REPRESENTING AND LEARNING AFFORDANCE-BASED BEHAVIORS

A Dissertation
Presented to
The Academic Faculty

by

Tucker Hermans

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

School of Interactive Computing
Georgia Institute of Technology
May 2014

REPRESENTING AND LEARNING AFFORDANCE-BASED BEHAVIORS

Approved by:

Aaron F. Bobick, Advisor
School of Interactive Computing
Georgia Institute of Technology

James M. Rehg, Co-Advisor
School of Interactive Computing
Georgia Institute of Technology

Mike Stilman
School of Interactive Computing
Georgia Institute of Technology

Henrik I. Christensen
School of Interactive Computing
Georgia Institute of Technology

Charles C. Kemp
Department of Biomedical Engineering
Georgia Institute of Technology

Dieter Fox
Department of Computer Science & Engineering
University of Washington

Date Approved: 5 March 2014

To my siblings,

Judson, Phillip, and Corrie,

who continue to remind me of how little I know.

ACKNOWLEDGEMENTS

I am fortunate to have had two extremely supportive advisors throughout graduate school. Aaron Bobick has challenged me intellectually since our first meeting. At the end of my PhD I am still reminded of conversations we had early in my studies. While many paths of inquiry have been abandoned over the years, Aaron has helped me to remember the original motivation that led to this dissertation. I will greatly miss my meetings with him when I leave Georgia Tech. Outside of our research discussions, Aaron has been a tremendous advocate for me. Aaron has done a wonderful job in promoting me and my work and has ensured I had all the necessary resources to complete my studies. Teaching computer vision with Aaron was one of the highlights of my time at Georgia Tech. I learned much running the course with Aaron and while my lectures paled in comparison to his, the experience cemented my desire to make teaching a part of my career.

I came to Georgia Tech in large part because of my meeting with Jim Rehg as a prospective student. Jim and I talked at length about research, both that of his lab and the honors project I was finishing at Bowdoin. Jim's enthusiasm for his work was infectious. His excitement in lab meetings and reading groups has frequently motivated me when my own methods and experiments have failed. I could not imagine a better pair of mentors than Aaron and Jim. I thank them deeply for their encouragement, support, and friendship over the past five years.

I am extremely grateful to the other members of my committee. Charlie Kemp provided me complete access to his laboratory and robot. I thank him for this and the many hours of discussion he provided me early in my PhD. Mike Stilman enthusiastically supported my turning a class project into one of my most cited publications.

Henrik Christensen was a regular source for references in addition to ensuring RIM provided all the support it could. For this I thank him. Dieter Fox has provided excellent advice since my PhD proposal. His comments ultimately inspired the core ideas of Chapter 4. I would also like to thank professors Irfan Essa and Frank Dellaert. Both have taught me much about computer vision and robotics and both have been strong supporters of my research efforts. I must thank all members of the Wall-Lab and CPL for giving me a great place to work each day. I cherish having had such a brilliant group of fellow students to help me through my PhD. Matt Flagg, Howard Zhou, and Matthias Grundmann were extremely welcoming and helpful in my first year. Matthias developed into a trusted colleague, who was often my first resource in answering computer vision questions. Although we often disagreed, Alireza Fathi was a great labmate for four years. Our discussions always helped me clarify my own position and I must admit that he was even right on a few occasions. I must also thank Arri Ciptadi, Ahmad Humayun, Yin Li, Abhijit Kundu, and Brian Goldfain for helpful and jovial discussions across my cubicle wall. Fuxin Li has been a wonderful asset during the final year of my PhD. He has consistently provided meaningful feedback on my writing, while aiding me in improving my machine learning knowledge. Misha Novitzky has been a persistent source of support. His determination in studying for the quals was inspirational and he has continued to inspire me through his unquenchable thirst for knowledge and understanding.

I am grateful to the members of Charlie Kemp's Healthcare Robotics Lab, who have been my partners in using the PR2. Hai Nguyen in particular spent many hours coding away with me in the Georgia Tech Aware Home. Other HRL members I wish to thank are Marc Killpack, Travis Deyle, Advait Jain, Kelsey Hawkins, Tapo Bhattacharjee, and Tiffany Chen. All have patiently helped me in some form improve my research on the PR2.

I must thank all participants past and present of CPL Drinks! The group has

blossomed into far more than I ever expected when I sent the first email to go to Cypress three years ago. CPL Drinks! has become a weekly staple in my life and helped me form many friendships. CPL Faculty all gave early support both in attending and buying rounds. While CPL Drinks! served to keep me mentally sound, RoboGrads FC encourage me to keep my body healthy. I must thank all team members and organizers for regularly getting me out of the lab and into the fresh air.

I could not have completed my PhD with the support of a number of people outside of Georgia Tech. My parents were extremely encouraging of my studies, never once questioning my choices. They have shown great patience and understanding when I fail to call and have made sure I have had the resources necessary to escape from Atlanta to visit friends and family when necessary. I thank my sister Corrie for her many encouraging text messages and phone calls. Talking to her always brightens my day. My brother Judson and his wife Emily were wonderful neighbors when I first moved to Atlanta. I must thank them for the countless times they got me to eat a delicious meal and enjoy their company when I had planned to snack alone in front of my computer. Madison Dotson has been equally encouraging in getting me out of the house. I am extremely grateful to her for her steady friendship and thoughtful advice over the past five years. My brother Phillip has answered my calls at all hours of the day. He has served as my sounding board and confidant on nearly every problem I have faced during my PhD. His dedicated attitude to his scholarship and research have been inspirational to my own efforts. Off campus Brian Goldfain has been a great friend and roommate during the final years of my PhD. I can not thank him enough for the support his friendship provides. Finally, I wish to thank my fiancé, Kelly Brooks. Her wit, friendship, and love have made the final months of my PhD far happier and far less stressful than had I not had her in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Autonomous robots deployed in complex, natural human environments such as homes and offices need to manipulate numerous objects throughout their deployment. For an autonomous robot to operate effectively in such a setting and not require excessive training from a human operator, it should be capable of discovering how to reliably manipulate novel objects it encounters. We characterize the possible methods by which a robot can act on an object using the concept of *affordances*. We define *affordance-based behaviors* as object manipulation strategies available to a robot, which correspond to specific semantic actions over which a task-level planner or end user of the robot can operate.

This thesis concerns itself with developing the representation of these affordance-based behaviors along with associated learning algorithms. We identify three specific learning problems. The first asks which affordance-based behaviors a robot can successfully apply to a given object, including ones seen for the first time. Second, we examine how a robot can learn to best apply a specific behavior as a function of an object's shape. Third, we investigate how learned affordance knowledge can be transferred between different objects and different behaviors.

We claim that decomposing affordance-based behaviors into three separate factors—a control policy, a perceptual proxy, and a behavior primitive—aids an autonomous robot in learning to manipulate. Having a varied set of affordance-based behaviors available allows a robot to learn which behaviors perform most effectively as a function of an object's identity or pose in the workspace. For a specific behavior a robot can use interactions with previously encountered objects to learn to robustly manipulate a novel object when first encountered. Finally, our factored representation allows a robot to transfer knowledge learned with one behavior to effectively manipulate an object in a qualitatively different manner by using a distinct controller or behavior

primitive. We evaluate all work on a bimanual, mobile-manipulator robot. In all experiments the robot interacts with real-world objects sensed by an RGB-D camera.

CHAPTER I

INTRODUCTION

The goal of this dissertation is to enhance the ability of robots to learn how to manipulate objects with which they may have no prior experience. Specifically we desire robots to autonomously explore methods of interacting with a given object. First, the robot should be able to interactively discover objects of interest in its environment. The robot should also be able to determine which available behaviors are effective in manipulating a particular object. Finally, the robot must transfer knowledge learned from objects, with which the robot has previously experimented, in order to be more capable when first encountering a novel object or in manipulating the same object in a novel manner.

1.1 Thesis Statement

A factored approach to autonomous, interactive learning provides a robot the means to learn what behaviors effectively manipulate specific objects, the capacity to leverage previous experiments in robustly manipulating novel objects, and the power to transfer this learned knowledge for use with other behaviors.

1.2 Motivation

As the goal of having robots operate in uncontrolled environments becomes more critical to the advancement of robotics, robots must be able to discover and effectively manipulate previously unseen objects. This has brought about research on the notion of *affordances* of objects with respect to a robot agent[34]. Within the context of robotics, affordances describe the possible actions an agent can take when acting upon an object along with the associated outcome resulting from the action [47, 68]. Specific

examples might include *graspable* (e.g. [24]) or *pushable* (e.g. [55]) that indicate a particular object can be grasped stably or pushed in a desired manner, respectively. Because one can cast affordances as state-action pairs that will transform the object state in some way, there has been further work in considering affordances as a basis of planning [6]. If the robot has a goal of clearing the path to an object being fetched, it might first push interfering objects to the side, assuming they can be pushed, that is have the affordance *pushable*.

However, while a planner may be able to leverage an abstracted description of the affordance as being true or not with respect to a particular object, such a high level description is not sufficient to actually execute the action required for the affordance. Indeed the method of performing the action may vary by object or object state: pushing a round cereal bowl might be quite different than pushing a TV remote control that has rubber feet that occasionally stick to the table surface. In order to bridge this gap between abstract, task-level descriptions of actions and the underlying continuous valued commands sent to a robot's actuators, we propose the use of affordance-based behaviors. Our affordance-based behaviors encode how a robot should act on an object when attempting to induce a desired outcome. The behaviors are comprised of three components: a perceptual proxy, a control policy, and a behavior primitive. The control policy used by a specific affordance-based behavior instance defines a task space controller that commands the robot actuators as a function of the state of the object and the current robot state. The perceptual proxy computes from the robot's sensory data the state description of the object necessary for the control policy. The behavior primitive constrains how the robot maintains those degrees of freedom not explicitly controlled by the control policy during manipulation. We explain these components in greater detail in Chapter 3.

Affordances provide more benefit to a robot when we examine their relation to

perception. The physical properties of objects, the agent, and the interacting environment determine the affordances present. As such an agent that correctly perceives object properties such as shape, weight, or material composition can accurately predict how an object should behave when acted upon. This has two further implications for perception. First, a robot needs to understand what object or other element of the environment (i.e. pile of stuff, liquid in a container, collection of objects, etc.) it can act on. The task of *singulation* helps in achieving this goal as the robot interacts with the environment to discover the objects present. Second, a robot should infer the how it can best manipulate novel objects based on its previous interactions. Finally, once a robot knows the affordances available to it from objects it regularly sees in its environment, it should become more effective at manipulating those objects. We thus investigate how a robot can specialize the execution of affordance-based behaviors for specific objects in order to be more robust in execution of the planner-level actions.

1.3 Scope

This dissertations investigates how a robot can learn about the elements of its environment and how to manipulate them through autonomous interaction. We examine this problem in the context of pushing objects. We consider our push behaviors to be useful as actions in a task-level planner. While we are fundamentally interested in robots learning autonomously, we believe that much of the work here could be helpful in learning from demonstration, where information about where objects are and how they should be manipulated can be guided by a human teacher.

Additionally, the experiments presented in this thesis are focused on applying affordance learning on man-made objects commonly found in home environments. However, we believe the work would scale nicely to domains such as search and rescue robotics or other emergency robotics fields, where semi-autonomous agents may need to understand what can be manipulated and how best to perform this

manipulation. Consider a semi-autonomous humanoid robot attempting to clear a path through rubble. It may be difficult for a human supervisor to identify the extent and size of individual pieces of stone or wood, while a robot that could autonomously interact with the environment to find individual objects would ease this burden on the user. Additionally, being able to infer and learn about what types of debris could be pushed versus which pieces need to be picked up could allow a robot to learn from its experience, so that a human user guiding the robot need not have expert knowledge of the materials present in the robot's environment.

The remainder of the dissertation is organized as follows. We explain our approach to autonomous discovery of objects through interaction in Chapter 2. Our behavior representation for autonomous affordance learning is presented in Chapter 3. Chapter 4 examines a method for learning how a particular affordance-based behavior, pushing, can be executed based on the properties of an object. A robot learns to predict as a function of shape the best location on an object to push, so that the object moves stably to the desired location. Chapter 5 describes a framework for using models of object dynamics to improve the performance of specific affordance-based behaviors. The chapter goes on to tell how learned knowledge can be transferred between behaviors. We overview the various areas of related research in Chapter 6. Finally, we summarize our problem and our contributions in Chapter 7.

CHAPTER II

HYPOTHESIS GUIDED OBJECT SINGULATION

The ability to detect previously unseen objects and separate them from surrounding objects holds great value for autonomous robots operating in household environments. This ability, termed object singulation, provides the most benefit by allowing a robot to segment novel objects of potential interest from the scene. This is especially true for a robot operating in a home, which will encounter a wide variety of objects during its lifespan.

Singulation provides many benefits to a general purpose mobile manipulator beyond the identification of previously unknown objects. Separating objects form one another can be used to assist grasping methods by creating fewer obstacles for an arm to navigate, while providing more clearance for the robot's end-effector. For uses that desire specific object identification or categorization, separating objects from one another gives recognition systems a better view of the object and provides detection locations avoiding the need to search the entire image.

Additionally, performing object segmentation through pushing allows us to understand objects at the operational level of the behaviors used during singulation. Such an identification is important for tasks such as organizing the objects on a table through pushing, where the available behaviors may not be capable of separating all objects at their semantic level. For example, a basket on a kitchen table may be filled with apples. While object recognition systems could potentially recognize each apple independently, a robot attempting to push the basket does not need this information if the fruit remains inside the basket while its being pushed. This holds particular importance in the context of affordance learning and prediction, where the *pushable*

affordance can be associated with acting on the filled basket, as opposed to *graspable* which may operate on the individual apples as well as the basket.

We propose a method for separating an unknown number of objects on a surface by performing selective pushes in such a way as to disambiguate potential object boundaries. Our method is predicated on the idea that such boundaries tend to produce visual edges in captured images. Thus visual edges represent hypotheses of plausible locations, where a single object cluster may split if it is indeed more than one object. Through successive pushes we accumulate evidence that such edges do or do not correspond to object boundaries. We do so without explicitly tracking edges during the push sequence. Instead, we introduce the use of a histogram of orientations as an aggregated representation of potential boundary edges: only the histogram has to be propagated through the push sequence. Furthermore we design a decision process for systematically testing these hypotheses, which is aware of the constraints of the robot and workspace. Figure 1a illustrates an example initial scene for our system to singulate. Figure 1b shows the robot performing a singulation push.

(a) (b)

Figure 1: Example initial cluttered scene encountered by the robot. Partial singulation result as the robot performs a pushing action.

In this chapter we present our approach to object singulation. The majority of this chapter covers our work on object singulation [41]. Section 2.1 gives an explanation of our approach, followed by a detailed description of the implementation in Section 2.2.

We give extensive experimental results in Section 2.3 including in depth quantitative comparisons and qualitative evaluation for a number of different scenarios. We then conclude with a discussion in Section 2.4.

2.1 Approach

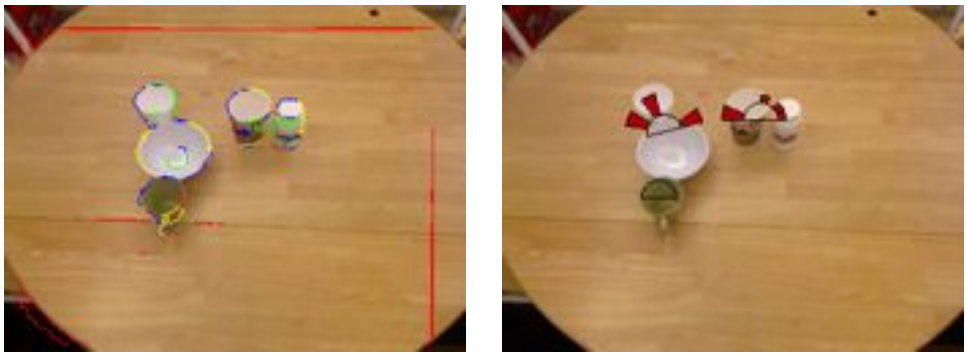
We base our approach on the fact that object boundaries in the world tend to generate visual edges in captured intensity and depth images. While edges in the depth image give the strongest cues as to the location of object boundaries, we can not rely on them alone as object boundaries often do not appear as depth edges, especially when the objects are in contact. As such we use the complementary evidence provided by intensity edges generated by color and texture differences between neighboring objects.

Of course, visual edges also arise as the result of other phenomena such as internal texture on an object surface, cast shadows, or specular reflection. While computer vision techniques have been develop that attempt to classify image edges as object boundaries (cf. [74]), a robot can much more reliably test if an edge corresponds to an object boundary by attempting to separate the objects that would generate such a visual edge. Thus for any edge that corresponds to a potential boundary we determine explicitly where the pair of objects generating the edge would lie in the scene, if such a pair exists. Then for a given object hypothesis the robot determines and performs a push action, which will generate evidence helping to confirm or deny the hypothesis. After each push action we segment the scene in such a way that physically separated objects will belong to separate clusters, but neighboring objects may be grouped together. Thus a push that successfully splits a cluster results in an increased number of clusters from the segmentation procedure. When no new clusters are formed, we have confirming evidence that the cluster may be a single object, but the robot must still test the remaining image edges, to verify that each

of the segmented clusters represent singular objects.

A naive application of this approach would attempt to split the clusters at each candidate image edge in order to exhaustively exclude every such edge from being a boundary between two objects. However, this exhaustive approach has a number of shortcomings. First, edge detection is not stable across illumination changes; some edges will appear or disappear after pushes. This causes great difficulty in tracking individual edges over time. More importantly, pushing actions that attempts to cleave a cluster along a particular edge will likely reveal any object boundaries parallel to that edge;additionally, table friction will often cause objects to begin to separate as long as the push is not directly perpendicular to the separating edge. As such we approximate this procedure by examining the distribution of boundary candidate orientations associated with each object cluster and accumulate evidence by tracking the history of pushes for each cluster. This allows us to achieve singulation with fewer pushes than the total number of candidate edges in the scene.

For each object cluster we quantize all candidate edges by orientation into one of n bins, creating a histogram of edge orientations. Each push performed by the robot is similarly quantized and the appropriate bin in the cluster's push history histogram is incremented. By using this quantization we avoid the need to explicitly track image edges and instead must only repopulate the edge orientation histogram after each push. This relies on the ability to track the transformations undergone by the clusters, so that the push history and edge orientation histograms can maintain consistent alignment with the cluster's internal frame of reference over time. Once the robot successfully pushes a cluster in all populated directions of the boundary orientation histogram, the robot deems the cluster singulated. By recursively applying this procedure to all clusters in the scene the robot asserts that all objects have been separated.



(a) (b)

Figure 2: (a) Example extracted and labeled image edges. Green edges correspond to candidate splitting edges. Yellow edges represent boundary edges between an object and free space. Blue edges have too few 3D points to stably estimate a 3D line, while red edges do not overlap with object clusters. (b) Boundary orientation histograms associated with the current object clusters.

2.2 Implementation

This section details our specific implementation of the general approach described in Section 2.1. We first describe the geometric segmentation and boundary detection methods used to form object hypotheses. We then explain how to generate an abstract push vector for a specific hypothesis. After this we introduce a mechanism for explaining the outcome of the pushing action and how the robot can use this explanation to update the push history for each object cluster. Finally, we detail how the robot translates an abstract push into a real-world action.

2.2.1 Object Hypothesis Generation

The first step in proposing the current set of objects requires segmenting spatially separate regions from one another. The algorithm takes as input a point cloud of the scene and fits a plane to the supporting surface using RANSAC [26]. We then remove these table points and all points below or beyond the edges of the table from the point cloud. The remaining points correspond to objects supported by the estimated plane. These points are grouped based on the euclidean distance between neighbors

into a set of "object clusters."

Each resultant cluster contains at least one separate object. In order to identify potential objects within these clusters we extract edges from the intensity and depth images and associate them with the object clusters. We compute image derivatives both on the RGB and depth images in both the x and y directions using Scharr filters [99]. We threshold the resulting depth and color derivative images and combine them into a single binary edge image. We then remove isolated pixels thin blobs and link neighboring edge pixels in the binary image to form edges. Finally we associate edges with the clusters they fall on, discarding those edges that do not lie on any of the object clusters or have too few 3D points associated with them. Extracted edges are shown in Figure 2a.

Each remaining edge corresponds to a specific boundary hypothesis. We generate the object hypothesis by splitting the object cluster point cloud by a vertical plane defined by the chosen edge's location and dominant orientation. We determine the edge orientation by fitting a 3D line to the set of edge points using RANSAC.

The vector, v, formed in the direction of the x-y component of the 3D line in the table plane defines the direction of the splitting plane. We compute the normal to the splitting plane, n, by taking the cross product of v and the vertical axis vector $(0, 0, 1)$. We then specify the splitting plane as the plane with surface normal, n passing through the point p returned from the 3D line fitting process.

Points from the object cluster are then labeled as belonging to hypothetical object A or B according to which side of the plane they fall on. We discard from consideration edges where A or B have few points, as edges generating such splits correspond to object boundaries on the outside of the object cluster neighboring free space. An example object hypothesis is shown in Figure 3.

The 2D orientation of each vertical splitting plane can be described in the internal reference frame of each object cluster with a single angle in the range $(-\frac{1}{2}\pi, \frac{1}{2}\pi]$

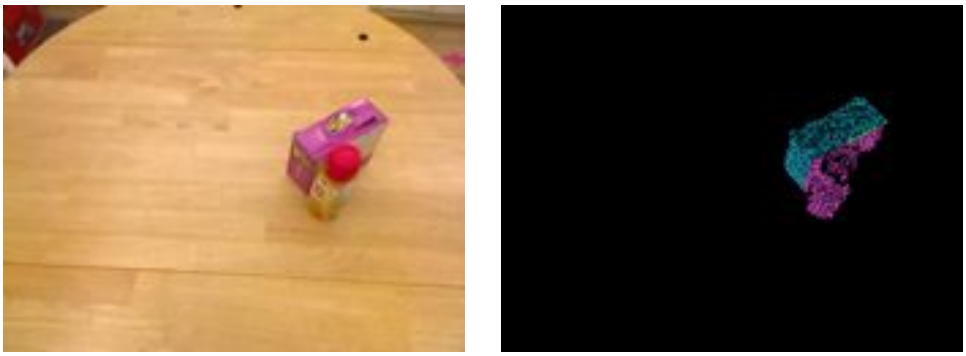


Figure 3: Object hypothesis generated by the image edge highlighted in green.

radians. We compute this angle for all edges associated with a given object cluster. We then construct a boundary estimate histogram for each object cluster by quantizing each edge orientation into one of n bins. Figure 2b illustrates these boundary orientation histograms for a scene with three object clusters.

2.2.2 Push Vector Selection

For any candidate boundary we generate a set of four possible abstract push vectors. We produce push vectors for all candidate boundaries associated with the object cluster of interest. The robot then ranks this set of push vectors and performs the highest ranked push. For a given boundary a push vector points parallel to the splitting plane running through the centroid of either hypothetical object A or B. Two potential push vectors run through each hypothetical object centroid giving us four candidate push vectors. We determine the start location and length of each push vector by locating the intersection of the line through the centroid with the hull of the hypothetical object point cloud. We present one such abstract push vector in Figure 4.

We rank the resulting set of abstract push vectors by preferring those that have start and end locations that do not leave the tabletop or robot workspace. After this, we rank higher those pushes that are less likely to collide with other object clusters. The remaining push options are ranked in decreasing order by the ratio, R_{AB}, between

the number of points in the hypothetical objects:

$$R_{AB} = \frac{\min(|A|, |B|)}{\max(|A|, |B|)}$$

Thus objects with near equal splits score close to 1 and splits that generate one object much larger than the other receive scores closer to 0. We prefer even splitting sizes for three main reasons. First objects in a given environment tend to be of relatively equal size and thus tend to generate an equal number of points in the point cloud split. Second smaller object sizes have less surface area and are thus more difficult to make contact with while pushing. Lastly, even if the generated split does not align with the actual object boundary pushing farther from the middle of the cluster, in the correct direction of an object boundary, will more likely push only one of the two objects present.

Figure 4: Example push vector generated for a single object cluster containing five objects.

2.2.3 Outcome Explanation and Evidence Accumulation

Once a push action is performed the robot first segments the table and clusters the remaining points as described in Section 2.2.1. It can then determine which clusters have moved by taking the difference of the point clouds before and after the push action. We estimate the motion underwent by each moved cluster using iterative closest point (ICP) [11]. ICP generates a rigid body transformation for each cluster

as well as a score describing the goodness of fit between the two input point clouds. We maintain object cluster identification by matching each moved object cluster after a push with the best fitting cluster from before the push, while maintaining uniqueness in correspondence. We compose the most recent transformation estimate with the previous estimates of the cluster's motion, so that its rotation since initialization can be recovered. The recovered transformation is then used to rotate the cluster's internal frame of reference in order to keep the push history aligned over time, as well as correctly populate the edge orientation histogram for the image observed after a push.

Before updating the cluster's push history we must analyze the resultant scene. We do not update the push history if clusters split or merge. If the number of clusters remains the same, then we updated the push history as follows. We first check that the intended object was in fact moved. If this is the case we only update the push history if the fitness score returned by ICP is better than a pre-determined threshold. We avoid updating the history for bad ICP fits as it likely means the push began to disrupt the configuration of multiple objects within a cluster, but was unsuccessful in separating them. Thus another attempt may well break the objects apart. For the case where clusters split, confirming an object boundary hypothesis, we discard the previously split object cluster and initialize two or more new object clusters associated with the split. For each new object cluster we set the push histories to be empty and define their tracked transformations to be the identity matrix. In rare cases clusters will unfortunately merge due to unaccounted for pushing dynamics or accidental collisions of the robot arm with the scene. We take a conservative view of these cases and reinitialize the merged cluster with no push history or tracked rotations.

We acknowledge that this tracking method will produce drift in the object orientation over time. While more sophisticated methods could be used to mitigate this

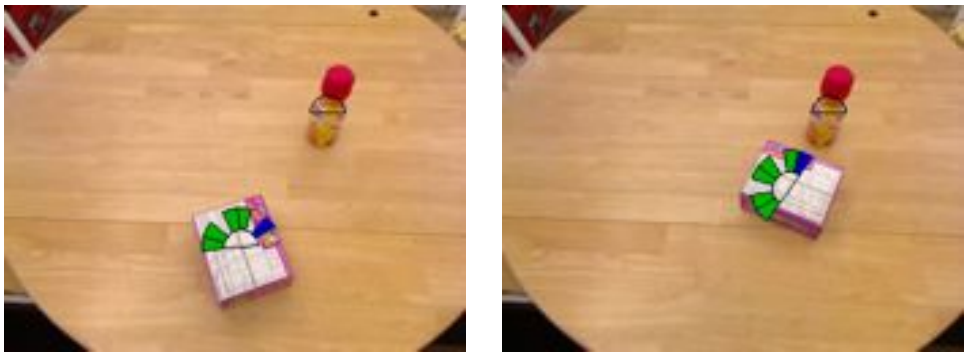
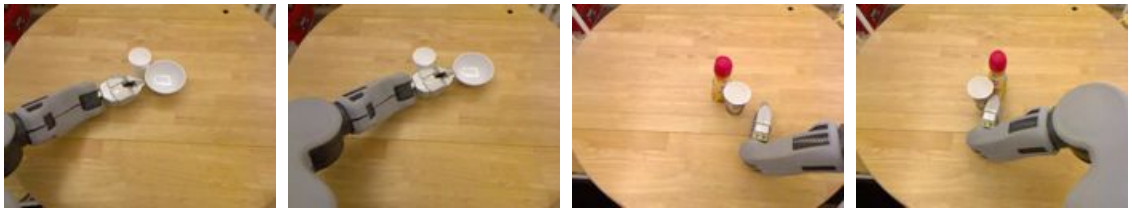


Figure 5: Example push history before and after pushing a cluster corresponding to a single object. The blue bin corresponds to the direction of the next selected push.



(a) *fingertip push* Before (b) *fingertip push* After (c) *gripper sweep* Before (d) *gripper sweep* After

(e) *overhead push* Before (f) *overhead push* After (g) *overhead pull* Before (h) *overhead pull* After

Figure 6: The four implemented push behaviors used by the robot.

issue, we find the precision of the rotational estimate is good enough for our purposes, since we are operating at the relatively coarse scale of quantized boundary and push orientations.

2.2.4 Push Behavior Selection

The robot translates a given abstract push vector into a real-world action using one of four different pushing behaviors. This set of behaviors allows the robot to more effectively accomplish the intended separation of hypothetical objects, when possible,

while accounting for constraints in the environment and its own joint limitations. Each behavior takes as input the abstract push vector defined as a start location (x, y) and push angle θ in the world frame as well as the distance d_p to push.

The *fingertip push* behavior, pictured in Figure 6a-6b, places the tip of the end-effector at the start pose (x, y, θ) keeping the top of the robot hand parallel to the table. *fingertip push* then moves the end-effector the distance d_p along the straight line defined from the gripper wrist to its tip. *gripper sweep*(Figure 6c-6d) uses the broad face of the hand to push objects while remaining close to the table surface. The center of the hand is positioned at the location (x, y) on the table with vector normal to the hand facing in the direction of θ. The arm is then controlled so that the hand moves in a straight line in the direction of the palm for the distance d_p. The third behavior, *overhead push* (Figure 6e-6f), where the robot bends its wrist downward and pushes in the direction of the back of the hand. Again we control the center of the hand to move in the direction normal to its face after positioning it at the (x, y, θ) pose.

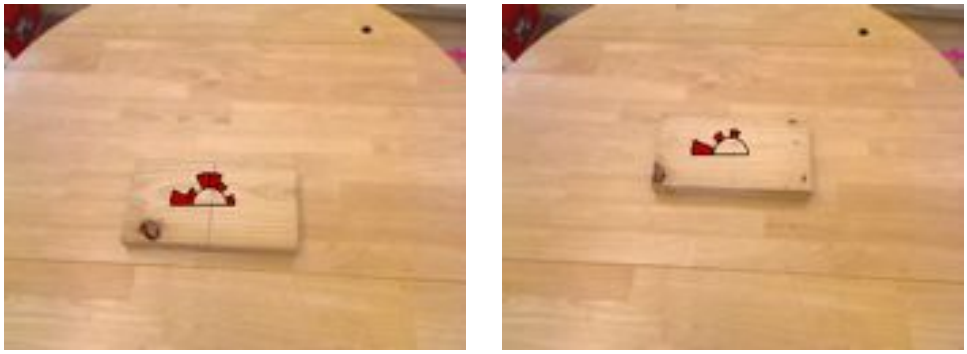
These three behaviors all move to the start pose from a predefined home position following a straight-line trajectory in the work space. The fourth behavior, *overhead pull* (Figure 6g-6h) performs the same action as *overhead push*, except that it will pull towards the robot and navigates to its start pose by first moving to the location (x, y, Z_h), where Z_h is high above the table to avoid colliding with the intended object to pull.

Pushing behaviors are selected based on the input pose (x, y, θ). The *overhead push* behavior is used for initial locations close to the robot, where there is little clearance for the arm to fit between the robot torso and body. For lateral pushing angles the *gripper sweep* behavior is chosen, while *overhead pull* is used for angles point towards the robot torso. *fingertip push* is used in the remaining cases. The decision between using the left or right arm is based on workspace limits of the arms.

If either arm is capable of making the push, the arm is chosen that pushes away from its side in an attempt to keep the workspace less crowded. We note that while we have used pre-programmed push behaviors and behavior selection mechanisms, we are interested in using learned behaviors and behavior selection mechanisms as discussed below in Chapters 3 and 5.

2.3 Experimental Results

We report comprehensive qualitative demonstrations of our method as well as quantitative comparisons to other push selection methods. All of our experiments were performed by a Willow Garage PR2 robot with a Microsoft Kinect camera mounted on the head in the Georgia Tech Aware Home.



(a) (b)

Figure 7: Distribution of detected potential splitting boundaries. (a) A single wood block with a line drawn on it. (b) Two wood blocks placed together.

2.3.1 Qualitative Results

In order to validate our proposed claim and understand its limitations, we first present results on a set of qualitatively different groups and arrangements of objects.

We begin by examining two visually similar situations. Figure 7 shows the initial view of two different scenarios. In one a single block of wood has a line drawn down its center in the other two blocks of wood are pressed together appearing as a single block. We note that the two different scenes generate quite similar images, which would be

hard to discriminate with standard object recognition methods. Additionally, large color and texture overlap exists between the objects and table, making unsupervised segmentation based on low-level visual cues challenging. Beyond this the large flat surfaces would be difficult for most robots to grasp.

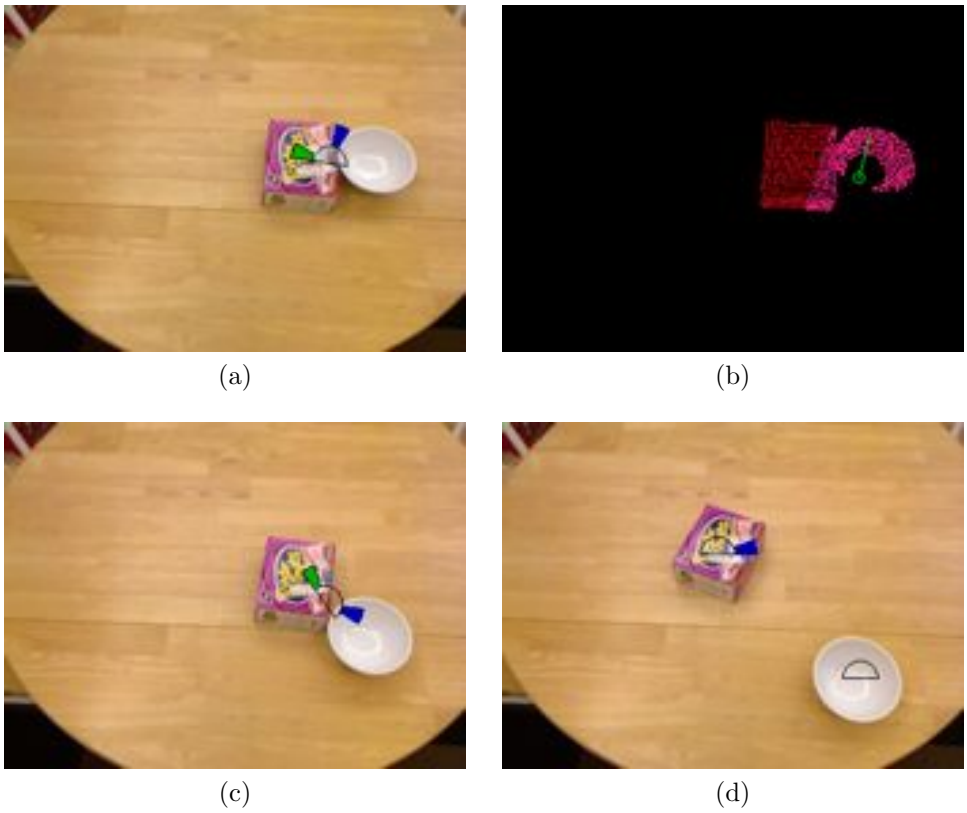
(a) (b)

Figure 8: Push histories for object clusters. (a) A single wood block with a line drawn on it after being determined to be a single object. (b) Two wood blocks correctly singulated after starting placed together.

(a) (b)

Figure 9: (a) Distribution of candidate boundary orientations after one push. (b) Push history accumulated after an additional 13 pushes.

However our method successfully determines that Figure 7a corresponds to a single object, while Figure 7b shows two separate entities. Figure 8 shows the end result and push history of our singulation approach. The robot performed a total of seven pushes on the solid object, as a number of edges were detected on regions of different color in the wood pattern.



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

Figure 10: Push histories for bad singulation pushes. Green shows populated push history directions. Blue represents the direction of the chosen push. (a) After pushing perpendicular to the object boundary a push is chosen (b) that should separate the objects. (c) The objects failed to separate. However, ICP returns a bad score, so the push history is not updated. (d) The next push successfully separates the objects.

A less contrived example is shown in Figure 1. The scene contains five objects in collision that the robot separates with four pushes. Figure 4 shows the first push chosen by the robot to singulate the objects. While the chosen edge to test corresponds to a true object boundary, multiple objects, the bowl and green mug, lie on the other side of the plane defined through this edge. Thus, while the push separates the bowl from the textured coffee cup, other objects remain in contact and must be separated through subsequent pushes. Figure 2 depicts the scene after the initial push. While our method efficiently singulates the objects in this scenario, we note that the majority of objects have low surface texture and as such have few potential object boundaries.

We show an example of textured objects in Figure 9, where the objects are separated on the first push. However, the robot must perform an additional number of pushes in order to accumulate evidence that all remaining boundaries are internal intensity edges and not object boundaries.

Figure 10 shows the importance of only updating the push history after successful ICP alignment. The attempted push in the direction of an actual object boundary does not push far enough to separate the two objects. However the system recognizes this and successfully separates the objects with the next push.

2.3.2 Quantitative Comparison

To test the robustness and efficiency of our method we compare directly to a push selection method, which does not reason about potential objects within object clusters. We compare to an unguided approach, similar to that of Chang et al., that pushes in a random direction through the centroid of a randomly chosen cluster. Pushes are discarded that are expected to push into other objects or out of the robot's workspace. We examine two termination criteria for this method. In the first we push each cluster a fixed number of times. In the analysis below, we term this method "fixed." Below we select the fixed number of pushes to be three per clusters. The second termination method is an attempt to have success with fewer pushes. We halt when the current set of clusters have all most recently been pushed without receiving a bad ICP fitness. We call this second method "rand-ICP." This second method is similar to that of Chang et al., as we rely on confirmation of a rigid body transform to assert object identity. However, rand-ICP is a simplification of their approach, since no image feature correspondences are used to seed the transform estimate and we do not compute an explicit likelihood calculation for classification of the cluster as a single object.

We experiment on a number of varying object sets and configurations. For each

set of objects we randomly generate five configurations for each of the methods. We construct random configurations by generating a random 2D pose in the robot's workspace for each object and then push the objects together so that they are in contact with one another. We classify our different tests by the number of objects present and whether the objects have low or high amounts of texture. For each set of objects we report the number of successful singulation trials. We also show median, mean, and minimum number of pushes executed in all successful trials. Additionally, we note failures due to our segmentation system, where the push selection system is itself not at fault. For all results using our guided approach we report the number of histogram bins used.

Table 1: Results for sets of two objects with low texture. Number in parenthesis is the number of histogram bins used.

Method	Success Rate	Min # Pushes	Median	Mean
Guided (4)	5/5	0	1	1
Guided (8)	4/5	0	1	1.7
Rand-ICP	4/5	2	2	2.5
Fixed	4/5	9	10	11

Table 2: Results for sets of three objects with low texture. Number in parenthesis is the number of histogram bins used.

Method	Success Rate	Min # Pushes	Median	Mean
Guided (4)	5/5	1	4	3
Guided (8)	4/5	1	2	3.5
Rand-ICP	4/5	4	4	4.75
Fixed	3/5	11	15	14.0

Table 1 and Table 2 show results for tests where all objects present had relatively low surface texture. We note that the guided pushing achieved the best singulation success rate when using four histogram bins. The guided approach with eight histogram bins performed as well as rand-ICP, but needed fewer pushes. Fixed performed worse in all respects. The one failure of our method in the two object case (Table 1) was a result of the segmentation claiming that the two separated objects

were actually three. Rand-ICP failed because it incorrectly asserted that the two objects that moved together after being pushed were one. The minimum scores of zero pushes resulted from the segmentation system correctly separating the objects even though they were in contact. Our method was able to recognize that no pushes were necessary, since no edges produced candidate boundaries. We show the estimated boundaries for this case in Figure 11. We note that the random methods would still have to perform at least two pushes in such situations, since they have no estimate of potential objects making up a given object cluster prior to pushing.

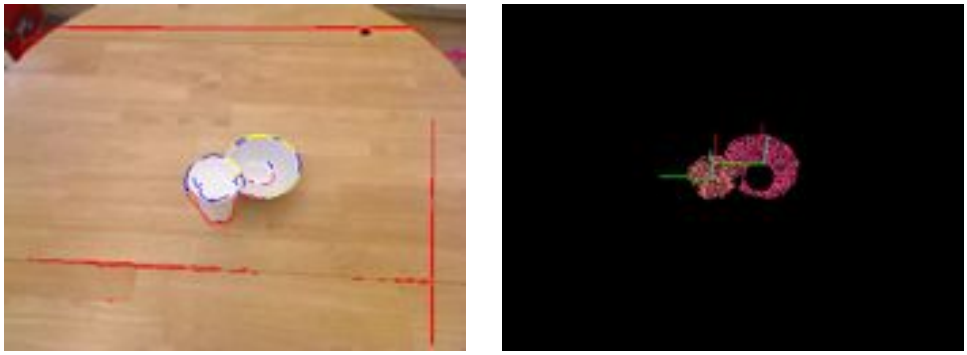


Figure 11: Boundary estimates and segmentation relating to a correct segmentation of objects in contact. Axes represent the internal object frame axes of the two object clusters.

We report results for a set of two objects with high surface texture in Table 3. Guided pushing using four histogram bins had the highest success rate, while guided with eight histogram bins produced success rates equal to rand-ICP method. However the high level of texture required more pushes to be made in order to singulate the objects. We note that all failure cases of our method and all failures of the fixed method were the result of objects being knocked over and falling off the table. One of the two failure cases for rand-ICP was due to the bottle present being knocked over and pushed off the table, but the other was a result of terminating prior to all objects being separated. Thus, while building evidence can increase confidence in the end result, it does so at the cost of performing more pushes that increases the chance

of unintentionally knocking rollable items out of the scene.

Table 3: Results for sets of two objects with high texture. Number in parenthesis is the number of histogram bins used.

Method	Success Rate	Min # Pushes	Median	Mean
Guided (4)	4/5	10	10	11.5
Guided (8)	3/5	14	14	15.3
Rand-ICP	3/5	2	3	2.7
Fixed	1/5	6	6	6

We report results for larger sets of five and six objects in Table 4 and Table 5. Following the success of using only four histogram bins on earlier experiments we do not compare to the guided case of eight histogram bins. For both cases our approach performs the best. In the first, where five low textured objects are used, we have the highest success rate. The fixed method was unsuccessful in all trials, while rand-ICP consistently believed to have segmented all objects and halted early in all but one trial.

Table 4: Results for sets of five objects with low texture. Number in parenthesis is the number of histogram bins used.

Method	Success Rate	Min # Pushes	Median	Mean
Guided (4)	3/5	6	24	18.3
Rand-ICP	1/5	7	7	7
Fixed	0/5	-	-	-

For the challenging case of six objects with varying levels of texture we attain an equal success rate to fixed. However, our guided approach achieves the same performance level with less than half the number of pushes of fixed. Again rand-ICP

Table 5: Results for sets of six objects with varying levels of texture. Number in parenthesis is the number of histogram bins used.

Method	Success Rate	Min # Pushes	Median	Mean
Guided (4)	1/5	12	12	12
Rand-ICP	0/5	-	-	-
Fixed	1/5	30	30	30

regularly halted early, believing the scene to have as few as one object in it and never more than four. Fixed had similar issues, believing only three or four objects to be on the table. Our method failed once from believing only five objects to be present. All other failures of our method were due to items being knocked out of view of the robot during testing.

2.4 Conclusions

We have presented a novel approach to object singulation that explicitly hypothesizes about the orientation and location of potential separation boundaries between possible objects within a single object cluster. This reasoning allows us to state that singulation has occurred more confidently than methods which rely on random robot actions to serendipitously separate objects from one another. We have confirmed through experimentation that our method achieves singulation more reliably than unguided approaches and often does so with fewer pushes. Additionally our method works in situations where all objects present have low texture, a challenge not addressed by previously developed methods.

One shortcoming of our singulation approach comes from the poor performance of our open-loop pushing controllers. We define and examine a number of closed-loop feedback controllers for pushing in Chapters 3 and 5. In Chapter 3 we additionally present a framework in which the robot can learn to choose between the different pushing behaviors, which were determined heuristically in this chapter.

CHAPTER III

A FACTORED BEHAVIOR REPRESENTATION FOR AUTONOMOUS LEARNING OF AFFORDANCES

As the goal of having robots operate in uncontrolled environments becomes more critical to the advancement of robotics, there has been much research on the notion of *affordances* of objects with respect to a robot agent[35]. Within the context of robotics, affordances describe the possible actions an agent can take acting upon an object and the resulting outcome [47]. Specific examples might include *graspable* (e.g. [24]) or *pushable* [55] that indicate a particular object can be grasped or pushed, respectively. Because one can cast affordances as state-action pairs that will transform the object state in some way, there has been further work in considering affordances as a basis of planning [6]. If the robot has a goal of clearing the path to an object being fetched, it might first push interfering objects to the side assuming they can be pushed, i.e. have the affordance *pushable*.

However, while a planner may be able to leverage an abstracted description of the affordance as being true or not of an object, such a high level description is not sufficient to actually *execute* the action required for the affordance. And, indeed the method of performing the action may vary by object or object state: pushing a round cereal bowl might be quite different than pushing a TV remote control that has rubber buttons that occasionally stick to a table surface.

The goal of this chapter is to introduce a mechanism by which a robot determines whether a given object has a particular affordance by systematically exploring *how* that affordance might be successfully achieved. We do this by decomposing an affordance-driven behavior into three components. We first define *controllers* that

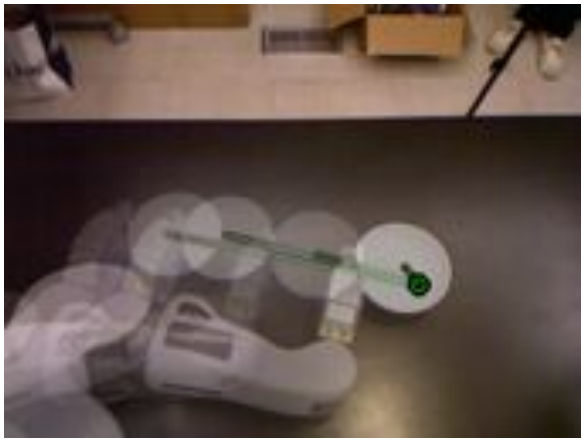


Figure 12: Composition of example frames of the robot performing a successful pushing behavior. The green line denotes the vector from the estimated centroid of the object to the goal location.

specify how to achieve a desired change in object state through changes in the agent's state. For each controller we develop at least one *behavior primitive* that determines how the controller outputs translate to specific movements of the agent. Additionally we provide multiple *perceptual proxies* where each defines the representation of the object that is to be computed as input to the controller during execution. Obviously, the proxy must be sufficiently rich to support estimation of the variables required by the controller. The novelty here is that multiple proxies may support the same controller and a given proxy representation may be selected for use with more than one controller. Additionally, a single behavior primitive may be compatible with multiple controllers. Separating these components into separate factors allows the robot to systematically explore a variety of strategies when determining the affordances of novel objects.

In this chapter we use as examples the affordances of *push-positionable* and *pull-positionable* where the goal is to move the object to a specified location on a table. We develop three different feedback controllers to implement these actions. Each of the two push-positioning controllers uses any of three behavior primitives of *overhead push*, *fingertip push*, and *gripper sweep*. Additionally each controller can choose to

use one of several perceptual proxies. These methods require no prior knowledge of the object being pushed and make no estimates of underlying model parameters. We show how a robot can autonomously determine the effectiveness of a particular affordance-based behavior combination of proxy, controller, and primitive for a given novel object. We examine which methods perform best for fifteen different household objects and explore the success and failure of the approaches as a function of where in the robot's workspace the object is located. Figure 12 shows the robot successfully push positioning a dinner bowl using the affordance-behavior combination of centroid perceptual proxy, centroid alignment controller, and gripper sweep behavior primitive.

The majority of this chapter was originally published in two previous works [42, 43]. We organize the remainder of this chapter as follows. In Section 3.1 we formally define the affordance assertion problem and the push and pull positioning tasks. Section 3.2 presents our three proposed feedback controllers whose effectiveness, as we will see, varies depending upon object. We give details of our implemented perceptual proxies in Section 3.3 followed by the implemented behavior primitives in Section 3.4. A short analysis of the pushing controllers is then presented in Section 3.5. Section 3.6 presents more comprehensive results of over 1500 behavior-controller-proxy trials performed semi-autonomously by a robot using our proposed system. We end the chapter with a short discussion in Section 3.7

3.1 Problem Statement

We define an affordance to exist between a robot and an object, if the robot can select a specific behavior primitive, controller, and perceptual proxy by which it can successfully perform the desired action. We take as example actions those of *push positioning* and *pull positioning*, where the robot must position an object at an arbitrary location by pushing or pulling with its arm. We assume that the object is being manipulated over a plane and thus the object state $X = (x, y)$ defines the

location of the origin of the object in a 2D space. We denote the goal pose as $X^* = (x^*, y^*)$. This state representation is sufficient at the level of a task level planner, however, a specific controller may require more state variables to be estimated by the relevant perceptual proxy.

The (unknown) dynamics of the pushing system are governed by the nonlinear relation $\dot{X} = h(X, Q, U)$ which defines the interaction dynamics between the object state, the robot configuration Q, and the input to the robot U. Importantly, we make no attempt to model h. In developing our visual feedback controllers to achieve the above defined task, we presume we do not have an exact measurement of the object state. Instead we will operate on the estimated state $\hat{X}$ that will be computed at each time step based upon properties of a perceptual proxy. In this work we control the arm through Cartesian control, both position and velocity, in the robot's task frame. We denote the specific forms of U and X used in our controllers in detail below. Our task thus becomes defining a feedback control law $U = g(\hat{X}, X^*)$ which drives the position error $X_{err} = X^* - \hat{X}$ to zero.

3.2 Feedback Controllers for Pushing and Pulling

In this section we define several visual feedback controllers for the robot either to push or pull an object to a desired location. Each controller has a necessary set of state variables to be estimated from the perceptual representation that is continuously updated. These representations serve as the proxies for the object with respect to the defined controllers.

3.2.1 Spin-Correction Control

Our first method of defining a push-positioning controller relies on the fact that the direction of an object's rotation while being pushed depends on which side of the center of rotation the applied force intersects. This fact is well described by the limit surface formulation [75, 36]. Mason derived the velocity direction of a sliding object as

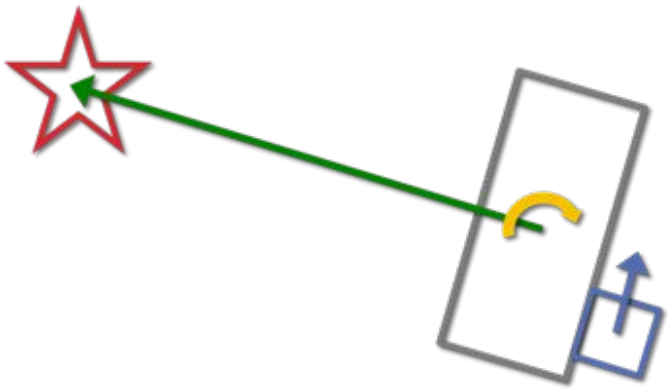


Figure 13: Visualization of the spin compensation feedback control policy. The green arrow depicts the goal oriented term of the controller from the object centroid to the goal (red star). The blue arrow show the term used to move the end-effector (blue) to compensate for the observed rotation the object (yellow). These two vectors are scaled and combined to create the commanded velocity of the end-effector.

a function of the forces applied by the pushing robot as well as the support locations and mass distribution of the object [75]. These parameters are difficult to know or estimate well for a given object and even when they are known, the exact resulting behavior is often indeterminate [75]. However, we make use of Mason's realization that the resulting rotation of the object abruptly changes direction when the input force passes directly through the center of rotation of the object. As such we can use the direction of the observed rotation of the object to infer which side of the center of rotation the applied forces are currently acting through. We can then correct the direction of our applied forces to compensate for any unwanted rotation of the object.

To reduce induced rotation, our controller attempts to push the object through its center in the direction of the goal position. This gives a simple procedure for determining the initial hand position. We cast a ray from the goal location through the centroid of the object and find its intersection with the far side of the object. This location defines the initial position for the hand. We further orient the hand so that its gripper is facing in the direction of the goal from the initial position. An example image of the initial hand placement can be seen in the upper left of Figure 15. Once positioned our feedback control process is initiated. The controller is defined in

equations 1 and 2 which operates on state $X = (x, y, \theta, \dot{\theta})$ and computes input U of x and y velocity of the end effector in the robot's workspace.

$$u_{\dot{x}} = k_g e_{goal_x} - sin(\phi_g)(e_{rot}) \tag{1}$$

$$u_{\dot{y}} = k_g e_{goal_y} + cos(\phi_g)(e_{rot}) \tag{2}$$

Our control is comprised of two terms. The first pushes through the object driving it to the desired goal, while the second displaces the contact location between the robot and object to compensate for changes in object orientation. The input control defined in equations 3 and 4 commands the robot to push in the direction of the goal. The overall effect of this component is controlled by the positive gain k_g. Since the object lies between the end effector and the goal this causes the object to translate towards the goal.

$$e_{goal_x} = (x^* - \hat{x}) \tag{3}$$

$$e_{goal_y} = (y^* - \hat{y}) \tag{4}$$

However, since the forces applied by the robot on the object are not pushing directly through the center of rotation, the object will undoubtedly spin. To compensate for this we apply additional input velocities proportional to the observed rotational velocity of the object. We desire not only that the object not rotate, but also that it maintains its initial orientation θ_0. We combine these terms to generate e_{rot}.

$$e_{rot} = k_{sd}\dot{\hat{\theta}} - k_{sp}(\theta_0 - \hat{\theta}) \tag{5}$$

We desire to displace the end-effector perpendicular to the current direction of the object's translational motion. Since our estimate of the instantaneous velocity is somewhat noisy, we instead rotate the velocity vector about the angle defined between the center of the object and the goal ϕ_g.

$$\phi_g = \texttt{atan2}(y^* - \hat{y}, x^* - \hat{x}) \tag{6}$$

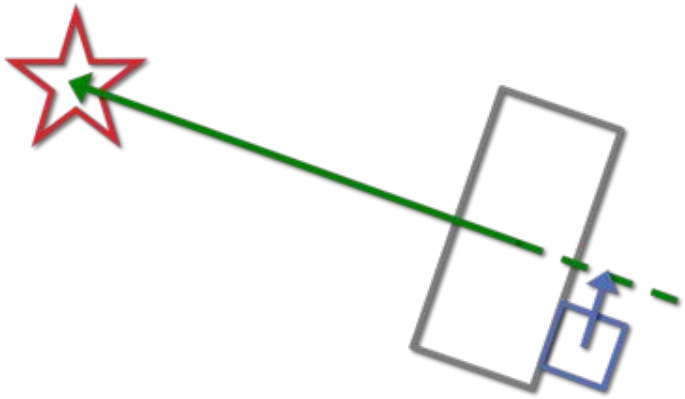


Figure 14: Visualization of the centroid alignment feedback control policy. The green arrow depicts the goal oriented term of the controller from the object centroid to the goal (red star). The blue arrow show the term used to align the end-effector (blue) with the center of the object. These two vectors are scaled and combined to create the commanded velocity of the end-effector.

Our pushing controllers halt once $x_{err} < \epsilon_x$ and $y_{err} < \epsilon_y$. For the purpose of developing this method as well as the controller in Section 3.2.2, the gains are manually adjusted, but remain fixed for all objects.

3.2.2 Centroid Alignment Control

Our second push-positioning controller replaces the monitoring of object orientation with a strategy based upon the relative locations of the object's centroid, the assumed location of the contact point on the end-effector, and the goal position. The simple intuition is that pushing the object can be achieved by positioning the end-effector at a location on the object boundary that intersects a line between the goal location and the object centroid. We visualize this control law in Figure 14.

The robot achieves this behavior by using a control law that includes a velocity term to move toward the goal and one that moves the end-effector to the line defined through the goal centroid locations:

$$u_{\dot{x}} = k_{gc} e_{goal_x} + k_c e_{centroid_x} \tag{7}$$

$$u_{\dot{y}} = k_{gc} e_{goal_y} + k_c e_{centroid_y} \tag{8}$$

where e_{goal_x} and e_{goal_y} are as before. The second term provides the additional velocity term toward the goal-centroid line; $e_{centroid_x}$ and $e_{centroid_y}$ are components of perpendicular vector from the presumed end-effector contact point to the goal-centroid line. This error term is computed using the following equations:

$$e_{centroid_x} = T_x - EE_x \tag{9}$$

$$e_{centroid_y} = T_y - EE_y \tag{10}$$

where T defines the point on the line passing through the centroid and goal locations closest to the end-effector pose

$$T_x = \hat{x} + m \cdot x_{err} \tag{11}$$

$$T_y = \hat{y} + m \cdot y_{err} \tag{12}$$

and m is the slope of the goal to centroid line defined as

$$m = \frac{(EE_x - \hat{x}) * x_{err} + (EE_y - \hat{y}) * y_{err}}{\sqrt{x_{err}^2 + y_{err}^2}} \tag{13}$$

The robot then pushes in the direction of the goal attempting to maintain this collinearity relation. This controller has the state $X = (x, y)$ and computes the same U as in Section 3.2.1. Additionally, the end-effector is initially positioned relative to the object as above.

3.2.3 Direct Goal Pull Control

We implemented a single feedback control law to be used with pulling objects. The controller assumes the object is already grasped by the gripper and simply moves with a velocity proportional to the direction of the goal from the current object centroid. The gripper is placed following a similar procedure to that used in pushing. However, the initial hand placement is chosen to be at the closest location on the object to the goal position. The gripper is opened prior to moving to this initial pose. The gripper then moves forward to surround the object and closes to grasp it. Upon halting of

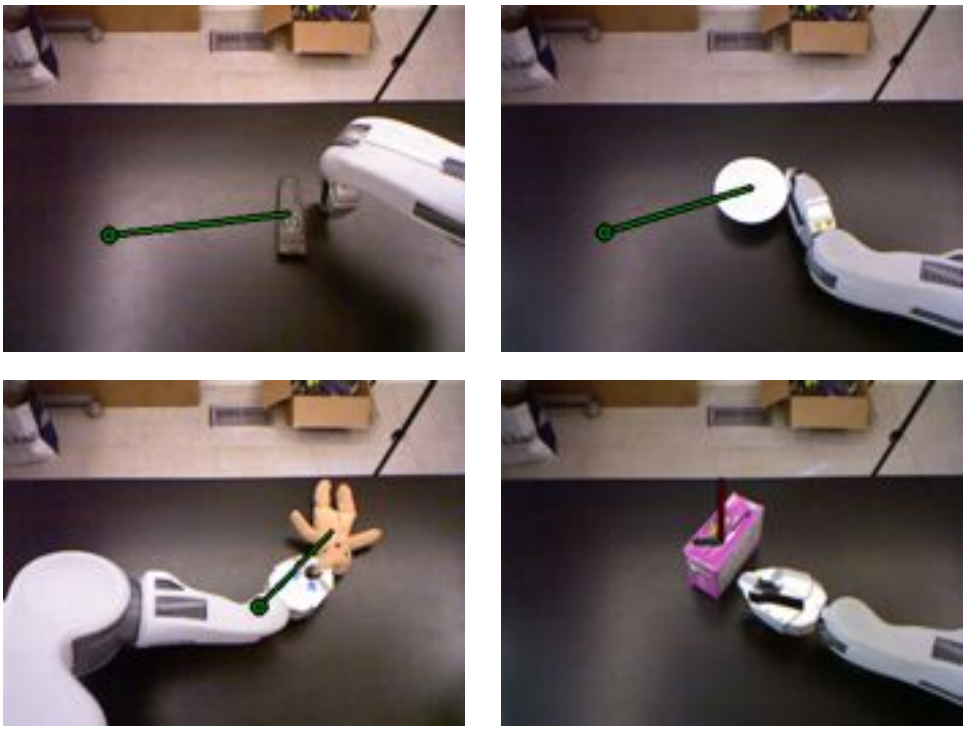


Figure 15: Array of behavior primitives. The first image shows the overhead push behavior primitive placed prior to pushing the television remote. The second image show the gripper sweep behavior primitive pushing the dinner bowl. The lower left image shows the teddy bear being pulled. The first three images show the vector from current centroid estimate to goal location. The food box is being pushed with the fingertip push in the final image. The first three images have drawn in green the vector from estimated centroid to goal location. The box has the estimated centroid location overlaid.

the controller, the gripper opens to release the object and moves backwards to clear the object prior to returning to the ready position.

3.2.4 Controller Monitoring

During execution of each feedback controller we monitor for certain conditions where execution should be aborted. The simplest of these is when no object has been detected. To avoid being stuck in strange configurations, we abort execution when neither the arm nor the object has moved after a short period of time. To keep the robot from manipulating free air execution stops when the robots gripper moves farther than some predefined threshold from the estimated object centroid. Finally, we halt execution for both push controllers when the estimated object centroid is not between the gripper and goal locations.

3.3 Perceptual Proxies

The above defined controllers have modest perceptual requirements. The orientation-velocity controller requires both the location of the object and its orientation whereas the centroid-driven push controller and pulling controller require only position as defined by the object's centroid. Here we describe the perceptual computations performed and the proxies that satisfy the requirements.

We begin with a simple depth-based segmentation and tracking method that currently assumes only a single object resting on the sliding surface (a table) is in the scene. The input is the RGB-D image of a Microsoft Kinect though in this simple implementation only the depth channel is used. We initialize the tracker by moving the robot's arms out of the view of the camera, capture the depth image and then use RANSAC [26] to find the dominant plane in the scene parallel to the ground plane. We then remove all points below the estimated table plane and cluster the remaining points. We filter out clusters with very few points and, because we're assuming only

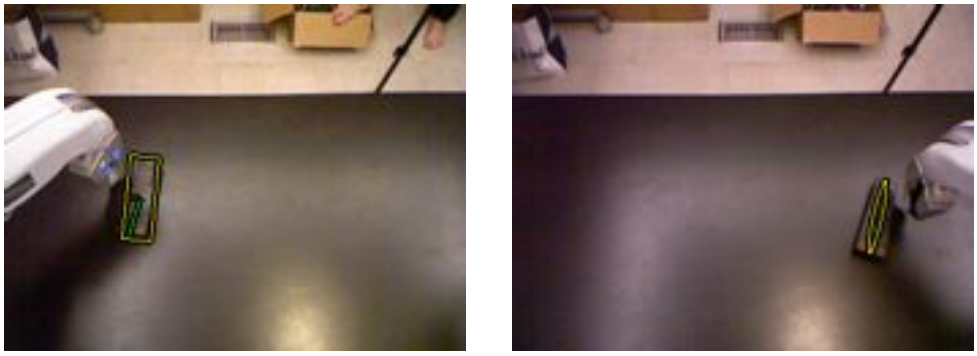


Figure 16: Examples of the robot performing pushing using feedback from the tracking. The left image shows the state estimated with the bounding box perceptual proxy. The right image displays the estimated ellipse.

one object is on the table, we accept the cluster with most points as the object.[1] We compute the 3D centroid of the points in the cluster and use the x and y components as the object's location on the table.

Once initialized we track the object by performing the same procedure with the added step of removing points belonging to the robot from the scene. We project the robot model into the image frame using the forward kinematics of the robot and remove points from the point cloud coincident with the robot arm mask. Because of noise in measurements and other calibration issues points belonging to the robot can sometimes remain. To prevent the tracker from selecting any of these points as the current object we perform nearest neighbor matching between current cluster centroids and the previous object state, selecting the closest as the current object. We then estimate the object velocity using the previous estimate of the object state.

Computing the perceptual proxies needed for each of the controllers is straightforward given the tracker described above. For the centroid based control methods, centroid alignment and gripper pull, we can immediately return the x and y values computed from the centroid of the object point cloud as input. We name this the centroid proxy. However, this estimate may not be the most accurate representation

[1] We note that we [41] and others (e.g. [15]) have previously developed methods for singulating objects form each other by pushing actions.

of the objects actual centroid, due to occluded regions of the object and non-uniform mass distributions. As such we implemented two additional proxies to estimate the centroid of the object. The first alternative approach uses RANSAC to fit a sphere model to the object point cloud. We then use the x and y estimates of the sphere's center as input to the controller. This allows for partially occluded spherical objects to have a more stable estimate of the object's center. Our other estimate approach fits the minimum area bounding box to the 2D footprint of the object. We then use the center of this bounding box as the object centroid. While this proxy is not robust to occlusion by the robot, it produces a result that is just a function of the convex hull of the point cloud and is not influenced by how many points occur in any single region of an object's interior.

For the orientation-velocity control we need a proxy that includes an estimate of object orientation, as well as its rotational velocity, with respect to the global robot frame. The bounding box proxy can be used to give us an estimate of the orientation. We simply set the orientation of the object to be the dominant axis of the bounding box. As an alternative proxy we fit a 2D ellipse to the x and y values of all points in the object point cloud and use the orientation of the major axis of the ellipse as the objects orientation θ. In both cases, the change in θ from one frame to the next is the estimated orientation velocity $\dot{\theta}$. Example of computed bounding box and ellipse proxies are shown in Figure 16.

3.4 Pushing and Pulling Behavior Primitives

We performed pushing with three behavior primitives: an overhead push, a gripper sweep, and a fingertip push. The overhead push has the robot place its hand such that the fingertips are in contact with the table with the wrist directly above. The gripper sweep places the length of the hand on the table with the flat of the hand facing the object. The fingertip push keeps the palm of the hand parallel to the

table and pushes with the tip of the gripper pointing in the push direction. As our controllers operate only within the 2D pose of the hand (x, y, θ), the configuration of the end-effector with respect to the arm and object remain fixed during operation. Specifically this means that the wrist remains above the hand throughout pushing for the overhead push. Likewise the gripper sweep keeps the long side of the robot hand along the table with the broad side of the hand perpendicular to the surface during pushing and the fingertip push keeps the palm parallel to the table. Images of the robot operating with these behavior primitives can be seen in Figure 15.

For all primitives the arm is moved to the initial pushing pose using Cartesian position control. The arm is first moved to a position directly above the table at the desired pose and desired orientation. The hand is then lowered in a straight line to the initial pushing pose. We use a Jacobian inverse controller to control the Cartesian velocity of the end-effector during feedback control.

The gripper pull behavior primitive is similar to the fingertip push primitive, except that the gripper is first opened prior to moving to the initial pose. The gripper then moves forward to surround the object and closes to grasp it. Additionally the initial hand placement is determined to be between the object and goal not behind the object, as in pushing.

3.5 Controller Evaluation

We examine the robot's manipulation performance with different combinations of proxies, control laws, and behavior primitives in pushing a television remote, a pink food box, and a dinner bowl. In all experiments $\epsilon_x = \epsilon_y = 0.05$ meters.

3.5.1 Goal Position Controller Evaluation

We first show an example of pushing a television remote using the overhead push controlled by the spin compensation controller. The perceptual proxy used is the ellipse model. The TV remote has a rather complicated set of support points and far

(a) (b)

Figure 17: The top and bottom of the television remote. The support distribution of the remote is much more complex than a simple polygon. Additionally the narrow end is significantly more massive than the wider end owing to the batteries inside.

from uniform mass or friction distributions. We show an up close picture of the remote in Figure 17. The tracked trajectory of the TV remote as well as the pose errors are shown in Figure 18. Midway through the pushing trajectory the remote becomes partially occluded by the robot arm which causes a jump in the estimated position. Figure 20 shows the controller compensating for this change which induces a larger velocity in the object, including its rotation. We show the velocities for the remote in Figure 21. Note that after the increased rotational velocity the controller most apply larger input velocities to maintain the initial orientation and continue pushing towards the goal. Regardless, the remote control converges within the desired bounds of the goal pose and the execution is successful.

We now show that this same affordance-based behavior instantiation of overhead push, ellipse proxy, and spin compensation controller can be used for a different object of a simple food box. The robot successfully pushed this box using the same behavior components as with the television remote. The results for a trial with this setting are shown in Figure 19.

We investigate the same manipulation settings of overhead push, ellipse proxy, and spin compensation controller for pushing a simple, white dinner bowl. We show

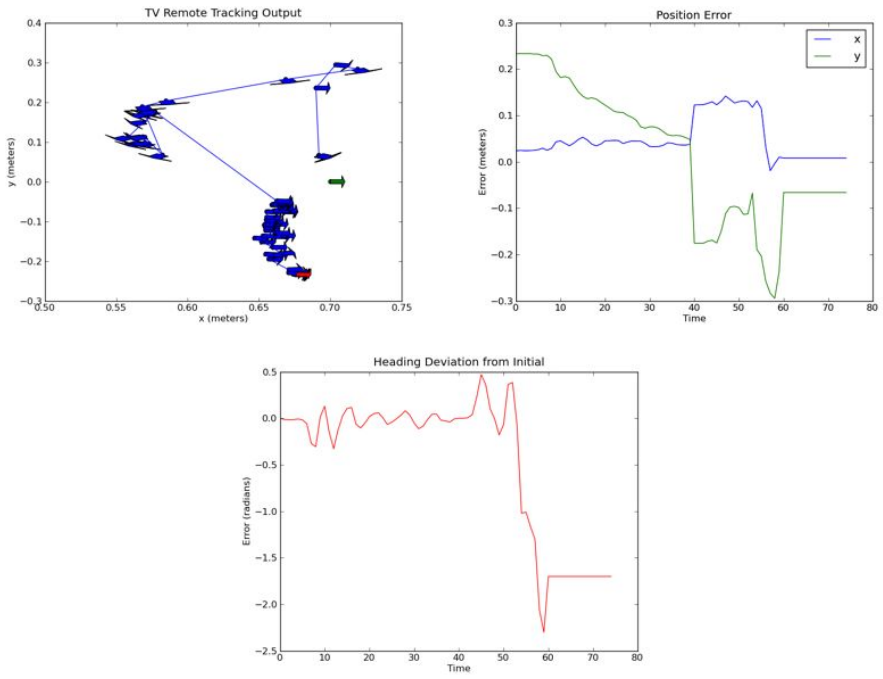


Figure 18: The first image shows the tracked TV remote pose trajectory. The red error shows the initial pose. The green arrow is the goal pose. The large jump in error near time 40 is a result of the TV remote becoming partially occluded by the robot arm, which results in poor visual tracking performance.

the robot pushing this bowl in Figure 15. The position error for the bowl and the input velocities during control are shown in Figure 22. We show the tracker output and rotational velocity estimates of the bowl in Figure 23. Applying this method to the bowl fails to push the bowl to the desired location. This failure can be attributed to the symmetric appearance of the bowl, which causes instability in estimating the object's orientation by the ellipse perceptual proxy. However, by pushing the bowl with the overhead push controlled by the centroid controller the robot can correctly position the object. We show error results and input velocities for these settings in Figure 24.

Following the success of the centroid controller in pushing the bowl, we investigate its use with the overhead pushing behavior primitive to push the television remote. Unsurprisingly, the centroid controller quickly loses contact with the remote since the

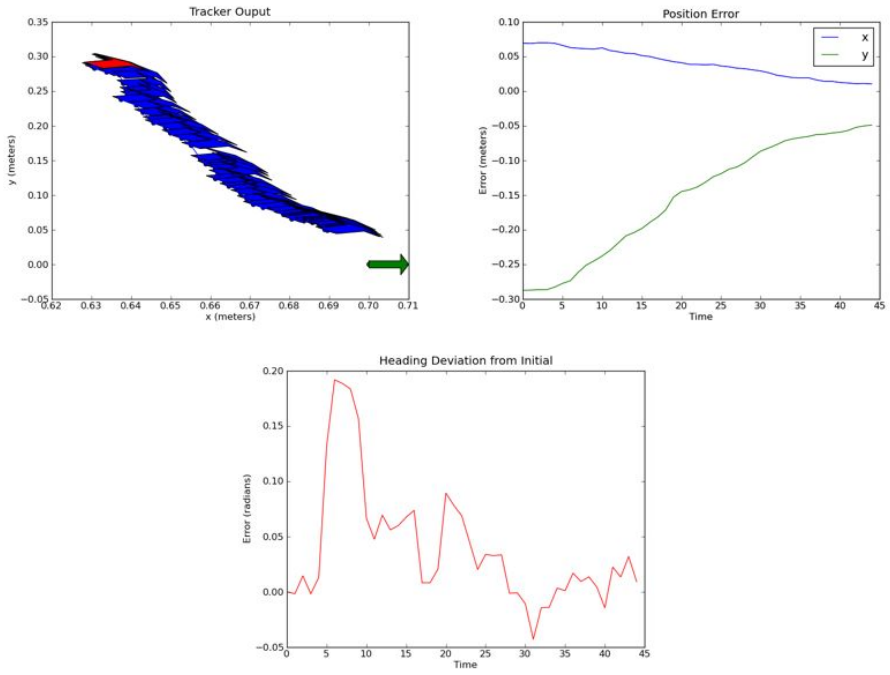


Figure 19: The first image shows the tracked box pose trajectory. The red error shows the initial pose. The green arrow is the goal pose. The second image is error in the food box position and the third shows change in orientation from the initial orientation.

visually estimated centroid is not the center of rotation and trying to push in line with it fails to compensate for the object's rotation. Figure 25 shows position errors and input velocities from the experiments.

3.5.2 Behavior Primitive Evaluation

We now examine using the gripper sweep behavior primitive with the controller proxy pairs. Following the success of positioning the TV remote with the spin compensation controller and overhead push, we tried the same setup with the gripper sweep behavior primitive. This performed quite poorly. Partially at fault was the occlusion of the remote by the arm causing unstable state estimates. Additionally the spin compensating control input, v_{rot} caused somewhat volatile control of the sweeping end-effector, that was much smoother with the overhead push. This could have perhaps been fixed by changing controller gains, however, we did not investigate this.

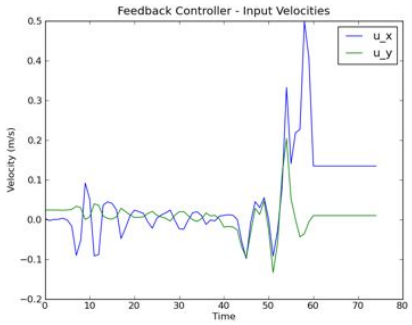


Figure 20: Plot of the input velocities to the arm controller commanded by our feedback controller during pushing of the TV remote.

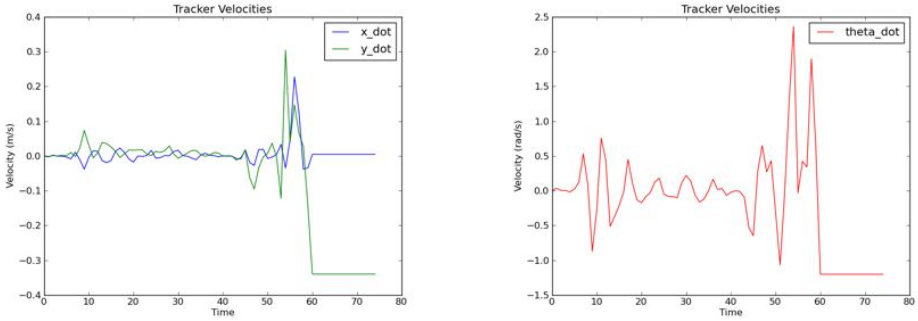


Figure 21: Plot of the tracked object velocities for the TV remote.

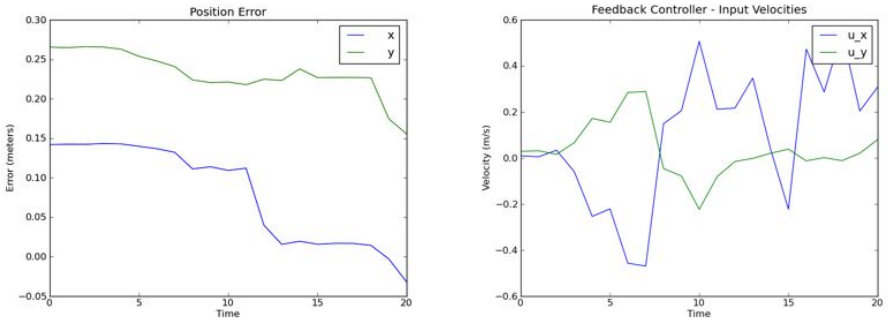


Figure 22: Position error and input velocities for pushing the bowl using the spin compensation controller with the overhead push.

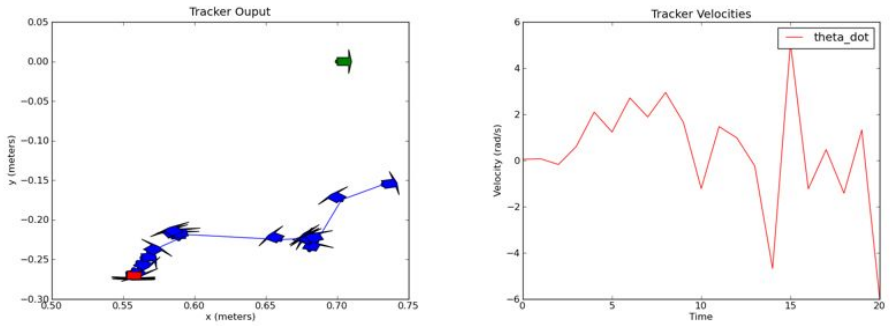


Figure 23: Position and orientation estimates as well as rotational velocities estimates for pushing the bowl using the spin compensation controller with the overhead push.

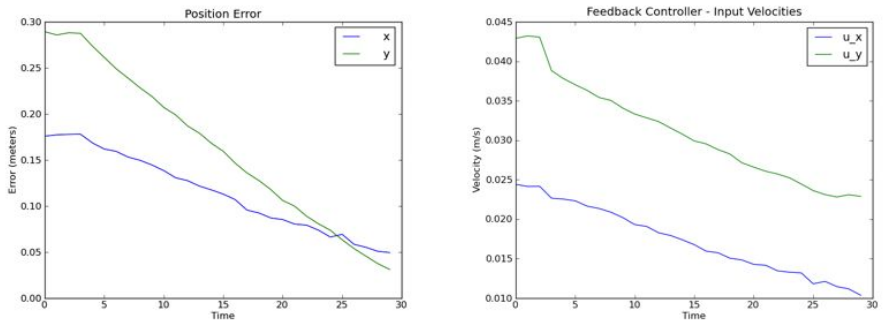


Figure 24: Position error and input velocities for pushing the bowl using the centroid controller with the overhead push.

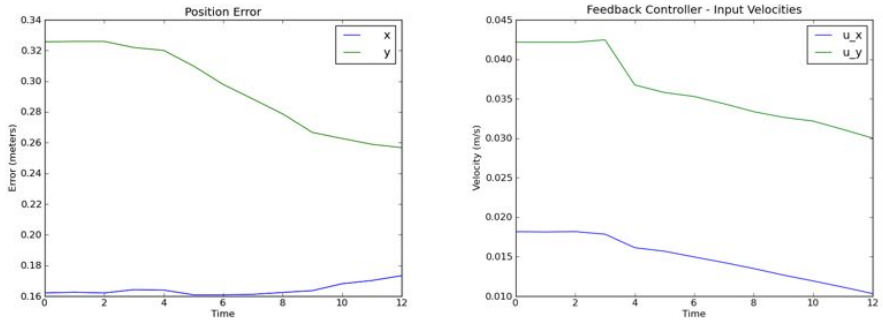


Figure 25: Position error and input velocities for pushing the television remote using the centroid controller with the overhead push.

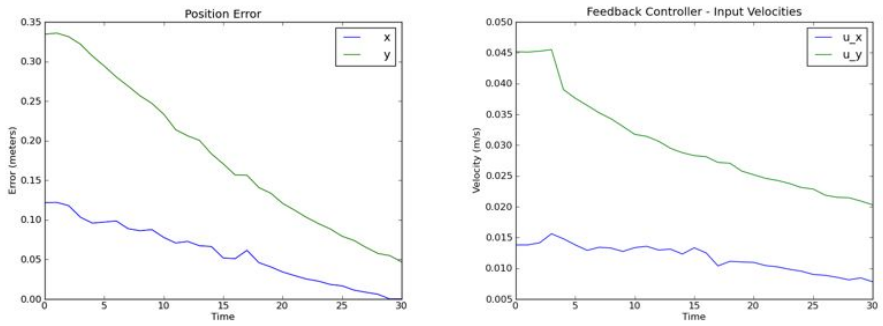


Figure 26: Position error and input velocities for pushing the bowl using the centroid controller with the gripper sweep.

Figure 27: The fifteen objects on which the robot performed experiments.

We also examined pushing the bowl using the centroid controller and the gripper sweep. This method successfully positioned the bowl. We show the position error and input velocities in Figure 26. The error results and control velocities were quite similar to those seen in pushing with the overhead push.

3.6 Experimental Validation

We experimentally validate our approach by having a mobile manipulator explore the possible combinations of affordance-based behavior actions over a set of 15 household objects, each displayed in Figure 27. For each object the robot attempted at least one left arm and one right arm push or pull action with every possible combination of proxy, behavior primitive, and controller. This produces a set of thirty-six possible

affordance-based behaviors.

For a given object we place it at an initial random position on the table. The robot generates a random goal position at least 0.2 meters from the current location and attempts the first instantiated affordance-based behavior. If the execution succeeds the robot generates a new goal pose as before and attempts the next behavior instantiation with the object positioned at its current location. If the controller aborts execution prior to reaching the goal the robot will reattempt the current combination up to two additional times. If the goal is not reached after the third attempt the robot moves on to the next affordance-based behavior combination. If the object is knocked off the table or out of a predefined workspace for the robot the robot asks a human operator to replace the object and continues exploration. If upright objects were knocked over during the exploration the robot continues to attempt the current affordance-based behavior. However once the robot completes the current trial, either by being successful or by aborting three times, the human operator pauses the search and returns the object to its canonical pose.

We implemented our system on a Willow Garage PR2 robot augmented with a Microsoft Kinect for visual input. In all experiments $\epsilon_x = \epsilon_y = 0.01$ meters. Below we show detailed results of the affordance-based behavior exploration consisting of more than 1500 total push/pull trials.

3.6.1 Object Affordances

For any given object we would like to know the best affordance-based behavior to use to effectively move it when a given task demands. Here we define the best affordance-based behavior to be the best choice of controller, perceptual proxy, and behavior primitive which together produce the lowest on average final position error. Here we average over trial and workspace location. The best choice of affordance-based behavior combination for each object along with it statistics is presented in Table 6.

Table 6: Lowest on average final position error affordance-based behaviors for the fifteen objects.

Object Name	Primitive	Proxy	Controller	Mean Score (meters)	Variance
Plate	Gripper Sweep	Centroid	Centroid	0.085	1.99e-05
Towel	Overhead Push	Centroid	Centroid	0.036	5.40e-05
Hair Brush	Overhead Push	Centroid	Centroid	0.136	0.001
Toothpaste	Overhead Push	Bounding Box	Centroid	0.126	0.015
Food Box	Gripper Sweep	Centroid	Centroid	0.056	0.002
Shampoo	Overhead Push	Bounding Box	Centroid	0.159	0.032
Telephone	Overhead Push	Bounding Box	Spin Comp.	0.032	5.21e-05
Soap Box	Overhead Push	Bounding Box	Spin Comp.	0.086	0.003
Mug	Overhead Push	Centroid	Centroid	0.034	2.69e-04
Medicine Bottle	Overhead Push	Bounding Box	Centroid	0.032	9.12e-05
Teddy Bear	Overhead Push	Bounding Box	Centroid	0.083	2.15e-04
Red Bottle	Overhead Push	Bounding Box	Centroid	0.084	0.001
TV Remote	Overhead Push	Bounding Box	Centroid	0.147	0.007
Bowl	Gripper Sweep	Centroid	Centroid	0.020	3.52e-05
Salt	Overhead Push	Centroid	Centroid	0.015	1.02e-04

We note several results: First, we see that the overhead push with the centroid controller performs best on average for most objects. Only the telephone and soap box, both of which tend to rotate when pushed, found better average performance with the spin compensation controller. Additionally the chosen behavior primitives were all either overhead push or gripper sweep for these objects. We believe this to be the case, as the fingertip push and gripper pull operate well only in restricted areas and angles of the workspace due to the constraint on the hand pose. We see that there is a nearly even split between the use of bounding box and centroid as perceptual proxy. The sphere is a much more specialized proxy that only works well on a few objects with mostly spherical shape.

We can gain further insight into the behaviors by examining which affordance-based behavior combinations produced a final goal error below a specified threshold. The results for all objects are presented in Table 7. Here we can examine individual attempts rather than average performance. For example, the TV remote—with it's rubber buttons that grip the surface—could only be controlled by the overhead push

behavior using the spin-compensated controller employing the ellipse proxy. Likewise, the shampoo was quite poorly controlled: only one behavior combination (overhead-push, bounding-box, spin-compensation) ever achieved the goal and then only once out of six attempts. Conversely, the food box could be successfully manipulated using a variety of combinations with only a slight preference for the same control combination as the TV remote. By such detailed experimentation and analysis the robot can develop a set of strategies not only for an initially available set of objects but also, potentially, for novel objects once the robot gains experience with them. For example, once a new object is observed to behave like a mug with respect to several behavior combinations a robot could rapidly develop a strategy for that new object based upon its familiarity with the mug's behavior.

3.6.2 Affordance-Based Behavior Performance as a Function of Object Workspace Location

The above analysis averages performance independent of target location under the assumption that the robot can equally well perform these actions throughout its workspace. Of course, mechanical limitations make certain actions more difficult at different positions. For example, the fingertip push behavior primitive has difficulty in pointing the fingertip towards the robot's torso. Additionally it may be impossible for the robot to reach objects far to the right with the left arm. Knowing which operations perform best in different areas of the workspace is difficult to predict. We would prefer learned models of where to apply certain behavior primitives as opposed to the heuristics previously used (cf. [41]). Having knowledge of which behavior primitive works best in a specific region of the workspace could be helpful in attempting to manipulate a previously unseen object. Behavior primitives that have been seen to consistently fail may be skipped in favor of those more likely to succeed.

To compare the various behaviors we grouped push trials by their starting (x, y) locations as well as the pushing angle, the angle from the initial object location

Table 7: Successful (final error within 0.02m or 0.04m of goal) affordance-based behaviors for each object.

Object Name	Primitive	Proxy	Controller	0.02m	0.04m	# Attempts
Plate	Overhead Push	Sphere	Centroid	0	2	5
	Overhead Push	Centroid	Centroid	0	1	6
	Fingertip Push	Centroid	Centroid	0	1	4
	Fingertip Push	Sphere	Centroid	0	1	6
Towel	Overhead Push	Bounding Box	Centroid	1	1	5
	Overhead Push	Centroid	Centroid	0	4	5
	Gripper Sweep	Ellipse	Spin Comp.	0	1	6
	Overhead Push	Sphere	Centroid	0	1	6
Hair Brush	Gripper Sweep	Centroid	Centroid	2	3	6
Toothpaste	Overhead Push	Bounding Box	Centroid	2	2	6
	Overhead Push	Ellipse	Spin Comp.	1	1	4
	Overhead Push	Centroid	Centroid	0	3	6
Food Box	Overhead Push	Ellipse	Spin Comp.	2	2	5
	Gripper Sweep	Centroid	Centroid	1	1	4
	Overhead Push	Sphere	Centroid	1	1	6
	Overhead Push	Bounding Box	Centroid	1	2	6
	Overhead Push	Bounding Box	Spin Comp.	0	3	6
	Overhead Push	Centroid	Centroid	0	1	6
Shampoo	Overhead Push	Bounding Box	Spin Comp.	0	1	6
Telephone	Overhead Push	Centroid	Centroid	0	3	6
	Overhead Push	Bounding Box	Spin Comp.	0	3	4
Soap Box	Overhead Push	Bounding Box	Spin Comp.	0	2	6
	Gripper Sweep	Bounding Box	Centroid	0	1	6
Mug	Overhead Push	Bounding Box	Centroid	2	2	5
	Overhead Push	Sphere	Centroid	2	2	6
	Overhead Push	Centroid	Centroid	1	2	4
	Gripper Sweep	Bounding Box	Centroid	0	3	6
	Gripper Sweep	Centroid	Centroid	0	1	4
	Gripper Sweep	Bounding Box	Spin Comp.	0	1	6
Medicine Bottle	Gripper Sweep	Sphere	Centroid	3	3	6
	Gripper Sweep	Centroid	Centroid	1	2	5
	Overhead Push	Bounding Box	Centroid	1	5	6
	Gripper Sweep	Bounding Box	Centroid	0	3	6
Teddy Bear	Overhead Push	Centroid	Centroid	3	3	6
	Gripper Sweep	Ellipse	Spin Comp.	0	1	6
	Gripper Sweep	Bounding Box	Centroid	0	1	5
Red Bottle	Gripper Sweep	Centroid	Centroid	1	1	5
	Overhead Push	Centroid	Centroid	0	3	6
	Gripper Sweep	Bounding Box	Centroid	0	1	6
	Overhead Push	Bounding Box	Centroid	0	1	6

Table continued on next page

Table 7 (continued)

Object Name	Primitive	Proxy	Controller	0.02m	0.04m	# Attempts
TV Remote	Overhead Push	Ellipse	Spin Comp.	0	3	6
Bowl	Overhead Push	Ellipse	Spin Comp.	2	3	6
	Overhead Push	Bounding Box	Centroid	2	3	6
	Gripper Sweep	Centroid	Centroid	1	3	3
	Overhead Push	Centroid	Centroid	0	1	4
	Gripper Sweep	Bounding Box	Centroid	0	1	6
Salt	Overhead Push	Bounding Box	Spin Comp.	2	2	6
	Gripper Sweep	Bounding Box	Centroid	2	3	4
	Overhead Push	Centroid	Centroid	1	2	2
	Overhead Push	Bounding Box	Centroid	1	1	4
	Gripper Sweep	Centroid	Centroid	0	2	6
	Gripper Sweep	Bounding Box	Spin Comp.	0	1	6

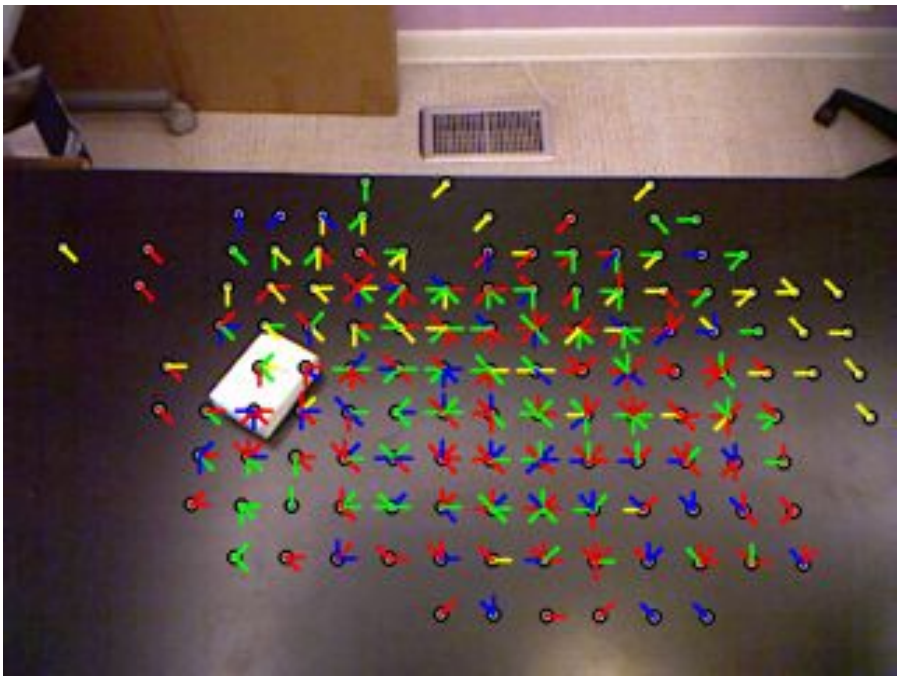


Figure 28: Best on average performing behavior primitives for different initial locations and pushing angles. Green is gripper sweep. Blue is fingertip push. Red is overhead push. Yellow is gripper pull. Bottom of the image is closer to the robot (smaller x). Left of the image is left for the workspace (positive y).

pointing towards the goal position. We quantized initial x and y locations to the closest 0.05 meter value and angles to the closest of eight directions at 45° increments. We visualize the best on average performance at each location and angle between the four behavior primitives in Figure 28. We note how at the edges of the workspace farthest from the robot the gripper pull behavior primitive performs best on average. The robot has discovered that at the farthest extent of its workspace the gripper pull behavior is the most effective choice it can make to push or pull an object to a desired location. We can perform similar analysis to allow the robot to automatically select between the left and right arms.

3.7 Discussion

We have presented a novel behavior representation by which a robot can systematically explore the affordances of objects. Our method allows us to find the most likely to succeed behavior as a function of object instance or location in the workspace. This representation forms the basis for all learning presented in the chapters following below.

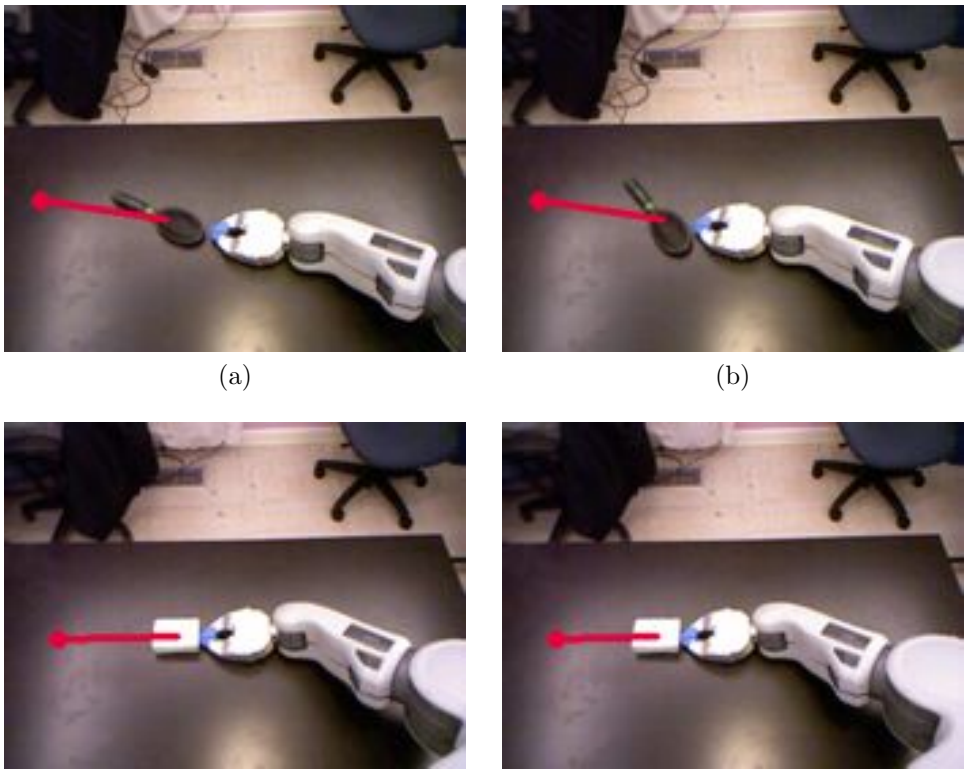
CHAPTER IV

LEARNING CONTACT LOCATIONS FOR PUSHING AND ORIENTING UNKNOWN OBJECTS

The ability to push objects purposefully can be of great utility to robots in performing many tasks of daily life, whether setting a table or searching through a cupboard or drawer. When performing these pushing tasks in real homes and other open, human environments a robot will often encounter novel objects it has never manipulated before. The goal of this chapter is to introduce a learning method by which a robot learns how to predict the effect of pushing actions on novel objects based upon object shape.

The approach developed here may be considered as an alternative to complete physical simulation. Physical models require specification of typically unobservable properties of the object such as support locations and friction distributions. For an unknown object these properties cannot be deduced without interactive experimentation. Even if a physical model is available, simulation may not be sufficient in solving the pushing control problem since many efficient solutions require simplifying assumptions of both the object and the supporting surface [75, 72, 94].

As an alternative to using a complete physical description, a robot can compute visual cues directly from camera input. Object shape encodes valuable information about effective pushing locations. In this chapter we develop a method for autonomously learning a shape-based push-prediction function that can be easily applied to new objects and whose applicability can be quickly ascertained through a small number of experimental manipulations. As an example, consider the scenario depicted in Figure 29a and 29b where the robot is pushing a hair brush. If the contact



(a) (b)

(c) (d)

Figure 29: Example Pushing instances. The first two images are two consecutive frames captured while the robot pushes the large hair brush from an unstable straight-line pushing location, which induces a rotation of the object. The second two frames show the robot pushing a soap box from a stable pushing location. In both examples the red line shows the vector from the estimated object center to the goal location denoted as the red circle.

between robot and object were to move a small amount to the left or right, the object would rotate significantly. Compare this to the example shown in Figures 29c and 29d. In this case a small variation in contact position will cause a relatively minor change in the pose of the object and it will essentially continue along the current pushing direction. Depending on the current task of the robot, it may wish to either push an object to a new location or rotate the object to a different orientation. As such, a robot capable of correctly choosing contact locations for straight-line pushes, as well as orienting pushes on unknown objects will have greater success in pushing objects than a system that does not directly reason about such contact locations.

We use a data-driven approach where the robot learns good pushing locations by interacting with objects during exploration. Each time the robot pushes an object it records both a shape description centered at the push contact location orientated towards the object's center. The robot then pushes the object at the selected location and computes a "push score" measuring the quality of the push. The push score encodes the robot's ability to either push the object along a straight path or to rotate it in a controlled manner. These shape features are extracted in such a way that they capture the necessary details of the object, while being able to generalize to novel object instances as well as novel object classes. As the robot interacts with more objects in more conditions, it uses non-linear regression to learn prediction functions for estimating these two push scores. The procedure for operating on a novel object or a previously seen object is identical, allowing the robot to seamlessly deal with new and old objects alike.

When presented with an object, whether new or previously encountered, and a desired pose the robot extracts shape features from all locations on the object boundary and uses the learned prediction function to estimate push-stability and rotate-push scores for each of these locations. The robot can then use a high scoring orienting-push location to rotate the object to a pose, such that a sufficiently good stable push location aligns with a straight line trajectory to the goal pose. The robot then performs the orienting push and straight line push, using the same feedback controllers as from training, to position the object as desired in its workspace.

This majority of this chapter has been published as the works [39, 38]. The remainder of this chapter continues as follows. Section 4.1 gives details of the scoring function used for the learning task, our shape features, and the method used for regression. We describe implementation details in Section 4.2. We validate our learning method offline in Section 4.3 and present the results of all robot experiments in Section 4.4. Finally, we conclude and discuss this work in the broader context of our

research program in Section 4.5.

4.1 Learning Task

We formulate our learning task as the estimation of a function $f : \mathbb{R}^m \rightarrow \mathbb{R}$, given n training example pairs $(\mathbf{z}_i, s_i)$, $i = 1, \ldots, n$, $\mathbf{z}_i = [z_{i1}, \ldots, z_{im}] \in \mathbb{R}^m, s_i \in \mathbb{R}$. All training pairs are collected autonomously by the robot through exploration with given training objects. The target value s encodes the quality of the push, while the input feature $\mathbf{z}$ encodes the overall shape of the object of interest and the local shape for the currently chosen pushing contact location. In the remainder of this section we first describe the details of the scoring function used for straight line pushing, as well as the scoring method for orienting pushes. We then explain the shape features used and finish the section with an overview of support vector regression (SVR), which performs the estimation of the function f.

4.1.1 Pushing Score Functions

We define our push-stability score to penalize pushes which deviate from the desired straight-line trajectory. We thus compute this score as the average distance of the observed object trajectory from the desired pushing trajectory. Equation 14 precisely defines this notion:

$$s_s = \frac{1}{K} \sum_{k=1}^{K} \texttt{dist}(\mathbf{X}[k], \ell_p) \tag{14}$$

where $\mathbf{X}[k]$ is the estimated (x, y) centroid location of the object in the table frame at time k, ℓ_p is the line passing through the objects initial location $\mathbf{X}[0]$ and the desired goal location $\mathbf{X}^*$, and `dist` is the Euclidean distance.

Figure 30 visualizes the scoring function for two synthetic trajectories. While both trajectories reach the desired goal location, the green trajectory follows the desired straight line trajectory more closely. The computed scores $s_s^0 = 0.62$cm for the green trajectory and $s_s^1 = 3.94$cm for the red trajectory reflect our preference with the green

trajectory receiving the lower score. There are a number of reasons to prefer this scoring measure to something simpler such as final position error. First, it allows the robot to have a more accurate prediction of how the object will behave when pushed. This is important for collision avoidance, where straight line push trajectories allow the robot to avoid pushing an object into other objects in the environment or off of the supporting surface. Additionally, the score allows the robot to predict how an object should move, allowing the robot to abort pushes early if they deviate significantly from the straight line path.

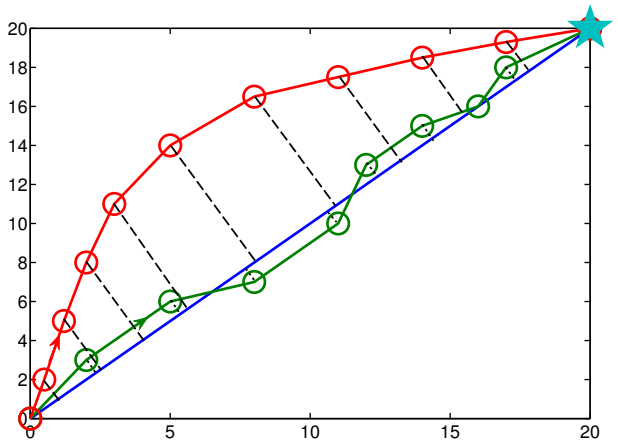
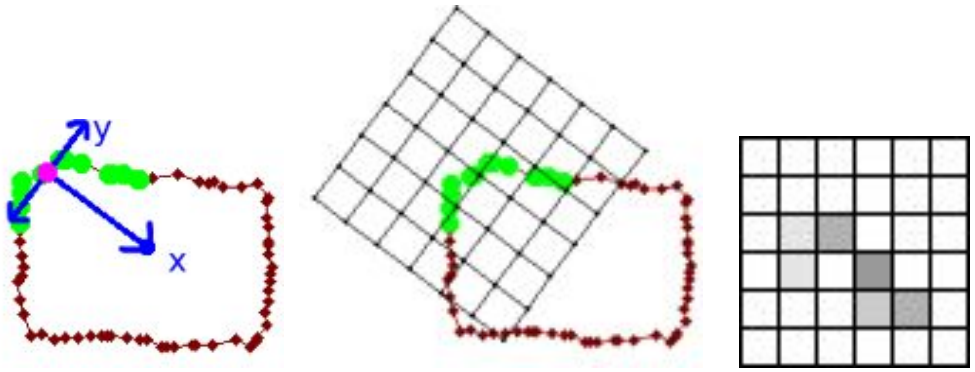


Figure 30: Two synthetic push trajectories going from initial location in the bottom left to the goal location (star) in the top right. The red trajectory receives a push-stability score of 3.94cm while the straighter green trajectory receives a score 0.62cm.

For the task of learning where to push in order to rotate an object to a desired heading we used the simple score of net change in orientation between the initial pose and final pose. Ideal orienting pushes should not only rotate the object, but also produce as little translation of the object position as possible. While this score can not differentiate between pushes which rotate the object while keeping the center fixed with those that also move the object, we found the simple scoring function sufficient for differentiating good and bad pushing contact locations. This reflects the nature of the feedback controller we used for orienting pushes, which tends to only cause objects to rotate, when they do not translate (c.f. Section 4.2.2). Thus penalizing translations

explicitly adds unnecessary complication. Formally the orienting push score is $s_r = |\theta[K] - \theta[0]|$ where $\theta[K]$ is the object's orientation defined in the supporting plane at final time index K and $\theta[0]$ is the orientation at the initial time step prior to pushing.

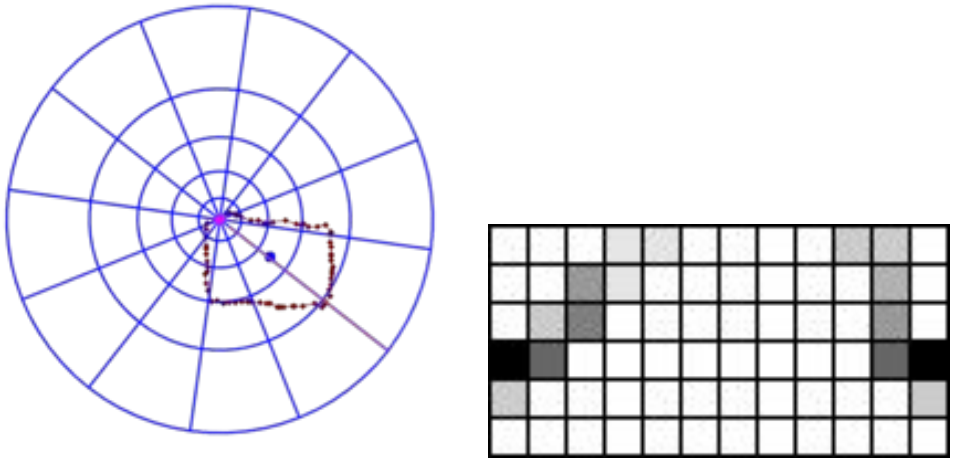


(a) (b) (c)

Figure 31: Local boundary selection and local feature descriptor extraction. The red points correspond to the 2D boundary of the object in the table plane. The magenta point shows the currently selected pushing location. The blue lines in 31a denote the dominant orientation of the local coordinate system towards the center of the object, as well as the distance in local y-direction used in selecting the green boundary points. We center a 2D histogram in this local frame and compute the distribution of the local boundary points.

4.1.2 Shape Features

Our feature extraction takes as input the point cloud associated with the segmented object of interest. We compute both local and global descriptions of shape encoded in a coordinate frame defined by the object center and chosen pushing contact location. We encode local object shape with a small 2D histogram, while we use a slightly modified version of the popular shape context feature for global object shape [8]. Given a 3D point cloud of an object we project all points to the 2D plane defined by the supporting surface and extract the boundary of this projected point cloud. The evaluated pushing location defines a specific point on the boundary. This unique point marks the center of the local coordinate system. The positive x-direction is oriented from the pushing point to the object center creating a consistent frame across objects



(a) (b)

Figure 32: Global shape feature extraction and descriptor. The pink line in 32a represents the zeroth bin direction of the shape context feature aligned to the center of the object. 32b shows the resulting two-dimensional array. A rasterized single form of this is used as the feature descriptor.

of similar shape.

To build the local feature descriptor we walk along the boundary to the left and right sides of the pushing point, selecting all points within a 0.05m band in the local y-direction around the chosen contact location. We visualize the point selection in Figure 31. We select these local points as they best define the object geometry with which the robot end-effector is most likely to interact while performing the pushing operation. These local points are then encoded into a 6×6 2-dimensional histogram with uniform bin sizes. An example histogram is shown in Figure 31c.

We again use the boundary of the object as extracted above, however all points are kept as input to our shape context feature, as opposed to only the points near the pushing location used for the local descriptor. For the given set of boundary points we follow the standard shape context extraction method: we compute the distance and angle to all other points on the boundary and a build a log-polar histogram of the distribution of all of the points. The polar coordinates allow for the shape to be encoded in a way that easily transforms between different contact locations. In order

to make this alignment consistent the first indexed angular bin of the histogram starts at the angle pointing from the contact location towards the 2D center of the original point cloud. The log transform of radial bins creates a more detailed description of the shape close to the contact location while points farther away are encoded more coarsely. Our global histogram has 12 orientation bins and 5 radial bins for a final histogram of size 60. Thus our final combined local-global shape descriptor has 96 dimensions. We visualize global feature extraction and the corresponding descriptor in Figure 32.

4.1.3 Support Vector Regression

We now turn to the estimation of the function, $f(\mathbf{z}) = s$, which predicts the push stability score, s, from the object shape feature, $\mathbf{z}$, defined at a potential push contact location. We can estimate f using kernel support vector regression. The regression function takes the form:

$$f(\mathbf{z}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{z}, \mathbf{z}_i) + b \tag{15}$$

where $K(\mathbf{z}, \mathbf{z}_i)$ is a positive semi-definite kernel comparing the similarity between the test example $\mathbf{z}$ and training examples $\mathbf{z}_i$, and b is a constant offset.

One can see that the prediction is largely based on similarity. In the extreme case that a testing example is only similar to one training example, such a function would be similar to nearest neighbor: predicting the test example by the value of the training example most similar with it. In general cases, the prediction is smoothed by the weighted average of similarities with multiple training examples, thus reducing the chance of over-fitting to a particular example and achieving provably better performance than nearest neighbor approaches.

The parameters α are found through a quadratic programming formulation. This

quadratic programming formulation is proven to be equivalent to the functional minimization problem in the reproducing kernel Hilbert space [45]:

$$\min_{f \in \mathcal{H}_K} \sum_i L_\epsilon(f(\mathbf{z}_i), s_i) + \lambda \|f\|^2_{\mathcal{H}_K} \tag{16}$$

where $\mathcal{H}_K$ is the reproducing kernel Hilbert space spanned by the kernel K, $\|f\|^2_{\mathcal{H}_K}$ is the Hilbert space norm of f which encodes the smoothness of f, λ is a regularization parameter on this smoothness norm (denoted C in the dual quadratic programming formulation and in Section 4.2), and

$$L_\epsilon(f(\mathbf{z}_i), s_i) = \begin{cases} 0, & |f(\mathbf{z}_i) - s_i| \leq \epsilon \\ |f(\mathbf{z}_i) - s_i| - \epsilon & |f(\mathbf{z}_i) - s_i| > \epsilon \end{cases} \tag{17}$$

is called the ϵ-insensitive loss function.

Support vector regression essentially finds a function that both fits the training data well, and is sufficiently smooth, as constrained by the Hilbert space norm term $\|f\|_{\mathcal{H}_K}$. Since such kernel methods are very flexible estimators that can fit almost all smooth functions, the L_ϵ loss function is designed so that the function does not have to fit exactly to the training data. This reduces the chances of over-fitting and improves generalization performance. In our experiments we observed that the ϵ-insensitive loss outperformed traditional L_1 and L_2 loss functions.

The kernel we use is the exponential χ^2 kernel [45]:

$$K(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\gamma \sum_{k=1}^{d} \frac{(z_{ik} - z_{jk})^2}{z_{ik} + z_{jk}}\right) \tag{18}$$

a proven excellent kernel for comparing histogram features that has been widely used in computer vision [113]. The parameter γ controls the width of the kernel, necessary when combining multiple kernels. This kernel corresponds to a symmetric version of the Pearson χ^2 test to determine whether a histogram comes from a certain probability distribution and has nice properties such as striking a good balance between large and small bins in the histogram, as well as being well-defined everywhere (as opposed to

the commonly used KL-divergence). We use separate kernels for the local and global features and take a weighted sum of these two measures as the ultimate similarity. Details of the learning parameters used are given in Section 4.2.3.

4.2 Implementation

We collected all pushing data using a Willow Garage PR2 robot. We performed all experiments using common household objects in the Georgia Tech Aware Home. All perception was conducted using a Microsoft Kinect sensor mounted on the head of the PR2. We segment the supporting table, robot arm, and object of interest from the point cloud captured from the Kinect and track the object throughout the course of pushing. For all experiments in this work the 3D centroid of the object estimates the location and an ellipse fit to the object point cloud determines orientation. For object tracking in our experiments we used the ellipse perceptual proxy defined in Chapter 3.3.

4.2.1 Straight Line Pushing

Straight line pushing is performed using the centroid alignment feedback controller define in Chapter 3.2.2, which attempts to align the tip of the robot gripper with the vector passing through the centroid of the object towards the goal, while pushing towards the goal location. We visualize this control law in Figure 14.

The hand configuration used can be seen in Figure 29. We orient the hand so that the closed fingertips of the robot gripper are in contact with the broad side of the hand facing up. We point the robot's hand down to make contact with the table in order to better manipulate low profile objects. This is a slightly modified version of the fingertip push behavior primitive discussed in Chapter 3.4.

To directly examine straight line pushing for a given object boundary location, we generated a goal location on the table 30cm past the center of the object along the vector formed by the sampled boundary location and the object center. This is

a natural choice for a straight line goal, given the controllers design to push through the center of the object towards the goal location. We stopped all pushing trials after five seconds in order to have consistent trial lengths for all samples. For each object the robot autonomously attempted between 19 and 43 trials. The robot collected a total of 163 pushing trials. The robot performed pushing trials with six different objects: a large brush, a small brush, a toothpaste tube, a box of soap, a food box, and a camcorder. We display the objects in Figure 33.

Figure 33: The six objects used in the experiments.

4.2.2 Rotate to Heading Pushing

The feedback controller used for orienting pushes applies a forward velocity in the robot's gripper frame proportional to the error in current heading, and a rotational velocity of the end-effector to track the rotation of the object. These feedback laws are defined as:

$$u_x[k+1] = \sigma_g|\theta^* - \theta[k]| \tag{19}$$

$$u_\omega[k+1] = -\sigma_s \cdot \dot{\theta}[k] \tag{20}$$

where $\theta[k]$ and $\dot{\theta}[k]$ are the estimates of the object orientation and angular velocity in the table frame, $u_x[k+1]$ is the forward velocity applied in the robot's gripper

frame and $u_\omega[k+1]$ is the desired rotational velocity of the end-effector about its vertical axis. This controller does not attempt to force a rotation on the object where it does not naturally rotate. Instead it pushes through the chosen location and, if the object begins to rotate, follows the dynamics of the object rotating the robot's wrist to maintain contact with object boundary. For all experiments we set the gain $\sigma_g = 0.1$ and $\sigma_s = 0.9$ which were set by hand to give good performance on a number of test objects. We visualize this control law in Figure 34.

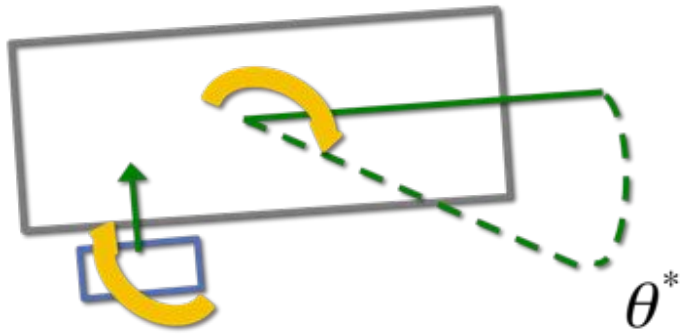


Figure 34: Visualization of the rotate to heading feedback control policy. The green arrow depicts the forward motion term in the end-effector (blue) frame of the controller. This forward motion is proportional to the difference between the current object heading θ (green solid) and desired orientation θ^* (dashed green line). The yellow arrow atop the end-effector shows the wrist rotation used to follow the observed rotation of the object.

The robot hand was oriented so that the fingertip touched the table and the broad side of the gripper was aligned with the object; this is the overhead push from Chapter 3.4. The robot determines the initial pushing direction of the hand by first finding the smallest bounding box around the object footprint with its major axis aligned to the dominant orientation of the object. The robot then selects the side of the bounding box closest to the current pushing location and chooses the push direction perpendicular to this side pointing inwards towards the object. We visualize this procedure in Figure 35. While this may not always produce the best pushing direction, it is consistent and embedded in the learning framework, which allows the robot to learn the best rotate pushing locations initialized following this procedure.

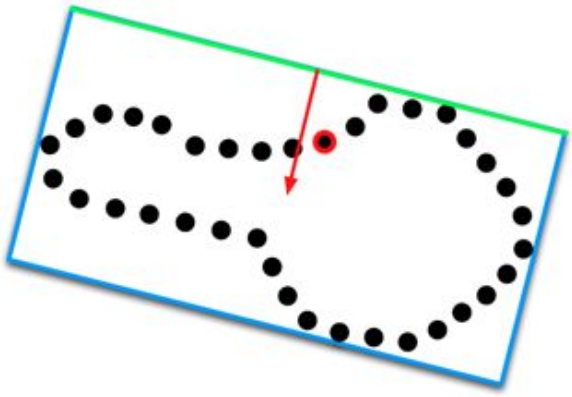


Figure 35: Visualization of determining the initial pushing direction for rotate pushes. For the chosen push location, highlighted in red, the green (top) edge of the bounding box is closest. Thus the initial pushing direction designated by the red arrow is chosen.

The goal heading for all trials was set to be 180° from the initial object heading to allow for the largest possible rotation of the object and trials were stopped after five seconds to give consistent samples as before. The robot collected 87 sample pushes for the camcorder object, 50 samples for the small brush, and 51 trials for each of the remaining four objects giving a total of 341 samples for learning.

4.2.3 Learning Details

We found that taking binary versions of the shape features helps slightly in learning. We converted both the local and global histograms to a binary version, where 1 indicates a nonzero bin in the original feature descriptor and 0 indicates a 0 bin in the original. Each histogram is then normalized, so that the vector sums to 1. For learning straight line pushes the weight of the global kernel is fixed to 0.7, and that of the local kernel is 0.3. The ϵ in SVR is fixed to 0.3, and C (the dual variable to λ in Equation 16) is fixed to 2 in all experiments. The kernel widths γ for the local kernel is fixed to 2.5, while the global kernel has a γ value of 2.0. We determined these values through cross-validation.

In order to improve the regression performance of straight line pushing, we take

the logarithm of Equation 14 as the regression target. Transforms like this are common in statistics in order to make the target distribution more balanced and better correspond to model assumptions. In this work, good pushing locations often have scores less than one-tenth of bad ones; taking the logarithm has the effect of both accentuating the differences between relatively good pushing locations as well as compressing the mapping of the poor choices to approximately equally bad scores. This remapping allows the regression to focus more on predicting good locations accurately, rather than aggressively fitting bad locations well.

Through cross validation on the rotate pushing data we determined a local γ value of 0.05 and the γ of the global kernel to be 2.5. We combined these two kernels with a global weight of 0.7 and the local kernel weighted with 0.3. The SVR loss ϵ value is set to 0.2 and $C = 1.0$. As with straight line pushing, we found taking the logarithm of the orienting push score to improve the learning performance.

4.3 Offline Learning Validation

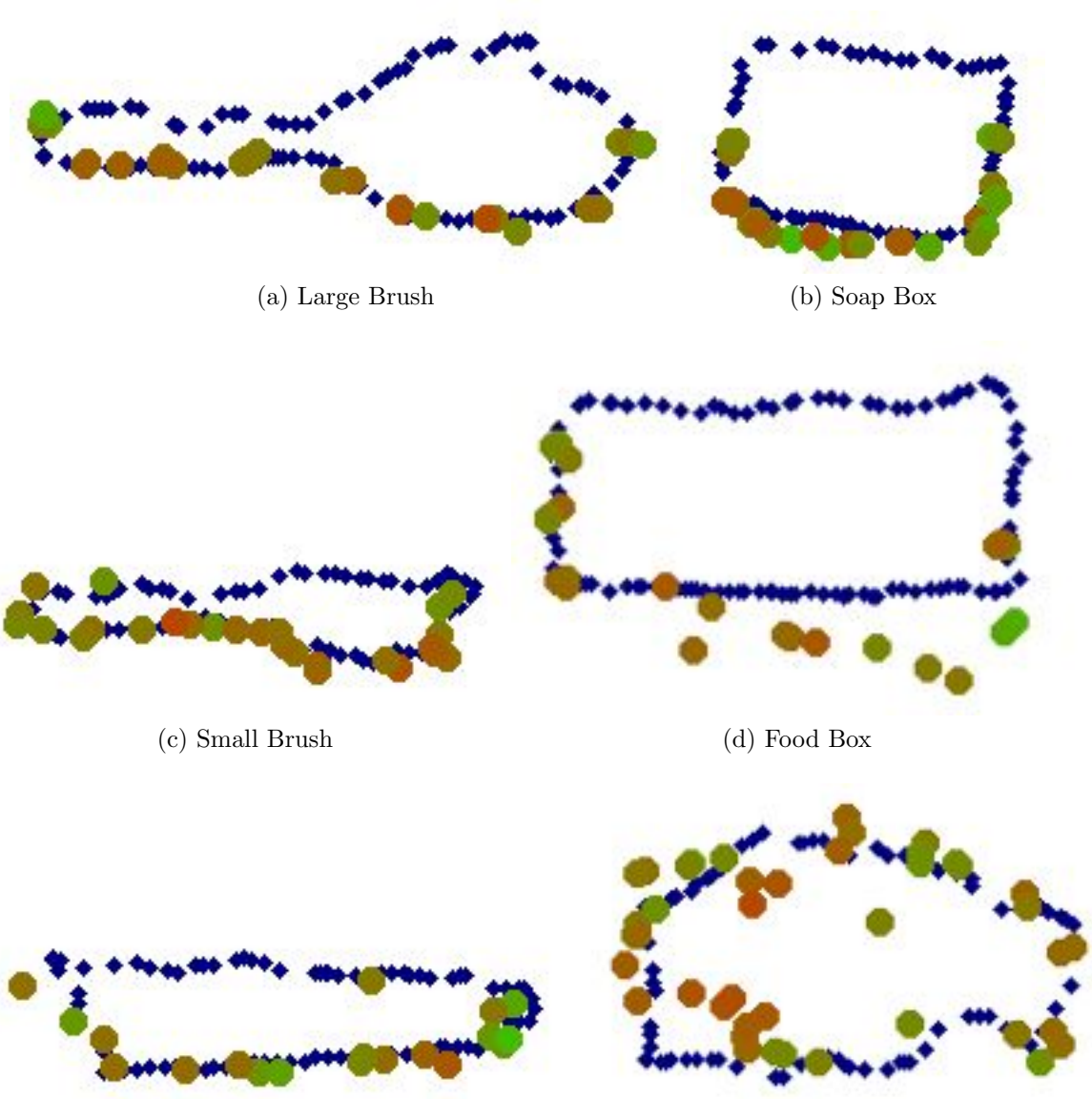
To measure the effectiveness of the learning, we perform leave-one-out cross-validation on the objects: for each object included in the experiment, we train on examples from all the other objects and validate on all the examples of the current object. This corresponds exactly to prediction of pushing behaviors on a novel object. Table 8 presents these cross validation results for straight-line pushing in terms of prediction error and effectiveness at predicting good push locations. To give some intuition for the distribution of push-stability scores, we visualize ground truth pushing scores for all six objects in Figure 36. The high curvature of the brush head, made pushing on the long side difficult for the robot. The brush would rotate quite a bit and the robot would not be able to push it directly towards the goal. However, pushing at the tip of the handle or the small end of the brush head allowed the robot to limit the degree to which the object rotated. For the soap box, many points worked well. We

attribute the high scores near corners to the fact that when pushed at a corner the object initially rotates, but when the robot compensate to push through the centroid, it now pushes near the center of the side and the object's center moves in a mostly straight line.

The first set of results presents the L_1 prediction error of the regression on the log of push stability score. Each column corresponds to the sequestered object of the leave one out methodology, while the rows correspond to the different learning functions. The support vector regression outperforms competing algorithms producing the best result on the mean and the 3 different objects: food box, small brush, and toothpaste. Linear ridge regression bests on only one object, the soap box, by a fairly small margin. Boosting stumps performs better on the camcorder and large brush, but fails to capture the details necessary in the food box and small brush categories, apparently the more predictable objects as seen from the results. Overall all regressors outperform the training mean baseline, except in the difficult case of the camcorder class.

More important than the actual regression error, however, is consideration of whether the prediction function actually allows the robot to more rapidly determine how to push a novel object than random experimentation. To measure this effect, for every object we trained the predictor on the other objects and then predicted good places to push. For each such novel object, the robot predicts the push score on all sampled points from the object boundary and then selects the 3 points with best predicted push-stability score. We define the *planning error* to be the actual error that was observed when the test object was pushed at the selected points. This corresponds to the error that would have resulted had the robot chosen that point for pushing.

As shown in the bottom half of Table 8, under such a planning error metric our approach performed well on all the objects, being able to predict a pushing location



(a) Large Brush (b) Soap Box

(c) Small Brush (d) Food Box

(e) Toothpaste (f) Camcorder

Figure 36: Visualization of ground truth pushing scores for all six objects. Green points represent better scores, while redder represent worse.

Table 8: Offline performance of the push stability prediction system. Regression errors are in the log-space of the predictions. The baseline 0th-order regressor reflects using the mean output in the training set to predict on all positions in the test set. Planning error reflects the error the robot would have incurred, had it used the predicted best, second best, or third best location (from the kernel SVR regressor) to push. As one can see using the predictor offers significantly lower pushing errors compared with pushing at a random location on the object boundary. The learned regressor also out performs deterministic selection strategies of choosing a minor or major axis boundary location.

Metric	**Mean**	Camcorder	Food Box	Large Brush	Small Brush	Soap Box	Toothpaste
L_1 Regression Error							
Kernel SVR	**0.720**	0.795	**0.436**	0.714	**0.569**	1.090	**0.717**
Ridge Regression	0.744	0.765	0.494	0.749	0.648	**1.037**	0.771
Boosting Stumps	0.756	**0.691**	0.602	**0.704**	0.719	1.060	0.762
Training Mean	0.823	0.781	0.739	0.793	0.636	1.137	0.853
Planning Error in cm							
1st Predicted Location	**0.347**	**0.36**	0.14	0.61	0.50	0.21	0.26
2nd Predicted Location	0.478	0.64	0.49	0.21	0.96	0.17	0.40
3rd Predicted Location	0.538	1.66	**0.10**	**0.13**	1.04	**0.12**	**0.18**
Random Location	1.370	1.56	1.30	1.62	1.48	1.31	0.95
Major Axis Location	1.355	2.16	1.76	1.95	1.64	0.44	0.18
Minor Axis Location	0.687	1.51	0.52	0.35	**0.34**	1.14	0.26

with an error of $0.14 - 0.61$ centimeters. Significantly, if one compares against pushing at a random location on the shape, the mean pushing error is reduced by 74.7% (from 1.37cm to 0.347cm) by using the system. The second and third locations have slightly larger planning errors, but are still significantly better pushing locations. Even more significant: using the best of the top 3 predictions for each object, the average reduction in push error was 83%.

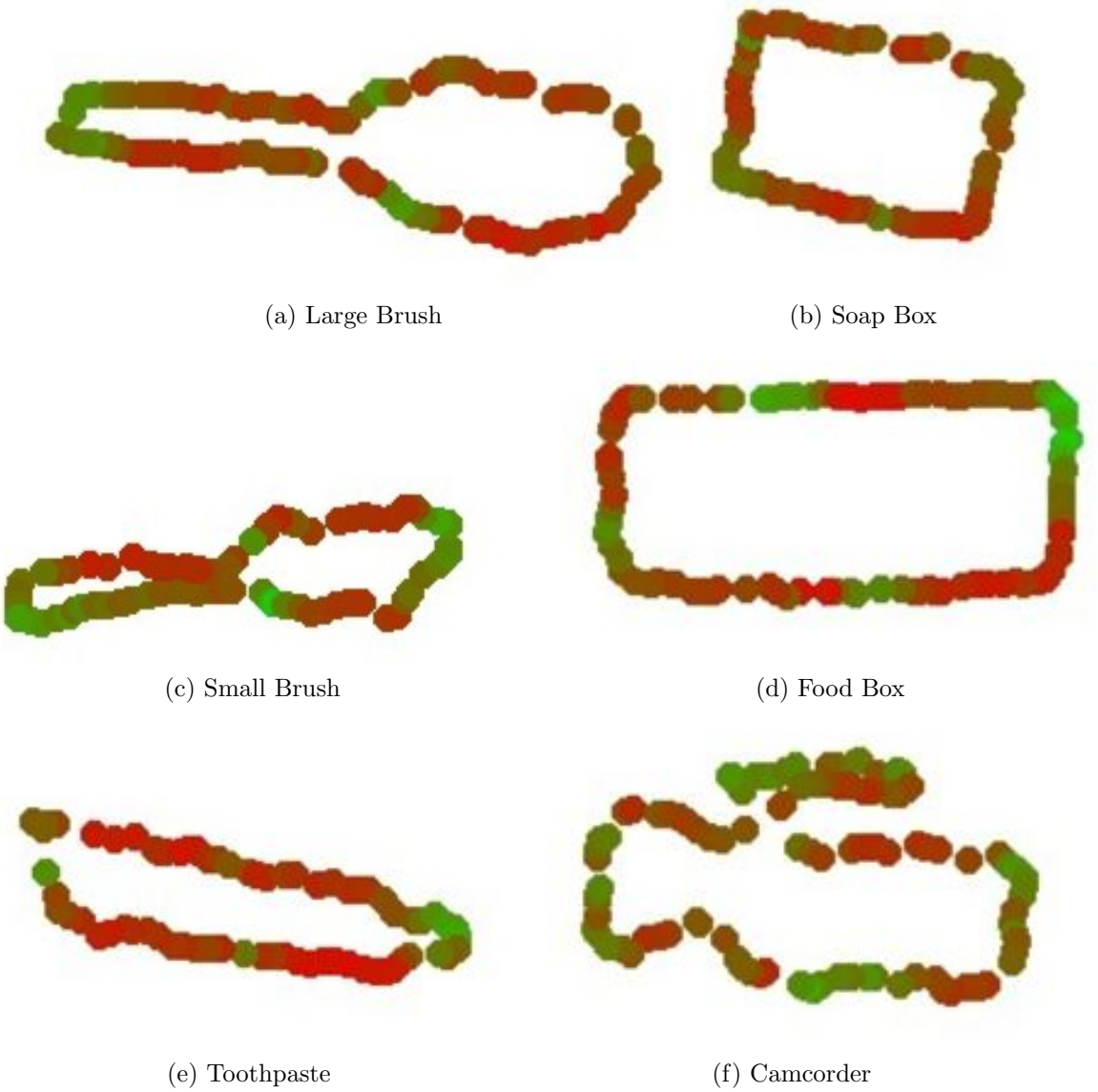
We additionally compare to the pre-programmed heuristic of selecting a point on the object's boundary lying on the major or minor axis of the object's footprint. This selection criteria is a simple geometric feature that requires no learning, but is more informed than simple random selection. Our method outperforms these baselines in all cases but one, the small brush category. However, this is not a damming results as our method produces its worst performance on this category and while the minor axis location is better than any of the top three determined by our learned method, we still outperform the major axis location selection. On average the top ranked pushing

location of the learned regressor out performs the major axis push location by 74.4% and the minor axis push location by 49.5% (from 0.687cm to 0.347cm). This result shows the feasibility of learning to predict from shape descriptions where best to push on a new object given experience with other objects of different shapes. Finally, the ability to predict the outcome for all boundary points makes the planning robust for scenarios where external constraints might prevent the robot from pushing at certain locations.

4.4 Robot Results

To measure the effectiveness of the learning on the robot, we use the learned regression functions to perform push tasks on novel objects. A leave-one-out setting is again used where the robot learns the regressor on five of the objects and performs new pushing tests on the held-out object. When performing push tests the robot extracts features from the sampled boundary points of the current object. It then predicts the push score for each point on the boundary and chooses the point with the best score (lowest for straight-line, highest for orienting pushes) as the initial contact point. A push is then attempted at this contact location, with the goal defined as either a straight line extending 30cm through the center of the object (for straight-line pushes), or 180° from the current heading (for orienting pushes), the same procedures as used for training in both cases.

In order to test the effectiveness of the learned prediction functions for each object we performed 15 new straight-line pushing trials as well as 15 new orienting pushes on the robot. For straight-line pushes, we reject attempts where the goal poses are off or near the edges of the table. Additionally, pushes are not attempted if the inverse kinematics solver fails to give a final position for hand placement. In such cases the robot then decides on the next best ranked pushing location. If no good locations are reachable, the object is moved to another position on the table. We



(a) Large Brush

(b) Soap Box

(c) Small Brush

(d) Food Box

(e) Toothpaste

(f) Camcorder

Figure 37: Visualization of predicted stable push scores for all six objects. Green points represent better scores, while redder represent worse.

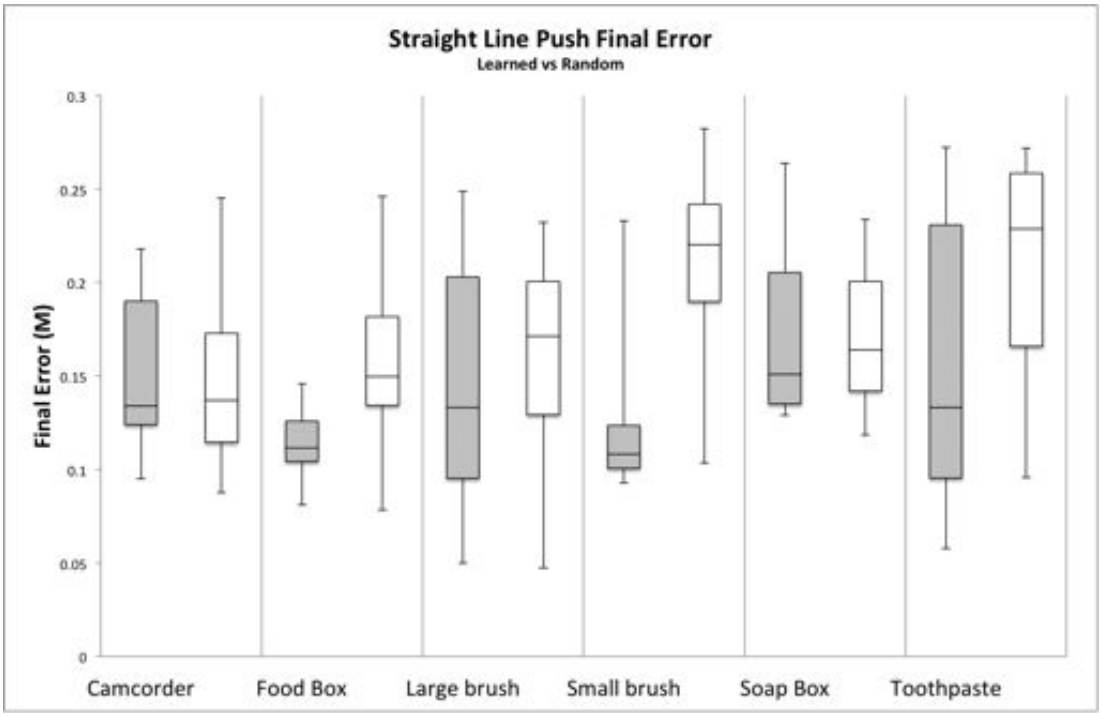


Figure 38: Box and whisker plots of the final position error for learned and random initial contact location prediction for straight-line pushing. The shaded boxes represent the pushing locations chosen through the learned regressor, while the unfilled boxes correspond to random initial locations. The vertical axis represents final position error in meters. The horizontal labels are the object categories. The lower most bar for each plot represents the minimum error for the set of trials, while the upper most bar is the maximum error. The lowest side of the box corresponds to the first quartile, the middle line is the median (second quartile), and the top of the box is the third quartile.

limit the pushes to five seconds, to give easily comparable results. As a comparison, we additionally performed 15 trials on each object for both pushing tasks choosing random initial start locations, subject to the same constraints mentioned above. A detailed explanation of these results follows.

4.4.1 Stable Pushing Locations

We first present results for straight-line pushing. To give some intuition for the distribution of push-stability scores, we visualize ground truth pushing scores from the training data for the two objects in Figures 36a and 36b. The high curvature of the brush head made pushing on the long side difficult for the robot. The brush would

rotate quite a bit and the robot would not be able to push it directly towards the goal. However, pushing at the tip of the handle or at the end of the brush head allowed the robot to limit the degree to which the object rotated. For the soap box, many points worked well. We attribute the high scores near corners to the fact that when pushed at a corner the object initially rotates, but when the robot compensate to push through the centroid, it now pushes near the center of the side and the object's center moves in a mostly straight line.

Contrast these with the visualization in Figures 37a and 37b of predicted push-stability scores taken from test trials of the same two objects. The colors in the predicted images have a narrower dynamic range, the greens and reds are not nearly as bright as those in the ground truth images. We attribute this to the smoothness constraint enforced in the SVR learning. Nevertheless, the predicted best locations on the objects correspond well to the ground truth locations with the best scores. This relative ranking is far more important than the exact prediction of the scores, as selecting good contact locations instead of bad ones is the ultimate goal of the learned function.

Examining the predicted push scores in Figure 37 in unison gives us more insight into what is being learned. We first note that the predicted good locations to push mostly lie on the narrower sides of the objects or near the center of the longer sides of the objects. We expect such a result as the robot attempts to push through the center of rotation of the object. However, as shown above in Section 4.3, we know that the predicted locations are better than simply choosing the point closest to the center of the major or minor axis. Looking at the food box results in Figure 37d shows that the best predicted locations are all slightly off center. If we look at these points jointly we see that they align to a slight, counter-clockwise rotation away from the center of the sides. While we are uncertain what to attribute this result to, we can provide a possible explanation.

There is likely some systematic bias in the way the robot places its hand with respect to the object given a selected contact point. The system would inherently learn this bias and learn to predict locations counter-clockwise to the actual initial contact point as good pushing locations. Additionally, reaching to points on the far side of the object during training was often less successful during training and perhaps this kinematic constraint provided a bias in the training data collected causing the observed deviation from the centers. Whatever the cause, the important thing to remember is that autonomous learning provides a means of directly incorporating these issues into the decision making of the robot. A number of interacting processes give rise to a good, stable push. The learner does not need to tease these processes apart to find good locations for pushing. As long as the procedure used during data collection and testing is consistent the robot must only learn to predict the combined result.

Figure 38 shows quantitative results for the test trial experiments. We plot pairs of box and whisker plots of the final position error for each test object. We plot all learned results as shaded boxes and random initial locations as unfilled boxes. We see that the learned results are better in many cases. First, the median result for the learned predictor outperforms the random contact location selection for all objects. In the case of the small brush and toothpaste, the learned prediction function has consistently lower error at all performance indices. The improvement is particularly pronounced for the small brush, where the top 13 trials consistently produced low errors. Performance on the food box and large brush is marginally better than random results, while the camcorder and soap box have close to identical performance. We attribute the difficulty in predicting the camcorder to its very different friction properties than all other objects used. The soap box results are also not surprising, since the performance is quite insensitive to initial contact location, as can be seen in Figure 36b. We note that many of the large errors come from pushes that result in

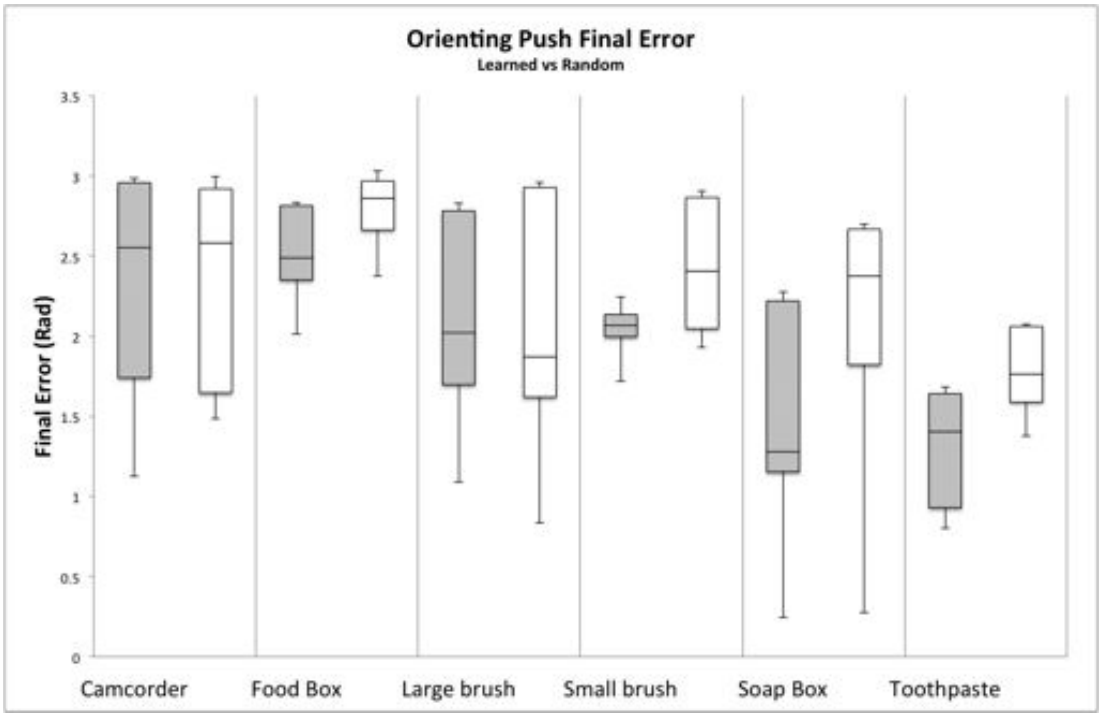


Figure 39: Box and whisker plots of the final heading error for learned and random initial orienting push location prediction for straight-line pushing. The shaded boxes represent the pushing locations chosen through the learned regressor, while the unfilled boxes correspond to random initial locations. The vertical axis represents final heading error from 0 radians to π radians. The horizontal labels are the object categories. Details of how to read the box plot can be found in the caption to Figure 38.

the object being pushed over a relatively large distance, but not towards the desired goal location. Regardless, the experiments demonstrate our ability to automatically learn stable contact locations for pushing from shape information; most importantly they demonstrate that learning improves the overall pushing performance achieved by the robot.

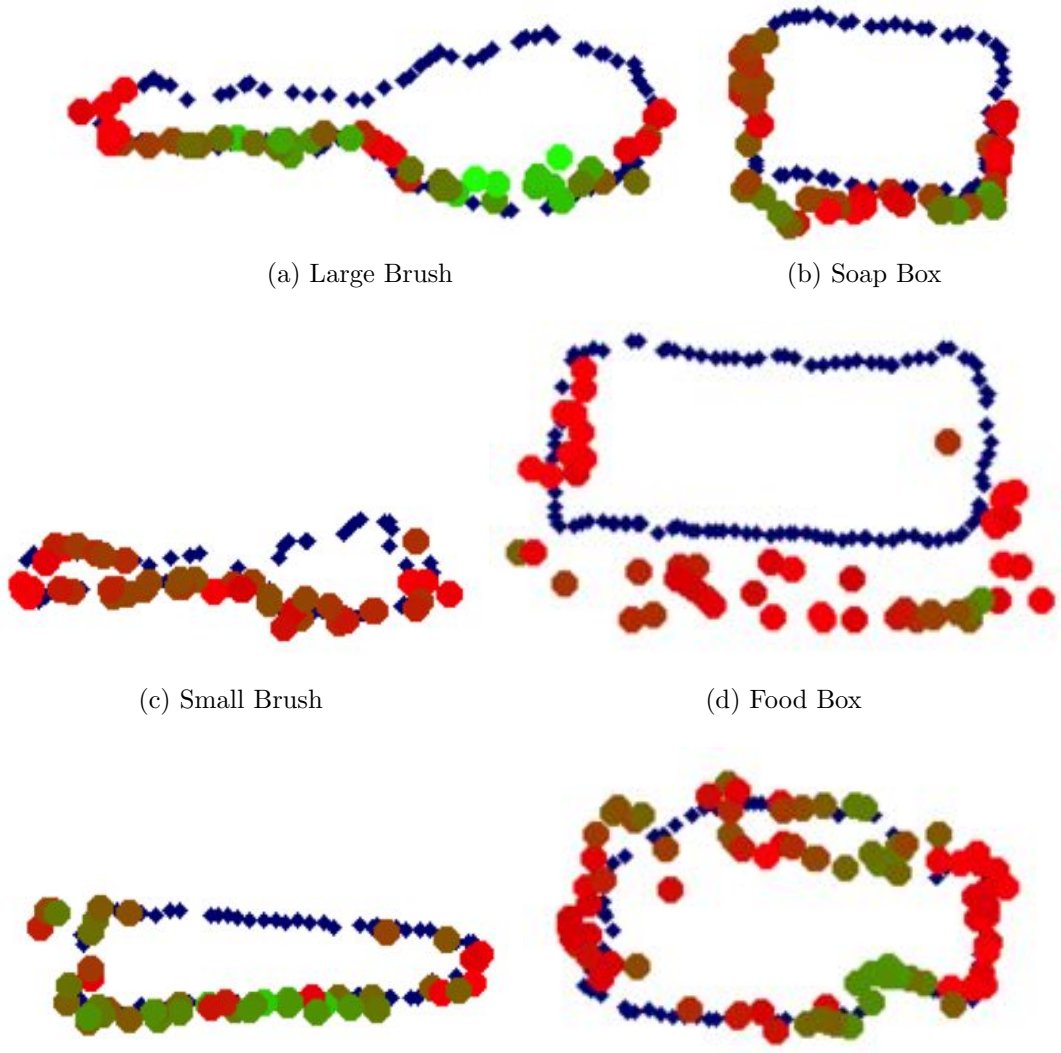
4.4.2 Orienting Push Locations

We present quantitative results for all rotate pushing test experiments in Figure 39. As before the shaded boxes represent learned results, while the unfilled boxes correspond to trial from random initial locations. The vertical axis represents final

heading error in radians. We note that the learned predictor consistently outperforms random initialization on the toothpaste, food box, small brush, and soap box categories. The performance on the toothpaste tube is particularly impressive with the best performance of the learned location achieving 0.80 radians of error, while the best performance for a random location was 1.38 radians. This represents a 42% decrease in final error. Performance is mostly equal on the remaining large brush and camcorder categories. We visualize the ground truth orienting push scores used for training in Figure 40. Example predicted scores for the objects appear in Figure 41. We see that while not all good pushing locations are correctly predicted, such as the center of the long side of the toothpaste and large brush. However the predicted best pushing locations correspond to good locations for orienting pushes based on the ground truth examples. While having more options for good pushing locations available to the robot would be ideal, we believe being that his predictor errs cautiously in generating false negatives instead of false positives.

Unlike in straight line pushing, the overall performance achieved across different categories is quite different for the orienting pushes. The food box never achieves a final error less than 2 radians, while the small brush attains better than 2 radians of error in just under half of all trials. The variability is much higher for the other objects, with both the random and learned predictors achieving a wider range of performance. While the rotation learning does appear to produce better results for some objects, the lack of increase on three of the objects indicates that some important information is not being encoded.

Figure 41 paints a very clear picture of what the robot has learned. We see that good locations are close to the end of the long axis of the objects. We would expect such a result, since points far from the center of rotation, when pushed will create a lever arm causing the object to rotate. Perhaps surprisingly the robot does not push at points at the extremes of the sides. We can attribute this to the fact that when

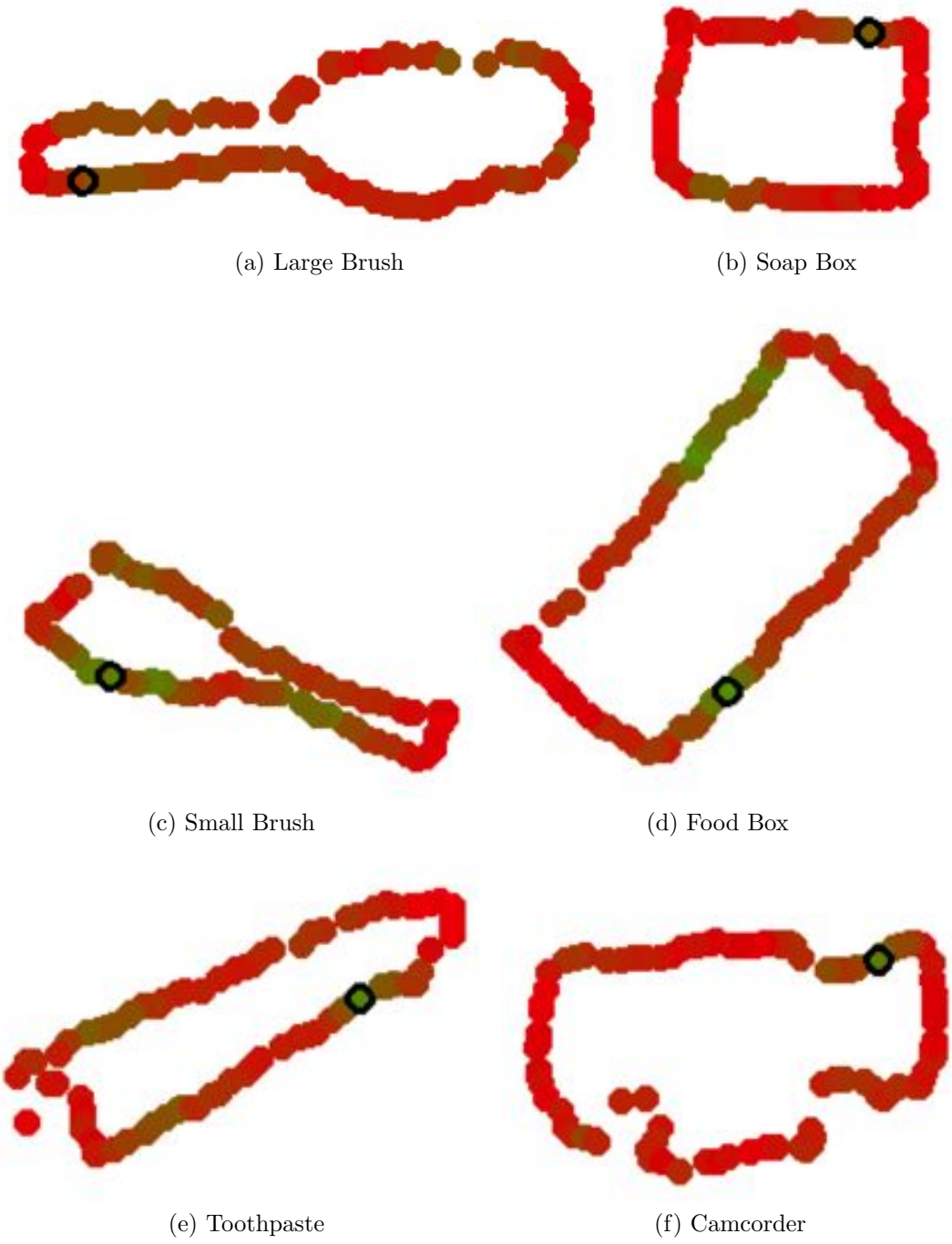


(a) Large Brush (b) Soap Box

(c) Small Brush (d) Food Box

(e) Toothpaste (f) Camcorder

Figure 40: Visualization of ground truth rotate push scores for all six objects. Green points represent better scores, while redder represent worse.



(a) Large Brush (b) Soap Box

(c) Small Brush (d) Food Box

(e) Toothpaste (f) Camcorder

Figure 41: Visualization of predicted rotate push scores for all six objects. Green points represent better scores, while redder represent worse. Black circles highlight the best ranked orienting push location.

pushing near the end of the side the robot is more likely to loose contact with the object while pushing and be less able to push the object to the desired heading. This represents a significant advantage of autonomous learning over direct reasoning about the physics of the object. A physics simulation could easily state that applying a force at the farthest edge of the object side induces the largest rotation; however this ignores the methods by which the robot is pushing. Even if the simulation was performed with the full robot model it is still likely that a more aggressive placement would be determined, since the causes of noise in the real system, such as the series elastic robot transmissions and nonuniform support friction would be difficult to correctly model in the simulator. This result reinforces those shown above for stable pushing, where learning produced results which may not have been obvious to a human programmer.

We note that not all locations on an object are selected that fit these criteria. For example looking at the food box in Figure 41d and soap box in Figure 41b we see only the bottom left and top right sides are highlighted in green. This corresponds to only selecting points which cause the object to rotate clockwise. We believe this to be a result of the data collection. For simplicity in data capture we only collected data pushing from one side of most objects as seen in Figure 40. This exposes a limitation of our feature which is only rotationally symmetric, not reflectively symmetric. We could potentially overcome this bias by either collecting training data from the opposite side of the object or simply reflecting the training data to remove this bias. Another additional correction could be in learning to predict the signed change in orientation, allowing the robot to separately model clockwise and counter-clockwise rotation. Regardless, we see that for the toothpaste tube in Figure 41e, the small brush in Figure 41c, and to a lesser extent the large brush in Figure 41a the robot predicts good locations to push on both long sides of the objects near the ends. Thus sufficient information is still present for this reflection to be learned at least in part.

4.5 Conclusions

We have presented a data-driven approach for learning good contact locations for pushing unknown objects. We demonstrate that a combined description of an object's local and global shape properties are effective in predicting initial contact locations for pushing an object in a straight line, as well as rotating an object to a new orientation. These results were validated with extensive experimentation on a mobile manipulator robot operating on common household objects. Importantly we have shown that what is learned differs from what a human programmer would have determined based on first principles of the object's physics. This reflects the learners ability to integrate the effects of numerous separate processes expressed during the training trials. This highlights the importance of autonomous learning over simulation, where such processes may be idealized, or human labeling for offline learning, where such processes may never be considered.

While the results presented in this chapter use only two affordance-based behavior instances: (*ellipse proxy*, *centroid alignment*, *fingertip push*) and (*ellipse proxy*, *rotate to heading*, *overhead push*), the method is not limited to these choices. Any behavior as defined in Chapter 3 could be used. In Chapter 5 below we examine transfer between behaviors of the prediction functions learned in this chapter. We also examine other elements of knowledge transfer using our affordance-based behavior representation.

CHAPTER V

MODEL LEARNING AND KNOWLEDGE TRANSFER FOR MANIPULATION OF NOVEL OBJECTS

Robots acting as assistants in homes, offices, and other human workplaces will regularly encounter new objects. In order to skillfully manipulate these objects with little experimentation the robot should leverage knowledge from previous experiences. Additionally, given more opportunities to manipulate a specific object the robot should improve its ability in efficiently achieving its task. Additionally, we desire for the behaviors being learned to map to higher level planner actions. While all pushing up to this point has focused on straight-line trajectories we introduce a method which can be given an arbitrary desired trajectory.

We presents as the first contribution of this chapter a framework for learning dynamics models of objects for use in robot manipulation. We show how these models can be used to perform visual feedback control on the task of pushing an object to a desired tabletop location. We make use of the model predictive control (MPC) paradigm to efficiently replan end-effector controls solved over a short time horizon. This provides the robot a desired reference trajectory to follow, breaking from the required straight-line paths of previous chapters. At the same time the robot can still use feedback provided from vision: a benefit over traditional open-loop model based planning methods. MPC additionally provides a means of incorporating learning into your control framework. At the heart of MPC is the system dynamics model. By learning models specialized to objects or groups of objects the robot should be able to improve performance in pushing these objects with more experience. We compare

performance of learned models in feedback control to using model-free feedback controllers, open-loop controllers, and feedback controllers with naive, analytic dynamics models. We give a detailed formulation of our model predictive control problem in Section 5.1 and initial evaluations of this framework performed on a Willow Garage PR2 Robot are given in Section 5.2.

Our second point of investigation examines methods for transferring learned dynamics models to novel objects. We wish for the robot to leverage the learned dynamics models in order to manipulate novel objects with lower error than if it had not performed any learning. Prior to interaction, shape is the most indicative feature of how an object will behave when manipulated. As such we examine how a robot can leverage shape descriptions in selecting from a set of learned dynamics models. Section 5.3.2 describes our approach to knowledge transfer using shape information. Once manipulation has been attempted the robot can make use of the observed object dynamics to check against the learned models, providing an additional means of selecting the best possible model for use in pushing a particular object.

In some situations a robot may wish to manipulate an object it has already encountered in a novel manner. The robot may be constrained in such a way that it should use a particular behavior primitive, which it has yet to try on the object at hand. Can the robot transfer knowledge learned about this object in using a different behavior primitive or control policy? We continue our investigation of knowledge transfer in Section 5.4 where we focus on transferring the contact location information learned in Chapter 4 to different control policies and behavior primitives. Implementation details of the system used for experiments are described in Section 5.5 Finally, we conclude in Section 5.6.

5.1 Model Predictive Controller

We demonstrate our model learning for control framework on the task of pushing objects to desired 2D locations on a table. We design our control task to follow a desired trajectory X^D by minimizing the cost function[1]:

$$c(X) = \sum_{k=1}^{H} ||X[k] - X^D[k]||_2^2 \tag{21}$$

We wish to solve for the desired control inputs, U^*, that minimize Equation 21 given fixed initial state $X[0]$. At the same time the solutions most be constrained to obey the system dynamics and actuator limits. We formally define this optimization problem as:

$$\begin{aligned} \underset{X,U}{\text{minimize}} \quad & c(X) \\ \text{subject to} \quad & X[k+1] = f(X[k], U[k]) \\ & |U[k]| \preceq U_{max} \forall k = 1, ...H \end{aligned} \tag{22}$$

where $f(X[k], U[k])$ describes the discrete time behavior of the system. For our pushing system we define the system state at time k to be the 2D pose of the object and the 2D pose of the end-effector in the robot's torso frame:

$$X[k] = (x_O[k], y_O[k], \theta_O[k], x_{EE}[k], y_{EE}[k], \phi_{EE}[k])^T$$

The control input for a given time step are the linear and rotational velocities controlling 2D position and orientation of the end-effector:

$$U[k] = (v_x[k], v_y[k], v_\phi[k])^T$$

We solve this optimization problem using the sequential quadratic programming (SQP) solver of Kraft [64]. This solver approximates all constraints with a linear model and as such any function f used for estimating the dynamics must be differentiable with respect to the decision variables X and U in order to solve the optimization

[1] We only specify desired 2D positions of the object, the desired orientation of the object remains unconstrained. Additionally while we only examine straight-line trajectories in this thesis, the controller can in theory take arbitrary, smooth trajectories as input.

problem quickly. A desired object trajectory, X^D, is generated by breaking the vector from the object's current pose to the goal position. Points are chosen evenly along this line such that the object moving slightly slower than the maximum control velocity could travel the distance between each point.

The method as proposed so far simply gives us an optimization based planner for following a desired trajectory. In model predictive control a control sequence is planned using some model of the underlying system, which is known to be wrong in some way [10]. Often the error in modeling is assumed to be some noise in sensing or actuation of the system, however the error may also be some systematic simplification in the dynamics model [57]. These modeling errors are overcome by replanning the control sequence periodically using a new estimate of the system state. The updated control solutions is then used changing the original open-loop solution into a feedback controller. In order to solve the optimal control problem efficiently a finite look-ahead horizon, much shorter than the total desired trajectory, is usually used. This allows the controller to not be myopic in choosing the first control to apply, while not wasting time solving for controls in the distant future. We examine the performance of our optimization based planner and the MPC feedback controller in sequel.

5.2 MPC Evaluation with Naive Model

As a first pass validation of our MPC controller we use a naive dynamics model, which we know to be incorrect. We use a dynamics model which assumes a rigid contact with the object as defined by Equation 23.

$$X[k+1] = X[k] + \begin{pmatrix} \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{pmatrix} U[k] \tag{23}$$

It is important to point out how weak this model is with respect to the underlying mechanics. The model has no knowledge of the forces being applied by the robot nor even if contact is currently being made. The model also has no ability to encode information regarding contact and friction between the object and the supporting surface. Nevertheless when the robot is in a stable pushing configuration with respect to the object this naive model roughly describes the qualitative behavior of the system.

We evaluate this naive dynamics model both in our proposed MPC controller as well as in a simple open-loop controller. The open-loop control method plans a control sequence for an entire desired trajectory using the SQP planner described above in Section 5.1. An optimal solution to this formulation equates to the robot simply moving its end-effector in a straight line from its initial placement to the goal location.

We performed experiments with fifteen different objects from those shown in Figure 43. The robot performed at least ten trials with each object. For a given trial the robot chose a random goal location in the workspace and pushed the object until it either reached the goal location (within 2cm) or reached an abort condition. For all experiments object pose estimates were computing using the method described below in Section 5.5.2. The robot performed all pushes using the overhead push behavior primitive. Figure 42 summarizes the results of our naive model evaluation. Each box and whisker plot visualizes the distribution of final position error for a given object

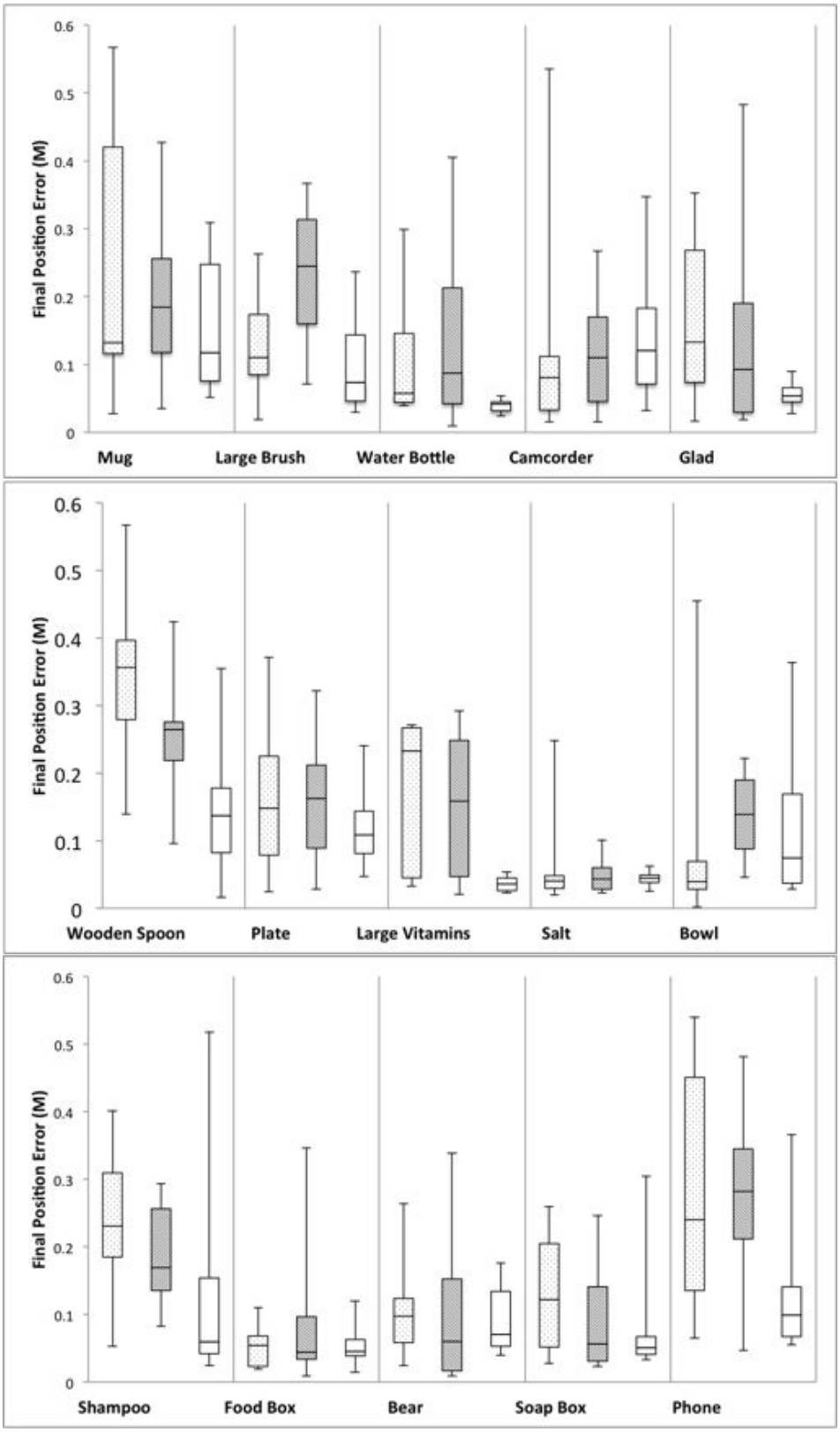


Figure 42: Box and whisker plots showing final position error for three different control methods across 15 different objects. Dotted plots correspond to centroid alignment controller, hashed correspond to open loop SQP with the naive model, and empty correspond to MPC with naive model.

Figure 43: The objects used to train and validate our model predictive controller.

and control method. A single plot shows the minimum, first quartile, median, third quartile, and maximum error values reached in the associated experiments. As a baseline we compare the open-loop and closed-loop model based controllers to a slightly modified version of the model-free centroid alignment controller[2] originally defined in Chapter 3. The first plot for each object (black dots) corresponds to this model free control method. The second series of plots (black hashes) shows the results for the open-loop naive model controller. The final series (empty boxes) corresponds to the MPC controller with naive dynamics.

The performance of the three controllers varies depending on the object being manipulated, however we note that the MPC controller almost always has a tighter distribution of final errors than the two competing methods. While the MPC controller does not always obtain the trial with lowest single error, it has the lowest median score for all objects except three: the camcorder, the salt container, and the bowl. All methods perform very well on the salt container and thus the small difference in medians is negligible. The bowl example is one of the best performing for the centroid alignment controller, with seven of its ten trials performing better than the

[2]The controller was modified to rotate the wrist while pushing. We describe this modification in detail along with the motivation for the change in Section 5.5

median result for MPC. The MPC controller still generally outperforms its open loop variant on this object and its worse performance is better than the worst attempt for the centroid alignment controller. The centroid alignment controller's performance can be attributed to the ease with which the bowl slips along the edge of the gripper while being pushed. Since neither model based method attempts to directly compensate for this slip, they more easily lose a stable pushing vector through the bowl towards the goal position. Nevertheless we clearly see that feedback improves over the open-loop control. It is less clear why the MPC controller performed poorly on the camcorder. We attribute this to the centroid alignment controller doing a better job of staying near the center of the long sides of the camcorder than the MPC controller. This in turn causes the camcorder to rotate less allowing it to more stably move towards the goal position.

Most surprising was the MPC controller's dominant performance in pushing the wooden spoon, shampoo bottle, and cordless telephone. All three of these objects rotate significantly while being pushed, a phenomenon not modeled at all by our naive dynamics model. Even the open-loop model based controller is competitive with the model-free controller on these objects. This supports the results shown in Chapter 3 that the spin compensation controller performs better than the centroid alignment controller for objects which tend to rotate significantly. It appears that the attempts to align with the centroid may actually cause the controller to perform worse, than simply pushing towards the goal from the current hand pose.

The results of this section strongly validate the model predictive control paradigm for pushing. At the same time these results demonstrate clearly the effect of feedback over performing a full open-loop plan. The remainder of this chapter examines how learning can be used to enhance the performance of MPC control.

5.3 Learning Models of State Dynamics

We seek to improve the performance of our model predictive controller by enhancing the expressiveness of the underlying dynamics model. This is a task well suited for machine learning, where the robot can experimentally push an object to collect training data and learn a model which predicts the behavior of the object as a function of the current end-effector configuration and control input. In the remainder of this section we examine several approaches to learning dynamics models along with methods of transferring this learned knowledge for use in manipulating previously unseen objects.

5.3.1 Learning Dynamics Models

We wish to learn a dynamics model describing the discrete time state dynamics of our pushing system $X[k+1] = f(X[k], U[k])$. The input space to this system is the entire workspace of the robot and sampling to fill this entire space would be quite time consuming. Thus instead of directly learning the system dynamics we seek to learn the change in system state defined in the object frame. We define this function to be $\Delta X^O[k] = f_O(Z[k])$. Where the superscript O denotes coordinates in the object centric frame at time k.

$$\Delta X^O[k] = (\Delta x_O^O[k], \Delta y_O^O[k], \Delta \Theta_O[k], \Delta x_{EE}^O[k], \Delta y_{EE}^O[k], \Delta \phi_{EE}[k])^T$$

and $Z[k]$ denotes the end-effector pose and controls expressed in the object frame:

$$Z[k] = (x_{EE}^O[k], y_{EE}^O[k], \phi_{EE}^O[k], u_x^O[k], u_y^O[k], u_\phi[k])^T$$

We can now treat $Z[k]$ as a feature vector for learning ΔX^O. We can then make use of our predictor f_O by transforming the output at each time step back into the global frame and adding it to the previous object estimate $X[k]$.

An important requirement of our predictor f_O is the need to be differentiable with respect to the decision variables of our optimization problem, X and U. This

stems from the fact that the dynamics function constrains the decision variables to follow the expected system behavior. Since all constraints our approximated by a linear function in the SQP formulation, gradients of these constraint functions must be used to search for feasible solutions. All training data were collected using the modified version of the centroid alignment controller defined in Section 5.5. The motivation for this controller is to ensure adequate sampling of both the linear and rotational velocities of the end-effector we desire to control. The robot collected ten example trials for each of the fifteen objects. These data are the same used as the centroid alignment comparison above in Figure 42.

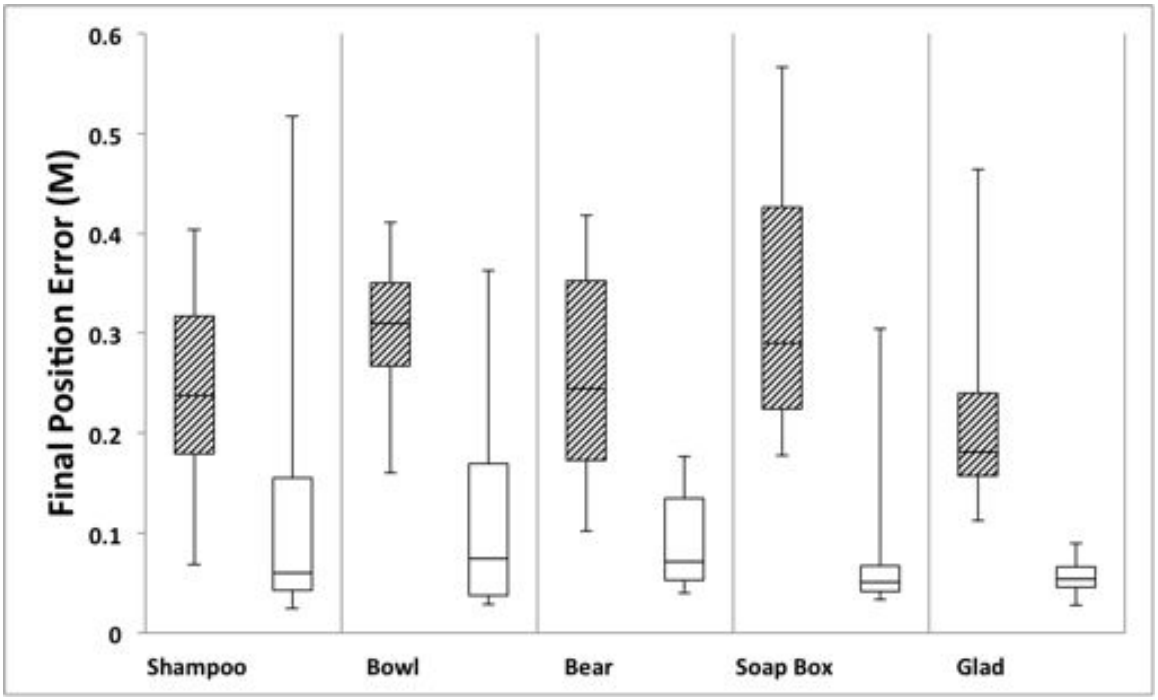


Figure 44: Box and whisker plots comparing learned and naive dynamics models with open loop control. Dashed plots correspond to learned SVR hold-out model dynamics. The empty boxes correspond to the naive model.

Our first learning scenario attempts to learn a single dynamics model using data from fourteen of the fifteen objects. This is a straight forward leave-on-out validation setup to examine how well the learned dynamics models scale to novel objects. We first performed learning using linear support vector regression (SVR). Details of

support vector regression can be found in Chapter 4. We learn a separate predictor for each of the six dimensions of $\Delta X^O[k]$. We chose $\epsilon = 0.001$ by performing cross-validation offline with a subset of the data. Figure 44 shows final position error using this learned hold-out model in the open-loop controller.We see that the learned model performs demonstrably worse in all scenarios than the naive dynamics model. We examine a number of possible reasons for this poor performance.

Our first inclination was to examine what performance gain adding feedback would have. Figure 45 shows results comparing MPC with learned SVR dynamics models to the naive dynamics model for six test objects. We see that the hold-out model SVR is only competitive with the naive model in pushing the bowl. We thought the linear model of the SVR may be insufficient in capturing the complexity of the dynamics. To address this we first learned separate models for each object and had the robot select the best matching model following the method described below in Section 5.3.2. The results correspond to the third series of box plots in Figure 45. We see that the performance decreases even further with these models. As such it would appear that more data is in fact helping the learner perform better for control.

We next examine if there is any sampling bias in the data collected for learning, which may cause poor models to be learned. We visualize sample distributions for several objects in Figure 46. We see that the boundaries of all objects are well sampled with varying push velocity vectors. While this is far from filling the entire space, it shows no obvious gaps in the data collected. We thus attempted to use more powerful learning techniques in estimating the dynamics. However neither kernel support vector regression with a radial basis function kernel nor Gaussian process regression improved the performance of the estimators. In fact both methods would often fail to solve the optimization problem with adequate time to perform feedback control due to the more costly constraint prediction and gradient computations.

As a final attempt at learning dynamics we examine a hybrid approach biased by

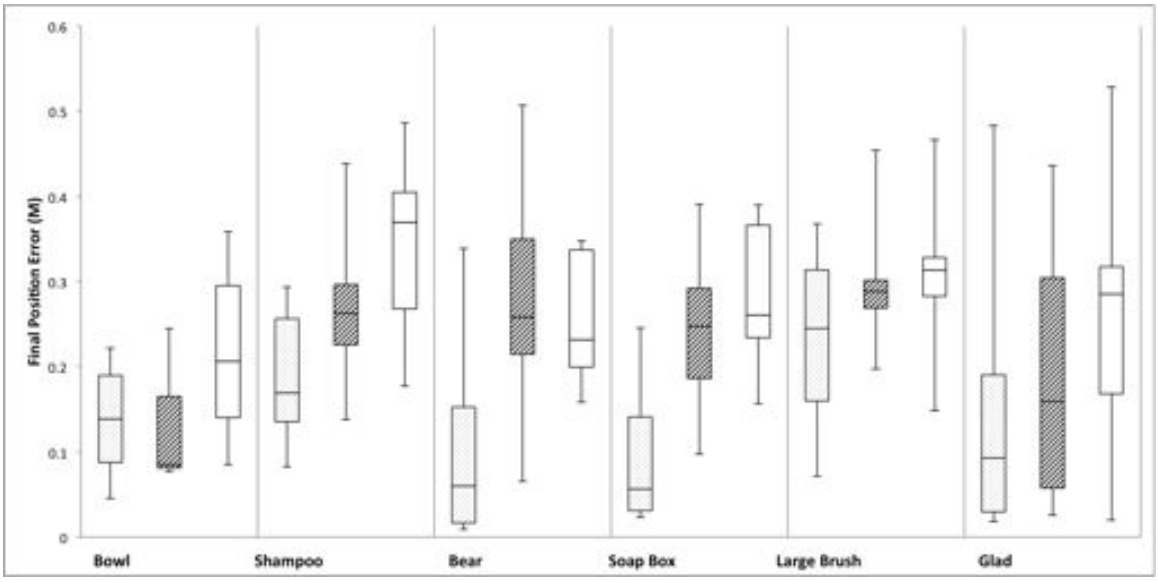
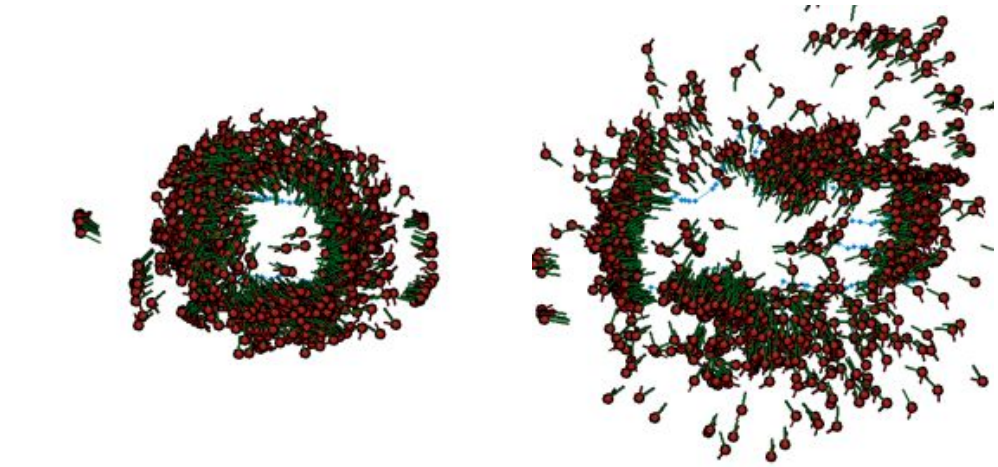


Figure 45: Final position error box and whisker plot for MPC control with learned SVR dynamics. Dotted plots show MPC with the naive dynamics model. Dashed plots correspond to MPC with the hold-out SVR model. Empty boxes represent results for MPC using an SVR model learned from the closest matched shape feature.

the naive dynamics model. To do so we change the targets of our learning problem from state deltas, ΔX^O, to be the error in the prediction function:

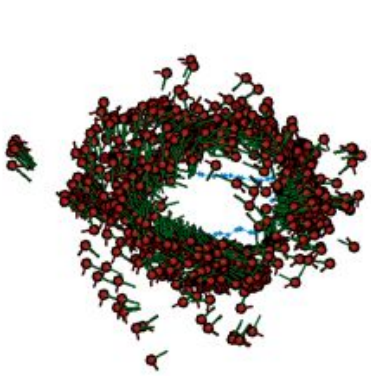
$$\tilde{X}[k+1] = X[k+1] - f(X[k], U[k])$$

The motivation for such a learning problem is that the majority of the complexity of the underlying dynamics is modeled by the naive dynamics model and thus the learner has a simpler function to model. We again use linear support vector regression for prediction. We compare two learning methods: single object class hold-out, as above, and learning a single model for each object. However, unlike before we remove the shape matching component to our model selection and use the single object model trained on the test object. We performed testing for three objects using these dynamics error learning formulations. Results are shown in Figure 47. We see again that the learning has produced no marked increase in performance over the naive model. It appears that the noise present in tracking is too significant to be overcome in learning. Figure 48 shows a typical example of tracking. While both position estimates as well

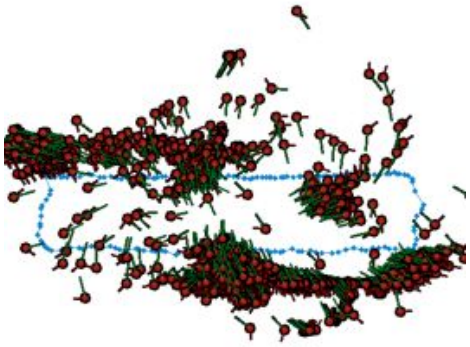


(a) Batteries

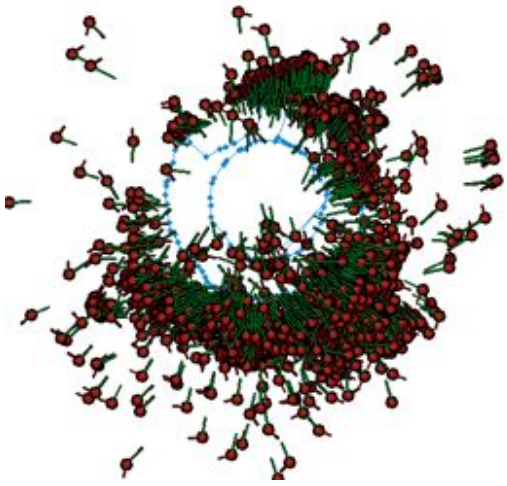
(b) Bear



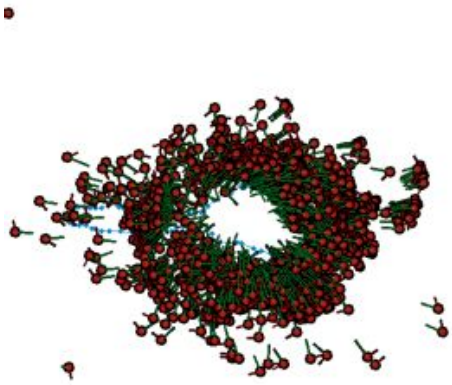
(c) Shampoo



(d) Glad box



(e) Bowl



(f) Large Brush

Figure 46: Distribution of sample locations used as training input. Red dots correspond to sample point locations of the end-effector, red lines show end-effector orientations, and green lines denote push velocity vectors. Object boundaries in blue.

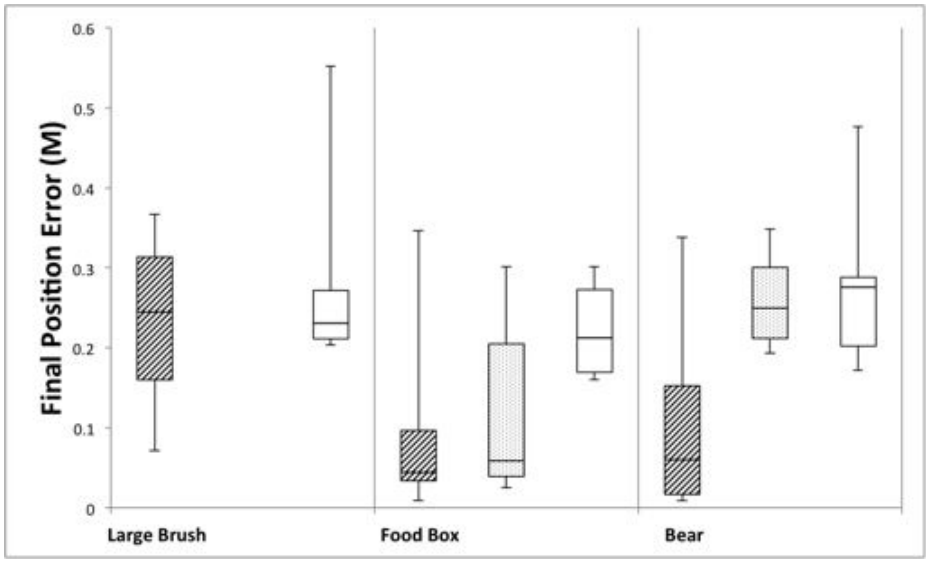


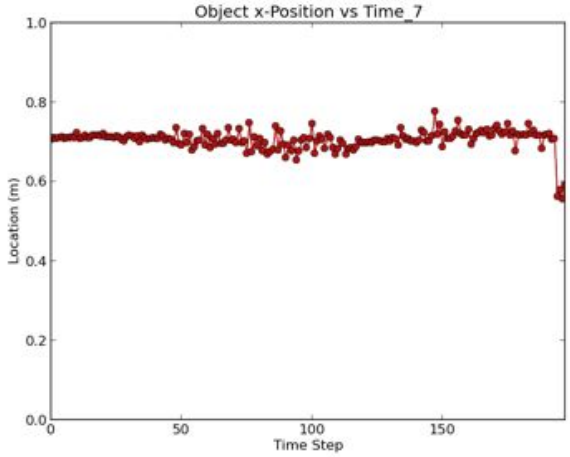
Figure 47: Final position errors comparing error dynamics learning to naive model MPC. The first series (slashes) corresponds to MPC with naive dynamics learning. The second series (dots) shows results for error dynamics learning with a single hold out model. The final series (blank boxes) shows the results for error dynamics learning with object specific models.

as the heading estimate appear mostly smooth, the small fluctuations are made more prominent when the much smaller values of position change and prediction error are computed. Thus it is unlikely the learning signal is strong compared to the noise present. Since we have no access to ground truth tracking we can not compute an explicit signal-to-noise ratio.

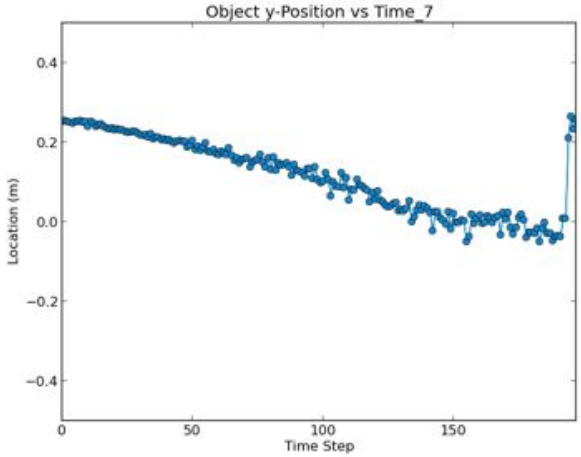
5.3.2 Transfer of Learned Models using Shape Features

We propose two methods in which shape information is used in selecting learned dynamics models for novel objects. In the first method the robot learns a dynamics model independently for each object it encounters. When encountering a new object the robot extracts a shape descriptor describing the object, finds the closest matching shape in its database, and selects the corresponding dynamics model for use in manipulation. Alternatively the robot can learn dynamics models jointly across multiple objects.

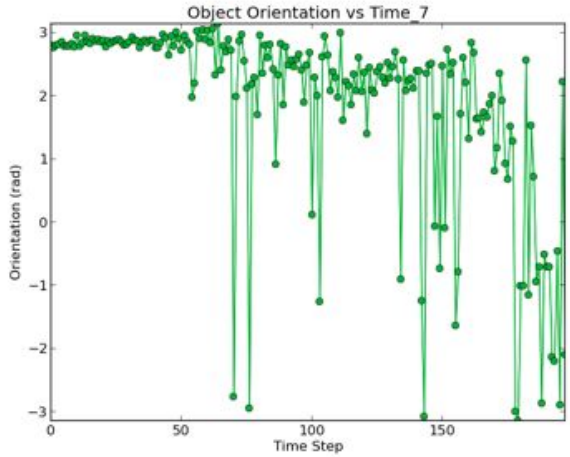
Shape features can be used to cluster similar objects into groups. The robot then



(a) Object x vs time



(b) Object y vs time



(c) Object Θ vs time

Figure 48: Example tracking performance on the bear object.

learns a dynamics model for each of these object groups. The robot can then select the best matching group model in a manner analogous to the selection method used for individual objects. Once a model has been selected and the robot performs its action, the robot can use feedback from the observed manipulation trial to improve its control. If the observed pushing trajectory could best be predicted by a model other than the one previously used by the robot, the robot can select this alternative model for its next pushing attempt.

We use the global feature designed in Chapter 4 for all shape comparisons. We chose this feature based on its good performance in describing objects for pushing previously. However, we remove the use of the local descriptor, as we are no longer trying to reason about the local contact location with the object. For each dynamics model we created shape exemplars by computing the average shape descriptor among all training examples. We examined performing k-means clustering to create multiple exemplars for each object, but in our offline testing this did not change the matching results. We perform matching by computing the L2 distance between the normalized shape descriptors.

In order to increase the training data available to learning a single model, shape clustering can be performed to group objects into shape based classes. A dynamics model is then learned and matching can be performed just as with the single object models. Such a shape based model hopes to find a middle ground between over generalizing across all previously seen objects and having a scarcely sampled model for a single object. Ideally, such shape based object classes would change as the robot gained experience with enough objects. In the extreme each individual object may become its own class when enough training examples were present. However, it is unclear if such specialized models would be best for use with novel objects or if more general models may better handle the uncertainty in modeling the new object's dynamics. Table 9 shows example clusters generated for a few hold out scenarios. We

Table 9: Example shape clusters for $k = 5$ clusters.

Test Object	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Bowl	large brush mug	food box phone glad, batteries camcorder shampoo	large vitamins, water bottle	bear plate	soap box salt
Bear	large brush mug	food box phone soap box camcorder glad, batteries shampoo	large vitamins water bottle	bowl plate	salt
Shampoo	large brush mug	food box phone camcorder glad, batteries	large vitamins water bottle	bear bowl plate	soap box salt
Soap Box	large brush mug	food box phone camcorder glad, batteries shampoo	large vitamins water bottle	bear bowl plate	salt

see that the clusters are generally stable with few changes dependent on the hold out objects. This is a good sign as to the robustness of the descriptor to building models. While some shape groupings make sense—bowl and plate, vitamin bottle and water bottle—other groups are less obvious, such as why the bear is grouped with the bowl and plate.

We present results of MPC with shape-based model selection in Figure 45. As discussed earlier the performance of this method was worse than both the naive dynamics model and the lumped hold-out SVR dynamics model. As such we will only further examine the single object model selection problem, ignoring any evaluation of the shape clustering methods.

We examine how predictive shape was given models available to the robot. To do this we compute a score of how predictive each single object model was of the observed

push trajectory. We use each possible model and simulate an object push trial using the controls applied from the previous trial. The average prediction error, in terms of object position, is then used as a scoring function to rank each model. We summarize results in Table 10. We see that the best ranked model is seldom the model selected based on object shape. At the same time, for each test object one model ranks best for a majority of the trials. These results show promise in selecting models based on post-push analysis. However, the poor predictive nature of the learned models in the control context should give pause in expecting significant generalization of this result. The fact that the glad box model is consistently chosen as the most predictive model lends more support to being cautious in leveraging these results. A variant of this post-push scoring method could be used as an affinity score for clustering training data for model learning. However, models learned in such a way would not be useful until the object had been manipulated at least once. Thus, we do not examine this idea further here.

5.4 Transfer of Initial Push Location Models

One finding of the previous section was that the noise currently present in object tracking prevents an adequate learning signal for learning dynamics models of objects. However, the results of Chapter 4 show that this noise does not prevent learning from trajectory wide signals. As such, we wish to examine if the knowledge previously learned about contact locations could be transferred to the novel MPC controller of this chapter. We compare using the learned stable-pushing prediction function to choose the initial contact location to randomly choosing initial contact locations in several different scenarios. Notably, the MPC controller with naive dynamics model was designed and tested for use with the overhead controller, but the contact locations learning of the previous chapter was done with the fingertip push. The experiments of this section thus also serve to examine transfer of the learned contact location

Table 10: Shape based model selection evaluation. For each trial we show the model selected based on shape in the second column. The third column shows the rank of this model based on the post-push analysis. The final column lists the model ranked best by the post-push analysis.

Test Object	Selected Model	Selected Rank	Best Model
Bowl	shampoo	12	glad
	food box	3	glad
	food box	9	camcorder
	food box	6	glad
	large vitamins	14	glad
	shampoo	9	glad
	shampoo	9	glad
	food box	3	glad
	food box	2	glad
	food box	2	mug
Bear	glad	1	glad
	glad	10	camcorder
	glad	14	camcorder
	glad	3	soap box
	batteries	10	glad
	glad	7	camcorder
	food box	4	glad
	food box	3	glad
	food box	2	glad
	food box	6	glad
Shampoo	phone	8	glad
	salt	6	glad
	batteries	13	glad
	batteries	13	glad
	batteries	1	batteries
	batteries	9	glad
	camcorder	9	glad
	batteries	11	glad
	large vitamins	3	batteries
	batteries	5	camcorder
	batteries	11	glad
	batteries	11	glad

Table continued on next page

Table 10 (continued)

Test Object	Selected Model	Selected Rank	Best Model
Soap Box	batteries	11	glad
	batteries	8	glad
	salt	4	glad
	salt	2	large vitamins
	salt	5	glad
	salt	7	glad
	salt	2	glad
	batteries	8	glad
	salt	2	glad
	salt	9	large brush
Large Brush	soap box	8	glad
	large vitamins	5	camcorder
	soap box	6	phone
	batteries	11	glad
	batteries	9	glad
	batteries	13	glad
	batteries	6	glad
	batteries	11	glad
	batteries	12	glad
	food box	3	bowl
Glad Box	batteries	13	camcorder
	batteries	11	camcorder
	batteries	11	camcorder
	batteries	9	camcorder
	batteries	11	camcorder
	shampoo	9	camcorder
	shampoo	13	camcorder
	food box	1	food box
	batteries	9	camcorder
	batteries	12	camcorder

between controllers and behavior primitives. At the same time we examine how well the MPC controller performs with the fingertip push behavior primitive.

We compare five testing scenarios, each containing ten trials. Figure 49 summarizes the findings of these experiments. We list the five scenarios as 3-tuples of the form (control policy, behavior primitive, random or learned contact location) in the same order they appear in the figure: (*MPC naive, overhead, learned location*), (*MPC naive, fingertip push, learned location*), (*MPC naive, overhead push, random location*), (*centroid alignment, overhead push, learned location*), and (*centroid alignment, fingertip push, learned location*). While no strong trends exist across the five testing objects, we see that learning tends to improve the performance of the MPC controller. This is definitely true for the difficult case of the large brush, where the MPC controller using learned contact locations outperforms the baseline MPC regardless of the behavior primitive used. This same improvement, although less pronounced, appears in the results for the camcorder, an object which was difficult for the MPC controller in the original experiments of Section 5.2.

No behavior primitive is dominant across all objects. This is consistent with the hypothesis of Chapter 3 that different behavior primitives will behave better as a function of the object being manipulated. However, the MPC controller with fingertip push is still competitive with all other methods present in the experiments. Additionally the centroid alignment controller also performs well with the overhead push when using learned locations, even though the contact location prediction was learned with the fingertip push. While not compared directly in Figure 49, we note that the centroid alignment controller using the overhead push behavior primitive, increases performance over the random location selection examined in Section 5.2.

5.5 Implementation Details

We now highlight some important implementation details of our pushing system.

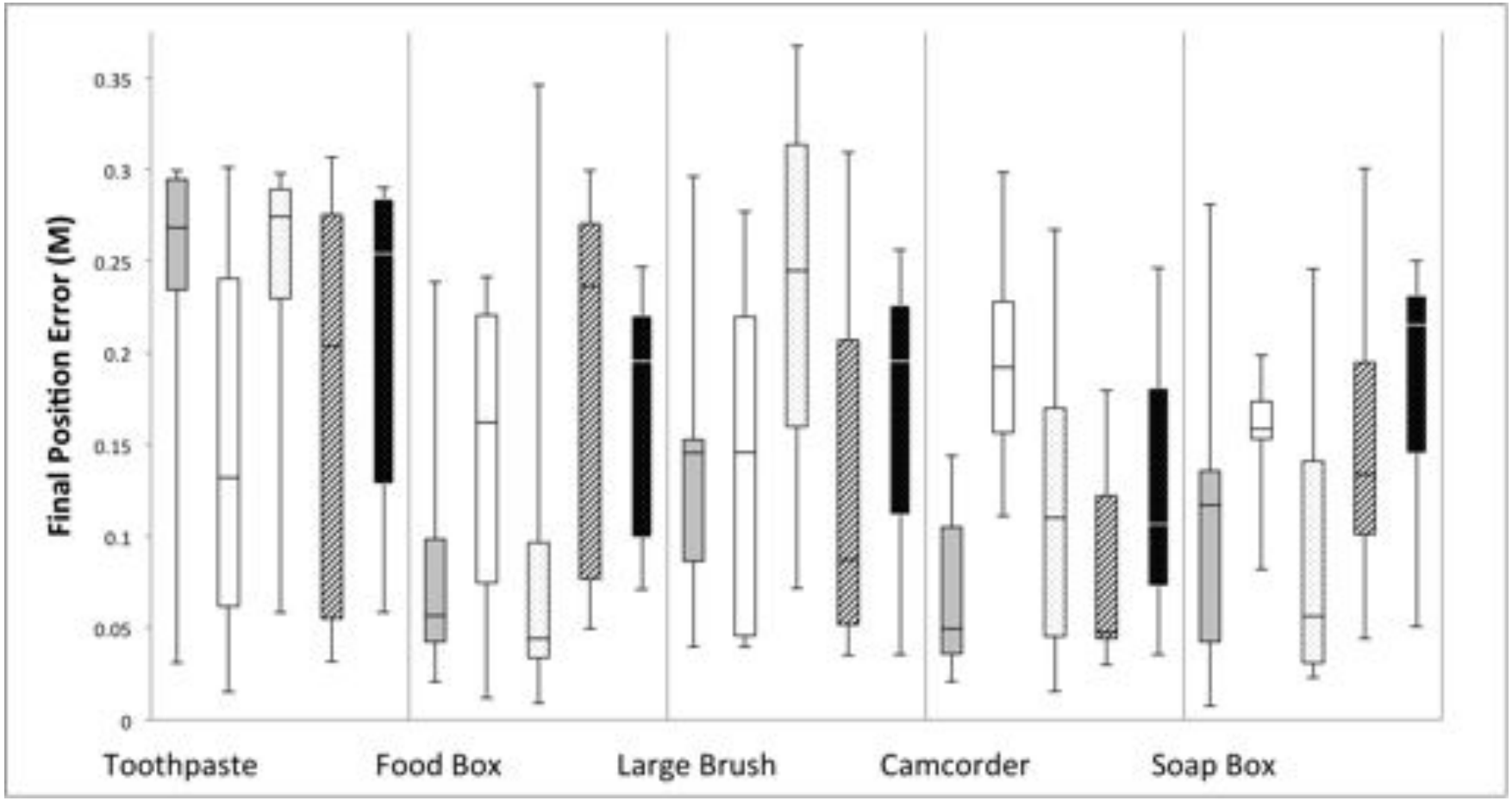


Figure 49: Comparison of push location learning final position errors for different controllers. The first series (small slashes) corresponds to MPC controller with overhead push behavior primitive using learned regressor. The second series (empty) is MPC with fingertip push and learned regressor. The third series (black dots) is MPC with overhead push and random locations. The fourth series (thick slashes) is the centroid alignment controller using the overhead push with learned push locations. The final series (white dots) is the centroid alignment controller with the fingertip push and learned push locations.

5.5.1 Pushing

For all experiments we follow the same hand pose initialization used in Chapter 3. The hand is placed in an initial configuration such that it points towards the goal location with a vector passing through the estimated center of the object. Of course in Section 5.4 the goal position is chosen to be a straight line from the chosen initial pushing location. Plans for our MPC controller are generated using a look-ahead horizon of $H = 10$ time steps. Only the initial control is applied and a new control sequence is planned at the next time step. We attempt to speed up the SQP solution by giving an informed initial solution. We initialize the first solution for each trial by setting a constant velocity from the initial position to the goal and apply these controls to the model, to ensure a feasible solution. For each subsequent solution we take the controls solved from the previous time step, remove the initial control and add a constant velocity term for the final control step.

In order to sample from all dimensions of our feature vector, we need the robot to collect data with a controller which not only moves the end-effector linearly in x and y but also rotates the wrist about its vertical axis. As such we modified the centroid alignment controller introduced in Chapter 3 to rotate its wrist to counteract any observed rotation of the object being pushed. This gives us the new control equation:

$$u_\phi = -k_s \dot{\hat{\theta}} \tag{24}$$

with gain k_s controlling how quickly the hand responds to the observed rotational velocity of the object.

5.5.2 Object Tracking

Feedback of the objects pose is provided by means of Microsoft Kinect attached to the robots head. The supporting table is segmented from the 3D point cloud using RANSAC [13]. The robot's arm is segmented away by projecting a CAD model of its

arm into the Kinect camera frame using the current forward kinematics. The remainder of the point cloud is segmented into objects by means of Euclidean clustering. The initial object pose is estimated using the ellipse proxy presented in Chapter 3. The system then extracts interest points in the color image within the region aligning with the segmented object. Interest points are extracted using the good features to track method of [103]. ORB feature descriptors are extracted for each interest point and stored as an object model [92]. In subsequent frames the same segmentation and feature extraction method is used to detect the object. Feature points extracted in the latest frame are matched to the initial model. A 3D rigid transform is estimated between the matching points to update the estimate of the object's pose.

5.6 Discussion

In this chapter we have presented a framework for performing model predictive control for a robot to autonomously push tabletop objects to desired locations. Our experiments demonstrate that a very simple dynamics model can be used in this framework to achieve state-of-the-art performance in pushing objects to follow straight lines. Furthermore we have demonstrated empirically the utility of the feedback provided by MPC over using a dynamics based planner without feedback. We have shown an attempt at learning dynamics models to provide more informative dynamics models to the MPC controller. While our learning failed to produce models with superior quality to the naive model presented, we have nevertheless proposed methods by which a robot can choose from multiple dynamics models for manipulating a novel object. We believe these methods ask many interesting questions open for future research. Finally, we have shown that knowledge about where initial contact should be made when pushing can be transferred between different controllers and between different behavior primitives. This result extends the results of Chapter 4 by incorporating the MPC controller of this chapter into the behavior representation of Chapter 3.

CHAPTER VI

RELATED WORK

This chapter serves to place this dissertation within the broader context of research on learning for robotics manipulation. Section 6.1 gives an overview of affordance learning work in robotics. Following this Section 6.2 describes work on object singulation. Information on pushing in robotics is given in Section 6.3. We then present work on control learning and improvement in Section 6.4. Last we describe work on visual and kinematic knowledge transfer in Section 6.5.

6.1 Affordance Learning

In early work on affordance prediction described in [27, 77], a humanoid robot learns to segment objects through actions such as poking and prodding. After interaction with a set of objects, the system could learn the rollable affordance for the objects and predict the result of hand-object interactions. The goal was to learn parameters such as initial location of the hand with respect to the orientation of an object that best induce the desired motion. The actions were atomic in the sense that they were applied in their entirety and the results measured. In [30], a classification method is applied to high-level image features to learn the affordance of liftable. Using decision tree classifiers with SIFT and patch features, they demonstrate the ability to learn liftable vs non-liftable objects. Other methods have also been proposed for predicting the manipulation capabilities of novel objects using visual features [97, 108, 40]. These methods, however, require a large training set containing examples similar in appearance to objects with which the robot is expected to interact.

A series of works [68, 79, 78] address the task of recognizing the graspable and tappable affordances, based upon experimentation through self-observation of actions.

Learning in a Bayesian network is employed to learn cuing rules for actions. The network models the relationship between object appearance and motion, end-effector motion, and action. In [105], a functional approach to affordance learning is developed in which subcategories of the graspable affordance (such as handle-graspable and sidewall-graspable) are learned by observation of human-object interactions. Interaction with specific object parts leads to the development of detectors for specific affordance cues (such as handles). The focus of that work was to learn a mapping from object features to grasp locations without unduly worrying about what method of grasping would work at that location. Ugur et al. also examine grasping as an affordance prediction problem, by first creating affordance labels through unsupervised clustering of grasping attempts [110]. A support vector machine is then trained to predict affordance labels as a function of the object and a parameter of the grasping behavior used. When attempting to create a specific outcome, a search is performed to find a behavior parameter that is far from parameters that give undesired outcomes using the predictions from the SVM as a forward model. Barck-Holst et al. learn in simulation to choose a side or top grasp given object appearance [5]. They additionally learn parameters for the grasp: the region of the object to grasp: *bottom, middle, top,* or *above* and the force to apply: *none, small, medium,* or *large*.

Related, Stoytchev [107] describes a method for learning the functionality of a tool through observation of the effects of exploratory behaviors, a process that he termed behavioral babbling. In experiments with a mobile manipulator, the system demonstrated the ability to learn the affordances of a set of tools that could be identified by their color. While not explicitly mentioning affordances, Klank et al. examine how a robot can choose from different perceptual and manipulation mechanisms to more reliably achieve a task on different objects in different scenarios [58].

With respect to planning, affordance-based modeling of robot-object interaction would allow a planning system to systematically select from a set of actions to achieve

desired sub-goals. An example of such an approach is given in [6] where the robot arrange plates and bowls on a table. In that work, however, there is an assumption of a priori knowledge as to which behaviors can successfully operate on which objects and what the resulting state of the action will be.

The concept of Instantiated State Transition Fragment (ISTF) is introduced in [33]. It encodes the pairing between an object and an action in the context of the state transition function for a domain-specific planner. The authors describe a process of learning Object Action Complexes (OACs) through generalization over ISTF's. Montesano et. al. [78] present a Bayesian network model that implicitly represents affordances as mappings from action to effect, which are mediated by the visual features of objects. A model for grasping, tapping, and touching actions is learned from both self-observation and imitation of a human teacher. The goal is to leverage such OACs in planning and executing a multi-step task. Ugur et al. present a planning architecture where behaviors define the actions of the planner and state transition outcomes are the learned outcomes of object affordances [111]. Full plans to achieve higher level tasks, such as lifting an object, are pieced together through chaining of affordance predictions.

Ridge et al. use a self organizing map to discover types of attributes as groupings of input-output relations from pushes [90, 89]. Does not predict what outcome an input will generate, instead attempts to differentiate rolling vs sliding learning affordances, not affordance values. Ridge et al. later perform bootstrap learning of affordances by first determining affordance classes as before and then train an SVM to classify objects based on visual features into the learned affordance classes [91]. The system thus learns to predict how one action performed by the robot induces different changes in the environment as a function of the object being acted upon. However, the number of affordance classes must be specified by the user [91]. Ridge et al. perform the same task through use of a neural network with learning vector quantization to jointly

cluster the output space and learn the affordance classifier [88].

6.2 Singulation and Interactive Segmentation

Object detection constitutes a major research effort in the computer vision community. State of the art object recognition and categorization methods can detect object instances or categories with high precision, but require large training sets enumerating all possible object categories of interest [25]. Additionally the execution time of such methods grows with the number of object classes presented to the system. The burden of obtaining training sets large enough to cover all objects a robot may potentially encounter means that alternative methods are needed for determining where objects are in the environment. Attempts at creating generic object detectors have been made, but they still require large training sets and computation time [2].

While singulation has been attempted using grasping, this requires dexterous capabilities to perform grasps on unknown objects and is limited in only being able to manipulate objects small enough to be grasped [67, 76, 59]. In contrast, non-prehensile pushing actions requires less precision, can be performed using much less capable manipulators, and can operate on a wider range of objects that may be too large to be grasped by the robot. As such, we restrict our discussion to those singulation works relying on pushing, where less dexterity is necessary in the manipulator.

The problem of interactive segmentation was introduced by Fitzpatrick and Metta, where a robot makes a sweeping motion inside its view frame and detects the objects that move [28, 29]. The robot detects its arm by calculating optical flow in the scene and segmenting the pixels with flow corresponding to the controlled motion of the arm [28]. Using this initial segmentation the end-effector of the arm is estimated at the farthest point of the arm being moved and any flow that appears in the image beyond the end-effector is assumed to be an object moving as result of the sweeping motion. Fitzpatrick extends this work in a graph-cut framework to better segment

the object being moved, allowing for textureless regions to be segmented in addition to those locations where optical flow occurs [29]. A limitation of this approach, not present in our work, comes from the assumption that all motion not explained by the robot's arm belongs to a single object. Additionally no method is given for determining where to push, thus pushing actions must exhaustively search the space in order to detect all objects present.

Li and Kleeman use small pushes, termed nudges, to segment symmetric objects [66]. Symmetry lines are detected in stereo cameras and are then grouped to form hypothesis for locations of symmetric objects. For a given hypothesis the robot performs a pushing action that will move the object in the stereo view allowing frame differencing to seed a segmentation constrained by the symmetric property of the object.

Similar to Fitzpatrick's work, Kenney et al. use image differencing to localize the robot arm in the scene and segment motion not belonging to the arm as objects [56]. Templates are built from the resulting object locations allowing for tracking of objects over time. This enables the objects to collide without their identity being lost. However, it still fails to separate objects in cases where multiple objects are in contact prior to the robot discovering them. Furthermore, these templates require highly textured objects to produce good results. Additionally it is assumed another process tells the robot where to initially push.

Katz et al. learn kinematic models of articulated objects through pushing, although segmentation of the objects in the scene is assumed known [55, 51, 52]. More recently this has been extended to extracting 3D models of the objects [54, 53]. Schiebener et al. generate object hypotheses by segmenting scene elements into 3D geometric primitives [100]. Pushing behaviors are then performed and the object hypotheses are refined based on what moved coherently when the object was pushed.

The work of Chang et al. is most similar to ours. Chang et al. perform object singulation through pushing of object piles [14, 15]. Groups of objects are first segmented by removing the supporting surface from the input point cloud. Pushes are planned to push through the object centroid in directions avoiding contact with other piles of objects. Rigid body hypotheses are proposed using point correspondences from the images taken before and after the push action and are confirmed by matching in the point cloud. Evidence for singulation is accumulated by consistent strong rigid body matches. Our work differs in that we explicitly form hypotheses for splitting locations in object clusters based on edge locations and orientations and accumulate evidence related to these locations in order to build confidence more quickly. Additionally, we do not rely on texture being present in the objects for correspondences nor do we need to perform pick-and-place operations to clear the workspace of singulated objects. Van Hoof et al. present a method for producing maximally informative pushes for object singulation [112]. They plan pushes which will give the most information in determining if over-segmented regions belong to one or multiple objects.

6.3 Robot Pushing

Robotic applications of pushing manipulation occur in many sub-fields of robotics. Here we review the fundamental results in robotic pushing as well as relevant applications to pushing in natural human environments and methods which apply learning in pushing. We will mostly ignore industrial applications such as pushing to orient parts [70].

Mechanics and Control Mason describes the qualitative rotational changes of sliding rigid objects being pushed by either a single point or single line contact [75]. Goyal et al. introduce the limit surface as well as other geometric examples to analyze the frictional effects of rate independent velocities applied to rigid bodies sliding on a surface [36]. Howe and Cutkosky present a method for approximating the limit

surface function with an ellipsoid [46]. Peshkin and Sanderson determine all possible support distributions for a given object and show how far a fence must push the object for it to become aligned with the fence [84]. Yoshikawa and Kurisu estimate the support distribution and center of friction for an object through use of a force sensor in pushing the object [114]. Lynch also uses force sensing to present a method for estimating the coefficient of friction and distribution of support for a number of objects through pushes by a robot. [69]. Balorda and Bajd show how pushing with a two fingered end-effector can reduce uncertainty in positioning polygonal objects. By using two point contacts and reasoning about the rotation with respect to the center of friction the object can be rotated into a desired orientation and then purely translate along the line of motion of the end-effector [4]. Lynch and Mason determine the controllability of pushing an object with a single point or line contact [71, 72]. They present a planner which returns a sequence of contact locations between the pusher and object where each contact point can be reached smoothly from the previous one while navigating the object around obstacles in the plane [71, 72]. Akella and Mason produce sequences of linear pushes with a fence to position a polygon at a desired pose in the plane [1]. Bernheisel and Lynch present a method for determining if a collection of planar parts in contact will maintain their configuration when pushed [9]. Ruiz-Ugalde et al. determine the static and kinetic fiction coefficients for multiple objects with rectangular footprints, both between the robot hand and object and between the object and table [94]. Additionally they present a robust controller using a cart model for the object being pushed. The controller takes object velocity as input to control the system to a desired 2D pose, as such the mapping from applied force to velocity is believed known from the estimation and is separate from the control of the object.

Outcome Prediction and Detection Gandolfo et al detect using optical flow patterns if an object will topple, slide, or oscillate when pushed [32, 31]. Ruiz-Ugalde et al. predict if an object will topple or slide when pushed [93]. Zrimec and Mowforth present a method for grouping changes in object pose in an unsupervised way which a human can qualitatively label with categories such as: {moved, not moved} or {moved left, moved right, etc.} [116]. Narasimhan uses vision to pose polygonal objects of known shape in the plane [81]. Three different methods are examined. The first method uses hand coded behaviors to rotate and translate the objects which have known center of mass. The second approach uses feedback control to rotate the object the desired direction about the center of rotation, while pushing it in the direction that will translate its centroid to the desired location. The third approach stores the results of different pushes and uses nearest neighbor classification to select the action that generated a result closest to the desired outcome.

Stoychev learns the effects of a robot moving different tools in the plane using a simple look-up table where actions are explored through motor babbling [106]. These learned tool motions are then used to push a puck into a desired goal region. Sinapov and Stoychev learn the pushing effects of different tools on a puck in simulation. Decision trees and nearest neighbor approaches are compared with different points of focus in the image used to predict the change. Generalization across different tools is also examined [104].

Omrčen et al. use a retinal image based neural network to learn the pushing dynamics for flat objects [82]. The learned models are specific to individual objects and generalization to similar objects is not examined. Scholz and Stilman learn object specific dynamics models for a set of object through experience [101]. Each object is pushed at a number of predefined points on the perimeter and Gaussian models of displacement in (x, y, θ) are learned. Salganicoff et al. present a method for learning and controlling the position in image space of a planar object pushed with a single

point contact [95, 96]. Slip of the object is avoided by pushing at a notch in the object.

Kopicki et al. also learn probabilistic models of outcome based on pushing manipulations [63]. The predictions are modified as a function of object shape, however only rectangular shapes are considered, unlike the variety of natural objects used in our work. Additionally the prediction models are not used to perform control movements of objects to desired poses. Ridge et al. learn as a function of visual object features to predict how a pushed object will behave [88]. They use a neural network with learning vector quantization to jointly cluster the output space into affordance classes (i.e. rolling vs pushing) and learn a classifier to predict if a given object will produce this outcome. In similar work by Ugur et al., a robot clusters observed outcomes of a number of learned manipulation strategies and learns to predict these outcome categories as a function of different input features, including object shape [109]. While these methods are definitely interesting, in neither work does the robot learn where to push in order to move the object to a desired pose, a problem we propose a solution to in Chapter 4.

Tabletop Clearing Planning Scholz and Stilman organize object on a table by specifying an optimization function as an abstract goal [101]. RRTs are used to plan a sequence of learned actions to achieve the desired result. Replanning is performed when the observed effects deviate too far from the original plan. Cosgun et al. plan pushes of tabletop objects in simulation to clear large enough space to place new objects into the scene [17]. Emeli et al. extend this for use on the PR2 robot [23]. Dogar and Srinivasa perform similar planning in order to clear a path through objects on a cluttered tabletop environment in order to grasp a desired object [22]. Zito et al. use RRTs to plan sequences of straight line pushes with a point contact to move objects to desired locations [115].

Grasping Dogar and Srinivasa plan and perform push grasps where pushing is used to reduce uncertainty in object pose relative to the robot gripper prior to closing [21]. Mason discusses push grasping in [75] describing the earlier work of Pingle et al. [85]. Rao and Goldberg also present work on push grasping [87]. Omrčen et al. push large planar objects to hang slightly off a table edge to facilitate grasps that are difficult when the object base is surrounded by table surface [82]. Gupta and Sukhatme present a method for separating blocks that are in piles or close to one another in order to more reliably grasp individual blocks [37].

6.4 Control Learning

For control systems where the underlying plant model is not well known, machine learning methods are often used to address the control problem. Schaal and Atkeson review control learning in [98]. Many approaches attempt to learn a forward model of the system dynamics and then attempt to invert this model to plan an appropriate action [98]. Locally weighted regression is often used for this modeling task. The alternative approach of directly modeling the policy as a function of the state space can be formulated as a reinforcement learning problem.

Deisenroth et al. use Gaussian process (GP) regression to estimate the state transition function in an MDP [19]. Using a policy with affine parameters an analytic gradient can be evaluated to update the policy. The expected cost of future actions is computed using the forward propagated Gaussian distribution through computation of an analytic gradient. This method was used to learn from scratch a task of stacking a series of blocks with a low-cost robot arm [20]. A separate control policy was learned for stacking each block on the tower, but training time for each additional controller could be decreased by initializing with the policy used on the previous block. This work was extended to allow for applying a single policy for controlling to multiple target values [18]. This was achieved using an internal change of coordinates so that

the target value of the task was the origin of the state space.

Another popular method of reinforcement learning for control learning builds on the use of dynamic motor primitives (DMPs) [48, 61, 83]. The DMP framework defines a set of dynamic systems, one for each of the system's degrees of freedom, which are coupled together by a single canonical system. A human teacher usually demonstrates the desired motion and then policy gradient methods are used to improve the controller to more reliably perform the task [61]. Pastor et al. use multiple MDP behaviors to give a robot a database of behaviors for use in different situations [83]. Kober et al. learn meta-parameters of the MDP system to adjust the robot controller to multiple target situations [60, 62]. A mixture of MDPs defines a set of MDP, where separate MDPs activate as a function of the robot's location in the state space [80].

6.5 Knowledge Transfer

Much work on affordance learning takes the flavor of transferring learned information between object categories using visual information and observed object behavior during manipulation, specifically predicting affordance labels for new objects based on previously learned information (cf. [108, 40, 65]). Here we focus on related work in robotics communities on learning methods that attempt to transfer other types of knowledge to novel situations in ways that could be helpful to affordance learning.

Madry et al. use object category information to transfer what types of grasps to perform on different objects [73]. Hillenbrand and Roa warp the contact locations of grasps between objects to transfer between objects [44]. Replanning refines the final placement location to determine a more stable grasp. Amor et al. use this contact warping method to enable robots to transfer grasps defined through human demonstration between different objects [3]. The non-rigid registration method of Chui and Rangarajan has recently been used in warping manipulation trajectories between different objects by Schulman and Abbeel [16, 102].

Finally, shape features have been used in a number of works on learning grasp locations on objects [12, 86, 49]. Bohg and Kragic use shape context as a feature for learning grasping locations [12]. Le et al. desire to learn grasping locations for each finger of a multi-fingered robot hand [86]. Local image features are extracted at each grasp point and depth features are extracted along the lines formed between the grasping locations. These features encode variation in depth in an attempt to encode smoothly changing shape. Jiang et al. use local histograms of depth values to rank grasping locations for a parallel jaw gripper [49]. In a different piece of work Jiang et al. learn placement locations for objects using features similar to the variance features discussed above as well as histograms of point locations of both the object being placed and the potential placement locations [50].

CHAPTER VII

CONCLUSION

This dissertation is motivated by the need for robots to act as assistants in the daily lives of humans. We address this concern in the forms of two concrete problems. The first desires a representation which allows a robot to encode goal directed behaviors it performs into discrete actions for use in a planner or to be commanded semantically by a human. Affordance-based behaviors provide an attractive representation for bridging this gap between high-level semantic knowledge and low-level sensing and actuation of robots. This dissertation presents a factored representation for encoding affordance-based behaviors. The representation defines an affordance-based behavior as a collection of three components: a *perceptual proxy*, a *control policy*, and a *behavior primitive*. The behavior representation provides a means of transforming the continuous space of actuator controls into a discrete selection process for a given goal situation and set of objects in the environment. Chapter 3 shows how a robot can explore a wide variety of behaviors from a small set of predefined atomic behavior components. The robot learns which affordance-based behavior is best suited for a specific task as a function of either the object's identity or its pose in the workspace.

For example our behaviors could provide the actions available to the planner proposed by Cosgun et al. [17]. The plan provides a sequence of straight line pushes to move the objects in the scene to clear a space on the table. The robot can use the learned knowledge of how each specific object is best pushed to select the behavior to complete these planner-level actions. Additionally the results of our push location learning presented in Chapter 4 could be used to select pushing directions as a function of the objects observed, instead of selecting locations only at compass points as

the planner currently works. Our behaviors would also work well with the planner of Barry et al. [7]. This planner explicitly reasons about diverse manipulation types: pushing, grasping, and transit of the robot base. While this planner provides geometric paths for transit and grasp planning, it only allows simple straight-line paths for pushing. Our behaviors and learned contact locations provide a means of selecting higher quality pushes to achieve the desired displacement, than the heuristics used in Barry's work. Finally, our proposed Model Predictive Controller from Chapter 5 could be extended to work with non-straight line trajectories. Having this extended set of pushing paths may allow the robot to create plans with shorter overall execution time, extend battery life by not needing to move the robot base as much, or possibly even achieve tasks which were otherwise infeasible. These examples highlight some ways in which our behaviors could improve current planning techniques, there are likely many other planners which could leverage this object specific manipulation knowledge, encapsulated as discrete actions.

The second main problem we address for assistive robots, asks how the robot can meaningfully interact with a novel object when seen for the first time. We examine this problem in three ways. First, we ask how a robot can discover objects in cluttered environments when none of the objects have previously been seen. Second, we examine how a robot can learn good locations for making contact for pushing objects in a controlled manner. Third, we examine if dynamics models can be learned for specific objects in order to improve pushing performance when encountering a novel object. We detail these contributions below.

Chapter 2 makes use of a set of behaviors in the setting of a robot autonomously discovering pushable objects in its environment. This problem of singulation requires the robot to segment and separate all objects in the environment. Since the actual set of objects is unknown, the robot selects a behaviors primitive as a function of where the objects lie in its workspace, keeping the control policy fixed. Singulation

provides a fundamental tool to a robot learning with affordances. Since, the robot determines empirically what constitutes an object in its environment, many ambiguities inherent to segmentation are removed, allowing the robot to confidently associate manipulation information with the object. Our singulation approach contributes an advancement in interactive segmentation over previous approaches, allowing a robot to effectively separate textureless objects, while providing confident reasoning about potential object boundaries.

Our approach to learning contact locations as a function of object shape represents a significant contribution of this dissertation. We show in Chapter 4 that a robot can learn to predict good initial contact locations for pushing previously unseen objects. The robot collects all training data autonomously through interaction. In our eyes this represents an extremely attractive property for learning in robotics. Needing little supervision from a human teacher allows the robot to continuously improve its performance through its deployment. One key to the autonomous data collection lies in characterizing good and bad pushing locations using a score derived from the entire push trajectory. These scores—characterizing stable, straight-line pushing and orienting pushes—inherently account for all processes involved in performing the push. In addition to being robust to noise introduced from the simple tracking methods used by the robot, more complicated information such as kinematic constraints and calibration errors become encoded in the scores as well. As long as the procedure used for collecting training data reflects how the robot will perform at deployment the robot has no need to tease apart the specific elements which make a particular push good or bad. This provides a benefit over simulation or human defined heuristics where it may be impossible a priori to define and account for all possible influences on the push quality. This robustness to noise and minor errors plays an especially important role in interacting with novel objects, where it is unreasonable to expect high quality object models to be available for tracking or simulation. This issue became even more

apparent to us in conducting the experiments presented in Chapter 5.

We examine in Chapter 5 the utility of dynamics models for control in pushing. We show that even when the model used is know to poorly represent the underlying dynamics, feedback can be used to achieve state-of-the-art performance. Given higher quality models a robot should achieve even better performance. As such, we examine the ability of a robot to learn dynamics models through interaction. Our attempts were insufficient in learning models useful to manipulation, which we attribute primarily to the level of noise present in our object tracking system. However, our results on contact location learning provide evidence that a useful learning signal is present in the trajectory recovered from our object tracker. Making use of our factored affordance-based behavior representation we show that the regressors learned for predicting stable contact locations produce improved pushing performance when used by our model predictive controller. These locations additionally provide benefit when used with a distinct behavior primitive from the one used at training. While this demonstrates how knowledge transfer can be performed using our behavior representation, the limits of generalization are unknown.

This dissertation presents a concrete use of affordance-based behaviors for autonomous learning on a robot. Using this representation our robot was able to learn meaningful strategies for experimenting with and effectively manipulating previously unseen objects. These results generate a number of questions we leave open to the research community. First, how can the robot use contextual information about its environment to more quickly search through the available behavior factors? For example, if we add as a component of the behavior the arm being used— left of right—the robot should be able to learn that pushing at the extreme right of its workspace is not possible with the left arm, independent of the controller being used. Second, many questions remain open in the context of task level planning. Specifically, how can learning and singulation be introduced as possible actions to a planner, so that the

robot may explore novel manipulation strategies while still making progress towards its goal? Finally, how extensible is our representation? Humans produce an enormous range of skillful manipulation strategies, beyond simple pushing and grasping. What must be added to our representation for a robot to make use of it for other tasks such as throwing or catching?

Enabling robots to operate autonomously alongside humans has the potential to help better the lives of countless people. Providing robots with the ability to generalize and improve their repertoire of available skills remains paramount to ensuring this success. This dissertation contributes a step towards this goal, presenting a mechanism by which robots can learn how to better interact with their environment, learning strategies that can be leveraged in novel situations and manged by higher-level planners or human operators.

REFERENCES

[1] Akella, S. and Mason, M. T., "Posing Polygonal Objects in the Plane by Pushing," *The International Journal of Robotics Research (IJRR)*, vol. 17, pp. 70–88, January 1998.

[2] Alexe, B., Deselaers, T., and Ferrari, V., "What is an object?," in *CVPR*, 2010.

[3] Amor, H. B., Kroemer, O., Hillenbrand, U., Neumann, G., and Peters, J., "Generalization of Human Grasping for Multi-Fingered Robot Hands," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, October 2012.

[4] Balorda, Z. and Bajd, T., "Reducing Positioning Uncertainty of Objects by Robot Pushing," *IEEE Transactions on Robotics and Automaton*, vol. 10, no. 4, pp. 535–541, 1994.

[5] Barck-Holst, C., Ralph, M., Holmar, F., and Kragic, D., "Learning Grasping Affordance Using Probabilistic and Ontological Approaches," in *International Conference on Advanced Robotics (ICAR)*, (Munich, Germany), pp. 1–6, June 2009.

[6] Barry, J., Hsiao, K., Kaelbling, L. P., and Lozano-Pérez, T., "Manipulation with Multiple Action Types," in *International Symposium on Experimental Robotics*, 2012.

[7] Barry, J., Kaelbling, L. P., and Lozano-Perez, T., "A Hierarchical Approach to Manipulation with Diverse Actions," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[8] Belongie, S., Malik, J., and Puzicha, J., "Shape Matching and Object Recognition Using Shape Contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, April 2002.

[9] Bernheisel, J. D. and Lynch, K. M., "Stable Transport of Assemblies by Pushing," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 740–750, 2006.

[10] Bertsekas, D. P., *Dynamic Programming and Optimal Control*, vol. 1. Athena Scientific, fourth ed., 2007.

[11] Besl, P. J. and McKay, N. D., "A Method for Registration of 3-D Shapes," *PAMI*, vol. 14, pp. 239–256, 1992.

[12] Bohg, J. and Kragic, D., "Learning Grasping Points with Shape Context," *Journal of Robotics and Autonomous Systems*, vol. 58, pp. 362–377, April 2010.

[13] Bolles, R. and Fischler, M., "A RANSAC-based approach to model fitting and its application to finding cylinders in range data," in *Seventh International Joint Conference on Artificial Intelligence, (Vancouver, British Colombia, Canada)*, pp. 637–643, 1981.

[14] Chang, L. Y., Smith, J. R., and Fox, D., "Interactive Singulation of Objects from a Pile," in *IROS: The PR2 Workshop*, 2011.

[15] Chang, L. Y., Smith, J. R., and Fox, D., "Interactive Singulation of Objects from a Pile," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[16] Chui, H. and Rangarajan, A., "A new point matching algorithm for non-rigid registration," *Computer Vision and Image Understanding*, vol. 89, no. 2-3, pp. 114 – 141, 2003.

[17] Cosgun, A., Hermans, T., Emeli, V., and Stilman, M., "Push Planning for Object Placement on Cluttered Table Surfaces," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2011.

[18] Deisenroth, M. P. and Fox, D., "Multiple-Target Reinforcement Learning with a Single Policy," in *ICML Workshop on Planning and Acting with Uncertain Models*, June 2011.

[19] Deisenroth, M. P. and Rasmussen, C. E., "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *International Conference on Machine Learning (ICML)*, June 2011.

[20] Deisenroth, M. P., Rasmussen, C. E., and Fox, D., "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," in *Robotics Science and Systems (RSS)*, June 2011.

[21] Dogar, M. and Srinivasa, S., "Push-Grasping with Dexterous Hands: Mechanics and a Method," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2010.

[22] Dogar, M. and Srinivasa, S., "A Framework for Push-Grasping in Clutter," in *Robotics Science and Systems (RSS)*, 2011.

[23] Emeli, V., Kemp, C. C., and Stilman, M., "Push Planning for Object Placement in Clutter Using the PR-2," in *IROS: The PR2 Workshop*, 2011.

[24] Erkan, A. N., Kroemer, O., Detry, R., Altun, Y., Piater, J., and Peters, J., "Learning probabilistic discriminative models of grasp affordances under limited supervision," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1586–1591, Oct. 2010.

[25] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D., "Object detection with discriminatively trained part based models.," *PAMI*, vol. 32, September 2010.

[26] Fischler, M. A. and Bolles, R. C., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, June 1981.

[27] Fitzpatrick, P., Metta, G., Natale, L., Rao, S., and Sandini, G., "Learning about objects through action - initial steps towards artificial cognition," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, pp. 3140–3145, Sept 2003.

[28] Fitzpatrick, P. M. and Metta, G., "Towards Manipulation-Driven Vision," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pp. 43–48, 2002.

[29] Fitzpatrick, P. M. and Metta, G., "First Contact: an Active Vision Approach to Segmentation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2003.

[30] Fritz, G., Paletta, L., Kumar, M., Dorffner, G., Breithaupt, R., and Rome, E., "Visual learning of affordance based cues," in *From Animals to Animats 9*, pp. 52–64, 2006.

[31] Gandolfo, F., Tistarelli, M., and Sandini, G., "Towards Vision Guided Manipulation," in *International Conference on Advanced Robotics (ICAR)*, pp. 661–667, 1991.

[32] Gandolfo, F., Tistarelli, M., and Sandini, G., "Visual Monitoring of Robot Actions," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pp. 269–275, 1991.

[33] Geib, C., Mourao, K., Petrick, R., Pugeault, N., Steedman, M., Krueger, N., and W org otter, F., "Object Action Complexes as an Interface for Planning and Robot Control," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, (Genova, Italy), Dec 4-6 2006.

[34] Gibson, J. J., "The theory of affordances," in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology* (Shaw, R. and Bransford, J., eds.), pp. 67–82, Hillsdale, NJ: Lawrence Erlbaum, 1977.

[35] Gibson, J. J., "The theory of affordances," in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology* (Shaw, R. and Bransford, J., eds.), pp. 67–82, Hillsdale, NJ: Lawrence Erlbaum, 1977.

[36] Goyal, S., Ruina, A., and Papadopoulos, J., "Limit Surface and Moment Function Descriptions of Planar Sliding," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 1989.

[37] Gupta, M. and Sukhatme, G. S., "Using Manipulation Primitives for Brick Sorting in Clutter," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[38] Hermans, T., Li, F., Rehg, J. M., and Bobick, A. F., "Learning Contact Locations for Pushing and Orienting Unknown Objects," in *IEEE-RAS International Conference on Humanoid Robotics*, October 2013.

[39] Hermans, T., Li, F., Rehg, J. M., and Bobick, A. F., "Learning Stable Pushing Locations," in *ICDL-EPIROB*, August 2013.

[40] Hermans, T., Rehg, J. M., and Bobick, A., "Affordance Prediction via Learned Object Attributes," in *IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*, May 2011.

[41] Hermans, T., Rehg, J. M., and Bobick, A., "Guided Pushing for Object Singulation," in *IROS*, October 2012.

[42] Hermans, T., Rehg, J. M., and Bobick, A. F., "Decoupling Behavior, Control, and Perception in Affordance-Based Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Cognitive Assistive Systems*, October 2012.

[43] Hermans, T., Rehg, J. M., and Bobick, A. F., "Decoupling Behavior, Perception, and Control for Autonomous Learning of Affordances," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

[44] Hillenbrand, U. and Roa, M. A., "Transferring Functional Grasps through Contact Warping and Local Replanning," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, October 2012.

[45] Hofmann, T., Schölkopf, B., and Smola, A. J., "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, pp. 1171–1220, 2008.

[46] Howe, R. D. and Cutkosky, M. R., "Practical Force-Motion Models for Sliding Manipulation," in *The International Journal of Robotics Research (IJRR)*, 1996.

[47] Iida, F., Pfeifer, R., and Steels, L., *Embodied Artificial Intelligence International Seminar, Dagstuhl Castle, Germany, July 7-11, 2003, Revised Papers*, vol. 3139 of *Lecture Notes in Computer Science*. Springer, 2004.

[48] Ijspeert, A. J., Nakanishi, J., and Schaal, S., "Learning Attractor Landscapes for Learning Motor Primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[49] Jiang, Y., Moseson, S., and Saxena, A., "Efficient Grasping from RGBD Images: Learning using a new Rectangle Representation," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[50] Jiang, Y., Zheng, C., Lim, M., and Saxena, A., "Learning to Place New Objects," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[51] Katz, D. and Brock, O., "Extracting Planar Kinematic Models Using Interactive Perception," in *In Unifying Perspectives In Computational and Robot Vision*, vol. 8 of *Lecture Notes in Electrical Engineering*, pp. 11–23, Springer Verlag, May 2008.

[52] Katz, D. and Brock, O., "A Factorization Approach to Manipulation in Unstructured Environments," in *Int. Symp. on Robotics Research*, (Lucerne, Switzerland), pp. 1–16, Springer Verlag, August 31-September 3 2009.

[53] Katz, D. and Brock, O., "Interactive Segmentation of Articulated Objects in 3D," in *Workshop on Mobile Manipulation at ICRA 2011*, 2011.

[54] Katz, D., Orthey, A., and Brock, O., "Interactive Perception of Articulated Objects," in *International Symposium on Experimental Robotics*, 2010.

[55] Katz, D., Pyuro, Y., and Brock, O., "Learning to Manipulate Articulated Objects in Unstructured Environments Using a Grounded Relational Representation," in *Proceedings of Robotics: Science and Systems IV*, (Zurich, Switzerland), pp. 254–261, June 2008.

[56] Kenney, J., Buckley, T., and Brock, O., "Interactive Segmentation for Manipulation in Unstructured Environments," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1377–1382, 2009.

[57] Killpack, M. D. and Kemp, C. C., "Fast reaching in clutter while regulating forces using model predictive control," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2013.

[58] Klank, U., Mösenlechner, L., Maldonado, A., and Beetz, M., "Robots that Validate Learned Perceptual Models," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[59] Klingbeil, E., Drao, D., Carpenter, B., Ganapathi, V., Khatib, O., and Ng, A. Y., "Grasping with application to an autonomous checkout robot," in *ICRA*, 2011.

[60] Kober, J., Oztop, E., and Peters, J., "Reinforcement Learning to adjust Robot Movements to New Situations," in *Robotics Science and Systems (RSS)*, 2010.

[61] Kober, J. and Peters, J., "Learning Motor Primitives for Robotics," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[62] Kober, J., Wilhelm, A., Oztop, E., and Peters, J., "Reinforcement Learning to Adjust Parametrized Motor Primitives to New Situations," *Autonomous Robots*, vol. 33, no. 4, pp. 361–379, 2012.

[63] Kopicki, M., Zurek, S., Stolkin, R., Morwald, T., and Wyatt, J., "Learning to predict how rigid objects behave under simple manipulation," in *ICRA*, pp. 5722–5729, 2011.

[64] Kraft, D., "Algorithm 733; TOMP - Fortran modules for optimal control calculations," *ACM Trans. Math. Softw.*, vol. 20, no. 3, pp. 262–281, 1994.

[65] Kroemer, O., Ugur, E., Oztop, E., and Peters, J., "A Kernel-based Approach to Direct Action Perception," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[66] Li, W. H. and Kleeman, L., "Autonomous Segmentation of Near-Symmetric Objects through Vision and Robotic Nudging," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pp. 3604–3609, 2008.

[67] Li, W. H. and Kleeman, L., "Interactive learning of visually symmetric objects," in *IROS*, 2009.

[68] Lopes, M., Melo, F., and Montesano, L., "Affordance-based imitation learning in robots," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 1015–1021, 29 2007-Nov. 2 2007.

[69] Lynch, K. M., "Estimating the Friction Parameters of Pushed Objects," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pp. 186–193, 1993.

[70] Lynch, K. M., "Toppling Manipulation," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[71] Lynch, K. M. and Mason, M. T., "Controllability of Pushing," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 112–119, 1995.

[72] Lynch, K. M. and Mason, M. T., "Stable Pushing: Mechanics, Controllability, and Planning," in *The International Journal of Robotics Research (IJRR)*, pp. 533–555, 1996.

[73] Madry, M., Song, D., and Kragic, D., "From Object Categories to Grasp Transfer Using Probabilistic Reasoning," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[74] Martin, D. R., Folkes, C. C., and Malik, J., "Learning to detect natural image boundaries using local brightness, color and texture cues," *PAMI*, vol. 26, pp. 530–549, May 2004.

[75] Mason, M. T., "Mechanics and Planning of Manipulator Pushing Operations," *The International Journal of Robotics Research (IJRR)*, vol. 5, pp. 53–71, September 1986.

[76] Mason, M. T., Srinivasa, S., and Vazquez, A. S., "Generality and simple hands," in *ISRR*, July 2009.

[77] Metta, G. and Fitzpatrick, P., "Early integration of vision and manipulation," *Adaptive Behavior*, vol. 11, pp. 109–128, 2003.

[78] Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J., "Learning object affordances: From sensory-motor coordination to imitation," *IEEE Transactions on Robotics*, vol. 24, pp. 15–26, Feb 2008.

[79] Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J., "Modeling object affordances using bayesian networks," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2007.

[80] Muelling, K., Kober, J., and Peters, J., "Learning Table Tennis with a Mixture of Motor Primitives," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2010.

[81] Narasimhan, S., *Task Level Strategies for Robots*. PhD thesis, Massachusetts Institute of Technology, 1994.

[82] Omrčen, D., Böge, C., Asfour, T., Ude, A., and Dillmann, R., "Autonomous Acquisition of Pushing Actions to Support Object Grasping with a Humanoid Robot," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, (Paris, France), 2009.

[83] Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S., "Learning and Generalization of Motor Skills by Learning from Demonstration," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[84] Peshkin, M. A. and Sanderson, A. C., "The Motion of a Pushed, Sliding Workpiece," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 569–598, December 1988.

[85] Pingle, K., Paul, R., and Bolles, R., "Programmable Assembly, Three Short Examples." Film, 1974.

[86] Quoc Le, D. K. and Ng, A. Y., "Learning to grasp objects with multiple contact points," in *International Conference on Robotics and Automation (ICRA)*, 2010.

[87] Rao, A. S. and Goldberg, K. Y., "On the Relation between Friction and Part Shape in Parllel-Jaw Grasping," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 461–466, 1993.

[88] Ridge, B., Skočaj, D., and Leonardis, A., "Self-Supervised Cross-Modal Online Learning of Basic Object Affordances for Developmental Robotic Systems," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, (Anchorage, USA), May 2010.

[89] Ridge, B., Skocaj, D., and Leonardis, A., "A System for Learning Basic Object Affordances using a Self-Organzing Map," in *International Conference on Cognitive Systems (CogSys)*, (Karlsruhe, Germany), 2008.

[90] Ridge, B., Skocaj, D., and Leonardis, A., "Towards Learning Basic Object Affordances from Object Properties," in *International Conference on Epigenetic Robotics*, 2008.

[91] Ridge, B., Skocaj, D., and Leonardis, A., "Unsupervised Learning of Basic Object Affordances from Object Properties," in *Computer Vision Winter Workshop (CVWW)*, 2009.

[92] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., "ORB: An Efficient Alternative to SIFT or SURF," in *International Conference on Computer Vision*, (Washington, DC, USA), pp. 2564–2571, IEEE Computer Society, 2011.

[93] Ruiz-Ugalde, F., Cheng, G., and Beetz, M., "Prediction of action outcomes using an object model," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2010.

[94] Ruiz-Ugalde, F., Cheng, G., and Beetz, M., "Fast Adaptation for Effect-aware Pushing," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2011.

[95] Salganicoff, M., Metta, G., Oddera, A., and Sandini, G., "A direct approach to vision guided manipulation," in *International Conference on Advanced Robotics (ICAR)*, 1993.

[96] Salganicoff, M., Metta, G., Oddera, A., and Sandini, G., "A vision-based learning method for pushing manipulation," in *AAAI Fall Symposium on Machine Learning in Computer Vision*, 1993.

[97] Saxena, A., Driemeyer, J., and Ng., A. Y., "Robotic grasping of novel objects using vision," *International Journal of Robotics Research (IJRR)*, vol. 27, pp. 157–173, Feb 2008.

[98] Schaal, S. and Atkeson, C. G., "Learning Control in Robotics," *IEEE Robotics Automation Magazine*, vol. 17, pp. 20–29, June 2010.

[99] Scharr, H., "Optimal Second Order Derivative Filter Families for Transparent Motion Estimation," in *EUSIPCO*, September 2007.

[100] Schiebener, D., Ude, A., Morimoto, J., Asfour, T., and Dillmann, R., "Segmentation and learning of unknown objects through physical interaction," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2011.

[101] Scholz, J. and Stilman, M., "Combining Motion Planning and Optimization for Flexible Robot Manipulation," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2010.

[102] Schulman, J. and Abbeel, P., "Personal email," October 2012.

[103] Shi, J. and Tomasi, C., "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pp. 593–600, June 1994.

[104] Sinapov, J. and Stoychev, A., "Learning and Generalization of Behavior-Grounded Tool Affordances," in *IEEE International Conference on Developmental Learning (ICDL)*, pp. 19–24, 2007.

[105] Stark, M., Lies, P., Zillich, M., Wyatt, J., and Schiele, B., "Functional object class detection based on learned affordance cues," in *6th International Conference on Computer Vision Systems (ICVS)*, (Santorini, Greece), 2008 2008. Oral presentation.

[106] Stoychev, A., "Behavior-Grounded Representation of Tool Affordances," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3060–3065, 2005.

[107] Stoytchev, A., "Behavior-grounded representation of tool affordances," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[108] Sun, J., Moore, J. L., Bobick, A., and Rehg, J. M., "Learning Visual Object Categories for Robot Affordance Prediction," *The International Journal of Robotics Research*, vol. 29, no. 2-3, pp. 174–197, 2010.

[109] Ugur, E., Sahin, E., and Oztop, E., "Self-discovery of motor primitives and learning grasp affordances," in *IROS*, pp. 3260–3267, 2012.

[110] Ugur, E., Oztop, E., and Sahin, E., "Going Beyond The Perception of Affordances: Learning How to Actualize Them Through Behavioral Parameters," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[111] Ugur, E., Sahin, E., and Oztop, E., "Unsupervised Learning of Object Affordances for Planning in a Mobile Manipulation Platform," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[112] van Hoof, H., Kroemer, O., Amor, H. B., and Peters, J., "Maximally Informative Interaction Learning for Scene Exploration," in *IROS*, October 2012.

[113] Vedaldi, A., Gulshan, V., Varma, M., and Zisserman, A., "Multiple kernels for object detection," in *International Conference on Computer Vision*, 2009.

[114] Yoshikawa, T. and Kurisu, M., "Identification of the Center of Friction from Pushing an Object by a Mobile Robot," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, pp. 449–454, 1991.

[115] Zito, C., Stolkin, R., Kopicki, M., and Wyatt, J. L., "Two-level RRT Planning for Robotic Push Manipulation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, October 2012.

[116] Zrimec, T. and Mowforth, P., "Learning by an autonomous agent in the pushing domain," *Robotics and Autonomous Systems*, vol. 8, pp. 19–29, November 1991.