2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Pushing Using Learned Manipulation Maps

Sean Walker
Department of Computer Science
Stanford University
Stanford, CA 94305
spw@cs.stanford.edu

J. Kenneth Salisbury
Departments of Computer Science, Surgery,
and Mechanical Engineering
Stanford University
Stanford, CA 94305
jks@robotics.stanford.edu

***Abstract*— Robot haptics ultimately consists of a set of models which interpret and predict a robot's physical interaction with the world. In this paper, we describe one approach to modeling support friction within a two-dimensional environment consisting of a single robot finger pushing objects on a table. Instead of explicitly modeling the friction distribution between the object and the table, we learn the mapping between pushes and the motion of the object using an online, memory-based model using local regression. The resulting manipulation map implicitly describes the support friction without a complex model. We also describe methods of acquiring object shape and localizing the object using a proximity sensor. Results are presented for objects with different friction distributions.**

I. INTRODUCTION

While human hands are amazing results of evolutionary engineering with a yet unmatched power to weight ratio and sensor density, they are surprisingly imprecise from a control standpoint. Most robot fingers can position themselves within a fraction of a millimeter in free space, while humans, even highly trained surgeons, are unable to position a finger within a millimeter of a desired location without visual feedback and some method to brace the hand. Yet, even a school-child is able to write a letter, pick up a bucket, and spin an apple within her fingertips within the space of a few minutes, a medley of tasks no robot hand can accomplish. The underlying kernel of our current research is the assumption that flexible and robust control algorithms are more important than precision robot hardware.

A major obstacle for hand control research is the sheer complexity of controlling the twelve or more degrees of freedom of the typical human-modeled robot hand [1] [2] [3]. Beyond just controlling all those motors and utilizing information from all those sensors, there is also the difficulty of maintaining a grasp in three dimensions. Our solution is to drastically reduce the number of degrees of freedom by focusing on pushing with a single 2-dof finger. While the 2-D problem is significantly less complicated, even a single finger pushing an object in the plane exhibits many of the difficulties of the full 3-D problem: modeling contact and friction, dealing with inaccuracies, and processing an avalanche of sensory information in time to act on that knowledge.

Our system, the Probabilistic Manipulation Experiment Table (PMET) consists of a single robot finger pushing objects within a plane (see Figure 1). The robot's structure is a modified version of the last two joints of the PHANTOM haptic device [4], made out of wood using rapid prototyping, techniques, and turned on its side for horizontal manipulation. Like the PHANTOM, it is driven through pretensioned cable reductions. It includes a proximity sensor [5], accelerometers, force sensing based on motor torques, and encoders. The full system is controlled with the Probabilistic Robotics Studio (PRS), our own hard-real time visual programming system [6].

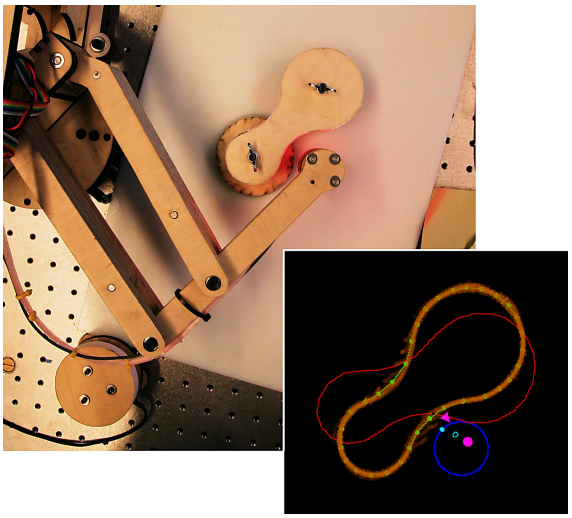


Fig. 1. The PMET pushing robot in action.

This paper represents our first major task using the PMET: to find, explore, and push an arbitrarily shaped 2-D object with an unknown support friction distribution to a desired position and orientation.

A. Previous Work

Manipulation by pushing is instrumental when moving an object that is too heavy to lift. Due to obvious manufacturing applications, pushing has been an active area of research for decades. Mason provides a good overview of pushing and planning techniques in [7]. Likewise, [8] demonstrates that object motion as a result of a push is based on the support friction distribution (the distribution of friction between the object and floor) and approaches the problem from a learning perspective. While the range of possible pushes is huge,

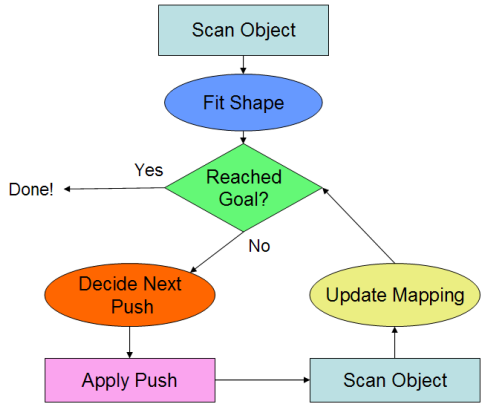


Fig. 2. The block diagram for the manipulation map algorithm.

especially if multiple contact points are allowed [9], we restrict our research to use only straight line pushes with a single contact point. Akella demonstrated that planning using such pushes could achieve any orientation in [10].

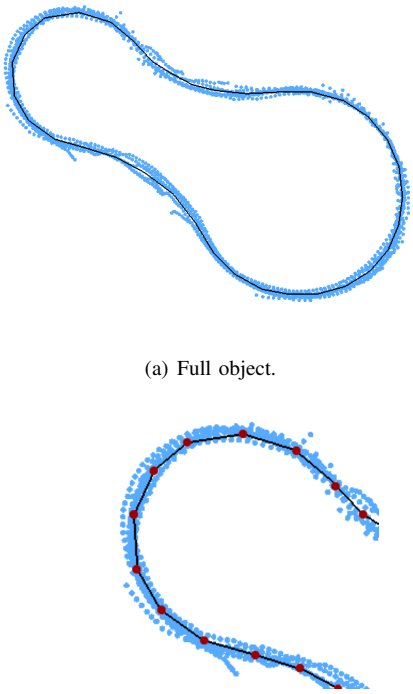
When pushing the most important haptic property of the object is support friction. Other researchers have estimated support friction using active exploration, such as [11] (which uses vision measurements to compute friction parameters) and [12] (which uses a force sensor and human-measured position and motion). Initially, we attempted to estimate and model the distribution, but we found that even in our simple environment, the interaction between the table (a plastic sheet) and the object 'foot' (wood) was hard to quantify with a simple model due to imperfections in the contact surfaces and bits of dirt between the object and table. Instead, we found that a memory-based model of previous push actions, or *manipulation map*, was more robust and easier to obtain.

We are not the first to use memory of past pushing operations to predict the result of future pushing operations. In particular, [13] also describes an unsupervised on-line method which selects each push direction using a weighted, stochastic, nearest-neighbor approach. Their approach is similar to ours, except we use a proximity sensor instead of optical flow to track the object and we do not restrict the point of contact to a single, notched location. Other researchers outside of pushing manipulation have used local regression of memory-based models, such as [14], which uses a neural network to provide nearest neighbor lookup.

B. Contributions

Our notable contributions described in this paper, are:

- Novel use of proximity sensor for both shape acquisition and localization.
- A memory-based technique that learns support friction by mapping pushes to motion (a *manipulation map*).
- A heuristic for updating the manipulation map which explores the object as it manipulates and implicitly generates a locally weighted regression.



(a) Full object.

(b) Detail of the narrow end, showing the object outline, a spline of degree 1.

Fig. 3. The proximity cloud for a peanut-shaped object.

Our work is presented in the form of the overall manipulation algorithm as shown in Figure 2. The algorithm is structured as a loop: planning a push, implementing the push, localizing the object, and updating the manipulation map with the new knowledge.

The rest of the paper describes the implementation of each of the basic capabilities required for the algorithm. Section II describes the use of the proximity cloud in acquiring object shape and localizing the object. Section III describes the manipulation technique for applying pushes and Section IV describes the manipulation map update method. Section V describes how each push is chosen from the set of candidate pushes. Section VI describes some experimental results and the final section provides a summary and areas of future research.

II. USING THE PROXIMITY CLOUD

The object shape and pose are both acquired using a cloud of proximity measurements from the proximity sensor. In order to make the cloud, the finger moves around the object at a distance of approximately 10mm and stores proximity measurements (corresponding to actual points on the object) as it goes. The contour following controller is described in [5]. Objects must be restricted to those with relatively low curvature (radii more than 10mm) to avoid confusing the proximity sensor since it cannot detect corners.

The proximity sensor measurements include a decent amount of noise (up to a 3mm displacement) but they are centered around the actual outline of the object. The next two

sections describe how the cloud of proximity measurements is used to construct a model of the object's shape and localize the object as it is moved around the table.

A. Object Shape

After the system initially acquires an object (by scanning through the workspace from side to side), it generates a proximity cloud and fits a first-degree spline (polyline) to the cloud. The initial control points of the spline are generated by sampling proximity measurements every centimeter. Figure 3 shows the proximity cloud and the fitted object shape.

After acquiring a rough model, a Bayesian optimizer refines the spline to better fit the cloud of points (based on [15]). If the object outline is defined by a set of n control points $\vec{x}$ and there are m proximity measurements $\vec{z}$, the best fit of the spline to the proximity cloud is defined as the set of control points which minimize:

$$f(\vec{x}, \vec{z}) = f_m(\vec{x}, \vec{z}) + f_p(\vec{x}) \tag{1}$$

Where $f_m(\vec{x}, \vec{z})$ is a function describing the sum of squares distance of the cloud to the spline and $f_p(\vec{x})$ applies a second derivative curvature constraint to avoid overfitting. Specifically:

$$f_m(\vec{x}, \vec{z}) = \sum_{0 \leq j < m} \|z_j - g(\vec{x}, z_j)\|^2 \tag{2}$$

$$f_p(\vec{x}) = \lambda \sum_{0 \leq i < n} \|x_{i-1} - 2x_i + x_{i+1}\|^2 \tag{3}$$

$g(\vec{x}, z_j)$ is a function which evaluates to the point on the first degree spline defined by $\vec{x}$ which is closest to z_j. In our implementation, we determine the closest point using a $O(n)$ technique which computes the closest point on each spline segment and returns the point which minimizes distances to z_j. λ is a constant that controls how 'smooth' the curve is. Since the object is a closed curve, we assume that the points wrap at each end (x_n maps to x_0 and x_{-1} maps to x_{n-1}). To minimize this sum of squares problem, we used the Levenberg-Marquardt non-linear least squares method.

B. Localization

Once the object shape is known, the object position and orientation can by found at any time by retracing the object contour and generating a new cloud of proximity measurements. The result of each push is found by optimizing the object pose based on this new cloud.

Given $\vec{x}$ and $\vec{z}$, we can find the best object center position $\vec{p}$ and object orientation θ by minimizing the sum of squared error:

$$f_L(\vec{p}, \theta) = \sum_{0 \leq j < m} \|z_j - g(R(\theta) * (\vec{x} + \vec{p}), z_j)\|^2 \tag{4}$$

Where $R(\theta)$ is the rotation matrix from θ. Again, we used the Levenberg-Marquardt to minimize f_L.

III. PUSHING (MANIPULATION)

The primary goal in defining a manipulation operation (a push) is repeatability: each push should produce results similar to previous pushes at that location. While many properties of the push can not be precisely controlled (the exact friction between the finger and object, the rotation of the fingertip as it moves, etc.), we found that straight line pushes are repeatable enough to give our desired degree of accuracy. After all, we aren't concerned with exact manipulation within fractions of a millimeter; most human-scale tasks require positioning within a few millimeters or even tens of millimeters.

The first step of the pushing procedure is to determine where and how far to push. Section V describes how the location and magnitude of each push is decided in more detail. It is important to mention that pushing only occurs at discrete points, specifically a subdivision of the control points of the shape spline model with approximately 3mm spacing. By pushing along the surface normal at specific points, we minimize tangential slip and reduce the number of possible pushes that need to be considered when planning.

Pushing procedure:

1) Line up 20mm from the object along the push line. Lining up this far from the object ensures consistency.
2) Move the fingertip toward the object along the push line until accelerometers detect contact and the proximity sensor says the object is less than 2mm away.
3) Record the starting point of the push.
4) Continue to move along the push line and push the object.
5) When the difference between the current finger position and the starting position is greater than the push length, stop.
6) Immediately move the finger backwards along the push line to avoid secondary manipulation of the object.

While pushes are relatively consistent, the inertia of the finger causes small pushes to produce approximately ten percent more motion than desired. Typically, this only happens with pushes of less than 3mm, and since the desired accuracy of the final manipulation is about 2mm, it does not impact manipulation significantly.

IV. ONLINE LEARNING

A major contribution of this work is a novel method to update and maintain a manipulation map over time. This mapping encapsulates the friction distribution between the object and the table. Ultimately, this method results in a local regression of past measurements at each of the possible push points (push points are a 3mm subdivision of the object spline control points).

The manipulation map is a representation of the expected change in position and orientation at a finite set of possible push points $\vec{y}_k$. For each $\vec{y}_k$ there is an associated $\vec{q}_k = [\Delta x, \Delta y, \Delta\theta]$ which defines the change in pose normalized to a 1mm push. Note that this change in pose is in reference to the center of the object (the centroid of the object shape

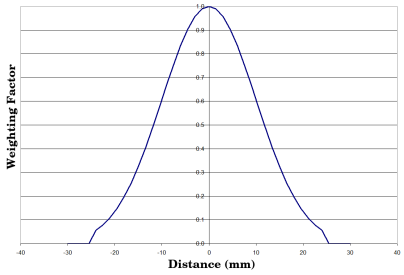


Fig. 4. The truncated Gaussian curve for location-based weighting.

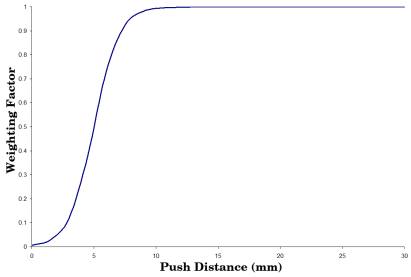


Fig. 5. The curve used for weighting based on push distance.

spline control points). There is also an associated scalar weighting, w_k, which represents how many measurements are contained in that pose change estimation (larger weights mean more confidence). The combination of all push points and their associated pose changes and weights makes up the manipulation map.

In our experiments, we found all pushes can be normalized to the scale of 1mm pushes as long as slip is minimized by pushing normal to the object surface. On initialization, assuming a 1mm push vector of $\vec{u}_k$ at $\vec{y}_k$, $\vec{q}_k = [u_x, u_y, R() * 0.01rad]$ and $w_k = 0.3$. $R()$ is a function that generates random numbers in the normal distribution with a standard deviation of 1. While the translation is a decent approximation of the true result for most objects, the random rotation component is often very far from the correct value. But, the random rotation helps the system to explore the surface during manipulation. The weights are initialized to the relatively small value of 0.3 (successful pushes result in weightings of 1.0) to allow new measurements to quickly change the initial map.

Each push modifies more than one entry of the manipulation map to allow localized spread of information. This spreading feature (along with averaging) produces an effective local regression in the manipulation map. The guiding feature in the algorithm is a set of adjustments, a_i, defined for each push point. For a given push of length l_k at push point $\vec{y}_k$, the adjustments are computed as:

$$a_i = exp(-\frac{\|\vec{y}_k - \vec{y}_i\|^2}{2 * \sigma^2}) * \frac{1}{1 + exp(-(l_k - 5mm))} \quad (5)$$

The first term of a_i controls the spread of the push information to adjacent push points with a standard deviation σ (typically set to 10mm). Figure 4 shows a plot of this function. The second term attenuates the weighting for small pushes to prevent noise from impacting the result. Figure 5 is a plot of this second term. As you can see, the final value of a_i is at most 1.0 ($a_k = 1.0$).

Once the adjustments are computed, the manipulation map is updated based on the new pose change, $\vec{r}$, according to:

$$\vec{q}'_i = \frac{\vec{q}_i * w_i + \vec{r} * a_i}{w_i + a_i} \quad (6)$$

$$w'_i = w_i + a_i \quad (7)$$

The end result is a sequence of updates as shown in Figure 6. Each T represents the change in position and orientation of the center of the object from a 1mm push. Any rotation of the T from upright means the object rotated in that direction, and the translation of the T off of the push point on the object represents how far the center of the object translated as a result of the push. The size of the T directly corresponds to w_i; a larger T means a high weighting and more confidence in the mapping. Portions (a) and (b) show the map before and after the first update. Likewise, (c) and (d) depict the results of the third push update. As the number of pushes increase (due to multiple goal points) the manipulation map becomes a detailed description of how the object moves.

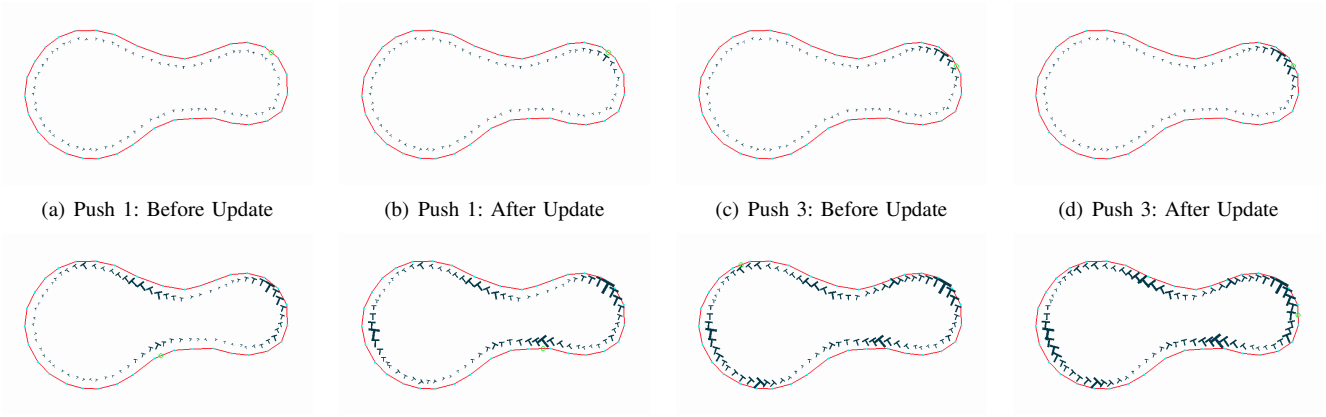
V. CHOOSING THE NEXT PUSH

The final component of the algorithm is a method of deciding where and how far to push the object at each step in order to move it to some goal position and orientation. The specific method is not important for the eventual repositioning of the object as long as each push moves the object toward the chosen goal. But, the method used will heavily influence how many pushes are required to get the object to the goal. In this paper we will describe one method which does not require any in-depth planning.

All push choice algorithms require two things: a method of simulating a push and a metric for measuring how effective the simulated push is. The manipulation map can be used directly to simulate the results of the push; just reorient the object model according to the corresponding mapping estimate, scaled based on the size of the push. The metric is defined as the sum of squared differences of the control points in the object spline for the simulated pose and the goal pose.

The optimal push location and distance could be computing by enumerating all possible pushes and distances, *and all sequences of all possible pushes*, and choosing the sequence which results in the shortest number of pushes (or another metric) to reach the goal. This is essentially planning to an unlimited horizon, but in practice, such planning would be too time consuming. Instead, we make some approximations, namely not planning beyond one step.

The method we currently use is a greedy one: for each push point, enumerate all possible pushes at 0.5mm resolution up to the maximum push length. The maximum push length is controlled by the weighting of that push point in a



(a) Push 1: Before Update (b) Push 1: After Update (c) Push 3: Before Update (d) Push 3: After Update

(e) Push 10 (f) Push 18 (g) Push 26 (h) Push 33

Fig. 6. A sequence of updates during the manipulation map learning algorithm. Each T represents the results of a push at the corresponding push point. The position and orientation of the T reflect the translation and rotation of the object respectively, while the size is directly proportional to the weighting.

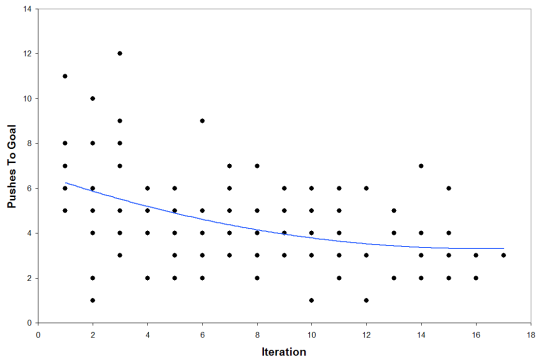


Fig. 7. Data from a sequence of runs.

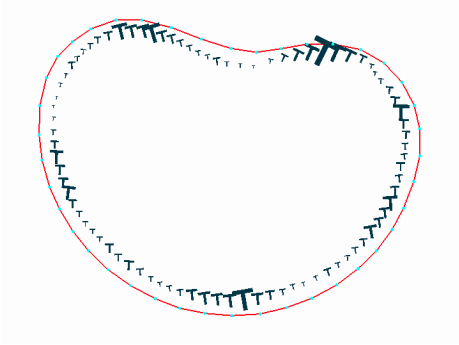


Fig. 8. Final manipulation map for a kidney-shaped object.

linear fashion so the smallest maximum push is 5mm and the largest is 30mm. For each of the enumerated pushes, simulate the result and compute a measurement metric. Then, execute the the push which results in the smallest metric value.

VI. EXPERIMENTAL RESULTS

A. Learning Speed

In order to test the speed and quality of the learned model, we ran a series of 10 trials with the 'peanut' object. Each trial produced an ordered set of 10 or more iterations and each iteration measured how many pushes it took to get to a randomly chosen goal position and orientation. Goal positions varied in x and y directions within a 4cm square and varied in orientation by 90 degrees. Success was declared when all control points of the object were within 1mm of the goal control points. The results are shown in Figure 7. The line in the figure is a curve fit to the average performance at each iteration. It clearly shows the decrease in pushes required to reach the goal as the algorithm learned.

Even after 10 iterations, the required number of pushes to get the object to the goal ranges from 1 to 6. Since the goal location is chosen randomly at each iteration, sometimes the goal is far from the current position, requiring many pushes, and sometimes the robot gets lucky and the goal is very close.

B. Analyzing Manipulation Maps

The manipulation map incorporates quite a bit of information about the shape of the object and the friction distribution. The restriction of possible pushes to those normal to the object surface allows us to ignore finger-to-object friction, but it also restricts the number of possible manipulations. Certain objects, especially those similar to circles, can be very difficult to reorient due to this restriction.

As an example, we ran the system on a kidney shaped object; the resulting manipulation map is in Figure 8. Note how most push points produce only minor rotation (primarily due to uneven friction between the base of the object and the table) except the two lobes near the top. In practice, manipulation of the kidney object required a lot of lobe pushes to change orientation and accompanying pushes in other locations to move the center of the object back toward the goal center.

We also ran tests to directly compare the result of different friction distributions on the 'peanut' object which uses two circular feet as shown in Figure 1. To change the support friction distribution, we placed a 'sticky foot' made from

high-friction silicone rubber under each lobe in turn and built the manipulation maps shown in Figure 9. For clarity, weights of the learned mapping are omitted. The two figures clearly show different manipulation maps due to different friction distributions.

VII. CONCLUSIONS AND FUTURE WORK

The memory-based learning technique presented in this paper allows manipulation of an unknown object with an unknown support friction distribution. While the specific algorithm given is far from optimal, we feel it is a good first step toward more robust manipulation. Truly robust manipulation needs to interact with unknown objects and unknown haptic properties.

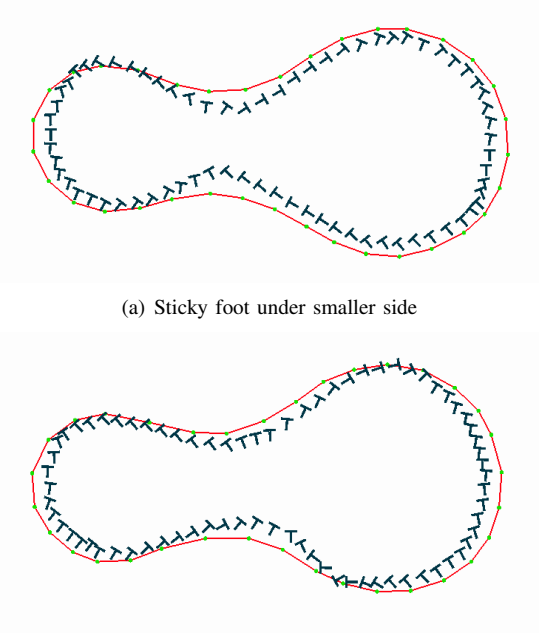
There are many ways to improve the existing algorithm. First, the granularity of the manipulation map could be improved by considering more points (possibly in a continuous representation). Also, allowing pushes that are not normal to the object surface would greatly increase the ability of the robot, but also increase the search space. Likewise, using vision or other sensing mechanisms would allow us to manipulate objects with sharp edges. Finally, deciding the next push using a multi-step look ahead would improve performance greatly.

A fundamental flaw with this approach is that it cannot generalize to other objects; once a manipulation map is learned, that same map cannot be applied to any other object, not even a larger version of the same shape.

In the broader problem, our next area of research will be other haptic properties outside of object-table friction using the PMET framework. In each case, we'll be focusing on robust techniques that do not require excessively sensitive sensors. Instead, our goal will be to use qualitative measurements from multiple sensors, as we accomplished with accelerometers, proximity, and proprioceptive sensors in this task. We believe that the next big leap ahead in robotics will be in software, not hardware.

VIII. ACKNOWLEDGMENTS

This work was supported by NSF Grant number 0535289. The authors would also like to thank Kevin Loewke and Paul Nangeroni for their hard work fabricating the PMET.



(a) Sticky foot under smaller side

(b) Sticky foot under larger side

Fig. 9. Peanut object manipulation maps for varying support friction distributions.

REFERENCES

[1] S. Jacobsen, J. Wood, D. Knutti, and K. Biggers, "The UTAH/M.I.T. dextrous hand: Work in progress," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 21 – 50, 1984.

[2] M. T. Mason and J. J. Kenneth Salisbury, *Robot hands and the mechanics of manipulation*. Cambridge, MA, USA: MIT Press, 1985.

[3] C. Lovchik, H. Aldridge, and M. Diftler, "Design of the nasa robonaut hand," in *ASME Dynamics and Control Division*, Nashville, Tennessee, November 1999.

[4] T. H. Massie and J. K. Salisbury, "The PHANTOM haptic interface: A device for probing virtual objects," in *ASME Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Chicago, IL, November 1994.

[5] S. Walker, K. Loewke, M. Fischer, C. Liu, and J. K. Salisbury, "An optical fiber proximity sensor for haptic exploration," in *IEEE International Conference on Robotics and Automation*, Roma, Italy, April 2007.

[6] S. Walker and J. K. Salisbury, "The Probabilistic Robotics Studio: Interactive robot programming," in *IEEE International Conference on Advanced Robotics*, Jeju, South Korea, August 2007.

[7] M. T. Mason, "Mechanics and planning of manipulator pushing operations," vol. 5, no. 3, pp. 53–71, 1986.

[8] M. T. Mason, A. D. Christiansen, and T. Mitchell, "Experiments in robot learning," in *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[9] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controlability, and planning," *International Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, December 1996.

[10] S. Akella and M. T. Mason, "Posing polygonal objects in the plane by pushing," vol. 17, no. 1, pp. 70–88, 1998.

[11] K. M. Lynch, "Estimating the friction parameters of pushed objects," in *Proc. 1993 IEEE/RSJ Int. Conf.*, 1993.

[12] T. Yoshikawa and M. Kurisu, "Identification of the center of friction from pushing an object by a mobile robot," in *Proc. of IEEE/RSJ International Workshop on Intelligent Robots and Systems*, 1991, pp. 449–454.

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

[14] C. Atkeson and S. Schaal, "Memory-based neural networks for robot learning," 1995. [Online]. Available: citeseer.ist.psu.edu/atkeson95memorybased.html

[15] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith, "Automatic Bayesian curve fitting," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 60, no. 2, pp. 333–350, 1998.

[16] R. Platt, A. H. Fagg, and R. A. Grupen, "Extending fingertip grasping to whole body grasping," in *IEEE International Conference on Robotics and Automation (ICRA'03)*, 2003, pp. 2677–2682.

[17] D. Wheeler, A. H. Fagg, and R. A. Grupen, "Learning prospective pick and place behavior," in *The 2nd International Conference on Development and Learning*, 2002.

[18] W. T. Townsend, "The BarrettHand grasper – programmably flexible part handling and assembly," *Industrial Robot*, vol. 27, no. 3, pp. 181–188, 2000.

[19] P. E. Gill and W. Murray, "Algorithms for the solution of the nonlinear least-squares problem," *SIAM Journal Numerical Analysis*, vol. 15, no. 5, pp. 977–992, 1978.