A Robust Pushing Skill For Object Delivery Between Obstacles

Senka Krivic [1], Emre Ugur[1,2] and Justus Piater [1]

Abstract— **Nonprehensile manipulation can play a significant role in complex robotic scenarios, especially for maneuvering non-graspable objects. A big challenge is to construct a robust skill for pushing highly-diverse objects. We present a strategy for pushing unknown objects that differ widely in their properties. For this purpose we introduce the concept of a *pushing corridor* for cluttered environments that leaves the robot sufficient space for corrective motions. We propose a reactive manipulation skill for pushing objects along this collision-free corridor. The motion of the robot is generated and adapted on the fly based on the observed reactions of the object in response to pushing actions. Our results show that the robot is able to successfully push objects in various complex scenarios.**

I. Introduction

Pushing is a commonly used activity by humans in different everyday activities. We push things out of the way, close doors, bring and relocate objects by pushing them. Introducing a similar skill for robots would enable new capabilities and increase dexterity of a mobile robot in complex scenarios. For example, if a mobile robot is not equipped with an arm, or already holding an object, the ability to push with the base can be used for clearing paths or conveying objects. In this paper we address the problem of delivering an object to a goal position in cluttered environments.

In various robotic scenarios a robot has to operate in environments with scattered objects and tangled heaps on the floor, limiting the robot's freedom to move. Therefore, we introduce a collision-free *corridor* which is defined by a path and its surrounding area with a fixed width as shown in Fig. 1. To push along such a corridor, we also introduce the concept of a *target point* towards which the robot pushes the object at every instant. This point is determined by a cost function which we define in Section IV. This allows cutting corners as long as the corridor is not violated by the robot or by the object.

In complex scenarios objects may have diverse, even anisotropic properties under pushing. For example, while some objects slide freely, other may even flip over. Generally, it is not straightforward to design a physical model describing the behavior of an object [1], [2]. Visual features can be non-informative, or even misleading, for the description of object behaviors while pushing [2]. Thus, a robot should react adaptively to different robot-object-environment dynamics. Humans are able to push various objects in a desired way by moving them approximately into designated directions and changing the hand motion based on the observed behavior of the object. We aim to incorporate a similar concept into the push-manipulation skill by adapting robot motion based on observations of the behavior of the object.

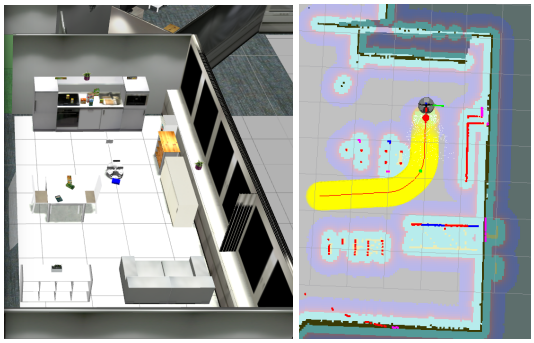

Fig. 1. Pushing in a complex environment. The collision-free path (red line) is obtained from a path planner, and defines the corridor (yellow). The current target point is marked with green dot on the path. (This and other figures are best viewed in color).

Most of the time, particularly along smooth trajectories, the robot should be able to push the object using smooth corrective motions while keeping contact with the object. Sometimes, however, especially at sharp turns, the robot will need to *relocate* by leaving the object and moving towards a new angle of attack. Thus, at every time step, we define robot movement directions both for smooth pushing and for relocating. The final motion of the robot is a result of blending these two motion vectors weighted by their respective activation functions. These functions are adapted over time based on observations from the robot-object interaction. At onset of the pushing manipulation, the robot assumes it can push the object if it is behind it with respect to the target point. Since properties of the object and its behaviour are unknown, we also introduce an additional feedback component to react to unexpected movements of the object.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 610532, Squirrel

[1]Authors are with the Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria. Contact information: senka.krivic@uibk.ac.at, emre.ugur@uibk.ac.at, justus.piater@uibk.ac.at

[2] The author is with the Department of Computer Engineering, Bogazici University, Istanbul, Turkey.

II. Related work

Nonprehensile manipulation raised interest in various scientific areas. Early work on pushing by Mason and Lynch [3] presented a theoretical model of the dynamics of pushing with a robotic manipulator. One of the first systems that took into account vision feedback for building a forward model of a planar object was created by Salganicoff et al. [4]. In later work, the action of pushing was often used for learning dynamic models, behaviors and object specific properties by observing results of the push [5], [2].

Ruiz-Ugalde et al. [2], [6] built a mathematical model for planar sliding motion. They executed a pushing behavior by determining the static and kinetic friction coefficients between the robot hand, the table and a rectangular object. Scholz and Stilman [7] learned object-specific dynamic models for a set of objects through experience. A robot learns Gaussian models of the pose displacement which are then used for the push selection. In a similar tabletop scenario Omrcen et al. [8] used a neural network to learn the pushing dynamics for flat objects. The learned models were specific to individual objects, and generalization to similar objects was not examined.

Several methods considered the problem of learning contact locations for successful pushes [9]. Most of them focus on extracting the shape of the object and determining contact locations that are good for pushing. Early work by Salganicoff et al. [4] presents an unsupervised online method for pushing in an obstacle-free environment by controlling only one degree of freedom. Their method selects each pushing direction using a nearest-neighbor approach. The point of contact is restricted to a single rotational contact point. With a similar approach, Walker and Salisbury [10] learn the support friction by mapping pushes to motion. They perform experiments by pushing the object from several contact points. Using local regression they model the support friction within a two-dimensional environment. Hermans et al. [11] develop an offline, data-driven method for predicting effective contact locations for pushing previously-unseen objects. They also develop a method for extracting the 2D shape of an object and generate points for performing pushing tests.

For the problem of delivery of the object from one position to another some scientists [12] focused on determining appropriate pushing actions and developing a push planner which will complete the task using these actions. Hermans at al. [11] were pushing towards the centroid of the object from a given point using the feedback controller. Input signals included the object-target displacement and the displacement of the robot from the pushing direction through the object's centroid.

Igarashi et al. [13] presented a novel pushing algorithm for circularly-shaped objects. Their controller consists of components for pushing and orbiting. They assigned two weighting functions to these movements where the vector of the robot movement resembles a dipole field (in analogy to physics). Although this is an elegant solution, it has many disadvantages in practice such as long trajectories while moving to correct positions, and excessive orbiting caused by even small displacements of the robot from the pushing direction.

Inspired by this approach, we create a feedback controller that is also based on the displacement angle of the robot from the desired object movement direction. However, in our approach, the robot learns when to relocate and when to push. In addition, we introduce a component to react to the changes in object movement behavior to overcome the strong assumptions about ideal properties of the object.

The state-of-the-art method presented by Mericli et al. [14] addresses the problem of pushing complex, massive objects on caster wheels with a mobile base in cluttered environments. Based on experience from self-exploration or demonstration via joy-sticking, they create experimental models used for planning the pushing action using an RRT-based method.

Among the many existing approaches and problems for push-manipulation in the literature, ours differs in several aspects:

- We address single-contact pushing of objects of various sizes, weight, mass and friction distributions.
- We introduce the notion of a pushing corridor within which the robot can interact with objects without having to take care of obstacle avoidance.
- The presented method is reactive and requires neither time-expensive learning phases nor prior knowledge.
- Our strategy does not make strong assumptions about the way the object reacts to pushes; it adapts the pushing motion based on current observations on the fly.

III. Problem description

Our case study involves an untidy room of a child which has to be tidied up. A robot needs to be able to push various objects like toys, boxes and small pieces of furniture to designated locations. In order to create a robust skill that can be applied directly in different scenarios, we decompose the problem into two subproblems:

- Define a pushing corridor and choose a target point for pushing the object at each time step allowing collision-free pushes.
- Determine a robust method for pushing objects towards a target point.

We assign a two-dimensional static frame g to the environment and a local frame r to the robot. We assume that the global position R of the robot and

the global position O of an object are available at each time step t and are given by $R(t) = \left[x_R^g(t)\ y_R^g(t)\right]^\mathsf{T}$ and $O(t) = \left[x_O^g(t)\ y_O^g(t)\right]^\mathsf{T}$, respectively. In addition, we assume that a map of the environment is available for path planing. The size of the robot is determined by its radius r_R, and the system is able to approximate the radius r_O of the bounding circle around the pushed object.

IV. Pushing Corridor

We define a pushing corridor $\mathcal{C}(\mathbf{Q}, w)$ as an area around the collision-free path $\mathbf{Q}$ from the start to the goal object pose given by a set of waypoints

$$\mathbf{Q} = \{Q_1, Q_2, Q_3, ..., Q_N\},$$

with the width w of the corridor which is defined as twice the distance from the path to the closest obstacle. A robot is allowed to move and push objects within this corridor ensuring collision avoidance.

At each time step, the robot pushes along the corridor by choosing the next target point. This allows straight-line pushes while constraints of the corridor are not violated. This procedure can be compared to the concept of the lookahead distance in navigation [15] which tells the robot how far along the path it should look from the current location to compute the velocity commands. In a pushing scenario, short lookahead distance ensures better path tracking but might lead to oscillations in object movements, and to long robot trajectories. By contrast, large lookahead distances enable short object trajectories but raise the possibility of corridor violations. Thus, to determine a target point we define a cost function that ensures long lookahead distances for pushing but also penalizes approaching corridor margins.

Let $J(O, Q)$ be a cost function defined as

$$J(O(t), Q) = \frac{1}{d(O(t), Q)} + \zeta d_{\max}$$

where (as shown in the Fig. 3):

- $O(t)$ is the current object position, Q is the waypoint and t is the time.
- $d(\cdot, \cdot)$ denotes the Euclidean distance between two points,
- $d_{\max}$ is the maximum distance between the path $\mathbf{Q}$ and the current push-line $\overline{OT}$, and

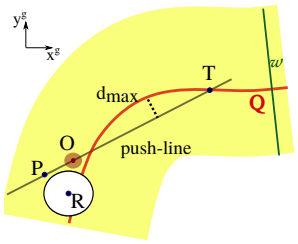

Fig. 3. Determining the next target point for pushing. R and O correspond to robot and object positions, the red line is the path, T is the current target point, and P denotes the desired position of the robot. $\mathbf{Q}$ is the path given by set of waypoints.

- ζ is the inflexibility factor of a pushing corridor given by the ratio of the corridor width and the minimum width needed for both robot and object to fit between the path $\mathbf{Q}$ and the closest obstacle:

$$\zeta = \begin{cases} \frac{w}{4(r_R + r_O)}, & \text{if } \frac{w}{4(r_R + r_O)} \leq 1 \\ 1, & \text{otherwise.} \end{cases}$$

Then a target point $T(t)$ can be determined at each time step as

$$T(t) = \arg\min_{Q \in \mathbf{Q}} J(O(t), Q) \quad (1)$$

$$s.t. \quad \frac{w}{2} - \zeta\left(d_{\max} + 2r_R\right) \geq 0 \quad (2)$$
$$\frac{w}{2} - \zeta(d_{\min} + r_R) \geq 0$$

where $d_{\min}$ is the minimal distance between the path $\mathbf{Q}$ and an ideal position of the robot for the current pushing direction denoted by P. This is a point on the push-line with distance $r_R + r_O$ to the object.

The conditions (2) ensure that neither the robot nor the object violate the corridor in case of high path curvatures or the relocation of the robot. It can be noticed that $\zeta < 1$ defines a corridor with less flexibility for the robot movement since the robot cannot fit between the corridor edge and the object. This factor allows mild shortcuts even in case of corridors so narrow that its non-violation cannot be guaranteed (Fig. 2).

Observe that our method can be extended to environments with moving obstacles by checking the validity of a corridor at each time step and recalculating it if required. Such a dynamic change of the corridor does not affect the pushing procedure since the target point is continually updated.

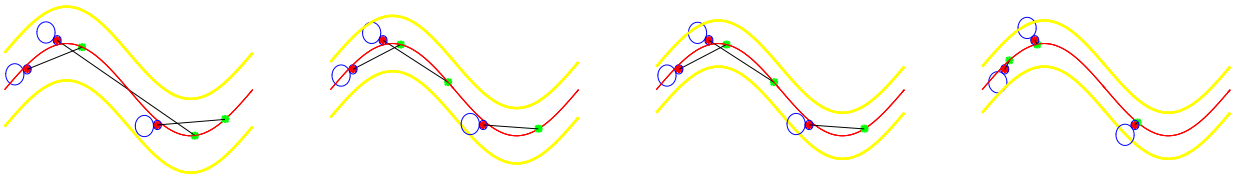

Fig. 2. Examples of the procedure for determining target points. Parameters were set as following: $r_R = 0.23$, $r_O = 0.10$. The path is defined as $y = \sin(x)$ for $x = [0, 2\pi]$. Two examples on the right side are given for the same corridor with and without effect of ζ.

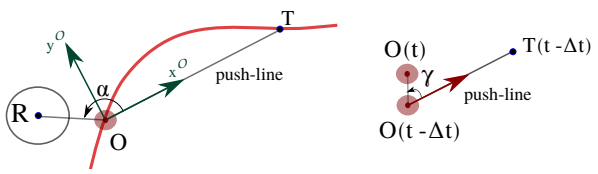

Fig. 4. The angle α between the robot, the object and the current push-target is used to determine the robot's movement direction (left). The angle of object movement displacement γ (right).

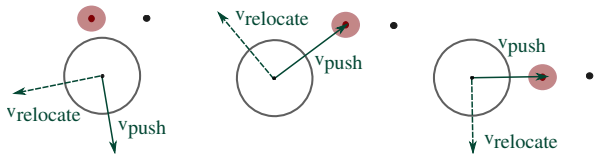

Fig. 5. Examples of push and relocate non-weighted directions defined in (3) and (4) for different locations of the robot, the object and the target point.

V. Reactive Pushing

In this section we introduce a reactive controller for pushing the object towards a target point. The target point is calculated at each time step based on the optimization routine introduced in the previous section. Therefore it is necessary that the robot adapts its movements to new pushing directions as well as to the behavior of an object. Thus two robot behaviors are needed: *pushing* and *relocating*.

Inspired by the dipole field method by Igarashi et al. [13] we define directions of the robot movement for both behaviors. The final robot movement direction is determined by blending these two. They are defined by the angle of robot displacement from the desired object movement direction denoted by $\alpha = \measuredangle TOR$ in Fig. 4. To simplify the description, let us assign a local frame to the object o where the x axis coincides with the desired object motion defined by line $\overline{OT}$. The pushing direction is determined in the object local frame as

$$v^o_{\text{push}} = \text{sgn}(\cos\alpha)\begin{bmatrix}\cos\alpha\\ \sin\alpha\end{bmatrix} \tag{3}$$

If the robot is in a position from which it cannot directly push the object, it should relocate to the position from which the push can be achieved. This relocation direction is given by

$$v^o_{\text{relocate}} = \text{sgn}(\sin\alpha)\begin{bmatrix}-\sin\alpha\\ \cos\alpha\end{bmatrix} \tag{4}$$

Examples for such movement directions are shown in the Fig. 5.

A. Activating the movement actions

To achieve smooth movements of the robot we blend these two behaviors. However, they have to be weighted to achieve successful object deliveries. If the robot is behind the object ($\alpha \approx \pi$), it should be able to push it towards the target. Thus, we build in this assumption by assigning the activation function ψ_{push} as a weight for the pushing behavior. The robot needs to learn which values of α are suitable for pushing. For that purpose, we define the push activation function as a Gaussian kernel over the angles α from which pushing towards the target point is achievable,

$$\psi_{\text{push}}(\alpha) = \exp\left(-\frac{(\alpha-\mu_\alpha)^2}{2\sigma_\alpha^2}\right),$$

where μ_α is the expected value of the robot displacement α for which the object moves towards the target, and σ_α is its variance. At the onset of a pushing action, the initial values are set to $\mu_\alpha = \pi$ and $\sigma_\alpha = \frac{\pi}{3}$. These values are then incrementally updated during the execution of pushing based on observations of the object behavior.

Desired values of the α for the pushing behavior are those that result in the object moving along the push-line $\overline{OT}$. Thus, we define the error of the object movement

$$\gamma(t) = \measuredangle T(t-\Delta t)\, O(t-\Delta t)\, O(t) \tag{5}$$

as the angle between the desired push-line and the resulting position of the object. Therefore the distribution of robot displacements α resulting in successful pushes can be described by the conditional probability density $p(\alpha|\dot\gamma < 0) \propto \psi_{\text{push}}(\alpha)$, where $\dot\gamma$ is the change of the error.

The relocation activation function is simply defined in terms of the push activation function, $\psi_{\text{relocate}}(\alpha) = 1 - \psi_{\text{push}}(\alpha)$. While relocation is active, the pushing component attracts the robot to the object, not allowing large distances between them.

B. Compensating displacements of the object movement

When the robot is pushing an object, it assumes the object moves on the push-line ($\gamma \approx 0$). However, this is not necessarily the case due to different mass distribution and the dynamics of objects. Thus, the robot tries to capture the model of the object behaviour by learning the expected value of the object movement error γ. We introduce an additional feedback term to compensate for constant deviations of the observed from the expected object movement:

$$v^o_{\text{compensate}} = \begin{bmatrix}-\sin\gamma\\ -\cos\gamma\end{bmatrix}(\mu_\gamma - \gamma), \tag{6}$$

where μ_γ is the mean value of experienced displacements, and all angles are expressed in radians. Larger values of μ_γ may indicate objects which do not have quasi-static properties or uniform mass/friction distributions. To produce more cautious movements of the robot in such cases, we modulate the velocity of the robot movement by this value:

$$V = \frac{V_{\max}}{1+|\mu_\gamma|}. \tag{7}$$

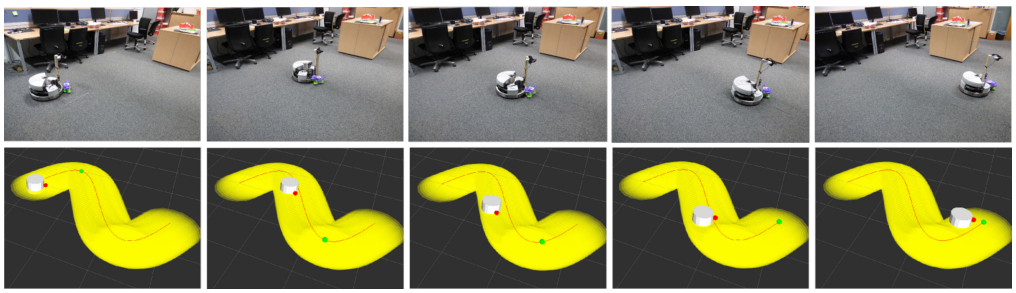

Fig. 6. A pushing trial in the real environment. Parameters were $r_R = 0.23$, $r_O = 0.12$, $w = 1.6$. The path was given as $y = \sin x$ for $x \in [0, 2\pi]$. The values are given in $[m]$. Time of delivery 58.70**s**

C. Robot motion

The direction of the robot motion is now defined as

$$v^o_{\text{motion}} = \psi_{\text{push}} \left(v^o_{\text{push}} + v^o_{\text{compensate}} \right) + \psi_{\text{relocate}} v^o_{\text{relocate}}$$

The control signal u^r_R is defined by a scalar term V (7) representing the desired velocity magnitude, and a unit vector in the desired robot movement direction v^o_{motion}:

$$u^r_R = R^o_r V \frac{v^o_{\text{motion}}}{\| v^o_{\text{motion}} \|}$$

where R^o_r is the rotation matrix from the object local frame o to the frame r of the robot.

VI. Experimental results

To evaluate the effectiveness of the proposed method, several tests were done both in simulation and in a real environment. The tests were performed with synthetic corridors of straight-line and sine-wave shape (5 times with each object for a particular path), as well as with corridors obtained automatically from a navigation system to demonstrate the robustness of the strategy. We used a holonomic robot with a circular base, the Robotino[1] by Festo, both in simulation and real environments. The simulator used is Gazebo, and the scenario is the one shown in Fig. 1. We performed tests with objects of different properties as shown in Fig. 7, and in environments with surfaces of different friction properties (carpet as in Fig. 6 and tiles as in Fig. 7).

Table I shows success rates of pushing tests performed by the real robot. The results show that method is robust to the width change of the corridor and the object shape and size. However, in most cases recorded as failures for violating the corridor, the robot still delivered the object to the target. In straight-line tests most of the failures occurred at the end of pushing procedures where pushing actions were biased towards presented initial values which did not change much due to a simple path. In sine-wave shape tests, robot was loosing the object the most at the places of high curvatures where it was not able to push object towards the push line due to unexpected object behavior. Line pushes lasted between 16 [s] and 22 [s], and sine pushes between $41\,[s]$ and $67\,[s]$ where the maximum speed of the robot was set to $0.2\,[m/s]$.

[1]http://www.festo-didactic.com/int-en/services/robotino

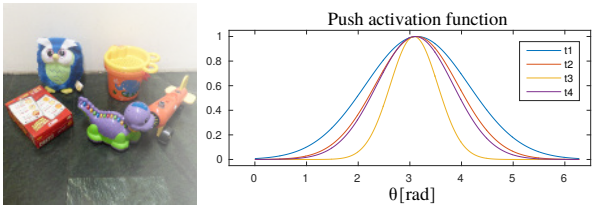

Fig. 7. Left: The set of objects used for pushing experiments contains objects of various shapes, sizes, friction properties, mass distributions and quasi-static properties. Right: Example of the change of the push activation function over time ($t_1 < t_2 < t_3 < t_4$). It can be noticed that the variance of α was decreasing and increasing again.

TABLE I

Success rates of pushing tests in real environment

path	$w = 1.8\,[m]$	$w = 1.6\,[m]$	$w = 1.4\,[m]$
straight line length: $3.14\,[m]$	100%	92%	88%
sine wave length: $6.28\,[m]$	96%	84%	76%

Examination of the robot trajectories reveals that the robot simplifies the pushing task by cutting corners as allowed by the corridor. One such sample execution is provided in Fig. 6. We analysed this effect with different corridor widths w. In the case of wide corridors, the robot tries to push towards the final goal at onset (illustrated in Fig. 8) but becomes more conservative as soon as deviations from desired direction are observed. When pushing towards the final goal is possible again, the robot relocates.

In the case of narrow corridors, the ζ parameter relaxes the corridor non-violation condition of the pushing strategy (2). The robot is allowed to violate the corridor at most for the distance $\frac{(1-\zeta)}{\zeta}\frac{w}{2}$ (see Eq.(2)). We did multiple tests also with narrow corridors with $w = 1.0$. The object delivery was achieved with 68% success.An example is shown in Fig. 8.

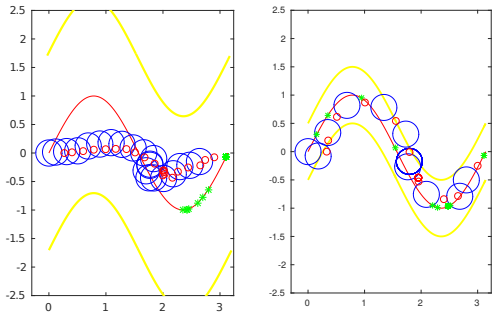

Fig. 8. Left: Result of pushing a small object of radius $r_O = 0.06$ in a very wide corridor of width $w = 3.4$. Right: Result of pushing the same object of in a very narrow corridor of width $w = 1.0$, $\zeta = 0.862$. Both results were produced by the real robot.

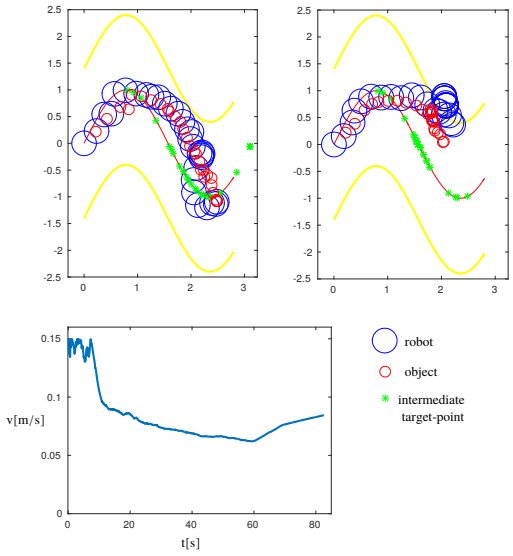

Fig. 9. Results of pushing with (top left) and without (top right) the compensation term (6). The bottom graph demonstrates the change of velocity (7); $V_{max} = 0.15[m/s]$.

The details of the pushing behavior depend on the observations of the object behavior, especially at the beginning of a push. The parameters μ_α, σ_α and μ_γ can be updated with incorrect/noisy perception or insufficient experience, which is the principal cause of the failures observed. During the pushing procedure the robot might become increasingly conservative (smaller values of σ_α) which causes increased relocating behavior, but due to new experience it increases the variance of α, generally settling at smooth and robust pushing behavior over time as can be seen in Fig. 7 (right).

We analyzed the effect of the compensation term and velocity modulation by performing tests with and without them. The results are shown in Fig. 9. It can be noticed that as soon as deviation of the object movement direction occurs, the robot slows down and corrects its movement direction by the compensation term.

VII. Conclusion

We proposed a strategy for object delivery by single-contact pushing with a robot base in cluttered environments. For this purpose we introduced the notion of a pushing corridor which enables following a collision-free path with constraints but in a flexible way. Instead of tuning control parameters with respect to the object class or the shape, we proposed identifying the object behavior by learning. The presented algorithm is computationally cheap and enables real-time push manipulation. It is purely reactive and does not require prior models or a training phase. Our results show it is robust for our set of objects of various shapes in different environments.

References

[1] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[2] F. Ruiz-Ugalde, G. Cheng, and M. Beetz. Fast adaptation for effect-aware pushing. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 614–621, Oct 2011.

[3] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[4] Marcos Salganicoff, Giorgio Metta, Andrea Oddera, and Giulio Sandini. A vision-based learning method for pushing manipulation. In *In AAAI Fall Symposium Series: Machine Learning in Vision: What Why and*, 1993.

[5] D. Katz and O. Brock. Manipulating articulated objects with interactive perception. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 272–277, May 2008.

[6] F. Ruiz-Ugalde, G. Cheng, and M. Beetz. Prediction of action outcomes using an object model. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1708–1713, Oct 2010.

[7] J. Scholz and M. Stilman. Combining motion planning and optimization for flexible robot manipulation. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 80–85, Dec 2010.

[8] D. Omrcen, C. Boge, T. Asfour, A. Ude, and R. Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 277–283, Dec 2009.

[9] M. Kopicki, S. Zurek, R. Stolkin, T. Morwald, and J. Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5722–5729, May 2011.

[10] S. Walker and J.K. Salisbury. Pushing using learned manipulation maps. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3808–3813, May 2008.

[11] T. Hermans, Fuxin Li, J.M. Rehg, and A.F. Bobick. Learning contact locations for pushing and orienting unknown objects. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 435–442, Oct 2013.

[12] Qingguo Li and Shahram Payandeh. Manipulation of convex objects via two-agent point-contact push. *Int. J. Rob. Res.*, 26(4):377–403, April 2007.

[13] Takeo Igarashi, Yoichi Kamiyama, and Masahiko Inami. A dipole field for object delivery by pushing on a flat surface. In *ICRA*, pages 5114–5119, 2010.

[14] Tekin A. Mericli, Manuela Veloso, and H. Levent Akin. Achievable push-manipulation for complex passive mobile objects using past experience. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 71–78, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[15] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, DTIC Document, 1992.