Learning Stable Pushing Locations

Tucker Hermans Fuxin Li James M. Rehg Aaron F. Bobick

Abstract— **We present a method by which a robot learns to predict effective push-locations as a function of object shape. The robot performs push experiments at many contact locations on multiple objects and records local and global shape features at each point of contact. The robot observes the outcome trajectories of the manipulations and computes a novel push-stability score for each trial. The robot then learns a regression function in order to predict push effectiveness as a function of object shape. This mapping allows the robot to select effective push locations for subsequent objects whether they are previously manipulated instances, new instances from previously encountered object classes, or entirely novel objects. In the totally novel object case, the local shape property coupled with the overall distribution of the object allows for the discovery of effective push locations. These results are demonstrated on a mobile manipulator robot pushing a variety of household objects on a tabletop surface.**

I. Introduction and Motivation

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 objects about which it has no or limited prior experience or knowledge. The only guidance a robot has for manipulating these objects is knowledge learned from prior experience with previously manipulated objects. The goal of this paper 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 such as support locations and friction distributions of the object. For an unknown object these properties cannot be deduced without interactive experimentation. Even if such a model is available, simulation may not be sufficient in solving the pushing control problem as efficient solutions require assumptions of uniform, stationary friction coefficients of both the object and the supporting surface [1–3].

In contrast to this complete physical description a robot can compute visual cues directly from camera input. Object shape encodes especially rich information about effective pushing locations. In this paper 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.

Tucker Hermans, Fuxin Li, James M. Rehg, and Aaron F. Bobick are affiliated with the Center for Robotics and Intelligent Machines and The School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA. {thermans, fli, rehg, afb}@cc.gatech.edu

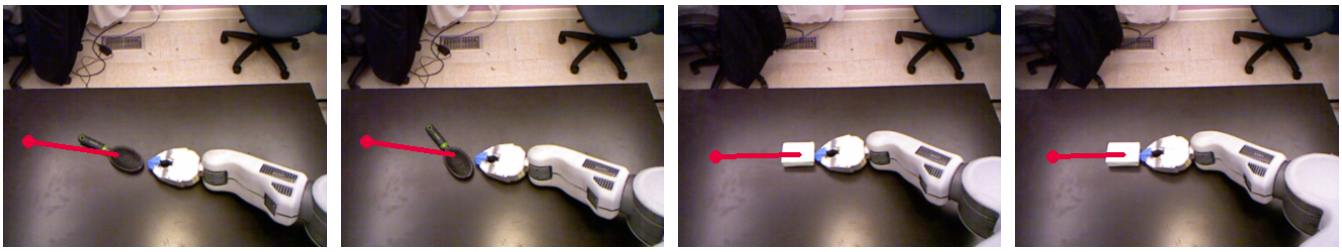
As an example, consider the scenario depicted in Figure 1a and 1b where the robot is pushing a hair brush. If it were to move its contact location a small amount to the left or right it causes a significant rotation of the object. Compare this to the example shown in Figures 1c and 1d. In this case a 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. The learning algorithm we develop allows a robot to predict from the shape of a novel object the best contact location on the boundary of the object for achieving a straight and stable push. Each time the robot pushes an object it records both a shape description relative to the push position and orientation as well as a "push score" measuring the quality of the push, essentially its ability to push the object along a straight path. As the robot interacts with more objects in more conditions, it uses non-linear regression to learn a prediction function. 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.

We organize the remainder of this paper as follows. Section II gives an overview of our approach to push learning and execution. We give details of the learning task, including the shape features and scoring function in Section III. We give details of the experimental procedure in Section IV and present the results of these experiments in Section V. Section VI discusses relevant related work in the robotics community. Finally we conclude and discuss future extensions of this work in Section VII.

II. Approach

We use a data-driven approach where the robot learns good pushing locations by interacting with objects during exploration. For a given object the robot chooses a location on the boundary and attempts to push the object in a straight line. While pushing the robot uses visual feedback control attempting to drive the object to the goal position. The robot performs a number of such push trials at various locations on a given object. The trajectory of the object during each push, obtained by the robot's visual tracking system used for feedback control, is recorded for each trial.

We derive a push-stability score from the observed trajectories to characterize each trial with the goal being to learn to predict this score from local and global shape features of the object. These features are extracted in the coordinate frame centered at the pushing location oriented in the direction of the goal location. When presented with



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

Fig. 1: Example pushing instances. The first two images are two consecutive frames captured while the robot pushes the large hair brush from an unstable pushing location. 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.

an object, whether new or previously encountered, the robot can then extract these shape features from locations around the object boundary, predict push-stability scores for each location, and push at a location that scores well.

III. Learning Task

Our learning task is to estimate a function $f : \mathbb{R}^m \rightarrow \mathbb{R}$, given n training example pairs (x_i, y_i), $i = 1, \ldots, n$, $x_i = [x_{i1}, \ldots, x_{im}] \in \mathbb{R}^m, y_i \in \mathbb{R}$. We can estimate this function using kernel support vector regression. The regression function has a form of:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i) + b \tag{1}$$

with $K(x, x_i)$ a positive semidefinite kernel comparing the similarity between the test example x and training examples x_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 overfitting 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 [4]:

$$\min_{f \in \mathcal{H}_K} \sum_i L_\epsilon(f(x_i), y_i) + \lambda \|f\|^2_{\mathcal{H}_K} \tag{2}$$

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 IV), and

$$L_\epsilon(f(x_i), y_i) = \begin{cases} 0, & |f(x_i) - y_i| \leq \epsilon \\ |f(x_i) - y_i| - \epsilon & |f(x_i) - y_i| > \epsilon \end{cases} \tag{3}$$

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 overfitting 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 used in this paper is the exponential χ^2 kernel [4]:

$$K(x_i, x_j) = \exp\left(-\gamma \sum_{k=1}^{d} \frac{(x_{ik} - x_{jk})^2}{x_{ik} + x_{jk}}\right) \tag{4}$$

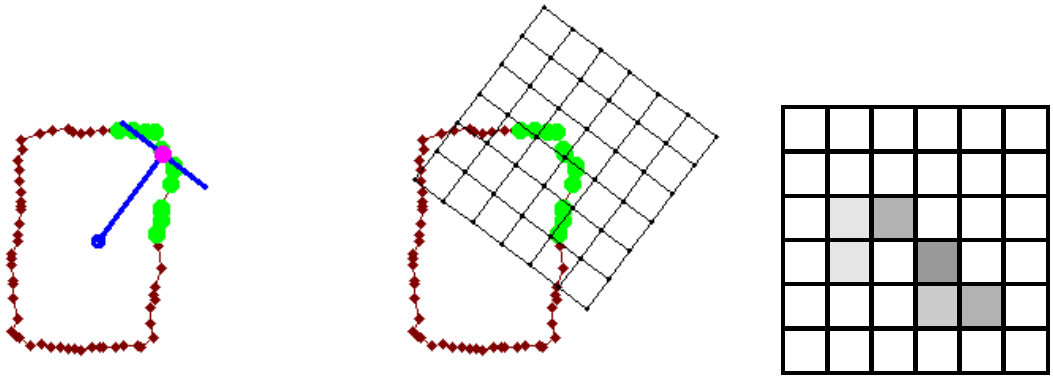
a proven excellent kernel for comparing histogram features that has been widely used in computer vision [5]. The parameter γ controls the width of the kernel, necessary when combining multiple kernels. This kernel corresponds to a symmetric version of the Pearson χ^2 test to determine whether a histogram 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).

A. Pushing Scoring Function

We wish to penalize pushes which deviate from the desired straight-line trajectory. As such our push-stability score is computed as the average distance of the observed object trajectory from the desired pushing trajectory. Equation 5 precisely defines this score:

$$y = \frac{1}{K} \sum_{k=1}^{K} \texttt{dist}(X[k], \ell_p) \tag{5}$$

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



(a) (b) (c)

Fig. 2: 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 2a 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.

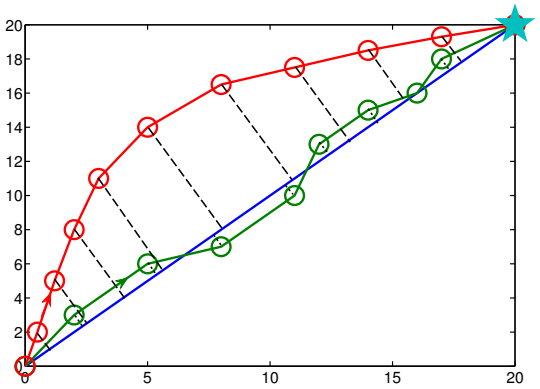


Fig. 3: Two synthetic push trajectories going from initial location in the bottom left to the goal location (star) in the top right. The red trajectory receives a push-stability score of 3.94cm while the straighter green trajectory receives a score 0.62cm.

Figure 3 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 $y_0 = 0.62$cm for the green trajectory and $y_1 = 3.94$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. Finally, this score naturally extends to arbitrary pushing trajectories. Given a suitable controller, this could allow a robot to learn if certain locations are good for pushing along other desired trajectories, such as causing an object to rotate to a desired orientation.

B. Shape Features

Our feature extraction takes as input the point cloud associated with the segmented object. We compute both local and global descriptions of shape encoded in the coordinate frame of the chosen pushing location. Given a 3D point cloud of the given object we project all points to the 2D plane defined by the supporting surface and extract the boundary of this projected point cloud. One point on the boundary corresponds to the evaluated pushing location. This point defines the center of the local coordinate system. The positive x-direction is oriented by the pushing direction, defined as the vector from the pushing point to the object centroid.

To build the local feature descriptor we walk along the boundary to the left and right sides, selecting all points within 0.025m in the local y-direction. We visualize the point selection in Figure 2. 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 2c.

We encode global object shape using a slightly modified version of the popular shape context feature [6]. We again use the boundary of the object as extracted above, however all points are kept as input to our shape context feature, not simply those in the local area of the pushing location. 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.

In order to align the features with the pushing direction we rotate the histogram so that the first angle bin is oriented towards the 2D center of the original point cloud. 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.

IV. Experimental Procedure

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. Pushing is performed using a feedback controller 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 4.

The robot achieves this behavior by using a feedback control law that includes a velocity term to move toward the goal and a second term that moves the end effector towards the vector defined from the goal location through the object's centroid:

$$u_{\dot{x}} = k_{gc} e_{goal_x} + k_c e_{centroid_x} \quad (6)$$

$$u_{\dot{y}} = k_{gc} e_{goal_y} + k_c e_{centroid_y} \quad (7)$$

where e_{goal_x} and e_{goal_y} define the goal error for goal locations (x^*, y^*):

$$e_{goal_x} = (x^* - \hat{x}),\ e_{goal_y} = (y^* - \hat{y}) \quad (8)$$

where $(\hat{x}, \hat{y})$ is the visually estimated object centroid at the current time step. All coordinates are defined in the table frame. The second term in Equations 6 and 7 provides the additional velocity term toward the goal-centroid line; $e_{centroid_x}$ and $e_{centroid_y}$ are components of the perpendicular vector from the presumed end effector contact point to the goal-centroid line. The robot then pushes in the direction of the goal attempting to maintain this collinearity relation. In all experiments $k_{gc} = 0.1$ and $k_c = 0.2$.

The hand configuration used can be seen in Figure 1. 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 our previous work [7, 8]. In the terminology of that work the controller used here is the centroid alignment controller and the perceptual proxy is the ellipse proxy. For more details on the pushing behaviors and object tracking used in our experiments consult these previous works.

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 5.

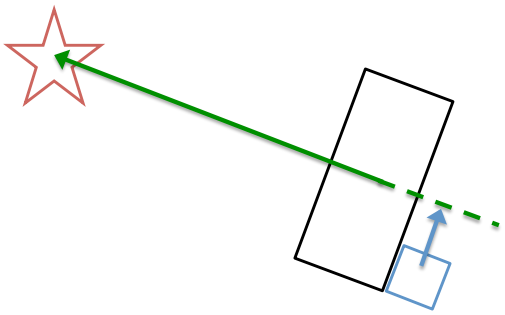


Fig. 4: Visualization of the feedback control policy. The green arrow depicts the goal oriented term of the controller from the object centroid to the goal (red star). The blue arrow show the term used to align the end effector (blue) with the center of the object. These two vectors are scaled and combined to create the commanded velocity of the end effector.

Fig. 5: The six objects used in the experiments.

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. 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 Eq. 2)

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.

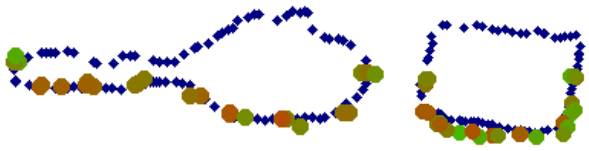
In order to improve the regression performance, we take the logarithm of Equation 5 as the regression target. Transforms like this are common in statistics in order to make the target distribution more balanced and better correspond to model assumptions. In this work, good pushing locations often have 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.

Finally, to learn the prediction function, we compared three different learning methods. The first — Kernel SVR — is the regression method discussed above. In addition, we implemented 2 other popular regression algorithms: linear ridge regression and boosting stumps. For boosting stumps, we used the L2 boosting framework [9] on regression stumps computed by the CART algorithm. The competing algorithms have been tuned to their respective best parameters. For completeness we present as baseline just using the training mean, also known as the 0th-Order regressor, to show the ability of the learning functions to improve upon average output.

V. Results

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 I presents these cross validation results both 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 two objects in Figure 6. 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 score. Each column corresponds to the sequestered object of the leave one out methodology, while the rows correspond to the different



(a) (b)

Fig. 6: Visualization of ground truth pushing scores for the large brush and soap box. Green points represent better scores, while redder represent worse.

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 is better on Camcorder and Large Brush, but fails to capture the details in Food Box and Small Brush, 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 I, under such a planning error metric our approach performed well on all the objects, being able to predict a pushing location 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

TABLE I: Performance of the 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

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.

VI. Related Work

Pushing

Effective pushing behaviors offer a number of benefits in robotics domains which complement standard pick-and-place operations. For example pushing can be used to move objects too large for the robot to grasp, to more quickly move objects to new locations, or to move an object while another object is already grasped. As such there has been considerable work at developing such capability. Early work that analyzed a complete model of the dynamics of pushing was developed by Mason who describes the qualitative rotational changes of sliding rigid objects being pushed by either a single point or single line contact [1]; representative examples of some more recent applications of pushing are available in [3, 8, 10–14].

Notably, Ruiz-Ugalde et al. execute a pushing behavior by determining the static and kinetic friction coefficients, both between the robot hand and object and between the object and table, for multiple objects with rectangular footprints [3]. Additionally they present a robust feedback controller using a cart model for the object being pushed. Hermans et al. present a method by which a robot can learn to select an appropriate pushing behavior as a function of object and workspace location [8]. Behaviors are represented as a combination of a perceptual proxy, a feedback controller, and a behavior primitive.

To address the inherent difficulty in estimating model parameters, there are data-driven methods that use an empirically derived characterization of the outcomes of specific actions applied to the object. For example, Narasimhan uses vision to determine the pose of polygonal objects of known shape in the plane [15] and then develops a variety of methods to push objects into the desired location and orientation including a data driven approach that learns the effect of different pushes on the object. Similarly, 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 [16]. Slip of the object is avoided by pushing at a notch in the object. Scholz and Stilman learn object specific dynamics models for a set of object through experience [17]. Each object is pushed at a number of predefined points on the perimeter and the robot learns Gaussian models of displacement of the object's 2D pose, (x, y, θ), at each location. These learned models are then used to select the input push location given a desired object pose.

Learning with Shape

Shape features have been used in a number of works on learning grasp locations on objects [18–21]. Bohg and Kragic use shape context as a feature for learning grasping locations [18]. Le et al. desire to learn grasping locations for each finger of a multi-fingered robot hand [19]. 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. Rao et al. use similar variance features on depth and height values of the object of interest as well as absolute range in height [20].

These features are combined with visual features to learn to classify good grasping locations. Jiang et al. use local histograms of depth values to rank grasping locations for a parallel jaw gripper [21]. In a different piece of work Jiang et al. use features similar to the variance features discussed above to encode shape information of both an object being placed and potential placement locations in learning to place objects [22]. Histograms of point locations are also used to encode information both about the object being placed and the placement location. Additional features are used to directly encode the interaction between the object and potential placement locations.

VII. Conclusions and Future Work

We have presented an interactive learning approach by which a robot autonomously performs experiments and learns object locations that afford stable pushing. We have shown that the robot can effectively learn to predict pushing locations on novel objects, and the results significantly outperform randomly sampling pushing locations, as well as a pre-programmed, shape-based heuristic. We are interested in extending this learning technique to learn also how to push objects to a desired orientation. We can couple this orienting behavior with the straight line pushing learned in this work to push an object to any desired pose. By first orienting such that a stable pushing location is inline with the desired goal point and then pushing in a straight line to the goal the robot can effectively push an object to a location in its workspace. If a specific final orientation is also desired, then an additional orienting push can be performed once the object has reached the final location.

Beyond this simple extension, we are interested in examining the benefits in using a combination of pre-programmed knowledge (such as the major-minor axis method in this paper), general, object independent knowledge (the learned pushing functions across objects), and object specific knowledge (such as the behavior selection done in our previous work [8]).

We envision a system which prior to any interaction could use heuristics to first manipulate the environment. This could help to bias the exploration to find good locations to push more rapidly. Once enough data is collected, the approach of this paper could be used to generalize to interaction with novel objects. However, if the robot is expected to interact with the novel object more than once, learning a new push location selection function, specialized to the object would be desirable. If it is the case that the general regressor works well, no such specialization should be required, but if the predictor is systematically wrong in someway, the robot should learn to specialize its actions to this object, without hindering the performance on other objects on which it already operates successfully.

VIII. Acknowledgments

This work was supported in part by NSF Award 0916687. We would like to thank the anonymous reviewers for their suggestion of comparing to the heuristic major and minor axis location selection scheme.

References

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

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

[3] F. Ruiz-Ugalde, G. Cheng, and M. Beetz, "Fast Adaptation for Effect-aware Pushing," in *Humanoids*, 2011.

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

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

[6] S. Belongie, J. Malik, and J. Puzicha, "Shape Matching and Object Recognition Using Shape Contexts," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 24, April 2002.

[7] T. Hermans, J. M. Rehg, and A. F. Bobick, "Decoupling Behavior, Control, and Perception in Affordance-Based Manipulation," in *(IROS) Workshop on Cognitive Assistive Systems*, October 2012.

[8] ——, "Decoupling Behavior, Perception, and Control for Autonomous Learning of Affordances," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

[9] P. Buhlmann and B. Yu, "Boosting with the l2 loss: Regression and classification," *Journal of American Statistics Association*, vol. 98, pp. 324–340, 2003.

[10] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous Acquisition of Pushing Actions to Support Object Grasping with a Humanoid Robot," in *Humanoids*, Paris, France, 2009.

[11] D. Katz, Y. Pyuro, and O. Brock, "Learning to Manipulate Articulated Objects in Unstructured Environments Using a Grounded Relational Representation," in *RSS*, Zurich, Switzerland, June 2008, pp. 254–261.

[12] M. Dogar and S. Srinivasa, "A Framework for Push-Grasping in Clutter," in *RSS*, 2011.

[13] ——, "Push-Grasping with Dexterous Hands: Mechanics and a Method," in *IROS*, 2010.

[14] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push Planning for Object Placement on Cluttered Table Surfaces," in *IROS*, 2011.

[15] S. Narasimhan, "Task Level Strategies for Robots," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

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

[17] J. Scholz and M. Stilman, "Combining Motion Planning and Optimization for Flexible Robot Manipulation," in *Humanoids*, 2010.

[18] J. Bohg and D. Kragic, "Learning Grasping Points with Shape Context," *Journal of Robotics and Autonomous Systems*, vol. 58, no. 4, pp. 362–377, April 2010.

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

[20] D. Rao, Q. V. Le, T. Phoka, M. Quigley, A. Sudsang, and A. Y. Ng, "Grasping Novel Objects with Depth Segmentation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2010.

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

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