Achievable Push-Manipulation for Complex Passive Mobile Objects using Past Experience

Tekin Meriçli
Department of Computer Engineering
Boğaziçi University
Istanbul, Turkey
tekin.mericli@boun.edu.tr

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
veloso@cmu.edu

H. Levent Akın
Department of Computer Engineering
Boğaziçi University
Istanbul, Turkey
akin@boun.edu.tr

ABSTRACT

The majority of the methods proposed for the problem of push-manipulation planning and execution deal with objects that have quasi-static properties and primitive geometric shapes, yet they usually use complex physics modeling for the manipulated objects as well as the manipulator. We propose an experience-based approach, where the mobile robot experiments with pushable complex real world objects to observe and memorize their motion characteristics together with the associated uncertainty in response to its various pushing actions. Our approach uses this incrementally-built experience to construct push plans based solely on the objects' predicted future trajectories without a need for object-specific physics or contact modeling. We modify the RRT algorithm in such a way to use the recalled robot and object trajectories as building blocks to generate achievable and collision-free push plans that reliably transport the object to a desired 3 DoF pose. We test our method in a realistic 3D simulation environment as well as in a real-world setup, where a variety of pushable objects with freely rolling caster wheels need to be navigated among obstacles to reach their desired final poses. Our experiments demonstrate safe transportation and successful placement of the objects.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Robotics

Keywords

Mobile push-manipulation planning

1. INTRODUCTION

There are many ways to perform robotic manipulation, which are determined by the requirements of the task and the constraints imposed by the physical properties of both the object and the robot. *Prehensile* manipulation is the most straightforward one, where the object is first grasped and then carried to the desired destination. On the other hand, it may be possible, or even necessary to relocate the object without grasping it first in cases where either the object is too large or heavy, the robot is not equipped with a manipulator arm, or the utilization of some properties of the object makes its transportation more efficient and convenient that way. This type of manipulation is referred to as *non-prehensile* manipulation [12], examples of which include sliding, rolling, throwing, and *pushing*.

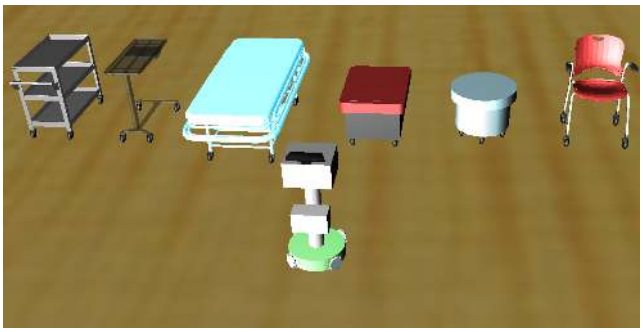


Figure 1: Realistically simulated passive mobile objects on freely-rolling caster wheels and our omni-directional mobile robot used as the pusher.

This paper focuses on the problem of push manipulation, the objective of which 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. Our problem setup consists of an omni-directional mobile robot without a manipulator arm [16, 2], and a set of passive mobile objects that are scattered in an environment cluttered with obstacles. The robot is expected to push-manipulate these objects in such a way to transport them to their desired poses while avoiding collisions. What makes our problem particularly challenging is that, in our experiments, we use real world objects moving on freely rolling caster wheels and their realistically simulated models. Caster wheels make the objects keep moving for some time even after pushing is terminated and they introduce some uncertainty into the objects' motions. Objects with this kind of uncontrolled motion properties are inherently more difficult to manipulate compared to objects sliding quasi-statically on surfaces with high friction.

We would like our robot to be able to manipulate a variety of such objects with different physical characteristics. Some of these objects are illustrated in Fig. 1. Our omni-directional mobile robot shown in front of the objects has a basket sticking out. Even though this basket makes con-

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

tact with the objects most of the time during pushing, due to the robot's and the objects' complex 3D shapes, some other parts of the robot may also be involved in the contact based on the pushing direction. Therefore, trying to model potential contacts becomes complex and infeasible. Under these circumstances, the robot needs a way to generate *achievable*, collision-free plans and execute them *reliably* as it is supposed to perform push-manipulation in a large environment cluttered with several objects and obstacles.

As a potential solution to these problems, we contribute an algorithm that does not require any explicit mathematical models for either the objects or the robot. Instead, following a case-based planning approach [18], the observed effects of previously experimented and memorized pushing moves, represented as *sequences* of state-action pairs, are utilized to generate push plans for manipulating passive mobile objects. A sampling-based planning algorithm is taken as basis and modified in such a way to construct complete plans by using the safe and achievable object trajectories as building blocks. The constructed plan is then executed by letting the robot replay the pushing motions of the corresponding sequences one after another. Due to the noise in actuation during plan execution and imprecision in the repeatability of the past observed object trajectories, the object may diverge from its expected pose along the foreseen path. If the divergence is significant, then a new plan is constructed to prevent potential collisions of the object and the robot with the obstacles in the environment. Using this method yields more successful executions compared to planning that does not take the achievability of the plan into account from both the robot's and the object's perspectives. Also, the robot more consistently starts generating plans that require less number of pushes as the variety of the memorized sequences increases.

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 [12, 6], push-manipulation has attracted several robotics researchers. An early work by Salganicoff *et al.* [17] presents a very simple, one 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.* [1] 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 pusher can push the object 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.* [14, 15] 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. Berg and Gerrits [4] computationally improve this approach and present both a contact preserving and an unrestricted push planning method in which the the pusher can occasionally let go of the object. Similar to the potential field based motion planners [7], Igarashi *et al.* [5] 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.* [9] 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 [19]. Their approach is similar to 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, their objects have quasi-static properties and they ignore the final orientation, which further simplifies the problem. Zito *et al.* [21] 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 robot arm's reach, which is relatively small, as they are using a stationary tabletop 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 modeling. Kopicki *et al.* [8] 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. Another recent study by Dogar and Srinivasa [13] uses push-manipulation in a tabletop manipulation scenario as a way to reduce uncertainty prior to grasping by utilizing the funneling effect of pushing.

According to our survey of the literature, the most common scenario seems to be 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 mathematical models are used for contact modeling and motion estimation, or physics engines of simulators are utilized for these purposes. Our approach differs from many of these proposed ones in the sense that

- we deal with complex 3D real world objects (some of which are shown in Fig. 1) that may contact the complex shaped robot on several points,
- the manipulated objects are not moving quasi-statically; rather, 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 environment cluttered with obstacles, requiring construction of *safe* and *achievable* plans,
- no explicit mathematical model is constructed 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 used for planning and execution.

3. PLANNING THE FUTURE USING THE PAST

Our algorithm consists of the following components:

- A set of sequences composed of robot motions for pushing an object from various directions for varying durations, and the object's corresponding motions,
- A generative planner that makes use of the past pushing experiences stored as sequences to construct achievable and collision-free push plans,
- An execution monitoring module to trigger re-planning when there is a significant discrepancy between the expected and the actual outcomes of the manipulation actions during push-plan execution.

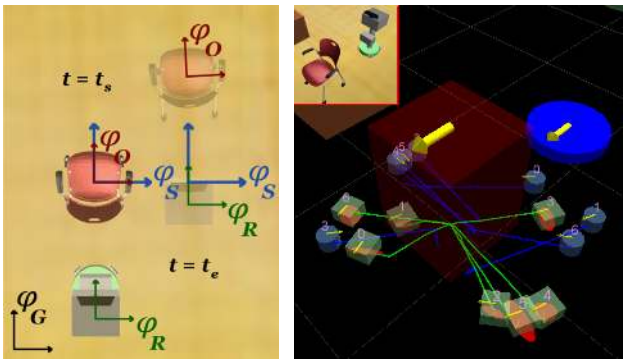
We elaborate on each of these components in the rest of this section.

3.1 Sequences

It is neither trivial nor efficient to try to manually define accurate mathematical models that will capture the complexity of the potential interactions between the robot, the objects, and the floor surface as well as the resulting motion characteristics in a push-manipulation scenario. Therefore, we make the robot *memorize* a small number of its pushing experiences from various directions as it interacts with the objects either through self-exploration or demonstration provided via joysticking the robot. These experiences are represented as *sequences* of pose-action pairs for the robot and the corresponding trajectory followed by the object. We make use of several frames of reference for constructing these sequences. A static global frame of reference, φ_G, is attached to the environment. The robot and the object of interest carry their own reference frames, denoted as φ_R and φ_O, respectively, that 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. 2(a) illustrates these frames of reference used for recording and executing the sequences. Invariance to the object's global pose (φ_O w.r.t. φ_G) is achieved by recording the pose of the robot relative to the object (φ_R w.r.t. φ_O) together with the corresponding motion commands, and the trajectory followed by the object relative to its stationary pose right before it starts being pushed (φ_O w.r.t. φ_S). Therefore, a sequence takes the form

$$(\wp_{R1}, a_1, \wp_{O1}), (\wp_{R2}, a_2, \wp_{O2}), ..., (\wp_{Rn}, a_n, \wp_{On})$$

where $\wp_{Ri}$ is the pose of the robot relative to the object (φ_R w.r.t. φ_O) denoted as $\langle x, y, \theta \rangle$, a_i is the action associated with $\wp_{Ri}$, denoted as $\langle v_x, v_y, v\theta \rangle$ indicating the omni-directional motion command composed of the translational and rotational velocities of the robot, and $\wp_{Oi}$ is the pose of the object relative to its last stationary pose before it starts moving (φ_O w.r.t. φ_S), also denoted as $\langle x, y, \theta \rangle$.



(a) (b)

Figure 2: (a) Various reference frames used during sequence recording and replay 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 (best viewed in color).

Fig. 2(b) shows the visualization of the robot and object trajectories within the stored sequences. Yellow arrows indicate pose orientations, the smaller blue cylinders represent the initial pose of the robot relative to the object before the push starts, and the green cubes represent the observed mean final poses of the manipulated object after the corresponding push. The robot trajectory (indicated with blue curves) and the object trajectory (indicated with green curves) that belong to the same sequence are marked with the same ID value. Final object pose uncertainty is depicted with the red ellipses drawn around the mean final poses.

The sequences are recorded at each step of the robot's perception cycle, in our case at a frequency of 30Hz. As the number and the length of the recorded sequences increase, such high recording rates may cause problems in terms of efficient processing and scalability. To address this issue, we use a sparsified sequence representation, and only check the *keyframes* defined at every k^{th} frame of the sequence for faster collision detection along the trajectories both for the robot and the object. The value of k can be adjusted according to the dimensions of the objects being pushed; that is, if the objects are too small, then it is better to check for collisions more frequently along the trajectories.

3.2 Learning How Objects Behave

Humans learn and further sharpen their manipulation and corresponding prediction-based planning skills through interactions with their environment. Ideally, robots should also learn from their experiences instead of being provided with detailed mathematical models of each and every object they are expected to interact with, and physics engines to compute the outcomes of these interactions, since such an approach quickly becomes infeasible with the increasing number and variety of objects that the robot is expected to be able to handle. Also, it is almost impossible to provide accurate mathematical models for the real world objects with complex physical properties and motion characteristics that we want our robot to push-manipulate. For these reasons,

we make our robot interact with the actual pushable objects to observe how they behave in response to various pushing actions for future use in the planning process.

There are two challenges to be addressed in our problem.

1. The first challenge is the uncontrolled motion of the object after the push is ceased. As a result of moving on freely-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 between the pushing attempts even from the same direction for the same duration.

2. The second challenge is the effect of the initial, stationary orientations of the object's wheels on the trajectory that the object follows when 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.

Since these two problems are partially related, we try to address them simultaneously by having the robot build its experience incrementally over several trials instead of relying on a single observation.

When the robot is given a new object to be manipulated, it picks m random push initiation poses immediately around the object that are oriented towards random points along the edges of the object together with the corresponding random pushing durations ranging from 1 to 3 seconds. We name these pose-duration tuples *push configurations*, $\varsigma = \{\varsigma_1, \ldots, \varsigma_m\}$. Informed experience in addition to purely random experimentation can be transferred to the robot by providing some of these configurations through demonstration via joysticking it. As distinct pushes cause different final wheel orientations, the robot alternates between ς_i in order to make sure that it experiments with each ς_i for varying initial caster wheel orientations. We ensure the coverage of all ς_i by picking m to be a prime number and iterating over ς by using a set of increments $\iota = \{\iota_0 = 1, \ldots, \iota_j = m-1\}$ in a way similar to a hash collision resolution strategy. Starting with ι_0, the robot alternates between ς_i using $i = ((i + \iota_j) \mod m)$ until each of them are covered. Then it picks another increment ι_j with $j = ((j+1) \mod (m-1))$ and continues its experimentation. This process is repeated until the robot collects n samples from each of the m push configurations. The first trials for each of the random push configurations are saved as new sequences, and the additional trials are used to replay these sequences to update their statistical parameters. That is, the robot, in a sense, tests how *reliably* and consistently it can reproduce the expected observation when a particular pushing motion is replayed.

Assuming that the observed final object poses will be normally distributed, the robot tries to capture the final object pose uncertainties by incrementally updating the parameters of these distributions after observing the outcome of each of the n trials of for each ς_i as shown in Eq. (1) and Eq. (2).

$$\bar{\wp}_{O_n} = \bar{\wp}_{O_{n-1}} + \frac{\wp_{O_n} - \bar{\wp}_{O_{n-1}}}{n-1} \tag{1}$$

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

In these equations, $\bar{\wp}_{O_n}$ denotes the mean of the observed object pose after the n^{th} trial for a specific ς_i, and $\Sigma_{\wp_{O_n}}$ is the corresponding covariance, which in our case is a 3×3 matrix. This compact representation eliminates the need for storing all of the previously observed individual poses. Fig. 2(b) depicts the pose uncertainty as red ellipses around the final projected object poses.

As previously mentioned, these distributions are also good indicators of how reliable and consistent individual push sequences are. Since the objects move in an uncontrolled manner after the pushing is ceased, we do not want them to end up in unforeseen poses which may happen to collide with the obstacles in the environment, or cause the next pushing motion to be unachievable due to the obstruction of the corresponding initial robot pose. Therefore, we eliminate the sequences with variances exceeding predefined thresholds to improve the safety and reliability of the plans generated using these sequences, potentially reducing the number of re-plans needed along the way during plan execution.

In addition to testing how reliably the sequences can be replayed, we also test how helpful they are in constructing plans that transport the object to desired placements. After experiencing a new batch of m sequences, we pick several random goals in the task environment and let the robot try to generate plans for each of these goals multiple times using the available sequences to see how the number of problems solved and the consistency of the solutions change with the increasing number and variety of the available sequences. The results of these tests are presented in Section 4.

3.3 Achievable Plan Generation

Having memorized the motion primitives for the pushable objects and the corresponding robot motions (i.e. sequences), now the robot needs to use them to construct plans that are collision-free and achievable for both the object and the robot itself. Rapidly-exploring Random Trees (RRT) [11, 10] is one of the most commonly used planning algorithms due to its simplicity, practicality, and probabilistic completeness property. 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. It is also possible to bias the tree growth towards the goal and to reuse past experience [3].

Instead of extending the tree towards the sample along the straight line that connects the closest tree node to the sample, our proposed planning algorithm 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 either the robot's or the object's projected poses being in collision with the obstacles. This is the key point in ensuring *achievability* from both the robot's and the object's perspective; that is, we cannot guarantee a straight 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.

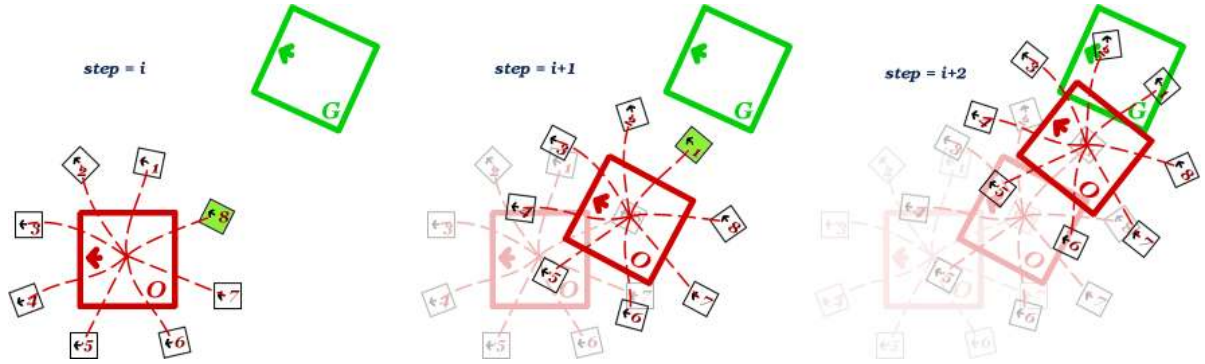


Figure 3: Illustration of the experience-based RRT construction process.

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. 3,

$$sim(p_1, p_2) = \frac{d_{max}}{dist(p_1, p_2)} cosSim(\hat{p_1}, \hat{p_2}) \quad (3)$$

where d_{max} is the maximum possible distance that can be obtained in the task environment, $dist(p_1, p_2)$ is the Euclidean distance between the locations of the poses, and $cosSim(\hat{p_1}, \hat{p_2})$ is the cosine similarity between the two poses assuming that they are unit vectors coinciding at the origin. Projecting the object on to the pose of the selected closest node, the mean final poses of the sequences originating from that pose are checked against the sample according the similarity function defined in Eq. 3. The tree is extended towards the sample by using the mean 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 continues until the projected object pose aligns with the goal pose with a certain error tolerance. Fig. 3 illustrates two steps of the tree construction process. The object trajectory components of the sequences are illustrated as dashed curves and the projected mean final object poses resulting from these trajectories are depicted as little black squares with attached sequence IDs. At each iteration, the projected object pose that gives the highest similarity value to the sample (in this particular example the sample is assumed to be the goal itself) is selected (highlighted in green), and the object is imagined to be pushed to that pose.

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. Specifically, collision check for the expected final object pose is performed using the associated distribution representing the pose uncertainty rather than a single pose. For this purpose, we derive $2L+1$ sigma points representing the extremes of the distribution from the mean and the covariance using Eq. (4)-(6), where L is the dimensionality of the state space. In our case $L = 3$ as we are dealing with 3 DoF poses.

$$\chi_0 = \bar{\wp}_O \quad (4)$$

$$\chi_i = \bar{\wp}_O + \zeta(\sqrt{\Sigma_{\wp_{O_n}}})_i, i = 1, \ldots, L \quad (5)$$

$$\chi_i = \bar{\wp}_O - \zeta(\sqrt{\Sigma_{\wp_{O_n}}})_i, i = L+1, \ldots, 2L \quad (6)$$

In these equations, $\bar{\wp}_O$ is the mean of the final object poses observed so far for a particular sequence, $(\sqrt{\Sigma_{\wp_{O_n}}})_i$ is the i^{th} column of the matrix-square-root of the covariance matrix $\Sigma_{\wp_{O_n}}$, and ζ is the scalar scaling factor that determines the spread of the sigma points around $\bar{\wp}_O$. 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 not used 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 $\wp_{R1}$ of the selected pushing sequences during plan execution.

3.4 Plan Execution and Monitoring

The constructed plan is executed by replaying one after another the robot trajectories of the chain of sequences that transports the object to the desired goal. 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 path, especially when it needs to be transported for a long distance. During plan execution, replanning 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. (7)

$$(\wp_{O_o} - \bar{\wp}_O)^T \Sigma_{\wp_O}^{-1} (\wp_{O_o} - \bar{\wp}_O) \leq \chi_k^2(p) \quad (7)$$

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 in order for the robot to re-plan.

Additionally, in order to relax the planning process a little, we use a heuristic that dynamically alters the desired final pose accuracy depending on the distance of the object from the goal. Eq. (8)-(9) define this heuristic

$$\delta = (dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \delta_{max} \quad (8)$$

$$\omega = \pi \cdot (dist(\wp_{O_o}, \wp_{O_g})/d_{max}) + \omega_{max} \quad (9)$$

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 to come up with a "rough" plan quickly when the object is far away from the goal, and forces it to generate more accurate plans each time it has to re-plan as the objects gets closer to the goal.

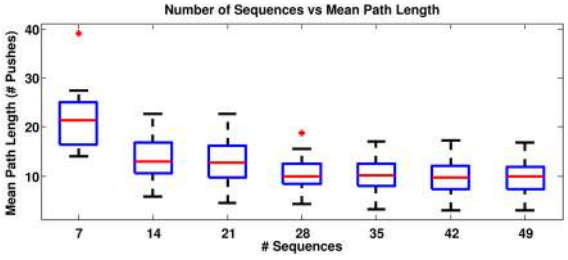
4. EXPERIMENTAL EVALUATION

We performed the majority of our experiments in the Webots mobile robot simulation environment [20], which enabled us to realistically simulate the pushable real world objects and their motions on freely rolling caster wheels. The final placement of an object was considered successful if the distance of the object to the desired goal was below 0.2m (i.e. $\delta_{max} = 0.2$ in Eq. 8) and the orientation difference was below $\pi/9$ radians (i.e. $\omega_{max} = \pi/9$ in Eq. 9). Considering the dimensions of the objects that our robot is manipulating, such as a $0.8m \times 0.45m$ serving tray and a $1.9m \times 0.9m$ stretcher, these constraints are quite tight.

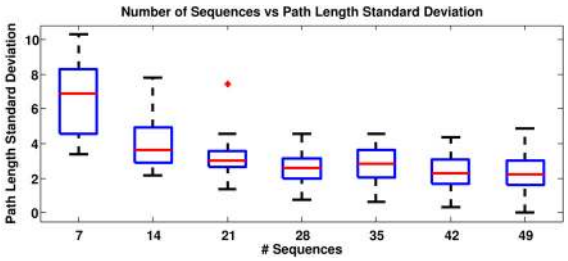
As briefly mentioned in Section 3.2, the first step in our experiments is to select the *reliable* set of sequences to be used for planning. We do this by eliminating the ones that cannot be replayed consistently; that is, the ones that have high variances in the final observed object pose. We determined 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 it starts reaching more than a certain percentage of the random goals, it 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 efficiency concerns.

The batch size used in our experiments was $m = 7$, and planning used during the sequence library evaluation process did not utilize the distance and orientation difference threshold relaxation heuristic we defined in Equations 8 and 9. 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 RRT nodes to be 33750 as we require 0.2m distance accuracy with at most $+/-\pi/9$ radians orientation difference in a $15m \times 15m$ environment.

Fig. 4 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, it makes 10 plan construction attempts for each of the 20 goals, so that that we can analyze the planning performance more reliably. Fig. 4(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. 4(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.



(a) Change of the mean path length with the increasing variety of sequences computed over 10 trials for each of the 20 goals.



(b) Change of the standard deviation of the mean path length with the increasing variety of sequences computed over 10 trials for each of the 20 goals.

Figure 4: Evaluation results of the increasing number of available sequences for the chair object. An increasing variety of available sequences results in consistently generated shorter paths.

Separate sets of sequences are learned and stored for each of our pushable objects. Fig. 5 demonstrates the generated achievable and collision-free plans from initial (S) to goal (G) poses for three of those objects, namely a chair, a food tray, and a pushable serving tray, by using their corresponding sequences as building blocks. 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 path during plan execution and for the robot to

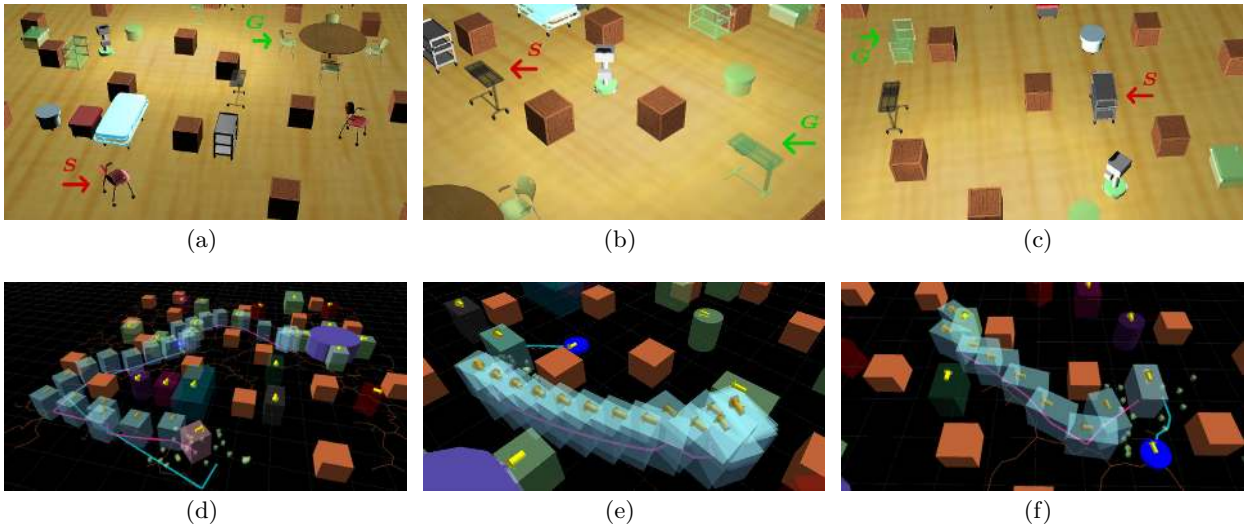


Figure 5: Generated plans (visualized as blue ghost figures over the pink path) using past experience for different pushable objects, namely a chair ((a) and (d)), a food tray ((b) and (e)), and a pushable serving tray ((c) and (f)) in very challenging environments cluttered with obstacles and other objects. The object of interests are pointed with red arrows and desired goal are pointed with green arrows in (a), (b), and (c). The robot's path to reach the next sequence to replay is depicted in blue (best viewed in color).

re-plan to guarantee the safe transportation of the object. In these three particular instances shown in Fig. 5, the robot had to re-plan for 4.33 times on the average.

It must be noted that we did not provide any explicit mathematical models or make use of physics engines for neither the pushable objects nor the robot. Our proposed method is able to handle any pushable object after the robot experiments with them to learn how they move in response to various pushes.

4.1 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 where our CoBot robot [16, 2] 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. 6 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 the 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, which can be quite noisy 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. 6), 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 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.

Figure 6: A snapshot from one of the real world experiments, 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.

During our preliminary tests, the robot was, in general, able to construct a decent 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.

5. CONCLUSION AND FUTURE WORK

Push-manipulation is one of the most interesting robotic manipulation modalities that has attracted many researchers. However, many of the proposed methods handle objects with quasi-static properties and primitive geometric shapes, yet they usually make use of complex mathematical models or utilize specialized physics engines to predict the outcomes of various pushes. On the other hand, we propose an experience-based approach that does not require any explicit mathematical 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 uncertainties in response to various pushing actions. It then uses this incrementally built experience as building blocks of a sampling based planner 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 shapes that may contact the robot on more than one point,
- the manipulated objects move on freely-rolling caster wheels and do not stop immediately after the pushing is ceased,
- the experiment environment is cluttered with obstacles; hence, achievable and collision-free plans should be constructed and manipulation should to be performed delicately,
- we do not use any explicit mathematical 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 and performed some preliminary tests in a physical setup, where a variety of pushable objects with freely rolling caster wheels need to be navigated among obstacles to reach their desired final poses. Our experiments demonstrate safe transportation and successful placement of several pushable objects in simulation and promising results for the task of chair arrangement in real world.

Future work includes extensive testing and detailed experimentation in the physical setup, partially repairing plans instead of complete re-planning, 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, and transferring learned manipulation sequences among objects with similar properties.

Acknowledgments

This research was partially supported by the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610, by award NSF IIS-1012733, and by ONR subcontract USC 138803. The views and conclusions contained in this document are those of the authors only.

6. REFERENCES

[1] P. K. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proceedings of ICRA*, 1997.

[2] J. Biswas, B. Coltin, and M. Veloso. Corrective Gradient Refinement for Mobile Robot Localization. In *Proceedings of IROS*, 2011.

[3] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proc. of IROS*, 2002.

[4] M. de Berg and D. Gerrits. Computing Push Plans for Disk-Shaped Robots. In *Proceedings of ICRA*, 2010.

[5] T. Igarashi, Y. Kamiyama, and M. Inami. A Dipole Field for Object Delivery by Pushing on a Flat Surface. In *Proceedings of ICRA*, 2010.

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

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

[8] M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Proc. of ICRA*, 2011.

[9] M. Lau, J. Mitani, and T. Igarashi. Automatic Learning of Pushing Strategy for Delivery of Irregular-Shaped Objects. In *Proc. of ICRA*, 2011.

[10] S. M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[11] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[12] K. M. Lynch. *Nonprehensile Robotic Manipulation: Controlability and Planning*. PhD thesis, Robotics Institute, Carnegie Mellon University, March 1996.

[13] M. Dogar and S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 33(3):217–236, 2012.

[14] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Path Planning for Pushing a Disk using Compliance. In *Proceedings of IROS*, 2005.

[15] D. Nieuwenhuisen, A. van der Stappen, and M. H. Overmars. Pushing Using Compliance. In *Proceedings of ICRA*, 2006.

[16] S. Rosenthal, J. Biswas, and M. Veloso. An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In *Proc. of AAMAS*, 2010.

[17] 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 Vision: What Why and How?*, 1993.

[18] M. M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, 1994.

[19] S. Walker and J. K. Salisbury. Pushing Using Learned Manipulation Maps. In *Proceedings of ICRA*, 2008.

[20] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[21] C. Zito, R. Stolkin, M. Kopicki, and J. Wyatt. Two-level RRT Planning for Robotic Push Manipulation. In *Proceedings of IROS*, 2012.