Noname manuscript No.
(will be inserted by the editor)

Push-Manipulation of Complex Passive Mobile Objects using Experimentally Acquired Motion Models

Tekin Meriçli · Manuela Veloso · H. Levent Akın

Received: date / Accepted: date

Abstract In a realistic *mobile* push-manipulation scenario, it becomes non-trivial and infeasible to build analytical models that will capture the complexity of the interactions between the environment, each of the objects, and the robot as the variety of objects to be manipulated increases. We present an experience-based push-manipulation approach that enables the robot to acquire experimental models regarding how pushable real world objects with complex 3D structures move in response to various pushing actions. These experimentally acquired models can then be used either (1) for trying to track a collision-free guideline path generated for the object by reiterating pushing actions that result in the best locally-matching object trajectories until the goal is reached, or (2) as building blocks for constructing *achievable* push plans via a Rapidly-exploring Random Trees variant planning algorithm we contribute and executing them by reiterating the corresponding trajectories. We extensively experiment with these two methods in a 3D simulation environment and demonstrate the superiority of the achievable planning and execution concept through safe and successful push-manipulation of a variety of passively mobile pushable objects. Additionally, our preliminary tests in a real world scenario, where the robot is asked to arrange a set of chairs around a table through achievable push-manipulation, also show promising results despite the increased perception and action uncertainty, and verify the validity of our contributed method.

Keywords Push manipulation · Manipulation planning · Experience-based manipulation

The first author was partly supported by The Scientific and Technological Research Council of Turkey under Programmes 2211 and 2214, and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610. This research was further supported by the National Science Foundation under grant number IIS-1012733, by the Office of Naval Research under grant number N00014-09-1-1031, and by the Air Force Research Laboratory under grant number FA87501220291. The views and conclusions contained herein are those of the authors only.

T. Meriçli
Department of Computer Engineering
Boğaziçi University
Bebek, 34342, Istanbul, Turkey
E-mail: tekin.mericli@boun.edu.tr

M. Veloso
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, United States
E-mail: veloso@cmu.edu

H. L. Akın
Department of Computer Engineering
Boğaziçi University
Bebek, 34342, Istanbul, Turkey
E-mail: akin@boun.edu.tr

1 Introduction

Pushing is one of the many modalities of *non-prehensile* manipulation (Lynch, 1996), which may be the most suitable option depending on the requirements of the manipulation task and the constraints imposed by the physical properties of both the object and the robot. For instance, the object may be too large or heavy, the robot may not be equipped with a manipulator arm, or the utilization of some properties of the object may make its transportation more efficient and convenient that way. The objective of push-manipulation is to come up with and execute a sequence of pushing actions to maneuver an object incapable of moving by itself from an initial configuration to a goal configuration. In this study, we expect our omni-directional mo-

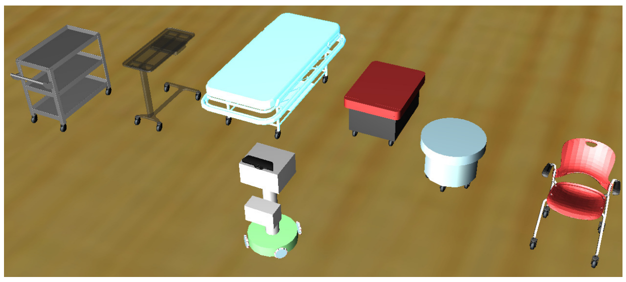


Fig. 1 Realistically simulated passive mobile objects and our omni-directional mobile robot used as the pusher.

bile robot CoBot (Rosenthal et al, 2010), which is not equipped with a manipulator arm, to push-manipulate a set of passive mobile objects (Fig. 1) in such a way to transport them to their desired poses while avoiding collisions in the task environment cluttered with obstacles. The following facts make our problem a particularly challenging one.

– Our pushable objects move on passive caster wheels, which introduce additional motion uncertainty as they continue moving for some time even after the push is ceased. Objects with these kinds of properties are inherently more difficult to push-manipulate compared to objects that slide quasi-statically on high-friction surfaces.
– Our objects have complex 3D structures. It is neither trivial nor feasible to write down analytical interaction and motion models for each and every one of such complex objects; hence, traditional model-based planning approaches will not solve the problem in a flexible way.

As a promising solution, we develop a method that does not require any explicit analytical models for neither the objects nor the robot (Meriçli et al, 2012, 2013). Following a case-based approach (Veloso, 1994) instead, the robot builds object-specific *experimental motion models* by memorizing the observed effects of its pushing moves on various passive mobile objects. The acquired models are then used in the following ways.

1. A collision-free path for the object is planned without taking into account whether the plan can be achieved with the available experimental models, and the robot tries to make the object track the path by reiterating the memorized pushing actions that result in the best locally-matching object trajectories until the goal is reached.
2. The acquired experimental models are used as building blocks for constructing safe and *achievable* push-manipulation plans via the *Exp-RRT* algorithm, a Rapidly-exploring Random Trees (RRT) variant we contribute (Meriçli et al, 2012, 2013) and executing the solution path.

The following sections elaborate on our experience-based push-manipulation methods.

2 Related Work

Pushing enables complex manipulation tasks to be performed with simple mechanics in cases where the object is too bulky or heavy to lift, or the robot simply lacks a manipulator arm. As a result of being one of the most interesting methods used within the non-prehensile manipulation domain (Lynch, 1996; K. M. Lynch and M. T. Mason, 1997), push-manipulation has attracted several robotics researchers. An early work by Salganicoff et al (1993) presents a very simple, 1-nearest neighbor based approximation method for the forward model of an object being pushed from a single rotational contact point in an obstacle-free environment by controlling only one degree of freedom. Agarwal et al (1997) propose an algorithm for computing a contact-preserving push plan for a point-sized pusher and a disk-shaped object using discrete angles at which the object can be pushed and a finite number of potential intermediate positions for the object. They assume that their pusher can place itself at any position around the object since it does not occupy any space; however, this approach cannot be used when real robots are considered as they have non-zero dimensions that can collide with the obstacles in the environment. Nieuwenhuisen et al (2005, 2006) utilize compliance of the manipulated object against the obstacles rather than trying to avoid them, and make use of the obstacles with linear surfaces in the environment to guide the object's motion by allowing the object to slide along the boundaries. de Berg and Gerrits (2010) computationally improve this approach and present both a contact preserving and an unrestricted push planning method in which the pusher can occasionally let go of the object. Similar to the potential field based motion planners (Khatib, 1986), Igarashi et al (2010) propose a method that computes dipole-like vector fields around the object that guide the motion of the robot to get behind the object and push it towards the target. Relatively slow robot motions and high friction for the objects are assumed, and robots with circular bumpers are used to push circular and rectangular objects of various sizes in single and multi-robot scenarios. As a promising step towards handling objects with more complex shapes, Lau et al (2011) achieve pushing of irregular-shaped objects with a circular robot by collecting hundreds of samples on how the object moves when pushed from different points in different directions, and using a non-parametric regression method to build the corresponding mapping, similar to the approach of Walker and Salisbury (2008).

Their approach resembles ours in the sense that they also utilize the observations of the object's motion in response to various pushing actions. Even though they use irregular-shaped objects in their experiments, those objects are flat ones with quasi-static properties and the final placement orientation is ignored in their experiments, which further simplifies the problem. Zito et al (2012) present an algorithm that combines a global sampling-based planner with a local randomized push planner to explore various configurations of the manipulated object and come up with a series of manipulator actions that will move the object to the intermediate global plan states. Their experiment setup consists of a simulated model of a tabletop robot manipulator with a single rigid spherical fingertip and an L-shaped object (a polyflap) to be manipulated. The setup is obstacle-free and the state space is limited to the reach of the robot arm, which is relatively small, as they are using a stationary manipulator. The randomized local planner utilizes a realistic physics engine to predict the object's pose after a certain pushing action, which requires explicit object and contact modeling. Kopicki et al (2011) use the same problem setup and present an algorithm for learning through interaction the behavior of the manipulated object that moves quasi-statically in response to various pushes. However, the learned object behavior is not used for push planning in their work. Scholz and Stilman (2010) use the observed outcomes of a set of four linear and two rotational pushes for planning in a quasi-static tabletop setup where a manipulator arm with a spherical rigid finger push-manipulates objects with simple geometric shapes, ensuring single point of contact. Another recent study by Dogar and Srinivasa (2012) uses push-manipulation in a tabletop manipulation scenario as a way to reduce uncertainty prior to grasping by utilizing the funneling effect of pushing. Along the lines of learning object kinematics and dynamics, Katz and Brock (2008) propose an interactive perception approach, where the robot interacts with the objects in a tabletop scenario and identifies individual rigid bodies that compose the object by tracking how the corresponding visual feature points move relative to each other. Using that information, the robot is able to extract the kinematics of the object of interest.

According to our survey of the literature, the most common push-manipulation scenarios seem to involve pushing of objects with primitive geometric shapes using circular or point-sized robots, or rigid fingertips on a surface with relatively high friction that makes the object stop immediately when the pushing motion is ceased. Even then, relatively complex analytical models are derived to define the object's behavior or realistic physics engines of simulators are used for contact modeling and motion estimation. Our approach differs from many of these proposed ones in the sense that;

- we deal with complex 3D real world objects that may contact the robot on various points (Fig. 2),
- the manipulated objects do not move in a quasi-static manner; they continue moving freely for a while after the push, and their caster wheels contribute to their motion uncertainty,
- *mobile* manipulation is performed in a large-scale environment cluttered with obstacles, requiring construction and execution of safe and *achievable* plans,
- no explicit analytical model is used or learning based mapping is built; only the pushing motions performed in the past and their corresponding observed effects along with the associated variances are utilized for planning and execution.

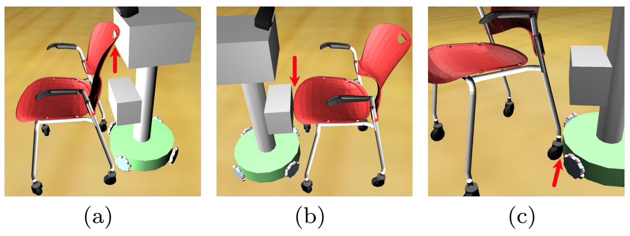


Fig. 2 The object may contact (indicated by a red arrow) the robot at (a) its body, (b) basket, or (c) base, or a combination of these depending on its 3D structure and the pushing direction, making it non-trivial to model explicitly.

3 Experience-based Push-Manipulation

Humans learn and further sharpen their manipulation and corresponding prediction-based planning skills by interacting with their environment and observing the outcomes. Ideally, robots should also learn from their experiences as opposed to the unscalable and inefficient approach of providing them with detailed analytical models of each and every object that they are expected to interact with, and physics engines to compute the outcomes of these interactions. In our case, due to the complexity of the potential interactions between the robot, the objects, and the floor surface as well as the resulting motion characteristics, it is neither trivial nor efficient to try to define such models manually. For these reasons, we let our robot interact with the pushable objects either through self-exploration or demonstration via joysticking to observe how they move in response to various pushes. These observations are then turned into *experimental models* to be used for planning and execution. In case of self learning, the properties of these interactions, such as the pushing directions and durations, are determined randomly. Our algorithm consists of the following components, which we explain in the rest of this section:

– A set of *experiences* represented as *sequences* composed of the robot's motion commands, its resulting active trajectory, and the object's corresponding observed passive trajectory,
– A *generative planner* that makes use of these past pushing experiences as building blocks to construct achievable and collision-free push plans,
– An *execution monitoring module* to stop execution and trigger re-planning whenever there is a significant discrepancy between the expected and the actual motion of the object during plan execution.

3.1 Sequences

Each individual interaction of the robot with the objects is stored as *sequences* of pose-action pairs for the robot and the corresponding poses for the object of interest, representing their active and passive trajectories, respectively. These trajectories are defined with respect to various frames of reference. A static global frame of reference, φ_G, is attached to the environment. We also attach separate frames of reference to the robot and the object of interest, denoted as φ_R and φ_O, respectively, to define their poses within φ_G. In addition, we define an auxiliary frame of reference, φ_S, to indicate the last stationary pose of the object before it starts being pushed. Fig. 3(a) illustrates these reference frames.

Let $\wp_R$ be φ_R w.r.t. φ_O, and $\wp_O$ be φ_O w.r.t. φ_S, both of which are denoted as $\langle x, y, \theta \rangle$. Invariance to φ_O is achieved by recording $\wp_R$ together with the motion command at that moment and the corresponding $\wp_O$. Therefore, a sequence S_i of length n takes the form

$$S_i : ((\wp_{R_0}, a_0, \wp_{O_0}), \ldots, (\wp_{R_{n-1}}, a_{n-1}, \wp_{O_{n-1}}))$$

where a_j is the action associated with $\wp_{Rj}$, denoted as $\langle v_x, v_y, v_\theta \rangle$ indicating the (in our case omni-directional) motion command composed of the translational and rotational velocities of the robot. Fig. 3(b) provides the visualization of the robot and object trajectories within the stored sequences. The transparent, scaled-down robot figures indicate the push initiation poses (i.e. $\wp_{R_0}$ of each sequence) whereas the scaled-down object figures indicate the mean observed poses of the object after the pushes (i.e. mean $\wp_{O_{n-1}}$ of each sequence). The robot trajectory (indicated by green curves) and the object trajectory (indicated by red curves) that belong to the same sequence are marked with the same ID value. Final object pose uncertainty is depicted with the yellow ellipses drawn around the mean final poses. The process of obtaining the projected final object pose distributions is elaborated in Section 3.2.

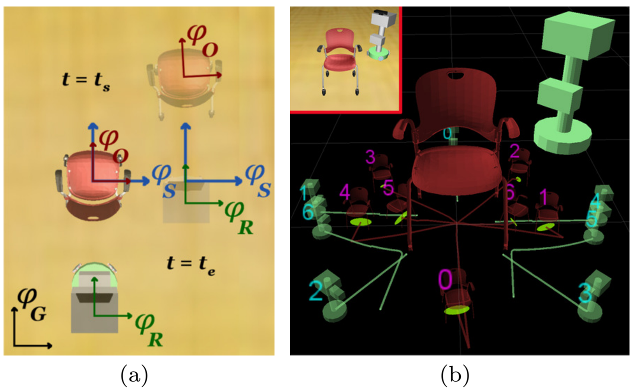


Fig. 3 (a) Various reference frames used during sequence recording and reiteration depicted before ($t = t_s$) and after ($t = t_e$) a push. (b) Visualization that corresponds to the scene shown in the upper left corner of the image. The robot trajectory and the corresponding object trajectory components of 7 different sequences are illustrated.

These sequences are recorded at each step of the robot's perception cycle, in our case at a frequency of 30Hz. In order to improve the efficiency of collision checking along the robot's and the object's trajectories, we define *keyframes* at every k^{th} frame of the sequence and perform collision checking only for the keyframes. The value of k can be adjusted according to the dimensions of the object being pushed; that is, the smaller the object, the better to check collisions more frequently along the active and passive trajectories.

3.2 Building Experimental Models

There are two challenges to be addressed during the experimental model building process.

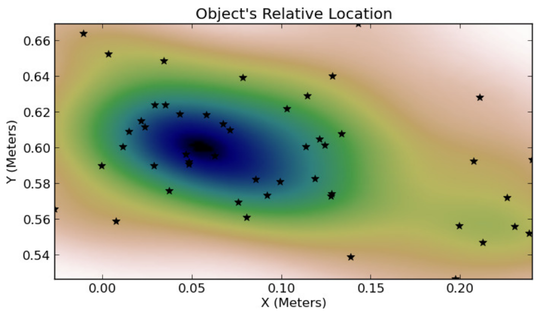
1. The first challenge is the uncontrolled motion of the object after the push is ceased. As a result of moving on passively-rolling caster wheels, the pushable objects used in our experiments do not stop immediately after the robot stops pushing, and the exact poses of the objects after they come to rest vary even between the pushing attempts from the same direction for the same duration.
2. The second challenge is the effect of the initial stationary orientations of the object's caster wheels on the trajectory that the object follows while being pushed. The wheels do not immediately align with the pushing direction after the push starts, which introduces additional uncertainty to the motion of the object and its final observed pose.

We address these partly interrelated problems simultaneously by having the robot build its experience *incrementally* over several trials instead of relying on a single observation.

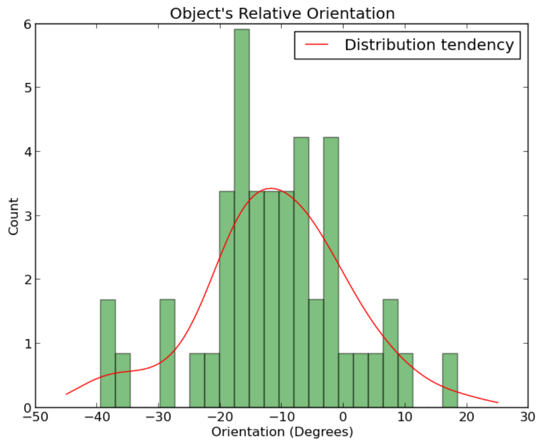
When the robot is asked to acquire experience about a given pushable object through self-exploration, it determines m random push initiation locations immediately around the object together with the corresponding random pushing durations ranging from 1 to 3 seconds. We name these tuples *push configurations*, $\varsigma = \{\varsigma_0, \dots, \varsigma_{m-1}\}$, $\varsigma_i : (\wp_{R_0}, t)$, which are used to carry out the first push trials on the object. Each ς_i represents a simple linear push performed while moving with constant velocity. On the other hand, we can also demonstrate the robot more sophisticated and informative pushing motions for the given objects via the use of a joystick. Regardless of the method that the robot uses to acquire its object related experience, a new sequence is created and stored whenever the robot tries a particular push for the first time. The additional trials are merely reiterating of these newly learned sequences to update the parameters of the distributions associated with each of them to represent the uncertainty in the observed final pose of the relevant object after a push.

Fig. 4 illustrates the visualization of the actual observed relative final object pose data recorded during the execution of one of the several learned sequences for the chair object. As an attempt to capture this motion uncertainty caused by the caster wheels, the robot experiments with each S_i^{new} of the newly gathered set of sequences S^{new} for varying initial wheel orientations. If there are more than two newly learned sequences, then the robot iterates over S^{new} by using a set of increments $\iota = \{\iota_0 = 1, \dots, \iota_j = |S^{new}| - 1\}$ in a way similar to a hash collision resolution strategy. Starting with $\iota = \iota_0$, the robot alternates between S_i^{new} using $i = ((i + \iota) \mod |S^{new}|)$ until each of them are covered. Then it keeps picking other increments ι_l with $l = ((l+1) \mod (j+1))$ and continues its experimentation until n samples from each of the S_i^{new} are collected. If there are only one or two newly learned sequences, then the robot either replays some of the other already learned sequences or executes random pushing motions to change the initial orientations of the wheels.

Based on the visualizations of the actual data as shown in Fig. 4, we decided to approximate these distributions with 3-dimensional Gaussians for the sake of simplicity. During the collection of these n samples for each S_i^{new}, the corresponding distribution parameters are incrementally updated according to Eq. (1) and Eq. (2), assuming that the observed final object poses will be normally distributed.



(a) Relative location



(b) Relative orientation

Fig. 4 Actual distribution of the relative final pose data of one of the sequences logged during push execution.

$$\bar{\wp}_{O_t^i} = \bar{\wp}_{O_{t-1}^i} + \frac{\wp_{O_t^i} - \bar{\wp}_{O_{t-1}^i}}{t} \tag{1}$$

$$\Sigma_{\wp_{O_t^i}} = \frac{(t-1)\Sigma_{\wp_{O_{t-1}^i}} + (\wp_{O_t^i} - \bar{\wp}_{O_t^i})(\wp_{O_t^i} - \bar{\wp}_{O_{t-1}^i})^T}{t} \tag{2}$$

In these equations, $\bar{\wp}_{O_t^i}$ denotes the mean of the observed final object pose after the t^{th} trial for a specific S_i^{new}, and $\Sigma_{\wp_{O_t^i}}$ is the corresponding covariance, which in our case is a 3×3 matrix as we are dealing with 3 DoF poses in the form of $\langle x, y, \theta \rangle$. This compact representation eliminates the need for storing all of the previously observed individual poses.

These distributions are also good indicators of how reliable and consistent individual push sequences are. Since the object moves in an uncontrolled manner after the pushing is ceased, we do not want it to end up in an unforeseen pose which may happen to collide with the obstacles in the environment, or cause the next pushing motion in the plan to be unachievable due to the obstruction of the corresponding push initiation pose.

Therefore, we eliminate the sequences with variances exceeding predefined thresholds to improve the safety and reliability of the plans generated using these sequences, eliminating potential failures and reducing the number of re-plans needed along the way during plan execution.

3.3 Reactive Push-Manipulation

One possible way of using the experimentally acquired models for push-manipulation is to plan a collision-free guideline path for the object without worrying about whether it can actually be followed using the available sequences, and then reiterating the ones that result in the *best locally-matching* non-colliding object trajectories until the goal is reached. First of all, the sequences with obstructed robot trajectories are filtered out as those would not be possible to reiterate. As the final placement orientation is important in our problem, the next check is performed to see if the current orientation of the object differs from the goal orientation by an amount greater than the allowed tolerance. If that is the case and the local environment of the object has enough space for the required maneuver, then the sequence, the execution of which will reduce the orientation difference the most, is selected for execution. If, on the other hand, the orientation difference is still acceptable, this time *cosine similarities* between the directions of each of the non-colliding object trajectories (τ_i) and the direction of the next waypoint on the guideline path ($\mathcal{G}$) are computed as shown in Equation (3). The sequence that results in the object trajectory with the greatest similarity to the guideline is selected for execution.

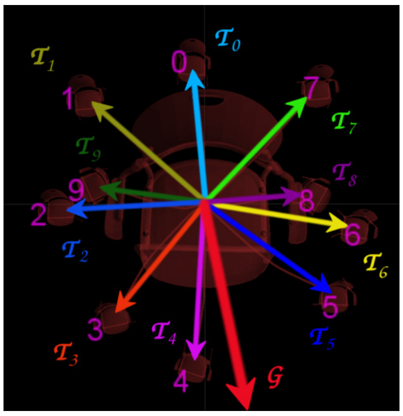


Fig. 5 The anticipated object trajectories are approximated with vectors (marked with τ_i) and compared against the vector representing the desired moving direction (the red vector marked with $\mathcal{G}$) using cosine similarity. τ_4 is the most similar direction to the desired one in this particular figure.

$$sim(\tau_i, \mathcal{G}) = \frac{\tau_i \cdot \mathcal{G}}{||\tau_i||\,||\mathcal{G}||} \tag{3}$$

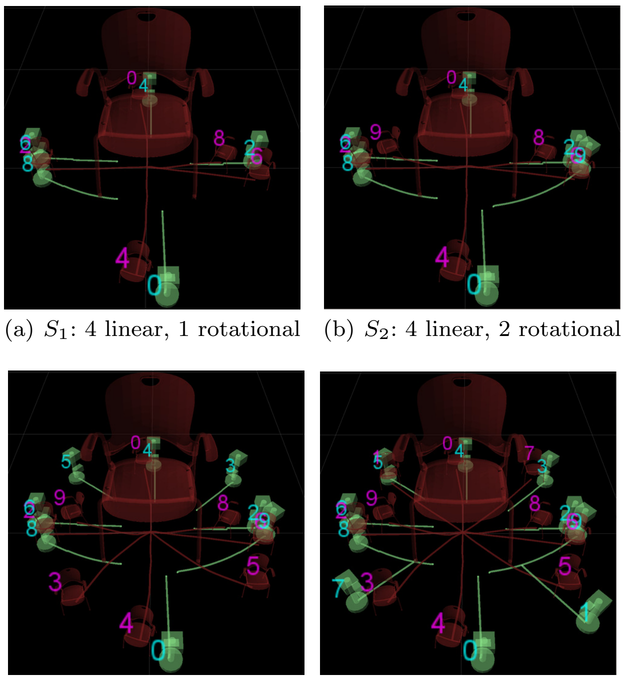
To better visualize the concept, Fig. 5 shows the bird's-eye view of the object trajectory components of a set of sequences learned for a chair together with the direction of the next waypoint along the guideline path generated at the very beginning of the process. The direction vectors are depicted as arrows superimposed on the anticipated and desired trajectories. Trajectories with IDs 8 and 9 are the results of the rotational movements of the robot; hence, they are usually more suitable for reducing the orientation difference between current pose and the goal pose of the object.

The sequence selected based on its directional and rotational similarity to the desired intermediate state may result in collision of the object with the environment when executed all the way to the end. It is important to keep in mind that the elegance of push-manipulation comes with the potential danger of *irreversibility*; that is, the robot may push the object to such an inconvenient configuration that it may not be able to recover. Considering these undesired consequences, it becomes very important to be able to control the amount of movement so that the object is pushed only so much that it does not collide with anything, and ends up in a state that is as close to the desired one as possible. Therefore, in addition to the directional and rotational similarity checks, the robot also tries to find the best matching and collision-free projected pose (i.e. keyframe) of the object along the trajectory and stops pushing right at the corresponding moment.

3.3.1 Experimental Evaluation

The Webots simulation environment (Michel, 2004) enabled us to realistically simulate the pushable real world objects and their motions on passively-rolling caster wheels. Even though we used identical caster wheels for all the pushable objects, their actual physical structures, weight distributions, and the varying wheel placements cause them to have distinct motion characteristics. For simulating the 2-axes rotation of the caster wheels, we set the Coulomb friction coefficient for the wheel axis as 0.1 and for the fork axis that rotates the wheel vertically as 1.0.

In the simulated setup, we first demonstrated a total of 10 pushing sequences to the robot; four linear pushes from the four main directions, four diagonal pushes, and two rotational pushes from either side of the object as shown in Fig. 6(d). The final placement of an object was considered successful if the distance to the desired



(a) S_1: 4 linear, 1 rotational (b) S_2: 4 linear, 2 rotational

(c) S_3: 4 linear, 2 diagonal, 2 rotational (d) S_4: 4 linear, 4 diagonal, 2 rotational

Fig. 6 The four different sets of sequences used in the experiments. Among these sets, S_1 is the most challenging one as it has only one rotational pushing sequence in addition to the four linear ones.

Table 1 Performances of different sets of motion sequences.

Sequence set	Push count	μ_{dist} (cm)	$\mu_{\triangle\theta}$ (deg)
S_1	25	14.11	26.12
S_2	8	14.82	14.06
S_3	7	14.87	23.52
S_4	7	14.86	20.49

goal location was below $0.15m$, and the orientation difference was below $\pi/6$ radians. In the experiments performed with the chair, the robot was able to place the object to its desired pose safely 6 out of 10 times. [1]

We also investigated the effects that the number and types of the available pushing sequences have on the overall success of the approach. Fig. 6 shows the four different sets of the motion sequences that the robot was allowed to use. In a sample setup, we placed the chair rotated $\pi/2$ radians counterclockwise 3 meters away from the the desired goal configuration so that the robot would need to perform rotational as well as translational manipulation moves in order to bring the chair to its desired pose. As we utilized a randomized generative planner to construct the guideline path, we repeated our experiments 10 times with each of the four sets of sequences. The average distances and orientation differences obtained with each of these sets as well as the number of pushes required to obtain these results are summarized in Table 1.

Among these sets, S_1 only had a counterclockwise rotational pushing sequence in addition to four linear pushing sequences; therefore, it took a lot more pushes for the robot to bring the chair to the desired orientation by using the available sequences. Looking at Table 1, the general tendency seems to be towards a decreasing number of pushes as the translational and rotational variety of the available sequences grows. In some challenging setups, we observed insufficient variety of sequences resulting in situations where the robot got stuck due to either potential collisions, or having all the push initiation poses obstructed. Those were the main reasons of the robot failing to safely transport the object to its desired pose 4 out of 10 times in the aforementioned experiments. The success rate could be increased by increasing the number and the variety of sequences. However, the best results are obtained when achievability is taken into account during planning, which is explained in the following section.

3.4 Achievable Push-Manipulation

The sequences encapsulate information about how the pushable objects move in response to various pushes; therefore, instead of trying to track paths that are generated in an uninformed manner, the learned sequences can be utilized for constructing informed plans that are collision-free and achievable for both the object and the robot. Due to its simplicity, practicality, and probabilistic completeness property, we decided to base our planning algorithm on a Rapidly-exploring Random Trees (RRT) (LaValle, 1998, 2006) approach. Starting from the initial configuration, the RRT algorithm incrementally builds a tree by uniformly sampling points from the state space and growing the tree in the direction of the sample by extending the closest node of the tree towards the sample along a straight line. It is also possible to bias the tree growth towards the goal and reuse past executions (Bruce and Veloso, 2002).

Our contributed planning algorithm modifies the original RRT algorithm and uses the previously observed object trajectories as building blocks for extending the tree towards the sample. In other words, we build the tree out of the memorized object trajectories that can be regenerated by the robot without neither the robot's nor the object's projected poses being in collision with the obstacles. This is the key point in ensur-

[1] A video showing the robot performing experience-based reactive push-manipulation in simulation can be seen here: `http://youtu.be/1Z8KW7fPGrA`

Algorithm 1 The Exp-RRT algorithm.

```
1: function BuildExpRRT($ooi, q_{init}, q_{goal}$)
2:     $Tree \leftarrow q_{init}$; $q_{new} \leftarrow q_{init}$;
3:     while $sim(q_{new}, q_{goal}) < THRESHOLD$ do
4:         $q_{rand} \leftarrow Sample()$;
5:         $q_{most\_similar} \leftarrow MostSimilar(Tree, q_{rand})$;
6:         $q_{new} \leftarrow Extend(ooi, q_{most\_similar}, q_{rand})$;
7:         $Tree.add(q_{new})$;
8:     end while
9:     return $Trajectory(Tree, q_{new})$;
10: end function
11: function MostSimilar($Tree, q_{target}$)
12:     return $\arg\max_{q \in Tree} sim(q, q_{target})$;
13: end function
14: function Extend($ooi, q_{source}, q_{target}$)
15:     $S_{trans} \leftarrow \{Transform(S_i, q_{source})\} \forall S_i \in ooi.S$;
16:     $S_{safe} \leftarrow \{S_{trans} \setminus \{Colliding(S_t)\}\} \forall S_t \in S_{trans}$;
17:     return $\arg\max_{S_i.q^f \forall S_i \in S_{safe}} sim(S_i.q^f, q_{target})$;
18: end function
```

ing *achievability* from both the robot's and the object's perspective; that is, we cannot guarantee a straight line extension towards the sample to be achievable with the available sequences, but we can indeed guarantee that an extension made with the most suitable non-colliding sequence is achievable, as the robot has already experienced that particular object motion. The pseudo-code of our contributed experience-based RRT (Exp-RRT) algorithm is given in Algorithm 1.

At each iteration, we sample a random pose with probability p or use the goal as the sample with probability $1-p$. The closest node of the tree to the new sample is the one that gives the maximum similarity value according to the similarity function defined in Eq. 4,

$$sim(p_1, p_2) = \frac{d_{max}}{dist(p_1, p_2)} cos(p_1.\theta - p_2.\theta) \tag{4}$$

where d_{max} is the maximum possible distance that can be obtained in the task environment and $dist(p_1, p_2)$ is the Euclidean distance between the locations of the poses. Therefore, the closer the locations of the two poses and the smaller the angular difference between their orientations, the more similar they are. After the closest node to the sample is determined, imagining the object to be on the pose of the closest node, this time the final expected poses of the sequences originating from that imaginary pose are checked against the sample according the similarity function defined in Eq. 4.

The tree is extended towards the sample by using the final projected object pose of the sequence that gives the highest similarity value and is collision-free for both the object and the robot. This process is repeated until the pose of the newly added node falls within predefined distance and orientation difference limits to the goal pose. Fig. 7 illustrates two steps of the tree construction process, assuming that the goal itself is used as the sample to be reached. Object trajectories within the sequences are illustrated as dashed curves and the projected final object poses are depicted as little squares.

During tree construction, the achievability of a sequence is determined by checking each keyframe for collisions along the robot and object trajectories within the sequence. Additionally, in order to incorporate the motion uncertainty of the object into the planning process, collision check for the final expected object pose is performed using the associated distribution rather than a single pose. Taking uncertainty into account during RRT planning has been studied in the literature (Melchior and Simmons, 2007; Berg et al, 2010), however, we do it for the manipulated objects instead of the robot itself in addition to performing it in a novel way. We use Eq. (5)-(7) to derive $2L+1$ sigma points for each sequence, which represent the extreme points of the associated distributions. L is the dimensionality of the state space, which, in our case is $L = 3$ as we are dealing with 3 DoF poses.

$$\chi_0 = \bar{\wp}_{O^f} \tag{5}$$

$$\chi_i = \bar{\wp}_{O^f} + \zeta(\sqrt{\Sigma_{\wp_{O_n^f}}})_i, i = 1, \ldots, L \tag{6}$$

$$\chi_i = \bar{\wp}_{O^f} - \zeta(\sqrt{\Sigma_{\wp_{O_n^f}}})_i, i = L+1, \ldots, 2L \tag{7}$$

In these equations, $\bar{\wp}_{O^f}$ is the mean of the final object poses observed so far for a particular sequence, $(\sqrt{\Sigma_{\wp_{O_n^f}}})_i$ is the i^{th} column of the matrix-square-root of the covariance matrix $\Sigma_{\wp_{O_n^f}}$, and ζ is the scalar scaling factor that determines the spread of the sigma points around $\bar{\wp}_{O^f}$. Increasing ζ increases the conservativeness of the planner. In our experiments, we used $\zeta = 3$. Each of these extreme poses are checked for collision and the sequence is marked as *unachievable* and excluded from the set of sequences to be considered for extending the tree in case any of these poses are in collision with the objects in the environment. A separate regular RRT planner is used for planning a collision free path for the robot that will take it to the starting pose $\wp_{R_0}$ of the selected pushing sequences during plan execution.

3.4.1 Plan Execution and Monitoring

The constructed plan is executed by reiterating one after another the robot trajectories of the chain of sequences that transports the object to the desired pose. Even though the plan is constructed by taking into account the uncertainties in the expected final object poses, the object inevitably digresses from its foreseen

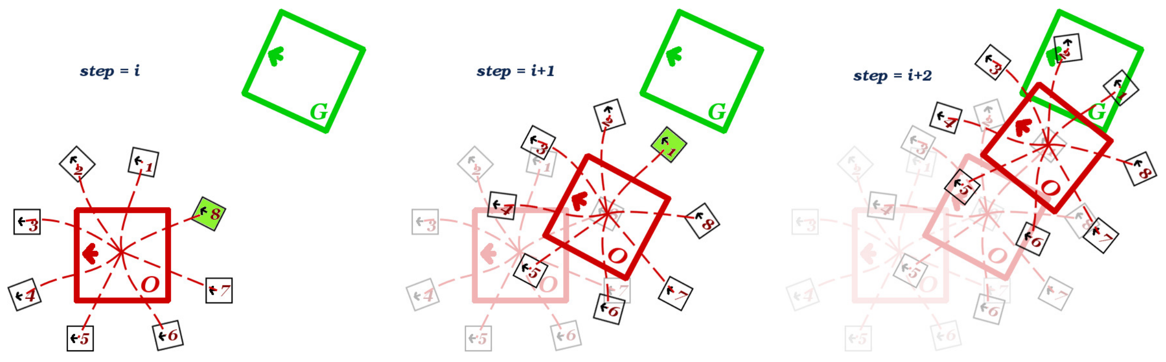


Fig. 7 Illustration of the Exp-RRT construction process. The sequences resulting in the most similar poses are highlighted.

path, especially when it needs to be transported for longer distances. During plan execution, re-planning may be triggered depending on whether the actual observed final pose of the object after a push falls within the *tolerance region* of the expected pose distribution, which is computed using Eq. (8)

$$(\wp_{O_o} - \bar{\wp}_O)^T \Sigma_{\wp_O}^{-1} (\wp_{O_o} - \bar{\wp}_O) \leq \chi_k^2(p) \quad (8)$$

where $\wp_{O_o}$ is the observed final pose of the object, $\bar{\wp}_O$ is the expected final pose, $\Sigma_{\wp_O}$ is the expected final pose covariance, and $\chi_k^2(p)$ is the quantile function for probability p of the chi-squared distribution with k degrees of freedom. In our case $k = 3$ and we use $p = 0.05$ to make sure that the observation is statistically significantly different from the expectation for the robot to trigger re-planning.

Additionally, in order to relax the planning process a bit, we design a heuristic that dynamically alters the desired final pose accuracy depending on the distance of the object from the goal. This heuristic is defined in Eq. (9)-(10) as;

$$\delta = (dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \delta_{max} \quad (9)$$

$$\omega = \pi(dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \omega_{max} \quad (10)$$

where $\wp_{O_g}$ is the goal pose, δ and ω are the distance and orientation difference thresholds, respectively, δ_{max} and ω_{max} are the maximum allowed final distance and orientation difference thresholds, respectively. This heuristic helps the robot plan a "rough" solution quickly when the object is far from the goal, and enforces more accurate planning at each re-planning attempt as the object gets closer to the goal.

3.4.2 Experimental Evaluation

We performed majority of our achievable push planning and manipulation experiments again in Webots. The final placement of an object was considered successful if the distance of the object to the desired goal was below $0.2m$ and the orientation difference was below $\pi/9$ radians. Considering the dimensions of the objects that our robot is expected to manipulate, such as a $0.8m \times 0.45m$ serving tray and a $1.9m \times 0.9m$ stretcher, these constraints are quite tight.

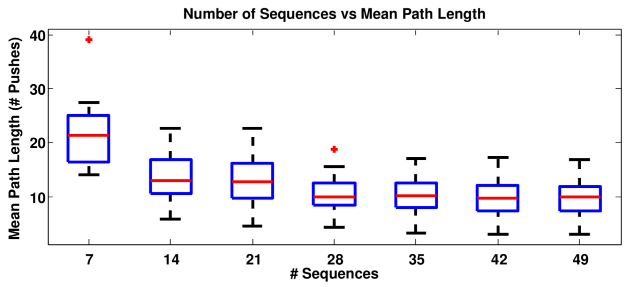
As briefly mentioned in Section 3.2, the first step in our experiments is to select the *reliable* set of sequences to be used for planning. We do that by eliminating the ones that cannot be reiterated consistently; that is, the ones that have high variance in the final observed object pose. We determined the maximum allowed position and orientation variances to have the same values as δ_{max} and ω_{max}. After this elimination, we evaluate the remaining set of sequences for their proficiency on generating solutions for randomly picked goals to see how good these solutions are in terms of the path length (i.e. the lowest the number of pushes required to transport the object, the better the solution is) and having consistently similar path lengths. In its evaluation mode, the robot picks a number of random, collision-free goals, and starts evaluating the proficiency of the available sequences by adding them to its library of sequences one batch at a time, and checking the number of goals that can be reached consistently with the available sequences. When more than a certain percentage of the random goals start to be reached, the robot stops adding new sequences to its library. It is always possible for the robot to learn some additional sequences for an object in case it encounters a problem that it cannot solve with the currently available set of sequences. On the other hand, it is also important to keep the number of stored sequences per object as low as possible due to storage and processing efficiency concerns.

The batch size used in our experiments was $m = 7$. During the evaluation and the actual planning processes, we consider a planning attempt unsuccessful if the total number of RRT nodes allowed is exceeded.

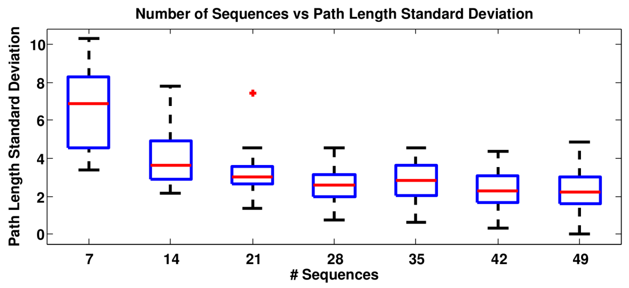
In our experiments, we determined the maximum number of Exp-RRT nodes to be 50625 as we require 0.2m distance accuracy with at most $+/-\pi/9$ radians orientation difference in a $15m \times 15m$ environment.

Fig. 8 illustrates the results obtained by following the sequence library evaluation procedure for the pushable chair object and 20 randomly picked collision-free goals. Since the robot is essentially using a random sampling-based planner, each planning attempt was repeated 10 times for each of the 20 goals, so that that we can analyze the planning performance more reliably. Fig. 8(a) shows how the mean path length computed over all 20 goals changes with the changing number and variety of the sequences in our library. It can easily be seen from the figure that the mean path length decreases with the increasing number of available sequences for a while and then settles around a certain mean path length value. Fig. 8(b) shows how the standard deviation of the mean path length changes with the increasing number of available sequences, which is a measure of how consistent the solutions are in terms of path length. Similarly, we can see from the figure that the robot starts finding solutions that have consistently lower path lengths as the variety of the available sequences increases. These figures are good indicators for the robot to understand when it has learned enough variety of sequences to solve a decent number of push manipulation problems for a specific object. Here we provide only the plots obtained for the chair object as the experiments with the other pushable objects in our inventory resulted in similar plots.

Separate sets of sequences are acquired and stored for each of our pushable objects. Fig. 9 demonstrates the generated achievable and collision-free plans from initial (S) to goal (G) poses for five of those objects, namely a chair, an overbed table, a pushable serving tray, a stretcher, and a cart by using their corresponding sequences as building blocks. Among these objects, the cart is a particularly challenging one as only the two front wheels are caster wheels and the back ones are stationary, resembling the wheel configuration of a traditional shopping cart. This wheel configuration results in totally different motion characteristics. The object and wheel configuration independent nature of our method enables the robot to learn and make use of push sequences even for these kind of objects with different kinematic and dynamic properties. As it can be seen from these screenshots, our experiment environment is much bigger and much more cluttered compared to the problem setups used in many of the related studies surveyed in Section 2. Considering the long distances that the robot is expected to navigate the object for, it is inevitable to have the object digress from its foreseen



(a) Change of the mean path length computed over 20 goals with the increasing variety of sequences.



(b) Change of the standard deviation of the mean path length computed over 20 goals with the increasing variety of sequences.

Fig. 8 Evaluation results of the growing variety of the available sequences for the chair object.

path during plan execution and for the robot to re-plan in order to guarantee the safe transportation of the object. In these five particular instances shown in Fig. 9, the robot had to re-plan for 5.5 times on the average. All of the generated plans were successfully executed in simulation with no failures. [2]

It must be noted that the simulation environment is essentially a black box for the robot, as the real world would be, and the only information that the simulator provides to the robot is the poses of the objects. It is a black box for us as well, since we only provide the object meshes, mass values, and wheel friction coefficients, and let the ODE physics engine of the simulator take care of the inter-object interactions. The only motivation behind using a realistic 3D simulator is to obtain a setup that "looks" an "behaves" reasonably realistic as we are not concerned with transferring any knowledge from the simulated environment to the real world. In other words, both the internal and external parameters of the simulator are totally irrelevant to the operation of our method. Therefore, even if we had not set the physics parameters realistically, the method would still work and learn how to push-manipulate objects under

[2] A video showing the simulated mobile manipulator successfully push-manipulating an overbed table can be seen here: `http://youtu.be/mw-PCjPaOXI`

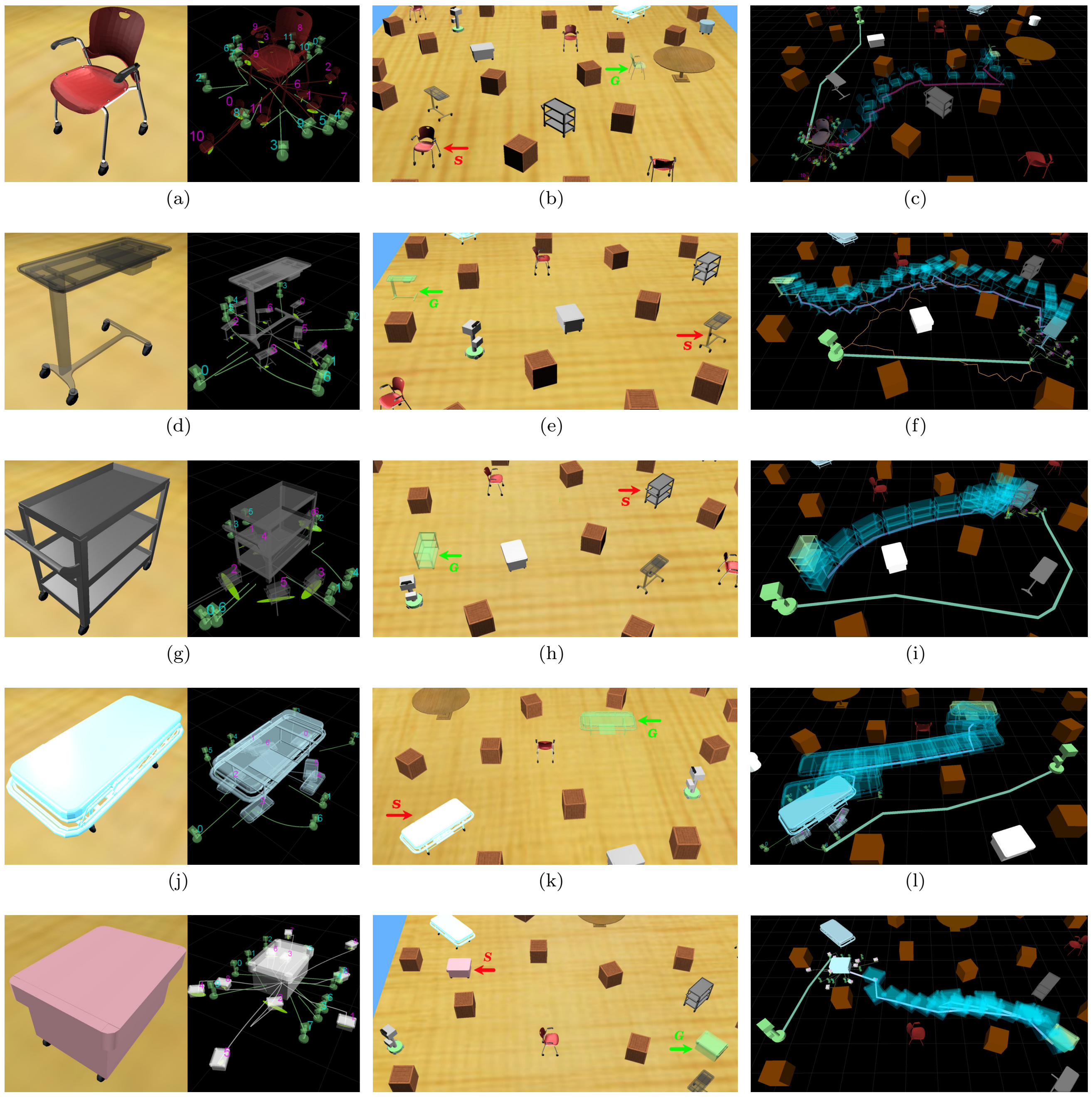


Fig. 9 Generated plans (shown as blue ghost figures over the corresponding path) using the past observed and memorized trajectories for various pushable objects, namely a chair ((a), (b), and (c)), an overbed table ((d), (e), and (f)), a pushable serving tray ((g), (h), and (i)), a stretcher ((j), (k), and (l)), and a cart ((m), (n), and (o)) in challenging environments cluttered with obstacles and other objects.

those circumstances. Our contributed method is able to handle any pushable object after the robot experiments with them to learn how they move in response to various pushes, and it is totally independent of the robot, the object, and the environment (simulated or real), as we verify in the following section.

3.5 Moving to the Real World

In addition to the detailed study we did in simulation, we also ran some preliminary tests in a physical setup to validate our contributed method in terms of its robot, object, and environment (i.e. real or simulated)

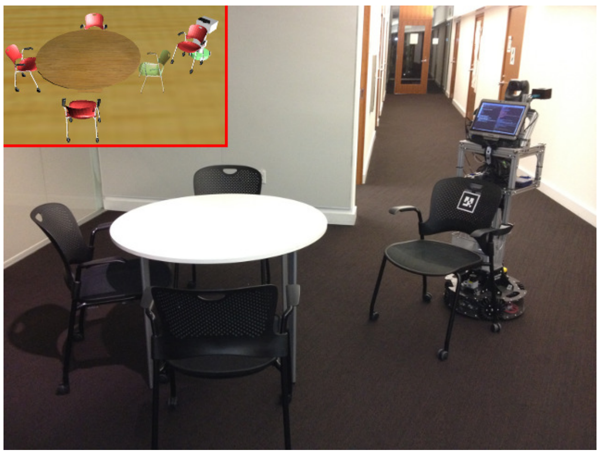


Fig. 10 A snapshot from one of the real world tests, where the task of the robot is to arrange the chairs around the round table. Visualization of the setup in simulation is provided on the top left corner of the image.

independence. In this test, our CoBot robot (Rosenthal et al, 2010) was asked to arrange a set of chairs in a predefined seating formation around a round table, some of which were already in place. Fig. 10 shows a snapshot from the physical setup in which we tested our proposed method.

There are a number of challenges that need to be addressed when switching from the simulated environment to the physical one. The first one is the construction of a stable world model. In simulation, we get the global pose information of all the objects in the environment directly from the simulator. However, in the physical setup, the robot's global pose information comes from the localization module (Biswas et al, 2011), which is noisier compared to the perfect information received in simulation. The pose of the chair is computed relative to the robot; hence, the calculated global pose of the chair is affected by the noise in the localization estimation of the robot. In order to make it easier to detect the chair visually, we placed Augmented Reality (AR) tags on both sides of the back of the chair (Fig. 10), which are visible most of the time from almost all directions. However, perception is not perfect either; therefore, additional noise comes from the perception of the AR tags. The second challenge is the maintenance of a reliable world model at all times. Since the Kinect sensor that we use as the primary visual sensing device is placed at a certain location on the robot with a certain angle to satisfy multiple requirements, and the field of view of the camera is limited, the AR tags cannot be seen anymore when the robot gets very close to the object to push it. Those cases need to be covered by a good tracker so that the robot can still have an idea of where the object is even if it is not visible within the robot's field of view.

During our preliminary tests [3], the robot was, in general, able to construct a stable world model by combining its perception with its localization information to generate and execute push-manipulation plans. Even though we have not performed detailed experiments in this setup, we observed that there was an overall increase in the frequency of re-planning due to the increased uncertainty in both perception and action in real world.

These preliminary real world tests run using the exact same code base verified the validity of our method and demonstrated its robot, object, and environment (simulated and real) independence as the only pieces of information needed were the robot's localization belief and the pose of the object inferred from the robot's own detection, processing, and transformation of the AR tags associated with the object of interest.

4 Conclusion and Future Work

Push-manipulation is one of the most interesting and challenging robotic manipulation modalities that has attracted many researchers. However, many of the proposed methods handle flat objects with primitive geometric shapes moving quasi-statically on high-friction surfaces, yet they usually make use of complex analytical models or utilize specialized physics engines to predict the outcomes of various interactions. On the other hand, we propose an experience-based approach, which does not require any explicit analytical model or the help of a physics engine. Our mobile robot simply experiments with pushable complex 3D real world objects to observe and memorize their motion characteristics together with the associated motion uncertainties resulting from varying initial caster wheel orientations and potential contacts between the robot and the object. It then uses this incrementally built experience either for trying to make the object of interest follow a guideline path by reiterating in a reactive manner the sequences that result in the best locally-matching outcomes, or as building blocks of a sampling based planner we contribute, the Exp-RRT, to construct push plans that are safe and achievable. In contrast to the proposed approaches in the literature, in our contribution;

- we handle real world objects with complex 3D structures that may contact the robot on more than one point,

[3] A video showing the physical CoBot acquiring and using push-manipulation cases in a real world setup can be seen here: `http://youtu.be/TORQdBPHJ3g`

– the manipulated objects move on passively-rolling caster wheels and do not stop immediately after the pushing is ceased,
– the experiment environment is cluttered with obstacles; hence, both collision-free and achievable plans should be constructed and manipulation should be performed delicately,
– we do not use any explicit analytical models or learn a mapping between the trajectories of the robot and the object; we only utilize the experimented and observed effects of the past pushing motions to anticipate the future, plan, and act accordingly.

We extensively tested our method in a realistic 3D simulation environment where a variety of pushable objects with passively-rolling caster wheels needed to be navigated among obstacles to reach their desired final poses. We also performed some preliminary tests in a physical setup to verify the validity of our method. Our experiments demonstrate safe transportation and successful placement of several pushable objects in simulation and promising results for push-manipulation tasks in real world, such as arranging chairs in predefined seating formations in a study area.

Future work includes;

– extensive testing and detailed experimentation in the physical setup,
– repairing only the problematic parts of the generated plans and reusing them instead of complete re-planning,
– active learning of the sequences as needed by asking for additional demonstrations,
– performing subset selection among the reliable sequences to find the minimum set of useful ones,
– expanding the skill set of the robot by accumulating new experiences over time,
– transferring learned manipulation sequences among objects with similar properties.

Acknowledgements We are grateful to Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal for their work with the CoBot's autonomy and task planning. We further thank Çetin Meriçli for his comments on this work and his help with the experiments with CoBot.

References

Agarwal PK, Latombe J, Motwani R, Raghavan P (1997) Nonholonomic path planning for pushing a disk among obstacles. In: Proceedings of ICRA

Berg JVD, Abbeel P, Goldberg K (2010) LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. In: Proceedings of Robotics: Science and Systems, Zaragoza, Spain

de Berg M, Gerrits D (2010) Computing Push Plans for Disk-Shaped Robots. In: Proceedings of ICRA

Biswas J, Coltin B, Veloso M (2011) Corrective Gradient Refinement for Mobile Robot Localization. In: Proceedings of IROS

Bruce J, Veloso M (2002) Real-Time Randomized Path Planning for Robot Navigation. In: Proc. of IROS

Dogar M, Srinivasa S (2012) A Planning Framework for Non-Prehensile Manipulation under Clutter and Uncertainty. Autonomous Robots 33(3):217–236

Igarashi T, Kamiyama Y, Inami M (2010) A Dipole Field for Object Delivery by Pushing on a Flat Surface. In: Proceedings of ICRA

K M Lynch and M T Mason (1997) Dynamic nonprehensile manipulation: Controllability, planning, and experiments. International Journal of Robotics Research 18:64–92

Katz D, Brock O (2008) Manipulating Articulated Objects with Interactive Perception. In: Proceedings of the IEEE International Conference on Robotics and Automation 2008, Pasadena, CA, pp 272–277

Khatib O (1986) Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. The International Journal of Robotics Research 5(1):90–98

Kopicki M, Zurek S, Stolkin R, Mörwald T, Wyatt J (2011) Learning to predict how rigid objects behave under simple manipulation. In: Proc. of ICRA

Lau M, Mitani J, Igarashi T (2011) Automatic Learning of Pushing Strategy for Delivery of Irregular-Shaped Objects. In: Proc. of ICRA

LaValle SM (1998) Rapidly-Exploring Random Trees: A New Tool for Path Planning. Tech. rep., Computer Science Dept., Iowa State University

LaValle SM (2006) Planning Algorithms. Cambridge University Press

Lynch KM (1996) Nonprehensile Robotic Manipulation: Controlability and Planning. PhD thesis, Robotics Institute, Carnegie Mellon University

Melchior N, Simmons R (2007) Particle RRT for Path Planning with Uncertainty. In: 2007 IEEE International Conference on Robotics and Automation, pp 1617–1624

Meriçli T, Veloso M, Akın HL (2012) Experience Guided Achievable Push Plan Generation for Passive Mobile Objects. In: Beyond Robot Grasping - Modern Approaches for Dynamic Manipulation, IROS'12, Algarve, Portugal

Meriçli T, Veloso M, Akın HL (2013) Achievable Push-Manipulation for Complex Passive Mobile Objects using Past Experience. In: 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Saint Paul, Minnesota, USA

Michel O (2004) Webots: Professional Mobile Robot Simulation. Journal of Advanced Robotics Systems 1(1):39–42

Nieuwenhuisen D, van der Stappen A, Overmars M (2005) Path Planning for Pushing a Disk using Compliance. In: Proceedings of IROS

Nieuwenhuisen D, van der Stappen A, Overmars MH (2006) Pushing Using Compliance. In: Proceedings of ICRA

Rosenthal S, Biswas J, Veloso M (2010) An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In: Proc. of AAMAS

Salganicoff M, Metta G, Oddera A, Sandini G (1993) A Vision-Based Learning Method for Pushing Manipulation. In: AAAI Fall Symposium on Machine Learning in Vision: What Why and How?

Scholz J, Stilman M (2010) Combining motion planning and optimization for flexible robot manipulation. In: Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on, pp 80–85, DOI 10.1109/ICHR.2010.5686849

Veloso MM (1994) Planning and Learning by Analogical Reasoning. Springer Verlag

Walker S, Salisbury JK (2008) Pushing Using Learned Manipulation Maps. In: Proceedings of ICRA

Zito C, Stolkin R, Kopicki M, Wyatt J (2012) Two-level RRT Planning for Robotic Push Manipulation. In: Proceedings of IROS