# Learning Contact Locations for Pushing and Orienting Unknown Objects

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

*Abstract*— **We present a method by which a robot learns to predict effective contact locations for pushing 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. Each trial attempts to either push the object in a straight line or to rotate the object to a new orientation. The robot observes the outcome trajectories of the manipulations and computes either a push-stability or rotate-push score for each trial. The robot then learns a regression function for each score in order to predict push effectiveness as a function of object shape. With this mapping, the robot can infer effective push locations for subsequent objects from their shapes, regardless of whether they belong to a previously encountered object class. 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 novel objects it has never manipulated before. 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 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 [1–3].

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 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. As an example, consider the scenario depicted in Figure 1a and 1b where the robot is pushing a hair brush. If the contact 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 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. 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 center as well as a "push score" measuring the quality of the push. The push score encodes the robot's ability either to 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 objects and 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 paper represents an extension of our previous work, which presented limited, offline results for the case of straight-line pushing [4].

The remainder of the paper continues as follows. Section II discusses relevant work in pushing and learning from object shape in the robotics community. Section III 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 IV and present the results of all experiments in Section V. Finally, we conclude and

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

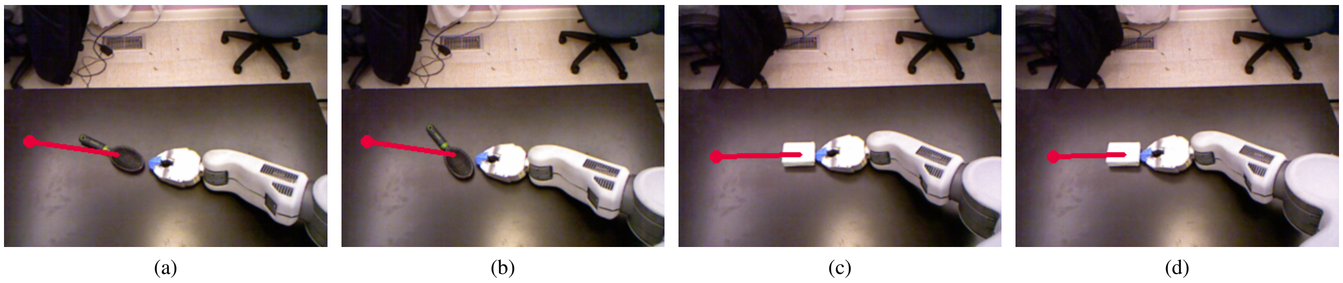


Fig. 1: Example pushing instances. The first two images are two consecutive frames captured while the robot pushes the large hair brush from an unstable straight-line pushing location, which induces a rotation of the object. The second two frames show the robot pushing a soap box from a stable pushing location. In both examples the red line shows the vector from the estimated object center to the goal location denoted as the red circle.

discuss this work in the broader context of our research program in Section VI.

## II. Related Work

Effective pushing behaviors offer a number of benefits for robot manipulation which complement standard pick-and-place operations. Early work that analyzed a complete model of the dynamics of pushing was developed by Mason who described 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, 5–9].

Ruiz-Ugalde et al. execute a pushing behavior by determining the static and kinetic friction coefficients, between the robot hand and object, as well as, 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. In our previous work, we presented a method by which a robot can learn to select an appropriate pushing behavior as a function of object and workspace location [9]. Behaviors are represented as a combination of a perceptual proxy, a feedback controller, and a behavior primitive.

A number of approaches use data-driven methods to empirically derive characterizations of the outcomes of pushing actions applied to objects. For example, Narasimhan uses vision to determine the pose of polygonal objects of known shape in the plane [10] 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 [11]. 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 objects through experience [12]. 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 at each location. These learned models are then used to select the input push location given a desired object pose. Omrčen et al. use a retinal image based neural network to learn the pushing dynamics for flat objects [5]. The learned models are specific to individual objects and generalization to similar objects is not examined. Kopicki et al. also learn probabilistic models of outcome based on pushing manipulations [13]. 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 [14]. 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 [15]. 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.

Finally, shape features have been used in a number of works on learning grasp locations on objects [16–18]. Bohg and Kragic use shape context as a feature for learning grasping locations [16]. Le et al. desire to learn grasping locations for each finger of a multi-fingered robot hand [17]. 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 [18]. 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 [19].

## III. Learning Task

We formulate our learning task as the estimation of a function $f : \mathbb{R}^m \to \mathbb{R}$, 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 $f$.

## A. 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 1 precisely defines this notion:

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

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

Figure 2 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.

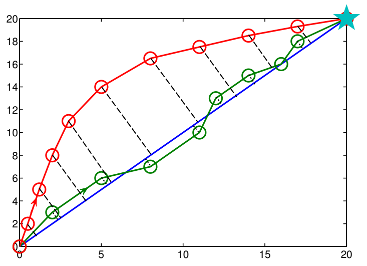


Fig. 2: Two synthetic push trajectories going from initial location in the bottom left to the goal location (star) in the top right. The red trajectory receives a push-stability score of 3.94cm while the straighter green trajectory receives a score 0.62cm.

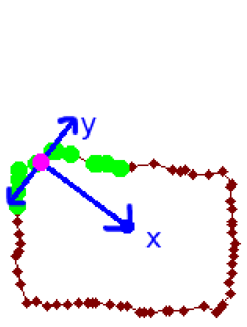
For the task of learning where to push in order to rotate an object to a desired heading we used the simple score of net change in orientation between the initial pose and final pose. Ideal orienting pushes should not only rotate the object, but also produce as little translation of the object position 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 IV-B). 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.

## B. Shape Features

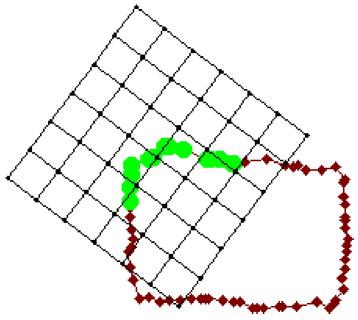
Our feature extraction takes as input the point cloud associated with the segmented object of interest. We compute both local and global descriptions of shape encoded in a coordinate frame defined by the object center and chosen pushing contact location. We encode local object shape with a small 2D histogram, while we use a slightly modified version of the popular shape context feature for global object shape [20]. 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 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 3. 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 $6 \times 6$ 2-dimensional histogram with uniform bin sizes. An example histogram is shown in Figure 3c.

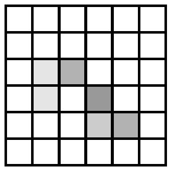
We again use the boundary of the object as extracted above, however all points are kept as input to our shape context feature, as opposed to only the points near the pushing location used for the local descriptor. For the given set of boundary points we follow the standard shape context extraction method: 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.



(a)



(b)



(c)

Fig. 3: 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 3a 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.

### C. 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:

$$f(\mathbf{z}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{z}, \mathbf{z}_i) + b \tag{2}$$

where $K(\mathbf{z}, \mathbf{z}_i)$ is a positive semidefinite 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 overfitting to a particular example and achieving provably better performance than nearest neighbor approaches.

The parameters $\alpha$ 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 [21]:

$$\min_{f \in \mathcal{H}_K} \sum_i L_\epsilon(f(\mathbf{z}_i), s_i) + \lambda \|f\|^2_{\mathcal{H}_K} \tag{3}$$

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 $f$, $\lambda$ is a regularization parameter on this smoothness norm (denoted $C$ in the dual quadratic programming formulation and in Section IV), and

$$L_\epsilon(f(\mathbf{z}_i), s_i) = \begin{cases} 0, & |f(\mathbf{z}_i) - s_i| \le \epsilon \\ |f(\mathbf{z}_i) - s_i| - \epsilon & |f(\mathbf{z}_i) - s_i| > \epsilon \end{cases} \tag{4}$$

is called the $\epsilon$-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 $L_\epsilon$ 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 $\epsilon$-insensitive loss outperformed traditional $L_1$ and $L_2$ loss functions.

The kernel we used in this paper is the exponential $\chi^2$ kernel [21]:

$$K(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\gamma \sum_{k=1}^{d} \frac{(z_{ik} - z_{jk})^2}{z_{ik} + z_{jk}}\right) \tag{5}$$

a proven excellent kernel for comparing histogram features that has been widely used in computer vision [22]. The parameter $\gamma$ controls the width of the kernel, necessary when combining multiple kernels. This kernel corresponds to a symmetric version of the Pearson $\chi^2$ 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 IV-C.

## IV. 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 more details on the object tracking used in our experiments consult our previous work, specifically the ellipse perceptual proxy [9].

### A. Straight Line Pushing

Straight line pushing is performed using a feedback controller which attempts to align the tip of the robot gripper

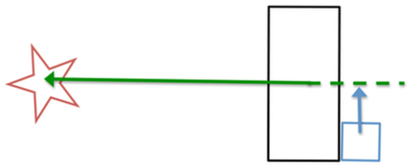


Fig. 4: Visualization of the feedback control policy. The green arrow depicts the goal oriented term of the controller from the object centroid to the goal (red star). The blue arrow show the term used to align the end effector (blue) with the center of the object. These two vectors are scaled and combined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

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:

$$u_x[k+1] = \sigma_{gc} e_{gx} + \sigma_c e_{centroid_x} \tag{6}$$

$$u_y[k+1] = \sigma_{gc} e_{gy} + \sigma_c e_{centroid_y} \tag{7}$$

where $u_x[k+1]$ and $u_y[k+1]$ are the desired end effector velocities in the table frame at time $k+1$ and $e_{gx}$ and $e_{gy}$ define the goal error for goal locations $(x^*, y^*)$ at time $k$:

$$e_{gx} = (x^* - x[k]),\ e_{gy} = (y^* - y[k]) \tag{8}$$

$(x[k], y[k])$ is the visually estimated object centroid at the current time step $k$. 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 $\sigma_{gc} = 0.1$ and $\sigma_c = 0.2$.

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 [9]. In the terminology of that work, the controller used here is the centroid alignment controller.

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.

### B. 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:

$$u_x[k+1] = \sigma_g |\theta^* - \theta[k]| \tag{9}$$

$$u_\omega[k+1] = -\sigma_s \cdot \dot{\theta}[k] \tag{10}$$

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. The robot hand was oriented

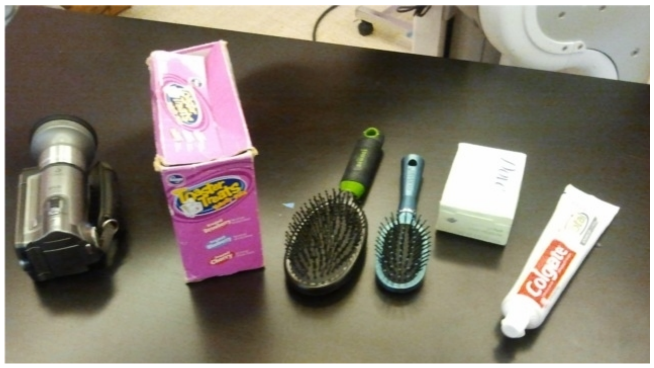


Fig. 5: The six objects used in the experiments.

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 [9]. 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 6. 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.

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

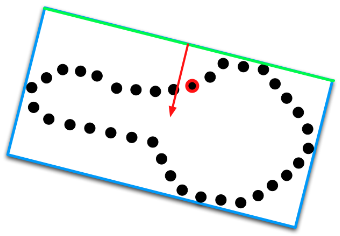


Fig. 6: Visualization of determining the initial pushing direction for rotate pushes. For the chosen push location, highlighted in red, the green (top) edge of the bounding box is closest. Thus the initial pushing direction designated by the red arrow is chosen.

brush, and 51 trials for each of the remaining four objects giving a total of 341 samples for learning.

## C. 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 $\epsilon$ in SVR is fixed to 0.3, and $C$ (the dual variable to $\lambda$ in Eq. 3) is fixed to 2 in all experiments. The kernel widths $\gamma$ for the local kernel is fixed to 2.5, while the global kernel has a $\gamma$ 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 1 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 $\gamma$ value of 0.05 and the $\gamma$ 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 $\epsilon$ 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.

## V. Results

To measure the effectiveness of the learning, we use the learned regression functions to perform push tasks on novel objects. A leave-one-out setting is 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

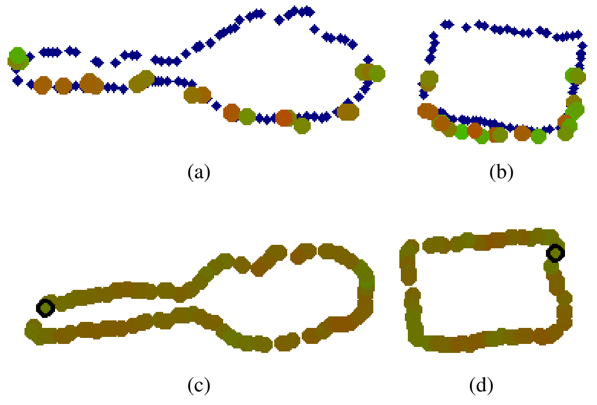


Fig. 7: Visualization of ground truth and predicted straight-line pushing scores for the large brush and soap box. The top row shows ground truth samples, while the bottom row shows predicted scores at all points. Green points represent better scores, while redder represent worse. The black circles on the predicted scores show the points with best predicted score.

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 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. Detailed explanation of these results follows.

## A. 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 7a and 7b. 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 7c and 7d of predicted push-stability scores taken from test trials of

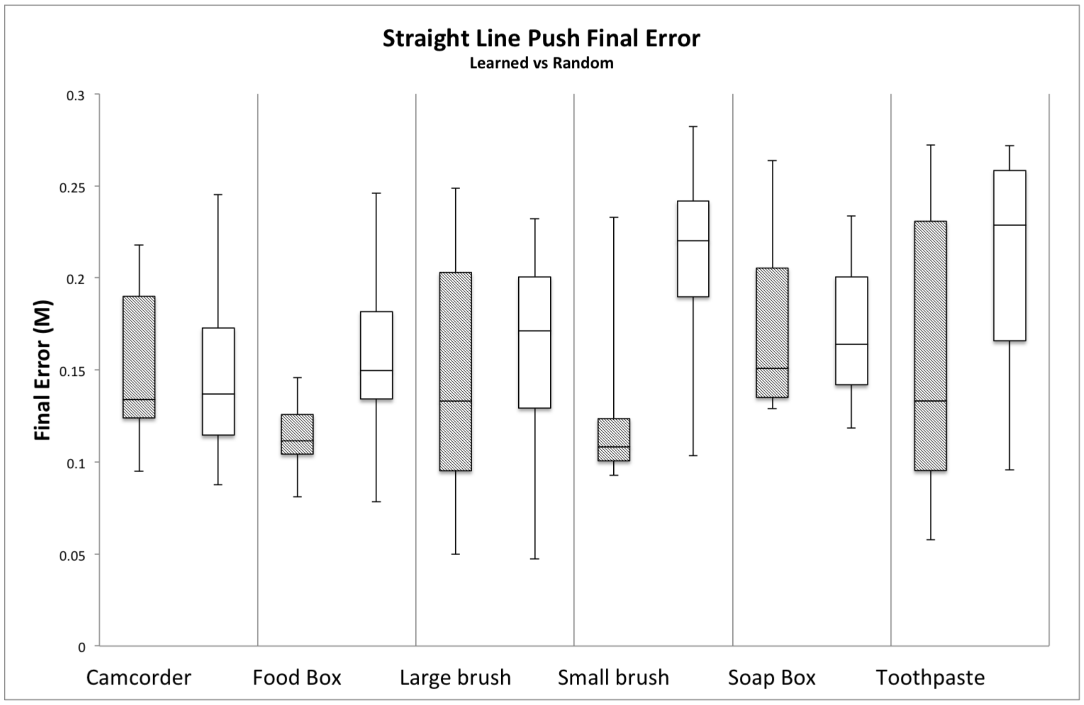


Fig. 8: Box and whisker plots of the final position error for learned and random initial contact location prediction for straight-line pushing. The shaded boxes represent the pushing locations chosen through the learned regressor, while the unfilled boxes correspond to random initial locations. The vertical 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.

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.

Figure 8 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 7b. We note that many of the large errors come from pushes that result in 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.

## B. *Orienting Push Locations*

We present quantitative results for all rotate pushing test experiments in Figure 9. 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.

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

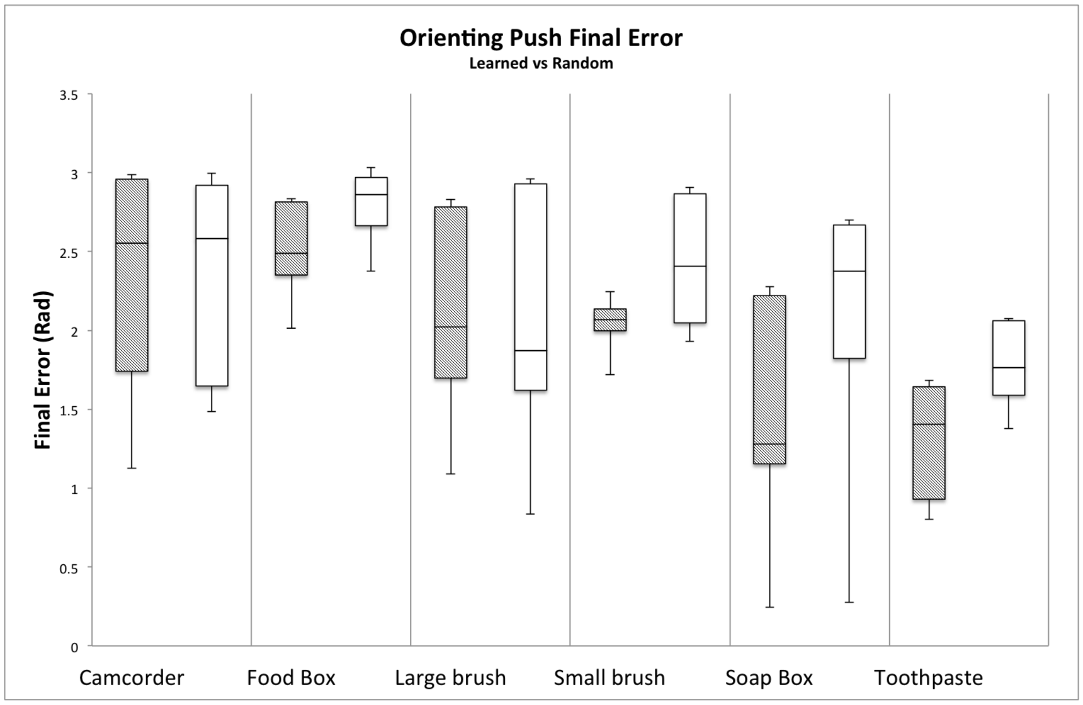


Fig. 9: Box and whisker plots of the final heading error for learned and random initial orienting push location prediction for straight-line pushing. The shaded boxes represent the pushing locations chosen through the learned regressor, while the unfilled boxes correspond to random initial locations. The vertical axis represents final heading error from 0 radians to $\pi$ radians. The horizontal labels are the object categories. Details of how to read the box plot can be found in the caption to Figure 8.

information is not being encoded.

## VI. Conclusions and Future Work

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. In the future we would like to examine whether using three dimensional shape features could improve performance beyond current levels. Additionally, we are interested in extending this work to learning dynamics models of objects from shape, so that more complicated push trajectories can be planned and performed.

## VII. ACKNOWLEDGMENTS

This work was supported in part by NSF Awards 0916687 and 1016772.

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

[5] 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.

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

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

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

[9] T. Hermans, J. M. Rehg, and A. F. Bobick, "Decoupling Behavior, Perception, and Control for Autonomous Learning of Affordances," in *ICRA*, May 2013.

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

[11] 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.

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

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

[14] B. Ridge, D. Skočaj, and A. Leonardis, "Self-Supervised Cross-Modal Online Learning of Basic Object Affordances for Developmental Robotic Systems," in *ICRA*, Anchorage, USA, May 2010.

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

[16] 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.

[17] 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.

[18] Y. Jiang, S. Moseson, and A. Saxena, "Efficient Grasping from RGBD Images: Learning using a new Rectangle Representation," in *ICRA*, 2011.

[19] Y. Jiang, C. Zheng, M. Lim, and A. Saxena, "Learning to Place New Objects," in *ICRA*, 2012.

[20] 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.

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

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