Automatic Learning of Pushing Strategy for Delivery of Irregular-Shaped Objects

Manfred Lau, Jun Mitani, Takeo Igarashi

Abstract— **Object delivery by pushing objects with mobile robots on a flat surface has been successfully demonstrated. However, existing methods can push objects that have a circular or rectangular shape. In this paper, we introduce a learning-based approach for pushing objects of any irregular shape to user-specified goal locations. We first automatically collect a set of data on how an irregular-shaped object moves given the robot's relative position and pushing direction. We collect this data with a randomized approach, and we demonstrate that this approach can successfully collect useful data. Object delivery is achieved by using the collected data with a non-parametric regression method. We demonstrate our approach with a number of irregular-shaped objects.**

I. INTRODUCTION

The use of mobile robots has increased greatly over the past decade as they can be found in practical use in factories, hospitals, and people's homes. This paper focuses on the use of mobile robots for pushing objects. Pushing is useful for object delivery, bulldozing/construction operations, cleaning tasks, and robot soccer. However, many existing approaches [1], [2], [3], [4], [5] push objects with regular shapes such as circles, squares, and rectangles.

Pushing an irregular-shaped object is challenging as it is difficult to accurately measure or specify the shape and physical parameters of the object by hand. Applying automatic methods for recognizing the parameters can be noisy and inaccurate. In addition, it is difficult to geometrically compute how such an object moves as it gets pushed, and to incorporate its motion into an algorithm for pushing it towards a goal location.

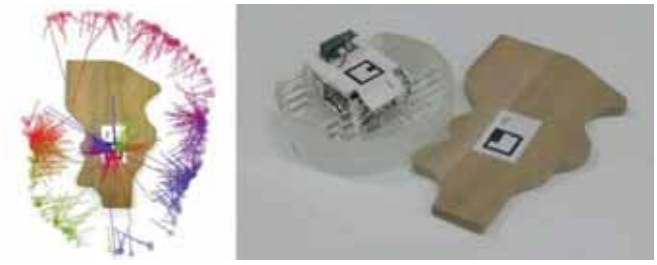

Fig. 1. We automatically collect a pushing strategy (left) for this irregular-shaped object, and use the data to control the robot to push the object towards user-specified goals.

This paper introduces a learning-based approach for pushing objects of any irregular shape (Fig. 1). Instead of explicitly measuring or specifying the object's shape and parameters, the idea is to first collect data on how the object moves based on the robot's relative position and pushing direction. We call such a set of data a pushing strategy. The pushing strategy is collected automatically. The user does not need to manually specify or provide it. We then use this pushing strategy to push the object towards a user-specified goal. The strength of our approach is that we can use the same algorithm for objects of different shapes and sizes. The object's shape is never explicitly specified or recorded, but is implicitly represented in the data. In addition, the data implicitly stores relevant physical parameters such as the weight distribution and the friction between the robot, object, and pushing surface. The learning-based method is also robust to slight inaccuracies in the data.

There are two main problems that we solve: how to collect the pushing strategy, and how to use the pushing strategy to deliver an object to a goal. For the first problem, we present an initial data collection method where we collect data by having the robot push the object from different positions and directions. We use a simple randomized approach to decide the robot's pushing position and direction. We can then experimentally evaluate the existing data by using them to solve random goal queries. This is an optional step. The motivation is that if the existing data can successfully solve random queries, then we can stop collecting data. For the second problem, we use a non-parametric regression technique to decide where to move the robot and which direction it should push in, given the final goal and the robot/object relative locations. We do not model the data parametrically, as it allows us to keep collecting data as the robot pushing is executed. Although we apply an existing non-parametric technique in our algorithm, our contributions are: (i) we solve the object-delivery-by-pushing problem for *irregular-shaped objects*, and (ii) we use the idea of *automatically learning a pushing strategy* to solve this problem.

We demonstrate that our algorithms are shape independent: they work with irregular-shaped objects of various shapes and weight distributions. Since our focus is on the pushing algorithm, we do not have obstacles in most of our test environments. For environments with obstacles, we first use a global planner to generate a high-level path and then use our method to push the object towards sub-goals along this path. We empirically compare our approach with an existing method for pushing circular objects [5].

II. RELATED WORK

The idea of using the memory of past pushes to predict future pushes has been previously studied [6], [7]. However,

All authors are with JST ERATO Igarashi Design Interface Project, Tokyo, Japan (contact: manfred.lau@gmail.com). Jun Mitani is also with University of Tsukuba. Takeo Igarashi is also with The University of Tokyo.

their pushing models are much simpler and they do not show results of real mobile robots pushing different types of irregular-shaped objects. Salganicoff et al. [6] uses a very simple push model where the point of contact between the robot and object is a single notched point on the object. There is only one rotational degree of freedom at the contact point. Walker et al. [7] also builds a mapping between pushes and object motion. However, they explicitly measure the object's shape by using a proximity sensor on a robot finger to detect a point cloud of the object, and then fit a shape to these points. Our method avoids potential measurement inaccuracies by not explicitly finding the object's shape. Furthermore, their objects are restricted to those with low curvature and without corners due to sensor issues. Their pushes are limited to be only at specific discretized points on the object, along the object's surface normal, and at single contact points with the object. In contrast, our objects can be of any shape. Our pushes can be at any point around the object, at any direction, and be in contact with the object at multiple points.

There exists much work in pushing objects for various tasks, but these methods handle objects of a circular/spherical or rectangular shape. Mobile robots for playing robot soccer can push a spherical ball or a rectangular-shaped box [8]. A watcher robot can lead a team of pusher robots to push a rectangular-shaped box [1]. Push plans for circular objects are computed that allow the object to touch and move along the wall/obstacles [2], [3], [4], or allow for multiple pushes of the object [4]. A method that computes a dipole field [5] can push circular objects for object delivery tasks. Our method differs in that we can handle objects of any irregular shape.

The mechanics of pushing objects on a surface has been studied [9]. This work uses knowledge about the mechanical properties of objects to generate stable pushing plans. On the other hand, our method is empirical and based on observing actual robot pushes of objects. Lynch et al. [10] perform experimental pushes of objects and observe the resulting motion to estimate the friction parameters. While we also perform empirical pushes of objects, we neither model the friction explicitly nor recognize the object geometry directly.

Our approach is related to work in the learning community. Reinforcement learning and vision methods have been used to allow mobile robots to learn to shoot a ball into a goal [11], and to learn behaviors such as obstacle avoidance and target pursuit [12]. Our work applies a different learning technique for the robot to learn a different task. Learning from demonstration methods [13], [14] allow the human to demonstrate a task to a robot, and a policy to perform the task is learned from the human data. Our method of using the example data to control the robot pushing is similar, although we collect the data automatically and there is no human demonstration of the task.

III. ALGORITHM

A. Problem Definition

There are two problems that we solve to achieve our pushing task. We assume that our system can continuously track the global position/orientation of the robot, object, and goal. We also assume that the object does not roll on the flat pushing surface. The first problem is to collect data on the object's movement based on the robot's position and push direction. We collect and store this pushing strategy beforehand so that it can be used for future goal queries. The inputs are the boundary of the workspace where the robot and object can move in, the radius of the circular robot, and the radius of the bounding circle of the object. We bound the object by a circle so that the robot can move around it regardless of its orientation. The output is a pushing strategy: a set of data where each sample is for one robot push and the corresponding object movement. We describe an initial data collection approach (Section III-C) where we start from no data, and we collect samples by having the robot push from a variety of positions and directions. We then present an experimental evaluation method (Section III-D) that can optionally be used to test if we should collect more data. This may be the case if the existing data cannot be used to successfully solve goal queries. We can also collect supplemental data with this method and immediately use them as the robot is solving goal queries.

The second problem is to use the pushing strategy for controlling the robot to deliver the object to any goal position (ie. we do not use goal orientation). We present a non-parametric regression algorithm (Section III-E) that takes as input the pushing strategy, the position/orientation of the robot, object, and goal, and returns as output the robot instantaneous push direction and in some cases the position that the robot should push from.

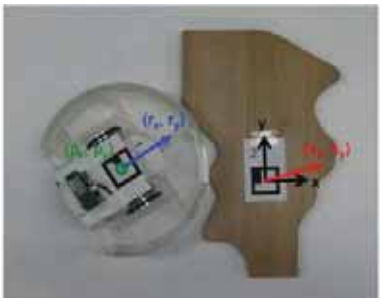

Fig. 2. We collect one data sample for each robot push. (p_x, p_y) is the position of the robot's center. (r_x, r_y) is the change in position of the robot's center. (s_x, s_y) is the change in position of the object. All values are in the object's coordinate frame (black axes).

B. Notation

Let the robot be R, the object be O, and the pushing strategy (or set of data) be D. Each sample of D contains the data for one robot push and the corresponding object movement: $(p_x, p_y, r_x, r_y, s_x, s_y, \theta)$ (Fig. 2). θ represents the change in orientation of the object and is not drawn in the figure. We only allow forward robot pushes in our current implementation, although the approach accepts general pushes such as moving forward and turning at the same time. The object's trajectory is not necessarily a straight line, although we only record its change in position and orientation. For

Algorithm 1: Data Collection

```
   Initial Data Collection:
1  for k = 1 to K do
2      if dist(prev_O_pos, O.pos()) < ε then
3          reset_robot();
4      if obj_outside_boundary() then
5          reset_robot_obj_boundary();
6      prev_O_pos = O.pos();
7      R.spin(random_angle());
8      R.forward(random_dist());
9      D.save_sample();

   Experimental Evaluation of Existing Data:
10 reset_robot();
11 while true do
12     if object_not_move() or over_time_limit() then
13         reset_robot();
14     if obj_outside_boundary() then
15         reset_robot_obj_boundary();
16     solve_query(random_goal());
17     if good_success_rate() then
18         break;
```

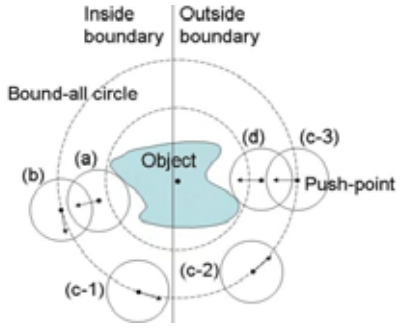

Fig. 3. The operations in *reset_robot_obj_boundary*() in Algorithm 1. Let the bound-all circle's center be the object's center, and radius be the sum of the robot's radius, the object's bounding circle's radius and a small amount of extra space. (a to b) The robot pushed the object outside the boundary, and we spin and move the robot to the closest point on the bound-all circle. Let the push-point be the point on the bound-all circle such that the vector from that point to the object's center is perpendicular to the boundary. (c-1,2,3) We keep spinning and moving the robot along points on the bound-all circle until it reaches the push-point. (d) We then spin and move the robot towards the object until it pushes the object inside the boundary (not shown).

the purpose of collecting data samples, the robot can be positioned anywhere as long as it is in contact with the object. The robot can be in contact with the object at multiple points. The robot's push direction can be in any direction such that it moves the object by at least a small distance.

C. Initial Data Collection

The objective of the initial data collection process is to start with no data, and collect a variety of robot pushes and object movement as quickly as possible. Although we can use more sophisticated methods to decide where and how the robot should push the object, we choose to use a randomized strategy that is simple but effective. The idea is to place the robot and object on a flat workspace (ie. a table) with a pre-defined boundary, and automatically run the data collection process without human intervention. The basic strategy is to choose a random direction and distance (within certain limits) for the robot to push with from its current position.

Algorithm 1 (Initial) collects K samples of data. *reset_robot*() is executed if a robot push does not move the object (ie. robot is not in contact with the object). In this case, we spin the robot to face the position of the marker attached to the object and move the robot forward until a small change in the object's position is detected. *reset_robot_obj_boundary*() is executed if the robot pushes the object outside the boundary. In this case, we move the robot around the object so that it can push the object back inside the boundary (Fig. 3). The collected data can be used immediately.

D. Experimental Evaluation of Existing Data

After we collect some data with the above method, we can experimentally evaluate this data (Algorithm 1 Experimental Evaluation) to test whether or not it has enough samples for successfully solving goal queries (ie. pushing object to goal). This is an optional process. The idea is to choose random goal queries, and try to use the current pushing strategy (with the non-parametric method in the next section) to solve them. If we can solve these goal queries with a good success rate, we have enough samples and can stop collecting data. Otherwise, this process keeps collecting supplemental data even if the goal queries are not successfully solved. We add the supplemental samples to the pushing strategy interactively and use them immediately.

In Algorithm 1 (Experimental Evaluation), *object_not_move*() is *true* if the robot pushes forward but the object does not move (ie. the robot has moved away from the object). *reset_robot*(), *obj_outside_boundary*(), and *reset_robot_obj_boundary*() are the same as in the initial data collection process. *over_time_limit*() is *true* if an execution of *solve_query*() is not solved before a certain time limit. *solve_query*() drives the robot to deliver the object towards the goal. It uses the data and the non-parametric method in the next section. The query fails if any of the conditions in lines 12 and 14 are true. New supplemental data is stored and used immediately during each query. *good_success_rate*() is *true* if the previous ten queries succeeded. The number of previous queries is a parameter.

E. Object Delivery by Using Collected Data

We describe how to use the collected pushing strategy to control the robot to push the object towards the goal. In the general case (Algorithm 2), we find the goal direction (from object to goal) and the robot position, and use the pushing strategy and non-parametric kernel regression to compute the robot push direction that results in that goal direction. We continuously perform this computation and execute the appropriate robot push until the object reaches the goal. Although pushing can be unstable, the continuous computation and execution of small pushes help to adjust for the noise and errors from pushing.

In Algorithm 2 (General Case), the values are in the

Algorithm 2: Object Delivery by Using Collected Data

General Case:

$(p_x, p_y) = R.pos$;
$(s_x, s_y) = offset * normalize(Goal.pos - O.pos)$;
foreach $sample_i \in D$ **do**
    $dist_i = dist(sample_i, (p_x, p_y, s_x, s_y))$;
$K = set\ of\ indices\ of\ k\ smallest\ dist_i$;
for $k \in K$ **do**
    $weight_k = exp\left(\frac{-dist_k^2}{K_w^2}\right)$;
$(r_x, r_y) = \frac{\sum_{k \in K}\ (weight_k * (r_{xk}, r_{yk}))}{\sum_{k \in K}\ weight_k}$;

Special Case (if last robot push moves object away from goal):

$(s_x, s_y) = offset * normalize(Goal.pos - O.pos)$;
foreach $sample_i \in D$ **do**
    $dist_i = dist(sample_i, (s_x, s_y))$;
$K = set\ of\ indices\ of\ k\ smallest\ dist_i$;
for $k \in K$ **do**
    $weight_k = exp\left(\frac{-dist_k^2}{K_w^2}\right)$;
$(p_x, p_y, r_x, r_y) = \frac{\sum_{k \in K}\ (weight_k * (p_{xk}, p_{yk}, r_{xk}, r_{yk}))}{\sum_{k \in K}\ weight_k}$;

object's coordinate frame. $offset$ adjusts the length of (s_x, s_y) such that it is near the range of the corresponding vectors in the pushing strategy. The $dist()$ function computes the Euclidean distance between (p_x, p_y, s_x, s_y) and the corresponding values in each $sample_i$. We take the nearest k samples for the kernel regression. K_w is the kernel width. We use $k = 5$ and $K_w = 8$ pixels (or about 1.8cm) in our tests. We execute the robot push direction given by (r_x, r_y).

The regression method described above only allows the robot to push the object from its current position. As the robot pushes the object towards the goal, it may deviate from the goal because it can only adjust its push direction. It may be possible that the best robot push direction pushes the object away from the goal (Fig. 4a). If this happens, we execute a special regression case to re-position the robot (Fig. 4b and Algorithm 2 Special Case). The idea is to first find the goal direction (from object to goal), and then use the pushing strategy to find *both the robot position and push direction* that results in that goal direction. We move the robot to the computed position, execute a push in the computed direction, and then return to the general case (or until a special case is needed again).

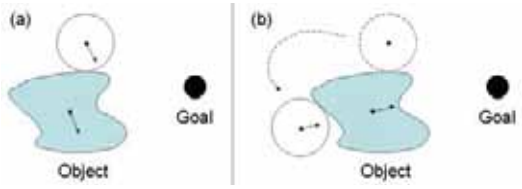

Fig. 4. (a) The "best" robot push direction may push the object away from the goal. (b) If this happens, we perform a special regression step where we use the pushing strategy to find both the robot position and push direction.

In Algorithm 2 (Special Case), the $dist()$ function uses only (s_x, s_y) and not the robot position. Instead the position (p_x, p_y) is computed. The robot moves to (p_x, p_y) by moving around the object similar to the way it does in Fig. 3. It then executes a push in the computed direction (r_x, r_y).

The runtime and storage space for Algorithm 2 are both in the order of the size of the pushing strategy. As there are about hundreds of samples in the pushing strategy, the runtime and storage space are not practical concerns.

As our focus is on local pushing algorithms, we focus on cases where there are no obstacles. If there are obstacles, we first compute a global collison-free path with existing planning techniques [15], and then execute the robot to push the object along sub-goals of this path.

IV. IMPLEMENTATION

Hardware. Our system (Fig. 5) consists of a robot, a ceiling-mounted USB web camera (Logicool Qcam Pro for Notebook, 2M pixels), and a host computer (Dell Latitude E6400, Intel Core2 Duo T9800 2.93GHz processor running Windows XP). The host computer continuously tracks the position/orientation of the robot, object and goal with the camera, and wirelessly sends control signals to the robot. We use a small custom-made differential drive robot with a circular bumper. We use unique markers [16] to identify the robot, object, and goal. These markers provide a simple method for recognition, but other methods can also be used.

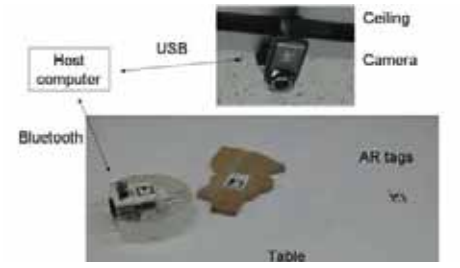

Fig. 5. Hardware configuration.

Software. The tracking and control program was written in Java™. It uses a 2D tracking system with proprietary visual markers, similar to the AR ToolKit [16]. The markers have a 3x3 black-and-white pattern enclosed in a black frame, and size of 5x5 cm. The system can uniquely detect the identity, position, and orientation of each marker. The camera resolution is 960x720 at 30 fps, with a delay of approximately 30 ms.

The control program continuously receives the markers' positions/orientations. Based on this information, it computes the desired robot movement using the algorithms described in the previous section. It then sends low-level control commands (move forward/backward, spin left/right, and stop) to the robot to execute the desired movement. To move the robot forward by a specific amount, the robot starts to move forward and stop when it has travelled within a small value of that amount. If the robot moves forward (or backward) by too much, we allow it to move backward (or forward). This resolves issues of noise in the captured positions/orientations. We use a two-level movement scheme: we first move the robot at a coarser level (by larger distances), and then move it at a finer level (by smaller distances). The coarser level is

useful for speed, while the finer level is useful for accuracy. The spin commands are executed in a similar way.

V. EMPIRICAL EVALUATION

Fig. 6 shows the robot and irregular-shaped objects that we used. The irregular shape with two markers are two separate cases. We cover one of the markers during our experiments. We intentionally put one marker near the middle of the shape and one marker near the corner to test the robustness of our approach. We only used the goal position (and not the orientation) in our experiments. We empirically show that our pushing strategy collection and robot control algorithms work well and, most importantly, are *independent of the shape and weight distribution of the object*.

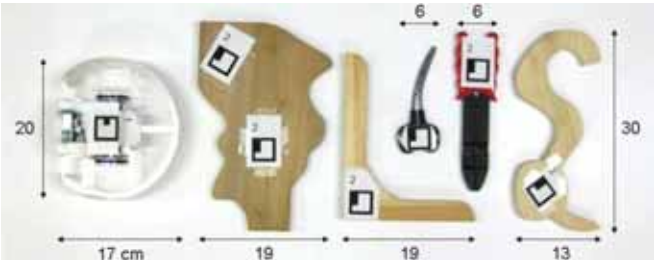

Fig. 6. Sizes of robot and irregular-shaped objects.

Pushing Strategy. Fig. 7 shows the pushing strategy for each object. In each case, we first collect samples with the initial method. We then performed the experimental evaluation process until we can successfully solve ten previous queries. This process also collects supplemental samples. Fig. 7 shows the total number of samples and total time. About a quarter of the total number and time is for the supplemental data. In our experiments, the initial data collection method already gives us good pushing strategies. Hence the evaluation process is mainly for verification purposes, and having the supplemental data was not a necessity. However, having more data can only help the algorithm.

An advantage of the initial method is that we can easily collect a variety of robot pushes with different positions and directions. The variety comes automatically from pushing with a random direction from the robot's current position, and from the robot turning around the object due to the boundary. An interesting observation is that the robot positions that are more likely to be encountered in practice (due to the overall shape of the object) are more likely to have more sample points in the pushing strategies. A disadvantage is that it can take a long time to collect a good variety of data. However, the data can be collected without human intervention and this is not a practical concern.

The advantages of the experimental evaluation method are that we can test if we have enough data with the current pushing strategy, and we can collect supplemental data at the same time. A disadvantage is that the additional samples are a weighted combination of existing samples and hence are not "new" samples. To resolve this issue, we can collect data by alternating the initial and experimental evaluation methods several times, and stopping when the evaluation method can successfully solve the previous ten queries. We found that this is not necessary in our experiments, as the initial data collection already gives us good pushing strategies.

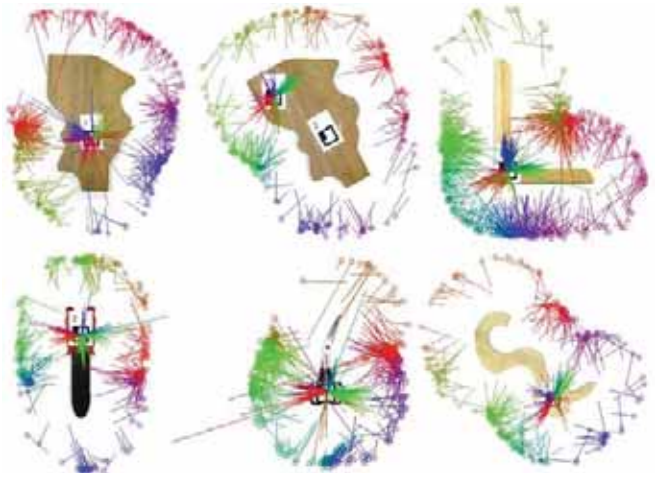

Fig. 7. Pushing strategy for (from top left) irregular, irregular-corner, L-shape, wire-cutter, scissors, and snake-shape. Each data sample (ie. change in robot position and corresponding change in object position) has its own color. There are 423, 217, 680, 245, 329, and 219 samples respectively. Collection times are 116, 99, 180, 56, 73, and 85 minutes respectively.

Shape Independence. We use the same data collection and object delivery algorithms, independent of the object's shape. Fig. 8 shows example trajectories of the robot and object. Please see the accompanying video for more examples.

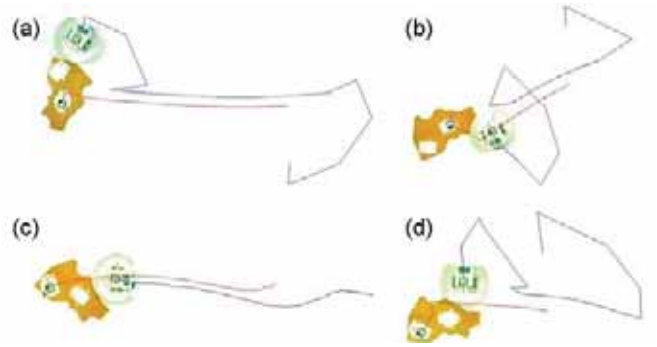

Fig. 8. Robot (blue) and object (red) trajectories for four trials. The robot and object are at the end of the trajectory in each case. In (a), (b), and (d), the robot re-positions itself around the object once at the beginning and once more before reaching the goal. A strength of the algorithm is that we do not need to specify when and the number of times to re-position the robot.

Parameters. We use $k = 5$ and $K_w = 8$ pixels. In general, we can use a larger K_w value if the environment is larger. We experimented with slightly different values of these parameters and found no significant influence on the result.

Learning Rate. Fig. 9 shows examples of the learning rate from our experiments. A trial succeeds if the distance between the goal and the object's marker position is less than 10 pixels in the camera view (or about 2.2cm). A trial fails if the robot or object goes outside the boundary, or the system

goes into an infinite loop (in which case we terminate the execution after 8 minutes).

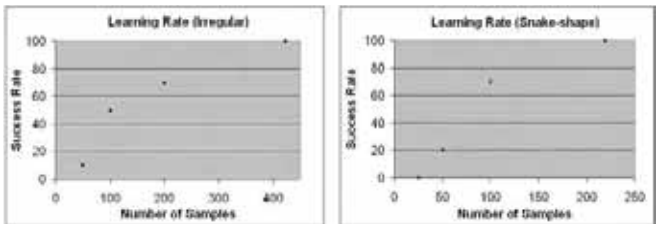

Fig. 9. Each point on the plots are for the same ten goal queries. In general, a larger number of samples in the pushing strategy leads to a higher success rate. The rate of increase (of the success rate) tends to decrease as the number of samples is large.

Comparison with Dipole Method. We compared our method with the dipole field method [5] (Fig. 10 and 11). Our method has a better success ratio than the dipole method in all cases. There are cases where the dipole method works by coincidence depending on the object's shape and initial configuration. For the L-shape, our method fails for two trials because the object went outside of the boundary. If the boundary were bigger (ie. our table was bigger), we believe that our method would have succeeded, as the L-shape required more space to maneuver.

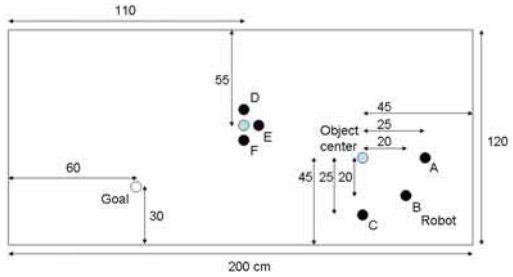

Fig. 10. Layout of test environment.

Initial Position	Irregular		Irregular-corner		L-shape	
	Dipole Field	Our Method	Dipole Field	Our Method	Dipole Field	Our Method
A	0	100	50	100	100	100
B	0	100	20	100	0	100
C	0	100	0	100	0	100
D	30	100	100	100	100	100
E	0	100	80	100	0	80
F	10	100	0	100	0	100

Initial Position	Wire-cutter		Scissors		Snake-shape	
	Dipole Field	Our Method	Dipole Field	Our Method	Dipole Field	Our Method
A	0	100	20	100	0	100
B	0	100	20	100	0	100
C	0	100	10	100	0	100
D	50	100	0	100	0	100
E	100	100	0	100	0	100
F	30	100	0	100	0	100

Fig. 11. Comparison between dipole field method and our method. We show the success percentage out of 10 trials.

VI. LIMITATIONS AND FUTURE WORK

Our approach requires collecting a pushing strategy for each object. It is possible to scale an existing pushing strategy for objects of the same shape but different scale. We may further use existing strategies and generalize them for different shaped objects in the future. In addition, some pushing strategies may contain more data than we need, and we can explore the idea of pruning the samples that do not improve the result significantly. We can also more formally analyze the relationships between the number of samples needed, the object shape, the location of the marker, and the success rate.

Our algorithm is not optimal and not complete. We also currently do not analyze the quality of the solution. These limitations are because our approach is greedy and learning-based. The tradeoff is that our approach works for any irregular-shaped object and is easy to understand.

Our current method for handling obstacles is simple and may fail. For example, we may ask the robot to push left (or push from the right) when there is an obstacle to the right. In the future, we can take into account such cases with our pushing approach.

For future work, we can define a formal measure of the variety of samples in a pushing strategy based on the robot positions and push directions. The idea is that if there are regions of the object without samples (or without as many samples), we should detect this automatically and explicitly instruct the robot to collect more samples in those regions. This can also improve the current runtime of the data collection process.

REFERENCES

[1] B. P. Gerkey and M. J. Mataric, "Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination," in *Int'l Conf. on Robotics and Automation (ICRA)*, May 2002, pp. 464–469.

[2] D. Nieuwenhuisen, A. Frank, and H. Overmars, "Path planning for pushing a disk using compliance," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 4061–4067.

[3] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars, "Pushing using compliance," in *ICRA*, 2006, pp. 2010–2016.

[4] M. de Berg and D. Gerrits, "Computing push plans for disk-shaped robots," in *Int'l Conf. on Robotics and Automation (ICRA)*, May 2010.

[5] T. Igarashi, Y. Kamiyama, and M. Inami, "A dipole field for object delivery by pushing on a flat surface," in *ICRA*, 2010.

[6] 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.

[7] S. Walker and J. K. Salisbury, "Pushing using learned manipulation maps," in *ICRA*, 2008, pp. 3808–3813.

[8] R. Emery and T. Balch, "Behavior-based control of a non-holonomic robot in pushing tasks," in *ICRA*, 2001.

[9] K. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," vol. 15, no. 1, December 1996, pp. 533–556.

[10] K. Lynch, "Estimating the friction parameters of pushed objects," in *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1993, pp. 186–193.

[11] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Vision-based reinforcement learning for purposive behavior acquisition," in *Int. Conf. on Robotics and Automation*, 1995, pp. 146–153.

[12] T. Nakamura and M. Asada, "Motion sketch: Acquisition of visual motion guided behaviors," in *IJCAI*, 1995, pp. 126–132.

[13] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *International Conference on Machine Learning*, 1997, pp. 12–20.

[14] D. C. Bentivegna and C. G. Atkeson, "Learning from observation using primitives," in *ICRA*, 2001, pp. 1988–1993.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://planning.cs.uiuc.edu/), 2006.

[16] H. Kato, M. Billinghurst, B. Blanding, and R. May, "AR ToolKit," in *Technical Report, Hiroshima City University*, Dec 1999.