 8th International Conference on Development and Learning*, pages 1–7. IEEE, 2009. 2.5

[28] Renaud Detry, Dirk Kraft, Anders Glent Buch, Norbert Krüger, and Justus Piater. Refining grasp affordance models by experience. In *Robotics and automation (icra), 2010 ieee international conference on*, pages 2287–2293. IEEE, 2010. 2.5, 3.7.2

[29] Renaud Detry, Dirk Kraft, Oliver Kroemer, Leon Bodenhagen, Jan Peters, Norbert Krüger, and Justus Piater. Learning grasp affordance densities. *Paladyn, Journal of Behavioral Robotics*, 2(1):1–17, 2011. 2.5

[30] Mehmet Dogar and Siddhartha S. Srinivasa. A Framework for Push-Grasping in Clutter. *Robotics: Science and Systems (RSS)*, 2011. 2.1

[31] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. Ieee, 2010. 3.7.1, 3.7.1.2

[32] Felix Endres, Jeff Trinkle, and Wolfram Burgard. Learning the dynamics of doors for robotic manipulation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3543–3549. IEEE, 2013. 2.5

[33] Michael Erdmann. Using backprojections for fine motion planning with uncertainty. *The International Journal of Robotics Research*, 5(1):19–45, 1986. 2.1

[34] Michael A Erdmann and Matthew T Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988. 2.1, 3.5

[35] Diego R. Faria, Ricardo Martins, Jorge Lobo, and Jorge Dias. Extracting data from human manipulation of objects towards improving autonomous robotic grasping. *Robotics and Autonomous Systems*, 60(3):396 – 410, 2012. 2.1

[36] MA Farooqi, Takashi Tanaka, Yukio Ikezawa, Toru Omata, and Kazuyuki Nagata. Sensor based control for the execution of regrasping primitives on a multifingered robot hand. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 4, pages 3217–3223. IEEE, 1999. 2.3

[37] Ronald S Fearing. Simplified grasping and manipulation with dextrous robot hands. *Robotics and Automation, IEEE Journal of*, 2(4):188–195, 1986. 2.3

[38] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems (NIPS)*, 2016. 2.4, 2.5

[39] Marc Freese, Surya Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform v-rep: a versatile 3d robot simulator. In *Simulation, modeling, and programming for autonomous robots*, pages 51–62. Springer, 2010. 4.3.3.2

[40] Jiaxin Fu, Siddhartha Srinivasa, Nancy Pollard, and Bart Nabbe. Planar batting under shape, pose, and impact uncertainty. *IEEE International Conference on Robotics and Automation (ICRA)*, April 2007. 2.2

[41] Noriatsu Furukawa, Akio Namiki, Senoo Taku, and Masatoshi Ishikawa. Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 181–187. IEEE, 2006. 2.3

[42] K Gadeyne, T Lefebvre, and H Bruyninckx. Bayesian Hybrid Model-State Estimation Applied to Simultaneous Contact Formation Recognition and Geometrical Parameter Estimation. *The International Journal of Robotics Research*, 24(8):615–630, August 2005. 2.1

[43] Kenneth Y. Goldberg. *Stochastic Plans for Robotic Manipulation*. PhD thesis, Carnegie Mellon University School of Computer Science, 1990. CMU-CS-90-161. 2.1

[44] Kenneth Y Goldberg and Matthew T Mason. Bayesian grasping. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1264–1269. IEEE, 1990. 2.1

[45] Corey Goldfeder and Peter K. Allen. Data-driven grasping. *Autonomous Robots*, 31(1): 1–20, 2011. 2.1

[46] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction. Part 1. Limit surface and moment function. *Wear*, 143:307–330, 1991. 2.4

[47] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012. 4.4.3.2

[48] R A Grupen and J A Coelho. Acquiring state from control dynamics to learn grasping policies for robot hands. *Advanced Robotics*, 16(5):427–443, 2002. 2.1

[49] László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer, 2002. 4.2.5.3

[50] Peter Hall, Jeff Racine, and Qi Li. Cross-validation and the estimation of conditional probability densities. *Journal of the American Statistical Association*, 99(468), 2004. 4.3.2.3

[51] Kensuke Harada, Torea Foissotte, Tokuo Tsuji, Kazuyuki Nagata, Natsuki Yamanobe, Akira Nakamura, and Yoshihiro Kawai. Pick and place planning for dual-arm manipulators. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2281–2286. IEEE, 2012. 2.3

[52] Klaus Havelund and Rajeev Joshi. Modeling and monitoring of hierarchical state machines in scala. In *International Workshop on Software Engineering for Resilient Systems*, pages 21–36. Springer, 2017. 3.7.3

[53] Anne Holladay, Jennifer Barry, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Object placement as inverse motion planning. In *IEEE Conference on Robotics and Automation (ICRA)*, 2013. 2.2

[54] Michael P Holmes, Alexander G Gray, and Charles Lee Isbell Jr. Fast nonparametric conditional density estimation. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 175–182. AUAI Press, 2007. 4.4.3.1

[55] Jiawei Hong, Gerardo Lafferriere, Bhubaneswar Mishra, and Xiaonan Tan. Fine manipulation with multifinger hands. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1568–1573. IEEE, 1990. 2.3

[56] Robert D Howe and Mark R Cutkosky. Practical force-motion models for sliding manip-

ulation. *The International Journal of Robotics Research*, 15(6):557–572, 1996. 2.4

[57] Kaijen Hsiao, Matei Ciocarlie, and Peter Brook. Bayesian grasp planning. In *ICRA 2011 Workshop on Mobile Manipulation: Integrating Perception and Manipulation*, 2011. 2.1

[58] H Inoue. Force feedback in precise assembly tasks. Technical report, MIT AIM-308, 1974. 2.1

[59] Yun Jiang, Marcus Lim, Changxi Zheng, and Ashutosh Saxena. Learning to place new objects in a scene. *The International Journal of Robotics Research*, 2012. 2.2

[60] D. Kang and K. Goldberg. Sorting parts by random grasping. *IEEE Transactions on Robotics and Automation*, 11(1):146 –152, feb 1995. 2.1

[61] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. Leveraging big data for grasp planning. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4304–4311. IEEE, 2015. 2.1

[62] Dov Katz, Arun Venkatraman, Moslem Kazemi, Drew Bagnell, and Anthony Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013. doi: 10.15607/RSS.2013.IX.039. 3.7.2

[63] Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3): 193–208, 2002. 5.1

[64] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *AAAI Conference on Artificial Intelligence*, 2018. 5.1

[65] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Mörwald, and Jeremy Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5722–5729. IEEE, 2011. 2.1, 2.4

[66] Steven M Lavalle and Seth A Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *The International Journal of Robotics Research*, 1(17):19–42, 1998. 2.1

[67] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *International Symposium on Experimental Robotics (ISER)*, 2016. 2.5, 3.7.2

[68] Minas V Liarokapis and Aaron M Dollar. Learning task-specific models for dexterous, in-hand manipulation with simple, adaptive robot hands. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2534–2541. IEEE, 2016. 3.7.2

[69] Tomas Lozano-Perez. Motion planning and the design of orienting devices for vibratory part feeders. *IEEE Journal of Robotics and Automation*, 1986. 3.7.2

[70] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):

3–24, 1984. 2.1

[71] Kevin M Lynch. Inexpensive conveyor-based parts feeding. *Assembly Automation*, 19(3): 209–215, 1999. 3.7.2

[72] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *IJRR*, 15(6):533–556, December 1996. 2.4

[73] Jeffrey Mahler, Sachin Patil, Ben Kehoe, Jur Van Den Berg, Matei Ciocarlie, Pieter Abbeel, and Ken Goldberg. Gp-gpis-opt: Grasp planning with shape uncertainty using gaussian process implicit surfaces and sequential convex programming. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4919–4926. IEEE, 2015. 2.1

[74] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1957–1964. IEEE, 2016. 2.1

[75] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017. 2.1

[76] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2308–2315. IEEE, 2010. 3.7.2

[77] Christopher R Mansley, Ari Weinstein, and Michael L Littman. Sample-based planning for continuous action markov decision processes. In *ICAPS*, 2011. 5.1

[78] James Stephen Marron et al. Optimal rates of convergence to bayes risk in nonparametric discrimination. *The Annals of Statistics*, 11(4):1142–1155, 1983. 4.3.2.2

[79] Matthew T. Mason. On the scope of quasi-static pushing. In *International Symposium on Robotics Research*, pages 229–233. Cambridge, Mass: MIT Press, 1986. 2.4

[80] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research*, 5(3):53–71, Fall 1986. 2.1, 2.4

[81] Matthew T. Mason, Alberto Rodriguez, Siddhartha S. Srinivasa, and Andres S. Vazquez. Autonomous Manipulation with a General-Purpose Simple Hand. *The International Journal of Robotics Research*, 31(5):688–703, 2012. 4.2.1, 4.2.1.2

[82] A. Morales, E. Chinellato, A. H. Fagg, and A. P. del Pobil. Using Experience for Assessing Grasp Reliability. *International Journal of Humanoid Robotics*, 1(4):671–691, 2004. 2.1

[83] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013. 3.7.2

[84] Matteo Munaro, Filippo Basso, and Emanuele Menegatti. Openptrack: Open source

multi-camera calibration and people tracking for rgb-d camera networks. *Robotics and Autonomous Systems*, 75:525–538, 2016. 3.7.1.1

[85] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005. 4.4.2.1

[86] John Oberlin and Stefanie Tellex. Learning to pick up objects through active exploration. In *Development and Learning and Epigenetic Robotics (ICDL-EpiRob), 2015 Joint IEEE International Conference on*, pages 252–253. IEEE, 2015. 3.7.2

[87] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011. 3.4, 3.5, 3.7.2.3

[88] Damir Omrcen, Christian Boge, Tamim Asfour, Ales Ude, and Rüdiger Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 277–283. IEEE, 2009. 2.4

[89] Robert Paolini and Matthew T. Mason. Symmetry of 3 dimensional objects using quaternions. Technical Report CMU-RI-TR-14-22, Carnegie Mellon University, October 2014. 3.7.1.4

[90] Robert Paolini and Matthew T. Mason. Data-driven statistical modeling of a cube regrasp. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016. 2.1, 3.7.2, 4.3

[91] Robert Paolini, Alberto Rodriguez, Siddhartha Srinivasa, and Matthew T. Mason. A data-driven statistical framework for post-grasp manipulation. *International Journal of Robotics Research (IJRR)*, 33(4):600–615, April 2014. 2.1, 3.7.2, 4.2

[92] Peter Pastor, Ludovic Righetti, Mrinal Kalakrishnan, and Stefan Schaal. Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE, 2011. 2.5

[93] Richard P. Paul. Modeling, trajectory calculation and servoing of a computer controlled arm. Technical Report AIM– 177, Stanford Artificial Intelligence Laboratory, 1972. 2.3

[94] Michael A. Peshkin and Arthur C. Sanderson. The motion of a pushed, sliding workpiece. *Journal of Robotics and Automation*, 4(6):569–598, December 1988. 2.1

[95] Anna Petrovskaya, Oussama Khatib, Sebastian Thrun, and Andrew Y Ng. Bayesian estimation for autonomous object manipulation based on tactile sensors. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 707–714. IEEE, 2006. 2.1

[96] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016. 2.5, 3.7.2

[97] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

4.4.2.1, 4.4.2.1

[98] Robert Platt. Learning grasp strategies composed of contact relative motions. In *7th IEEE-RAS International Conference on Humanoid Robots*, pages 49–56. IEEE, November 2007. 2.1

[99] C. E. Rasmussen and C. K. I Williams. *Gaussian Processes for Machine Learning*. Press, MIT, 2006. 4.2.2.1

[100] Dan Reznik and John Canny. The coulomb pump: A novel parts feeding method using a horizontally-vibrating surface. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 869–874. IEEE, 1998. 3.7.2.5

[101] Andrew Richardson, Johannes Strom, and Edwin Olson. AprilCal: Assisted and repeatable camera calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013. 3.7.2.3

[102] Alberto Rodriguez, Matthew T. Mason, and Siddhartha S. Srinivasa. Manipulation Capabilities with Simple Hands. *International Symposium on Experimental Robotics (ISER)*, 2010. 4.2.1, 4.2.1.2

[103] Alberto Rodriguez, Matthew T. Mason, Siddhartha S. Srinivasa, Matthew Bernstein, and Alex Zirbel. Abort and Retry in Grasping. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2011. 4.2.5.1

[104] Alberto Rodriguez, Garth Zeglin, Robert Paolini, Matthew T. Mason, and Siddhartha Srinivasa. The MLab Hand - Submitted. In *International Conference on Robotics and Automation (ICRA)*, 2013. 4.2.1.2

[105] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837, September 1956. 4.3.2.2

[106] Daniela Rus. In-hand dexterous manipulation of piecewise-smooth 3-d objects. *The International Journal of Robotics Research*, 18(4):355–381, 1999. 2.3

[107] J Kenneth Salisbury Jr. *Kinematic and Force Analysis of Articulated Hands*. Phd dissertation, Stanford University, 1982. 2.3

[108] Stefan Schaal and Christopher G Atkeson. Robot juggling: implementation of memory-based learning. *IEEE Control Systems*, 14(1):57–71, 1994. 2.1

[109] T. Schlegl, M. Buss, T. Omata, and G.K. Schmidt. Fast dextrous re-grasping with optimal contact forces and contact sensor-based impedance control. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 103–108 vol.1, 2001. 2.3

[110] Thomas Schlegl and Martin Buss. Hybrid closed-loop control of robotic hand regrasping. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3026–3031. IEEE, 1998. 2.3

[111] Mario Thomas Sgriccia. Feeder bowl. *US Patent 2654465*, 12 1950. 3.7.2

[112] S N Simunovic. *An information approach to parts mating*. Sc.d., Massachusetts Institute of Technology, 1979. 2.1

[113] Sascha Stoeter, Stephan Voss, Nikolaos P Papanikolopoulos, and Heiko Mosemann. Planning of regrasp operations. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 245–250. IEEE, 1999. 2.3

[114] Alexander Stoytchev. Learning the affordances of tools using a behavior-grounded approach. *Towards affordance-based robot control*, pages 140–158, 2008. 3.7.2

[115] F. Stulp, E. Theodorou, J. Buchli, and S. Schaal. Learning to grasp under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5703–5708, May 2011. 2.1, 2.5

[116] P. Tournassoud, T. Lozano-Pérez, and E. Mazer. Regrasping. In *IEEE International Conference on Robotics and Automation*, pages 1924–1928, 1987. 2.3

[117] J. C. Trinkle, J. M. Abel, and R. P. Paul. An Investigation of Frictionless Enveloping Grasping in the Plane. *The International Journal of Robotics Research*, 7(3):33–51, June 1988. 2.1

[118] Daniel E Whitney and Eric F Junkel. Applying Stochastic Control Theory to Robot Sensing, Teaching, and Long Term Control. In *American Control Conference, 1982*, pages 1175–1183, 1982. ISBN VO -. 2.1

[119] David Wingate. *Exponential Family Predictive Representations of State*. PhD thesis, University of Michigan, 2008. 2.1

[120] Liao Wu and Hongliang Ren. Finding the kinematic base frame of a robot by hand-eye calibration using 3d position data. *IEEE Transactions on Automation Science and Engineering*, 14(1):314–324, 2017. 3.7.1.1, 3.7.2.3

[121] Ximing Wu. Robust likelihood cross validation for kernel density estimation. *Journal of Business & Economic Statistics*, (just-accepted), 2018. 4.4.3.1

[122] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*, 2016. 3.7.2

[123] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 30–37. IEEE, 2016. 2.5, 3.7.2

[124] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699. ACM, 2002. 4.4.2.1, 4.4.2.1

[125] Jiaji Zhou, Robert Paolini, J. Andrew (Drew) Bagnell, and Matthew T. Mason . A convex polynomial force-motion model for planar sliding: Identification and application. In *International Conference on Robotics and Automation (ICRA) 2016*, May 2016. 2.4, 3.3

[126] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A probabilistic planning framework for planar grasping under uncertainty. *IEEE Robotics and Automation Letters*, PP(99), 2017. 3.4, 3.7.2

[127] Zoran Zivkovic and Ferdinand Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7): 773–780, 2006. 3.7.2.3