Autonomous Robots manuscript No.
(will be inserted by the editor)

A Planning Framework for Non-prehensile Manipulation under Clutter and Uncertainty

Mehmet R. Dogar · Siddhartha S. Srinivasa

Received: date / Accepted: date

Abstract Robotic manipulation systems suffer from two main problems in unstructured human environments: *uncertainty* and *clutter*. We introduce a planning framework addressing these two issues. The framework plans rearrangement of clutter using non-prehensile actions, such as pushing. Pushing actions are also used to manipulate object pose uncertainty. The framework uses an action library that is derived analytically from the mechanics of pushing and is provably conservative. The framework reduces the problem to one of combinatorial search, and demonstrates planning times on the order of seconds. With the extra functionality, our planner succeeds where traditional grasp planners fail, and works under high uncertainty by utilizing the funneling effect of pushing. We demonstrate our results with experiments in simulation and on HERB, a robotic platform developed at the Personal Robotics Lab at Carnegie Mellon University.

Keywords Manipulation among movable obstacles · Manipulation under uncertainty · Non-prehensile manipulation · Pushing

1 Introduction

Humans routinely perform remarkable manipulation tasks that our robots find impossible. Imagine waking up in the morning to make coffee. You reach into the fridge to pull out the milk jug. It is buried at the back of the fridge. You immediately start rearranging content — you push the large heavy casserole out of the way, you carefully pick up the fragile crate of eggs and move it to a different rack, but along the way you push the box of leftovers to the corner with your elbow.

Humans perform such manipulation tasks everyday. The variety of actual situations we encounter are endless, but our approach to them share common themes: the list of manipulation primitives that we use are not limited to grasping and include *non-prehensile* actions such as pushing, pulling, toppling; we are fearless in *rearranging clutter* surrounding our primary task — we care about picking up the milk jug and everything else is in the way; we are acutely aware of the *consequences of our actions* — we push the casserole with enough control to be able to move it without ejecting it from the fridge.

Successful robotic manipulation in human environments requires similar characteristics. In this work we propose a framework for robotic manipulation that plans a rearrangement of clutter, uses non-prehensile pushing actions as well as grasping actions, and tracks the consequences of actions by reasoning about the uncertainty in object pose and motion.

We present an example scene in Fig. 1. The robot's task is retrieving the red can which is surrounded by clutter. The robot first pushes the large box to the side and then uses that space to grasp the red can. It produces these actions autonomously using our planning framework. The planner identifies the objects to be moved: in the example the box is chosen among other objects in the scene. The box is a good choice but it is a big object that does not easily fit inside the robot hand, i.e. it is not graspable. Since our framework can work with non-prehensile actions, the box can be moved without grasping it.

Our planner reasons about the uncertainty before and during the motion of objects. Fig. 2 illustrates

Mehmet R. Dogar · Siddhartha S. Srinivasa
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA, USA
Tel.: +1-412-973-9615
E-mail: {mdogar,siddh}@cs.cmu.edu

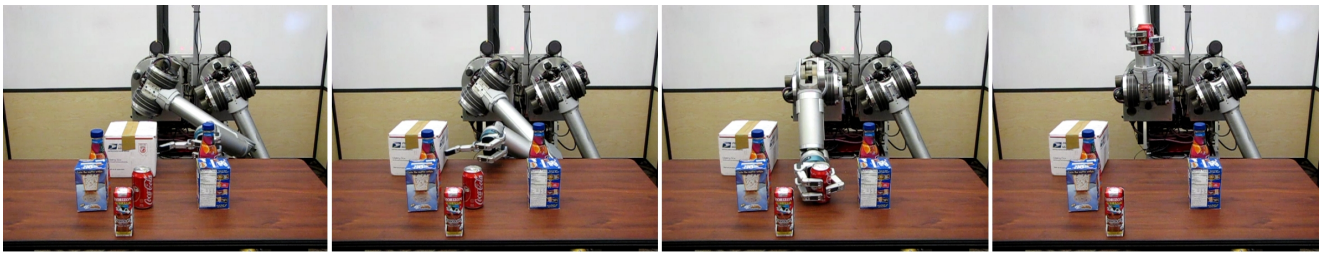


Fig. 1 The planner rearranging clutter to reach to a goal object. Pushing actions are useful for moving large objects that do not fit inside the hand, i.e. are not graspable. Planning time for the full sequence of actions in this example is 16.6 sec.

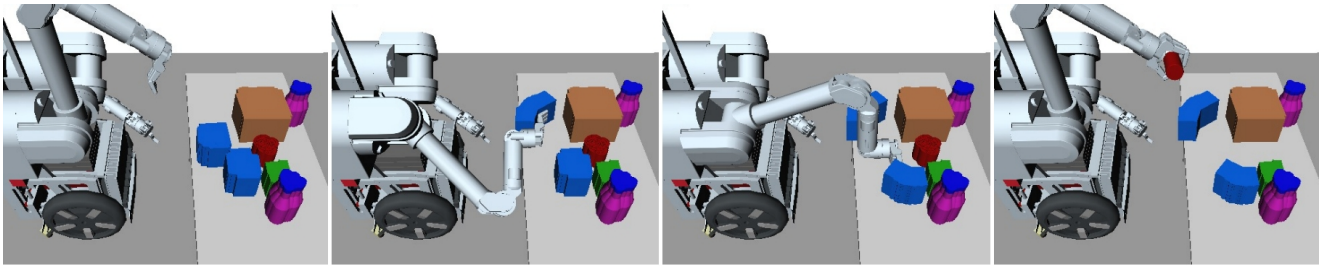


Fig. 2 The planner generates pushing actions that are robust to the pose uncertainty of objects. Uncertainty is represented using copies of the same object at different poses. Planning time for the full sequence of actions in this example is 23.4 sec.

the problem of planning under object pose uncertainty more clearly. One source of uncertainty is perception: robot cameras are used to detect and estimate the poses of objects but these pose estimates come with some amount of error. The second source of uncertainty is the action of the robot on an object: our predictions of how an object moves when it is pushed are not exact. Our framework accounts for both types of uncertainty when generating manipulation plans. When possible, our framework utilizes pushing actions to funnel large amounts of uncertainty into smaller amounts. In Fig. 2 the uncertainty of the red can is funneled into the hand using a pushing action before it is grasped.

The idea of rearranging objects to accomplish a task has been around for a few hundred years. We encounter this idea in games like the Tower of Hanoi (Chartrand 1985), the 15-Puzzle and numerous others. The blocks-world problem (Winograd 1971) introduced this idea to the AI community. STRIPS (Fikes and Nilsson 1971) is a well-known planner to solve this problem. In robotics, the problem is named *planning among movable obstacles*. The general problem is NP-hard (Wilfong 1988). Most of the existing planners work in the domain of two-dimensional robot navigation and take advantage of the low-dimensionality by explicitly representing, or discretizing, the robot C-space (Ben-Shahar and Rivlin 1998b; Chen and Hwang 1991; van den Berg et al 2008). These approaches are not practical for a manipulator arm with high degrees of freedom (DOF). Another group of planners are based on a search over different orderings to move the obstacle objects in the environment (Ben-Shahar and Rivlin 1998a; Overmars et al 2006; Stilman and Kuffner 2006). Planners that solve similar rearrangement problems in manipulation using real robotic hardware are also known (Stilman et al 2007). The planner from Stilman et al (2007) works backwards in time and identifies the objects that needs to be moved by computing the swept volume of the robot during actions. Recently, Kaelbling and Lozano-Perez (2011) proposed a planner that also identifies obstacles by computing swept volumes of future actions. In all of these cases, the physical act of manipulating an object is abstracted into a simple action, like pick-and-place. While extremely successful and algorithmically elegant, the simplified assumptions on actions severely restrict versatility. For example, such an algorithm would produce a solution whereby the robot carefully empties the contents of the fridge onto the countertop, pulls out the milk jug and then carefully refills the fridge. A perfectly valid plan, but one that is inefficient, and often impossible to execute with heavy, large, or otherwise ungraspable objects.

Pick-and-place actions are, however, easy to analyze. Once an object is rigidly grasped, it can be treated as an extension of the robot body, and the planning problem reduces to one of geometry. Performing actions other than pick-and-place requires reasoning about the non-rigid interaction between the robot effector and the object.

A separate thread of work, rooted in Coulomb's formulation of friction, uses mechanics to analyze the consequences of manipulation actions (Mason 1986; Goyal

et al 1991; Howe and Cutkosky 1996; Peshkin and Sanderson 1988; Brost 1988). Mason (1986) investigates the mechanics and planning of pushing for robotic object manipulation. One of the first planners that incorporates the mechanics of pushing was developed by Lynch and Mason (1996). Using the planner, a robot is able to push an object in a stable manner using edge-edge contact to a goal position. Goyal et al (1991) show that, in the quasi-static case, the motion of a pushed object is determined by the *limit surface*, which we use in predicting consequences of pushing actions. Manipulation planners and robot actions that use these physical models have been developed (Lynch and Mason 1996; Lynch 1999a; Akella and Mason 1998; Peshkin and Sanderson 1988; Agarwal et al 1997; Hauser and Ng-Thow-Hing 2011; Kappler et al 2012). Our planner uses pushing to address uncertainty and as a pre-grasp strategy, similar to these planners. A key difference of our framework is its ability to address *clutter* through rearrangement planning.

In this work we make an attempt at merging these two threads of work: geometric rearrangement planning and mechanical modeling and analysis. We present a framework that plans sequences of actions to rearrange clutter in manipulation tasks. This is a generalization of the planner from Stilman et al (2007). But our framework is not restricted to pick-and-place operations and can accomodate non-prehensile actions. We also present mechanically realistic pushing actions that are integrated into our planner.

Through the use of different non-prehensile actions, our planner generates plans where an ordinary pick-and-place planner cannot; e.g. when there are large, heavy ungraspable objects in the environment. We also show that our planner is robust to uncertainty.

2 Framework

In this section we present our framework to rearrange the clutter around a goal object. The framework uses non-prehensile actions that respects quasi-static mechanics. It produces open-loop plans which are conservative to the uncertainty in object poses. This uncertainty may be coming from either the non-stable non-prehensile actions or from the perception system that initially detects the objects. The framework consists of a high-level planner that decides on the sequence of objects to move and where to move them. The high-level planner uses a library of lower level actions to plan the actual robot trajectories that move the objects. The lower-level actions are also open-loop and do not require sensor feedback during execution.

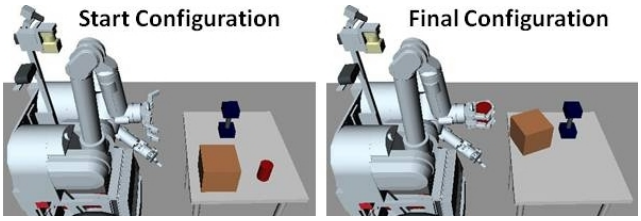


Fig. 3 An example scene. The robot's task is picking up the red can. The robot rearranges the clutter around the goal object and achieves the goal in the final configuration. The robot executes the series of actions shown in Fig. 4. We present the planning process in Fig. 5.

We first present the high-level planning framework, and then present the quasi-static pushing actions used by the high-level planner.

2.1 Planning Framework

In a given scene with multiple movable objects and a goal object to be grasped, the planner decides which objects to move and the order to move them, decides where to move them, chooses the lower-level actions to use on these objects, and accounts for the uncertainty in the environment all through this process. This section describes how we do that.

We describe our framework with the example in Fig. 3. The robot's task is picking up the red can. There are two other objects on the table: a brown box which is too large to be grasped, and the dark blue dumbbell which is too heavy to be lifted.

The sequence of robot actions shown in Fig. 4 solves this problem. The robot first pushes the dumbbell away to clear a portion of the space, which it then uses to push the box into. Afterwards it uses the space in front of the red can to grasp and move it to the goal position.

Fig. 4 also shows that the actions to move objects are planned backwards in time. We visualize part of this planning process in Fig. 5. In each planning step we move a single object and plan two actions. The first one (e.g. *Push-grasp* and *Sweep* in Fig. 5) is to manipulate the object. The second one (*GoTo* in Fig. 5) is to move the arm to the initial configuration of the next action to be executed. We explain the details of these specific actions in §2.1.6. We discuss a number of questions below to explain the planning process and then present the algorithm in §2.1.5.

2.1.1 Which objects to move?

In the environment there are a set of movable objects, `obj`. The planner identifies the objects to move by first attempting to grasp the goal object (Step 1 in Fig. 5).

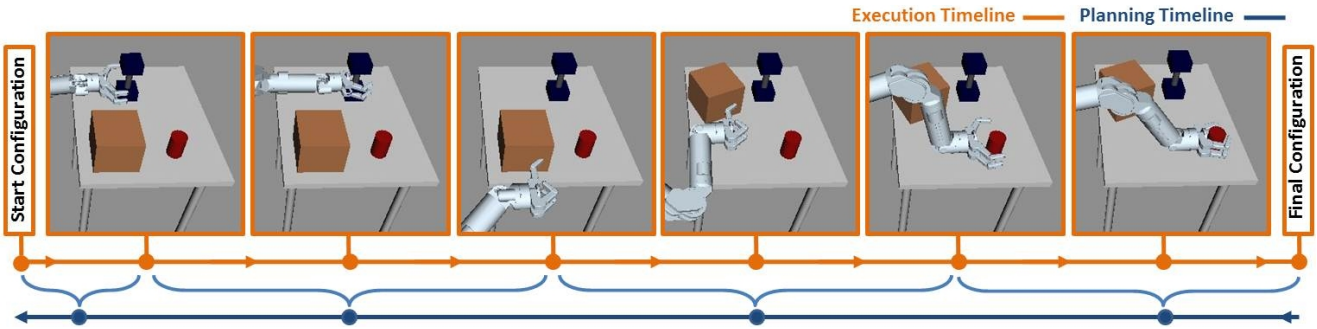


Fig. 4 We show the snapshots of the planned actions in the order they are executed. The execution timeline goes from left to right. Each dot on the execution timeline corresponds to a snapshot. Planning goes from right to left. Each dot on the planning timeline corresponds to a planning step. The connections to the execution timeline shows the robot motions planned in a planning step. Details of this planning process are in Fig. 5.

During this grasp, both the robot and the red can, as it is moved by the robot, are allowed to penetrate the space other objects in `obj` occupy. Once the planner finds an action that grasps the red can, it identifies the objects whose spaces are penetrated by this action and adds them to a list called `move`. These objects need to be moved for the planned grasp to be feasible. At the end of Step 1 in Fig. 5, the brown box is added to `move`.

We define the operator **FindPenetrated** to identify the objects whose spaces are penetrated:

$$\textbf{FindPenetrated}(vol, \texttt{obj}) = \{\texttt{o} \in \texttt{obj} \mid \text{volume } vol \text{ penetrates the space of } \texttt{o}\}$$

In the case of identifying objects to put into `move`, *vol* is the volume of space swept by the robot during its motion and by the object as it is manipulated. We compute the volume of space an object occupies by taking into account the pose uncertainty (§2.1.2).

In subsequent planning steps (e.g. Step 2 in Fig. 5) the planner searches for actions that move the objects in `move`. Again, the robot and the manipulated object are allowed to penetrate other movable objects' spaces. We add the penetrated objects to `move`.

This recursive process continues until all the objects in `move` are moved. The objects that are planned for earlier should be moved later in the execution. In other words, we plan backwards in identifying the objects to move.

Allowing the actions to freely penetrate other objects' spaces can result in a plan where objects are moved unnecessarily. Hence, our planner tries to minimize the number of these objects. This is described in §2.1.6.

We also restrict the plans to *monotone* plans; i.e. plans where an object can be moved at most once. This avoids dead-lock situations where a plan to move object A results in object B being moved, which in turn makes object A move, and so on. But more importantly restricting the planner to monotone plans makes the search space smaller: the general problem of planning with multiple movable objects is NP-hard (Wilfong 1988). We enforce monotone plans by keeping a list of objects called `avoid`. At the end of each successful planning step the manipulated object is added to `avoid`. The planner is *not* allowed to penetrate the spaces of the objects in `avoid`. In Fig. 5 in Step 2 the

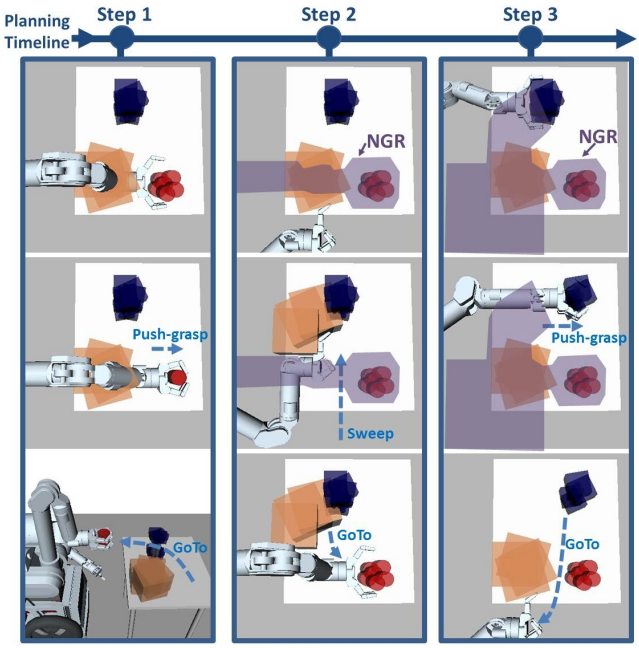


Fig. 5 The planning timeline. Three snapshots are shown for each planning step. The planner plans two consecutive arm motions at each step, from the first snapshot to the second snapshot, and from the second snapshot to the third snapshot. These motions are represented by blue dashed lines. The purple regions show the *negative goal regions* (*NGR*s), which are the regions the object needs to be moved out of (§2.1.4). The object pose uncertainty is represented using a collection of samples of the objects.

`avoid` list includes the red can, in Step 3 it includes the red can and the brown box.

2.1.2 How to address uncertainty?

Robots can detect and estimate the poses of objects with a perception system (in our experiments we use Martinez et al (2010)). Inaccuracies occur in pose estimation, and manipulation plans that do not take this into account can fail. Non-prehensile actions can also decrease or increase object pose uncertainty. Our planner generates plans that are robust to uncertainty. We explicitly represent and track the object pose uncertainty during planning.

Given a probability density function, $f_{\mathtt{o}}$, over the set of possible poses, we define the *uncertainty region* of an object $\mathtt{o}$ as the set of poses it can be in such that $f_{\mathtt{o}}$ is larger than ϵ:

$$U(\mathtt{o}) = \{q \in \mathbb{SE}(3) | f_{\mathtt{o}}(q) > \epsilon\}$$

We define the uncertainty region to be in $\mathbb{SE}(3)$ because we assume no uncertainty in objects' height and we also assume that the objects are standing upright on a surface.

Before the planning starts, the robot's perception system suggests a pose $\hat{q}_{\mathtt{o}}$ for each object $\mathtt{o}$ in the scene. We estimate the initial pose uncertainty of $\mathtt{o}$ as a multivariate normal distribution centered at $\hat{q}_{\mathtt{o}}$ with the covariance matrix Q. We estimate the covariance matrix Q by empirically modeling the error profile of our perception system (§3 presents the values we used to build the matrix Q in our experiments). In the rest of this paper we use $U(\mathtt{o})$ specifically to refer to the initial pose uncertainty of an object o.

The manipulation actions change the uncertainty of an object $\mathtt{o}$. We represent this as a trajectory $\nu_{\mathtt{o}}$:

$$\nu_{\mathtt{o}} : [0, 1] \rightarrow \mathcal{R}$$

where $\mathcal{R}$ is the power set of $\mathbb{SE}(3)$. We call $\nu_{\mathtt{o}}$ the *evolution of the uncertainty region* of object $\mathtt{o}$. $\nu_{\mathtt{o}}[0]$ is the same as $U(\mathtt{o})$. $\nu_{\mathtt{o}}[1]$ refers to the final uncertainty region of the object after manipulation. Each manipulation action outputs $\nu_{\mathtt{o}}$, i.e. how it evolves the uncertainty region of the object. §2.2.7 describes how $\nu_{\mathtt{o}}$ is estimated for pushing actions as a series of shrinking *capture regions*.

We used random sampling to represent all uncertainty regions. We present the number of samples we use for different uncertainty levels in §3. Fig. 5 illustrates the pose uncertainty using such samples.

During planning, we compute the volume of space an object occupies using U, not only the most likely pose. Likewise we compute the space swept by a manipulated object using $\nu_{\mathtt{o}}$. We define the operator **Volume**, which takes as input an object and a region, and computes the total 3-dimensional volume of space the object occupies if it is placed at every point in the region. For example, $\textbf{Volume}(\mathtt{o}, U(\mathtt{o}))$ gives the volume of space occupied by the initial uncertainty region of object $\mathtt{o}$.

We overload **Volume** to accept trajectories of regions and robots too; e.g. $\textbf{Volume}(\mathtt{o}, \nu_{\mathtt{o}})$ gives the volume of space swept by the uncertainty of the object during its manipulation, and $\textbf{Volume}(\mathtt{robot}, \tau)$ computes the three-dimensional volume the robot occupies during a trajectory τ. We compute this volume using a high-resolution sampling of configurations along the trajectory. We place three-dimensional models of the robot links at the corresponding poses at all the sampled points and sum them up to get the volume needed by the full trajectory.

2.1.3 How to move an object?

At each planning step, our planner searches over a set of possible actions in its action library. For example in Step 1 of Fig. 5 the planner uses the action named *push-grasp*, and in Step 2 it uses the action *sweep*. *Push-grasp* uses pushing to funnel a large object pose uncertainty into the hand. *Sweep* uses the outside of the hand to push large objects. Each low-level action, in turn, searches over different action-specific parametrizations to move the object; e.g. different directions to push-grasp an object, or different trajectories to use when moving the arm from one configuration to the other. We will describe the details of specific actions we use (e.g. push-grasp and sweep) and the search over the action-specific parametrizations in §2.1.6 and §2.2. Below we present the general properties an action should have so that it can be used by our high-level planner.

In grasp based planners robot manipulation actions are simply represented by a trajectory of the robot arm: $\tau : [0, 1] \rightarrow \mathcal{C}$ where $\mathcal{C}$ is the configuration space of the robot. The resulting object motion can be directly derived from the robot trajectory. With non-prehensile actions this is not enough and we also need information about the trajectory of the object motion: the evolution of the uncertainty region of the object. Hence the interface of an action $\mathtt{a}$ in our framework takes as an input the object to be moved $\mathtt{o}$, a region of goal configurations for the object G, and a volume of space to avoid $avoidVol$; and outputs a robot trajectory τ, and the evolution of the uncertainty region of the object during the action $\nu_{\mathtt{o}}$:

$$(\tau, \nu_{\mathtt{o}}) \leftarrow a(\mathtt{o}, G, avoidVol) \quad (1)$$

The returned values τ and $\nu_{\mathtt{o}}$ must satisfy:

- $\nu_{\mathtt{o}}[1] \subseteq G$; i.e. at the end all the uncertainty of the object must be inside the goal region.
- **Volume**($\mathtt{robot}, \tau$) and **Volume**($\mathtt{o}, \nu_{\mathtt{o}}$) must be collision-free w.r.t *avoidVol*; where `robot` is the robot body.

If the action cannot produce such a τ and $\nu_{\mathtt{o}}$, it returns an empty trajectory, indicating failure.

We also use a special action called *GoTo*, that does not necessarily manipulate an object, but moves the robot arm from one configuration to another. *GoTo* is also used to plan from the end of one object manipulation action to the start of other.

2.1.4 Where to move an object?

The planner needs to decide where to move an object — the goal of the action. This is easy for the original goal object, the red can in the example above. It is the goal configuration passed into the planner, e.g. the final configuration in Fig. 3. But for subsequent objects, the planner does not have a direct goal. Instead the object (e.g. the box in Step 2 of Fig. 5) needs to be moved out of a certain volume of space in order to make the previously planned actions (Step 1 in Fig. 5) feasible. We call this volume of space the *negative goal region* (*NGR*) at that step (shown as a purple region in Fig. 5) [1]. Given an *NGR* we determine the goal G for an object `o` by subtracting the *NGR* from all possible stable poses of the object in the environment: $G \leftarrow$ **StablePoses**(`o`) $-$ *NGR*.

The *NGR* at a planning step is the sum of the volume of space used by all the previously planned actions. This includes both the space the robot arm sweeps and the space the manipulated objects' uncertainty regions sweep. At a given planning step, we compute the negative goal region to be passed on to the subsequent planning step, NGR_{next}, from the current *NGR* by:

$$NGR_{next} \leftarrow NGR + \textbf{Volume}(\mathtt{robot}, \tau) + \textbf{Volume}(\mathtt{o}, \nu_{\mathtt{o}})$$

where τ is the planned robot trajectory, `o` is the manipulated object, and $\nu_{\mathtt{o}}$ is the evolution of the uncertainty region of the object at that planning step.

2.1.5 Algorithm

In our problem, a robot whose configurations we denote by $r \in \mathcal{C} \subseteq \mathbb{R}^n$ interacts with movable objects in the set `obj`. We wish to generate a sequence of robot motions `plan` that brings a goal object `goal` $\in$ `obj` into a goal pose $q_{\mathtt{goal}} \in \mathbb{SE}(3)$. We initiate the planning process with the call:

$$\mathtt{plan} \leftarrow \textbf{Rearrange}(\mathtt{goal}, \{q_{\mathtt{goal}}\}, \{\}, \{\}, \{\}, *)$$

The arguments to **Rearrange** are described in Alg. 1. The $*$ passed as the last argument here means that the final configuration of the robot arm does not matter as long as the object is moved to $q_{\mathtt{goal}}$.

Each recursive call to the **Rearrange** function is a planning step (Alg. 1). The function searches over the actions in its action library between lines 1-21, to find an action that moves the goal object to the goal configuration (line 4), and then to move the arm to the initial configuration of the next action (line 7). On line 11 it computes the total volume of space the robot and the manipulated object uses during the action. Then it uses this volume of space to find the objects whose spaces have been penetrated and adds these objects to the list `move` (line 12). If `move` is empty the function returns

Algorithm 1:
`plan` $\leftarrow$ **Rearrange**(`o`, G, *NGR*, `move`, `avoid`, r^{t+2})
`o`: The goal object;
G: The goal region; i.e. set of acceptable configurations `o` should end up in;
NGR: The negative goal region as described in §2.1.4;
`move`: The list of objects as described in §2.1.1;
`avoid`: The list of objects as described in §2.1.1;
r^{t+2}: The goal configuration for the robot after manipulating the object.

```
1  repeat
2      a ← next action from action library
3      avoidVol ← Σ_{i∈avoid} Volume(i, U(i))
4      (τ1, νo) ← a(o, G, avoidVol)
5      if τ1 is empty then
6          Continue at line 2
7      τ2 ←
       GoTo(τ1[1], r^{t+2}, avoidVol + Volume(o, νo[1]))
8      if τ2 is empty then
9          Continue at line 2
10     τ ← τ1 + τ2
11     vol ← Volume(robot, τ) + Volume(o, νo)
12     move_next ← move + FindPenetrated(vol, obj)
13     if move_next is empty then
14         return {τ}
15     NGR_next ← NGR + vol
16     avoid_next ← avoid + {o}
17     foreach i ∈ move_next do
18         plan ← Rearrange(i, StablePoses(i) −
           NGR_next, NGR_next, move_next −
           {i}, avoid_next, τ[0])
19         if plan is not empty then
20             return plan + {τ}
21 until all actions in action library are tried
22 return empty
```

[1] Note that the *NGR* has a 3D volume in space. In Fig. 5 it is shown as a 2D region for clarity of visualization.

the plan. On line 15 the function adds the volume of space used by the planned action to the *NGR*. On line 16 it adds the current object to `avoid`. Between lines 17-20 the function iterates over objects in `move` making recursive calls. If any of these calls return a plan, the current trajectory is added at the end and returned again (line 20). The loop between 17-20 effectively does a search over different orderings of the objects in `move`. If none works, the function returns an empty plan on line 22, indicating failure, which causes the search tree to backtrack. If the planner is successful, at the end of the complete recursive process `plan` includes the trajectories in the order that they should be executed.

2.1.6 Action Library

The generic interface for actions is given in Eq. 1. In this section we briefly describe the actions in our action library and explain how they satisfy this generic interface. There are four actions in our action library: Push-grasp, Sweep, GoTo, and PickUp.

- *Push-grasp*:
  Push-grasping is a robust way of grasping objects under uncertainty. It is a straight motion of the hand parallel to the pushing surface along a certain direction, followed by closing the fingers. In effect, a push-grasp sweeps a region on the pushing surface, so that wherever an object is in that region, at the end of the push it ends up inside the hand, ready to be grasped.
  For a given object, the *capture-region* of a parametrized push-grasp is the set of all object poses that results in a successful grasp. Example capture regions are shown in Fig. 6 and Fig. 9. We compute capture regions using a realistic quasi-static pushing analysis. We use the capture regions to decide whether a push-grasp will succeed given an object and its uncertainty region.
  We present a detailed analysis of push-grasping in §2.2, where we also explain how to compute and use capture regions.
- *Sweep*:
  Sweep is another action we use to move obstacles out of negative goal regions. Sweep uses the outside region of the hand to push an object. Sweeping can move objects that are too large to be grasped (Fig. 7a). Similar to Push-grasp, we parametrize a Sweep by a direction and distance to push.
  A push-grasp requires a minimum pushing distance because it has to keep pushing the object until it completely rolls into the hand. Since sweeping only needs to move an object out of a certain volume of space, it does not require a particular pushing distance. But we still use the capture region to guarantee that the object will not escape the push by rolling outside during the sweep. When computing the capture region for sweep (Fig. 7b) we use a realistic model for the side of the fingers but approximate the other side with a straight line located at the end of the wrist link.
  The sweep action can also address initial object pose uncertainty. Similar to Push-grasp, we check that the capture region of the Sweep includes all the poses sampled from the uncertainty region of the object (Fig. 7b).
- *GoTo*: The GoTo action moves the robot arm from one configuration to the other. The search space of the GoTo action is the configuration space of the arm. To implement this action we use an extension of the Rapidly-Exploring Random Tree (RRT) (Lavalle and Kuffner 2000) planner, namely the Constrained Bi-directional RRT planner (CBiRRT) (Berenson et al 2009a).
  The GoTo action either does not manipulate an object or moves an already grasped object. At the end the object pose is derived from the forward kinematics of the arm.
- *PickUp*: This is the prehensile manipulation of an object. We implement PickUp as a Push-grasp followed by a GoTo.

2.2 Push-grasping

In this section we present details of push-grasping.

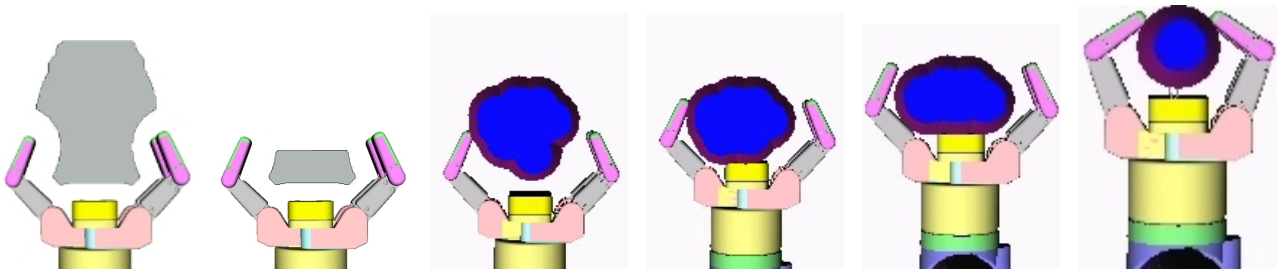
2.2.1 The push-grasp

The *push-grasp* is a straight motion of the hand parallel to the pushing surface along a certain direction, followed by closing the fingers (Fig. 8). We parametrize (Fig. 9(a)) the push-grasp $G(p_h, a, d)$ by:

- The *initial pose* $p_h = (x, y, \theta) \in SE(2)$ of the hand relative to the pushing surface.
- The *aperture* a of the hand during the push. The hand is shaped symmetrically and is kept fixed during motion.
- The *pushing direction* v along which the hand moves in a straight line. In this study the pushing direction is normal to the palm and is fully specified by p_h.
- The *push distance* d of the hand measured as the translation along the pushing direction.

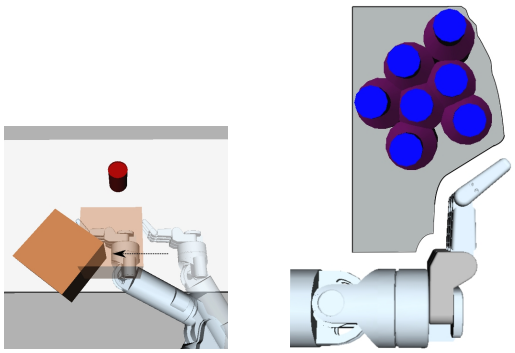
We execute the push-grasp as an open loop action.

We make certain assumptions while modeling and executing push-grasps:



(a) Capture Region (b) Object poses after the push (c) Uncertainty (d) Push grasp (e) Push grasp (f) Push grasp

Fig. 6 (a) The capture region of a for a rotationally symmetric bottle for a push-grasp. Every point corresponds to a bottle position where the coordinate frame of the bottle is at its center. (b) Uncertainty region of the object after the push, before closing the fingers. (c-f) A push-grasp funneling the uncertainty into the hand.



(a) Example sweep (b) Capture Region

Fig. 7 (a) Sweeping can move objects that are too large to be grasped. (b) The capture region of the sweep action for the a cylindrically symmetric bottle.

- The interaction between the robot hand and the object is quasi-static, meaning that the inertial forces are negligible.
- The objects do not topple easily. To prevent objects from toppling, the robot pushes them as low as possible.
- The robot has the three-dimensional models of the objects.
- The coefficient of friction between the objects and the pushing surface is uniform.
- Additionally, our model of how a pushed object moves is most realistic if the object's pressure distribution has rotational symmetry. This includes not only circularly symmetric distributions, but also rectangles, equilateral triangles; any distribution that repeats itself in a revolution about the center of pressure Howe and Cutkosky (1996).

2.2.2 The Capture Region of a Push-Grasp

A successful push-grasp is one whose execution results in the stable grasp of an object. Given the push-grasp, the object's geometry and physical properties, which we term O, and the object's initial pose, we can utilize the mechanics of manipulation described before to predict the object's motion. Coupling the simulation with a suitable measure of stability, like caging or force-closure, we can compute the set of all object poses that results in a stable push-grasp. We call this set the *capture region* $C(G, O) \subset SE(2)$ of the push-grasp.

We use a simulation of quasi-static pushing to compute a capture region. We performs this simulation offline once for each object.

Howe and Cutkosky (1996) show that the limit surface can be approximated by a three-dimensional ellipsoid. We use the aspect ratio of this ellipsoid, in calculating the normal to a point on it. The equatorial radii are found by calculating the maximum friction force (f_{max}) that the supporting surface can apply to the object, which occurs when the object is translating. The polar radius is found by calculating the maximum moment (m_{max}) that the supporting surface can apply, which occurs when the object is rotating around its center of friction. Then the quasi-static motion of the object is determined by the ratio $c = m_{max}/f_{max}$. The mass of the object and the coefficient of friction between the object and the supporting surface (μ_s) are multipliers in both the numerator and denominator of this fraction, and cancel out. Hence, as long as the ellipsoid approximation holds, we do not need to know the object mass or μ_s to predict the motion (These parameters simply inflate or deflate the ellipsoid but do not change the normal at a point on it). The pressure distribution supporting the object on the surface and the coefficient of friction between the robot finger and the object, μ_c, do affect the motion of the object. We

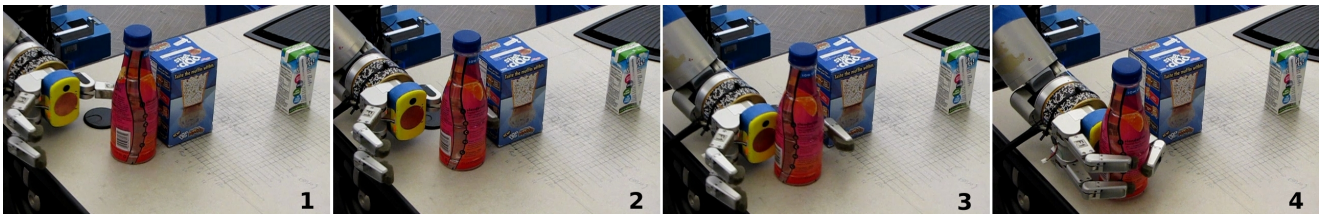
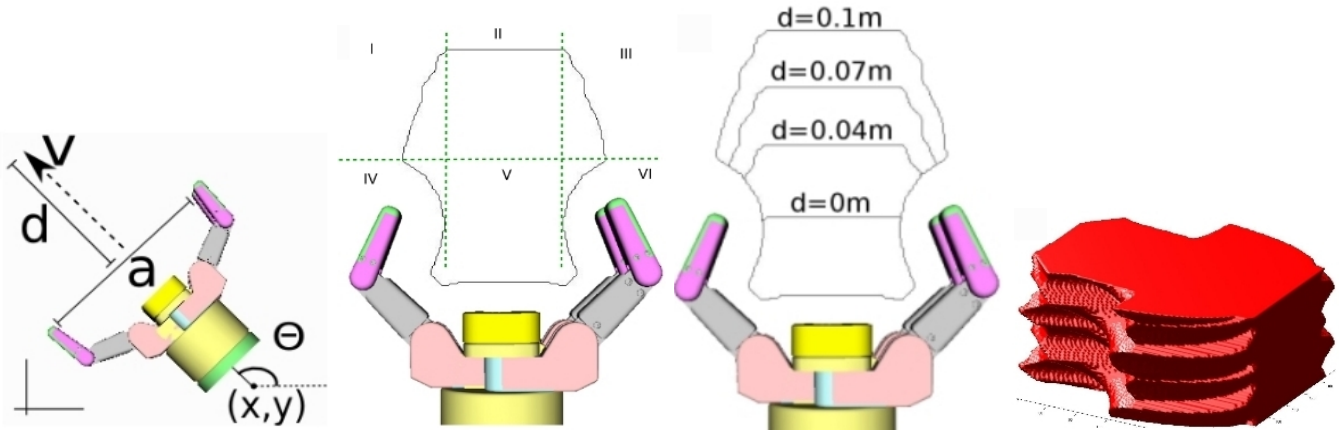


Fig. 8 An example push-grasp of an object in contact with the surrounding clutter.



(a) (b) (c) (d)

Fig. 9 (a) Parametrization of a push-grasp. (b) The capture region of a radially symmetric bottle is the area bounded by the black curve. We divided the plane into different regions using the green dashed lines. (c) Capture regions for push-grasps of different distances. (d) 3D capture region of a rectangular box.

compute capture regions conservatively with respect to these parameters, so that the capture region will be valid for a wide range of values these parameters can take. For a cylindrical object the conservative capture region is given by assuming the pressure distribution to be at the periphery of the object, and assuming μ_c to have a very large value. A proof of this is presented in Appendix A.

We present the capture region of a juice bottle produced by our pushing simulation in Fig. 9(b), which is a 2D region as the bottle is radially symmetric. The capture region is the area bounded by the black curve. The shape of the curve represents three phenomena. The part near the hand (inside regions IV, V, and VI) is the boundary of the configuration space obstacle generated by dilating the hand by the radius of the bottle. The line at the top (inside region II) represents the edge of the fingers' reach. We conservatively approximate the curve traced out by the fingers while they are closing by the line segment defining the aperture.

Regions I and III of the capture region curve are the most interesting. Let us consider the left side of the symmetric curve. If an object is placed at a point on this curve then during the push-grasp the left finger will make contact with the object and the object will eventually roll inside the hand. If an object is placed slightly to the left of this curve, then the left finger will push that object too, but it will not end up inside the hand at the end of the push: it will either roll to the left and out of the hand or it will roll right in the correct way but the push-distance will not be enough to get it completely in the hand. We can observe the critical event at which the object starts to slide on the finger, producing a discontinuity on the upper part of the curve.

We also present the three-dimensional capture region of a rectangular box in Fig. 9(d). We compute it by computing the two-dimensional regions of the object at different orientations.

2.2.3 Efficient Representation of Capture Regions

Each push-grasp G for an object O produces a unique capture region $C(G, O)$. By computing $C(G, O)$ relative to the coordinate frame of the hand, we can reduce the dependence to the aperture a and the pushing distance d. Every other capture region is obtained by a rigid transformation of the hand-centric capture region. This can be formally stated as $C(G(p_h, a, d), O) = T(p_h)C(G(0_h, a, d), O)$.

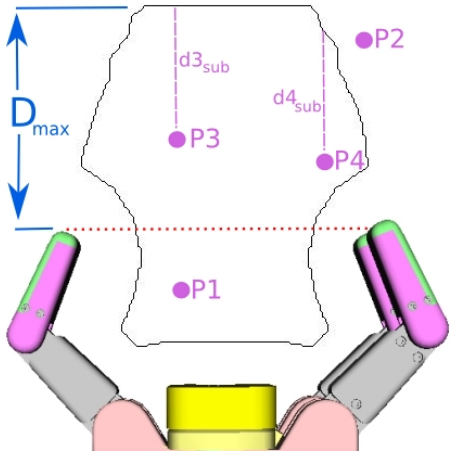


Fig. 10 Given an object pose, the minimum required pushing distance d to grasp that object can be found using a precomputed capture region of a push-grasp with pushing distance D_{max}. In the figure, $d = 0$ for P1 since it is already in the hand; P2 can not be grasped with a push shorter than D_{max} since it is outside the capture region; for P3 and P4 the required pushing distances can be found by computing $d = D_{max} - d3_{sub}$ and $d = D_{max} - d4_{sub}$ respectively.

To illustrate the effects of the pushing distance d on the shape of a capture region, we overlaid the capture regions produced by different pushing distances in Fig. 9(c). We can see that as the pushing distance gets smaller, the upper part of the larger capture region (regions I, II, and III in Fig. 9(b)) is shifted down in the vertical axis. To understand why this is the case, one can divide a long push into two parts, and think of the last part as an individual push with the remaining distance.

This lets us pre-compute the capture region for a long push distance, D_{max}, and use it to produce the capture regions of shorter pushes. Given all the other variables of a push-grasp, our planner uses this curve to compute the minimum push distance d required by an object at a certain pose (Fig. 10). The cases to handle are:

- If the object is already inside the hand (see *P1* in Fig. 10), no push is required; $d = 0m$.
- Else, if the object is outside the capture region (see *P2* in Fig. 10) there is no way to grasp it with a push shorter than D_{max}. Reject this object.
- Else, the minimum pushing distance required can be found by using the formula

$$d = D_{max} - d_{sub}$$

where d_{sub} is the distance between the object and the top part of the capture region curve along the pushing direction v (see *P3* and *P4* in Fig. 10). d_{sub} can be interpreted as the value we can shorten the push-distance D_{max} such that the object is exactly on the boundary of the capture region.

In our implementation we use $D_{max} = 1m$ as an overestimate of the maximum distance our robot arm can execute a pushing motion.

The effect of changing the hand aperture, a, is straightforward. Referring again to the regions in Fig. 9(b), changing a only affects the width of the regions II and V, but not I and III. Therefore, we do not need to compute capture regions for different aperture values. Note that this is only true assuming the fingertips are cylindrical in shape, hence the contact surface shapes do not change with different apertures. If the fingertip contact surfaces dramatically change with different apertures of the hand, one can compute the capture regions for a predefined set of different apertures.

2.2.4 Validating Capture Regions

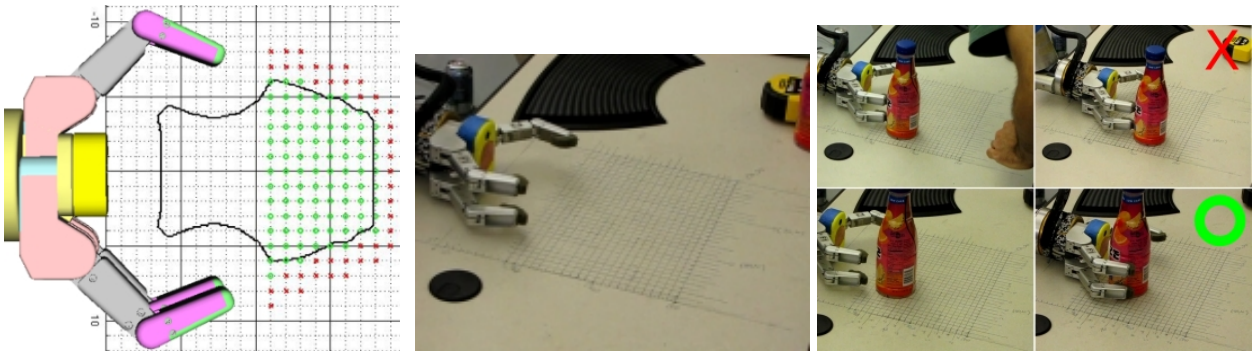
We ran 150 real robot experiments to determine if the precomputed models were good representations of the motion of a pushed object, and whether they were really conservative about which objects will roll into the hand during a push.

To validate the capture region, we repeatedly executed a push of the same d and placed the object in front of the hand at different positions on a grid of resolution $0.01m$ (Fig. 11b). Then we checked if the object was in the hand at the end of a push. The setup and two example cases where the push grasp failed and succeeded are shown in Fig. 11c.

The results (Fig. 11a) show that, the simulated capture region is a conservative model of the real capture region. There are object poses outside the region for which the real object rolled into the hand (green circles outside the black curve); but there are no object poses inside the curve for which the real object did not roll into the hand. This is in accordance with our expectations, since, for the system parameters that are hard to know (the pressure distribution underneath the object, and the coefficient of friction between the finger and the object) our simulation of pushing uses conservative values. This guarantees success, in the sense that our planner always overestimates the pushing distance needed. But this is a tight overestimate, as can be seen in Fig. 11a.

2.2.5 Overlapping Uncertainty and Capture Regions

The overlap between a capture region and an uncertainty region indicates whether a push-grasp will succeed under uncertainty. To guarantee that a push-grasp



(a) Simulation and real-world experiments. Green circles: real world successes; red crosses: real world failures.

(b) Push-grasping validation setup

(c) Two example cases where the push fails (top row), and succeeds (bottom row).

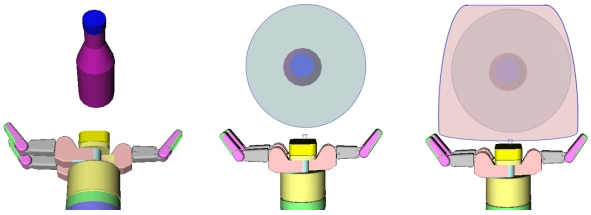
Fig. 11 Capture region generated with our push-grasping simulation and validated by robot experiments. 150 validation tests were performed in total.

will succeed it is sufficient to make sure that the uncertainty region of the goal object is included in the capture region of the push-grasp.

We illustrate this idea in Fig. 12. Here the robot detects a juice bottle (Fig. 12a). We illustrate the uncertainty region of the juice bottle in Fig. 12b, and the capture region of the push-grasp in Fig. 12c. If the uncertainty region is completely included in the capture region as in Fig. 12c, then we can guarantee that the push-grasp will succeed.

The uncertainty and capture regions are two-dimensional in Fig. 12 only because the bottle is radially symmetric. In general, these regions are three-dimensional, non-convex and potentially even disjoint (e.g. multi-modal uncertainty regions). Checking inclusion/exclusion of two generic three-dimensional regions is a computationally expensive problem.

We use a sampling strategy to overcome this problem. We draw n random samples from the uncertainty region, and check if all of these samples are in the capture region of a push-grasp. Samples are drawn according to the probability distribution of the uncertainty region: poses of higher probability also have a higher chance of being sampled.



(a) Detected object (b) Uncertainty region (c) Capture region

Fig. 12 If the uncertainty region of an object is included in the capture region of a push-grasp, then the push-grasp will be successful.

2.2.6 Finding a successful push-grasp

The planner searches for a push-grasp such that the hand can grasp all the samples drawn from the uncertainty region of the object, and the resulting hand motion can be executed with the arm.

Given a goal object in the environment, the planner searches for a push grasp by changing the parameters v, a, and the lateral offset in approaching the object, o. The lateral offset o changes the initial pose of the hand by moving it along the line perpendicular to the pushing direction v. During the search, these parameters are changed between certain ranges, with a user defined step size. v changes between $[0, 2\pi)$; a changes between the maximum hand aperture and the minimum hand aperture for the object; and o is changed between the two extreme positions, where the object is too far left or right relative to the hand.

The push-grasp is allowed to penetrate the space of other movable objects as explained in §2.1.1. But we try to minimize the number of such objects to get more efficient plans. Therefore we compute a heuristic value for the different directions to push-grasp an object. We rotate the robot hand around the goal object and check the number of objects it collides with. We prefer directions with a smaller number of colliding objects.

2.2.7 Evolution of uncertainty region during pushing

We use the capture region to also represent the evolution of the uncertainty region of a manipulated object, ν_o.

As explained in §2.2.3 we can use a capture region for a push of length d to compute capture regions for

Table 1 Planner Performance.

	No Clutter		Medium Clutter		High Clutter	
	TSR	PG	TSR	PG	TSR	PG
σ_1	10 0.01	10 0.02	10 0.01	10 0.04	5 0.54	8 1.98
σ_2	9 0.52	10 0.58	9 1.02	10 1.17	0 1.97	5 12.93
σ_3	0 0.86	10 1.00	0 1.61	10 5.17	0 3.22	3 28.16
σ_4	0 0.86	5 1.44	0 1.63	0 3.91	0 3.08	0 7.46

shorter pushes (Fig. 9(c)). Using the same formulation, at any future point before the end of a push, we construct the capture region for the remaining distance. A push with a pushing distance d is executed only if its capture region contains the initial uncertainty region of the object, which indicates that the evolved uncertainty of the object will always stay in the subsequent capture regions until the object is inside the hand at the end of the push. Hence, we discretize a push into smaller steps, and use these series of capture regions to conservatively approximate the evolution of the uncertainty region, ν_o, of the object o.

3 Experiments and Results

3.1 Push-grasping Experiments

In this section we present our experiments with the push-grasping action. We will present our experiments with the complete framework in §3.2. We conducted experiments in simulation and on HERB (Srinivasa et al 2009) to evaluate the performance of our planner. Simulation experiments are performed in OpenRAVE (Diankov and Kuffner 2008).

3.1.1 Robotic Platform

HERB has two 7-DoF WAM arms and 4-DoF Barrett hands with three fingers. A camera is attached to the palm to detect objects and estimate their poses.

3.1.2 Planner performance

We compared the performance of our push-grasp planner with another grasp planner that can handle uncertainty about the object pose. We used the *uncertainty task space regions (TSRs)* algorithm from Berenson et al (2009b). In our implementation, to supply the TSRs with a set of hypotheses we used samples from the uncertainty region of our objects. We used the same number of samples that we use for our push-grasp planner.

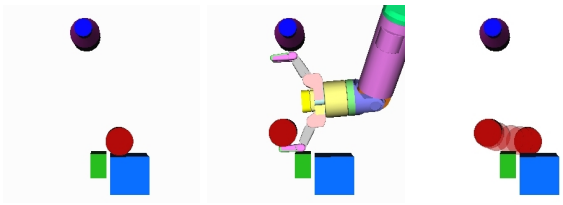


Fig. 13 A high-clutter scene where the TSR planner fails but push-grasp planner is able to find a plan.

Table 1 presents results in simulation comparing the performance of our push-grasp planner (PG) and the Uncertainty TSR planner. We categorize scenes as no clutter (1 object), medium clutter (2-3 objects placed apart from each other), and high clutter (3-4 objects placed close to each other). For each category we created ten different scenes. For each scene we added increasing amount of uncertainty, where σ_1 is no uncertainty, and σ_4 is the highest uncertainty.

In each cell of Table 1 we present four numbers. The top left number indicates in how many of the ten scenes Uncertainty TSR planner was able to come up with a plan. The same value for the Push-Grasp planner is in the top right. We indicate the average planning time in seconds, for TSR, on the lower left corner. The same value for the push-grasp planner is at the lower right. We used three-dimensional multivariate normal distributions (in x, y, and θ) as the uncertainty regions. We modeled each dimension as mutually independent and used the following standard deviations in object translation and rotation for different uncertainty levels: σ_1: no uncertainty; σ_2: (0.005m, 0.034rad); σ_3: (0.02m, 0.175rad); σ_4: (0.06m, 0.785rad). The number of samples, n, we used for these uncertainty levels are: 1, 30, 50, 50.

Table 1 shows that the push-grasp planner is able to plan in environments with higher uncertainty. When the uncertainty is high, the Uncertainty TSR planner is not able to find any static pose of the hand that grasps all the samples of the object. The push-grasp planner, on the other hand, is not limited to static grasps, and can sweep larger regions over the table than any static hand pose can. Note also that a push-grasp with no real pushing ($d = 0$) is possible, hence the push-grasp planner is able to find a solution whenever the TSR planner finds one.

We can see from Table 1 that push-grasp planner also performs better in high clutter. One example scene of high clutter, where push-grasp planner is able to find a grasp but the Uncertainty TSR planner cannot, is

presented in Fig. 13. Here the goal object is right next to other objects. The Uncertainty TSR planner cannot find any feasible grasps in this case since any enveloping grasp of the object will collide with the obstacle objects. In this case, the push-grasp planner comes up with the plan presented in Fig. 13, which moves the object away from the clutter first and then grasps.

The planning times also shown in Table 1. We see that in environments where the Uncertainty TSR planner succeeds, the push-grasp planning times are compatible. Push-grasp planner takes a longer but reasonable amount of time (tens of seconds) in difficult environments where Uncertainty TSR planner fails.

3.1.3 Real Robot Experiments

We conducted two sets of experiments on our real robot. In the first, we used the actual uncertainty profile of our object pose estimation system. In the second set of experiments, we introduced higher noise to the detected object poses.

In the first set of experiments we created five scenes, detected the objects using the palm camera and planned to grasp them using both the Uncertainty TSR planner and our push-grasp planner. As described in §2.1.2 we empirically modeled the uncertainty profile of our object pose estimation system as a multivariate normal distribution. We assumed each dimension to be mutually independent and used the standard deviation values σ: (0.007m, 0.07rad) to build the covariance matrix Q. The number of samples we used, n, was 30. Uncertainty TSR planner was able to find a plan three out of five times, and the push-grasp planner was able to find a plan four out of five times. All the executions of these plans were successful. Again the Uncertainty TSR planner was not able to find a plan when the goal object was right next to another obstacle object, making it impossible to grasp the goal object without colliding with the obstacles.

In another set of experiments on the real robot we introduced higher uncertainty by adding noise to the positions of the objects reported by the object detection system. For Gaussian noise in translation with $\sigma = 0.02m$ and $n = 50$, the Uncertainty TSR planner was not able to find a plan for any of the five scenes, while the push-grasp planner found a plan and successfully executed them in three of the five scenes. The failures were due to the collision of the arm with the clutter while moving the end-effector to the starting pose of the push-grasp. In the next section we present our experiments with the complete framework, where the arm motions are also planned by taking into account the uncertainty in the environment.

Execution of some of the push-grasps can be seen in Fig. 14. Videos of our robot executing push-grasps are online at

www.cs.cmu.edu/~mdogar/pushgrasp

3.2 Planning Framework Experiments

This section describes our experiments with the complete framework, including not only the Push-grasp but also the Sweep, GoTo, and PickUp.

We created scenes in simulation and in real world. The robot's goal was to retrieve objects from the back of a cluttered shelf and from a table. We used everyday objects like juice bottles, poptart boxes, coke cans. We also used large boxes which the robot cannot grasp.

We present snapshots from our experiments in the figures of this section. The video versions can be viewed at

www.cs.cmu.edu/~mdogar/pushclutter

3.2.1 Pushing vs. Pick-and-Place

Here, we compare our planner in terms of the efficiency (planning and execution time) and effectiveness (whether the planner is able to find a plan or not) with a planner that can only perform pick-and-place operations. To do this, we used our framework algorithm to create a second version of our planner, where the action library consisted of only the PickUp and GoTo actions, similar to the way traditional planners are built using *Transfer* and *Transit* operations. We modified the PickUp action for this planner, so that it does not perform the pushing at the beginning, instead it grasps the object directly. We call this planner the *pick-and-place planner*, and our original planner the *pushing planner*.

An example scene where we compare these two planners is given in Fig. 15. The robot's goal is to retrieve the coke can from among the clutter. We present the plans that the two different planners generate. The pushing planner sweeps the large box blocking the way. The pick-and-place planner though cannot grasp and pick up the large box, hence needs to pick up two other objects and avoid the large box. This results in a longer plan, and a longer execution time for the pick-and-place planner. The planning time for the pick-and-place planner is also longer, since it has to plan more actions. These times are shown on the figure.

In the previous example the pick-and-place planner was still able to generate a plan. Fig. 16 presents a scene where the pick-and-place planner fails: the large ungraspable box needs to be moved to reach the goal object, the can. The pushing planner generates a plan and is presented in the figure.

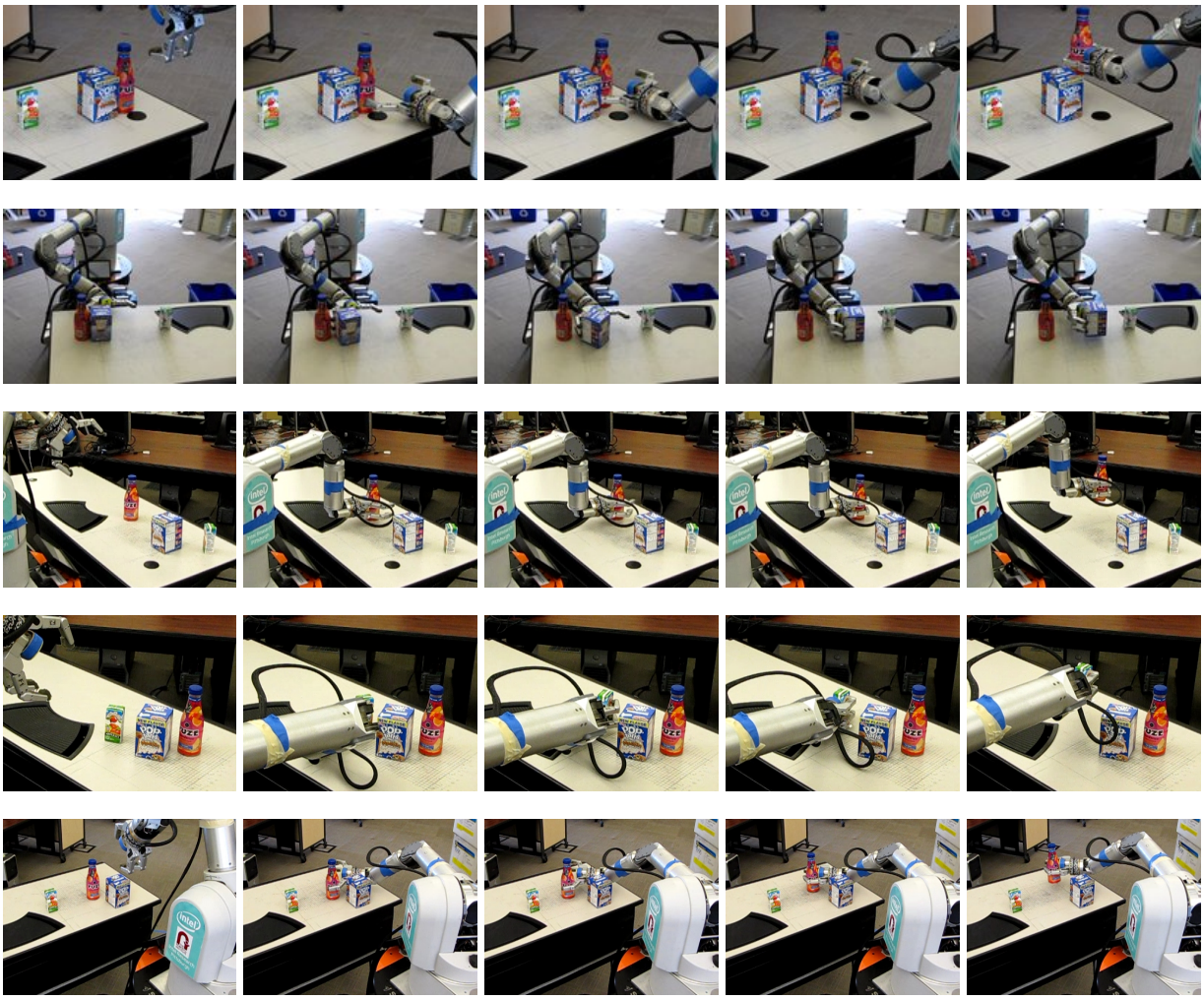


Fig. 14 Example push-grasps executed by our robot.

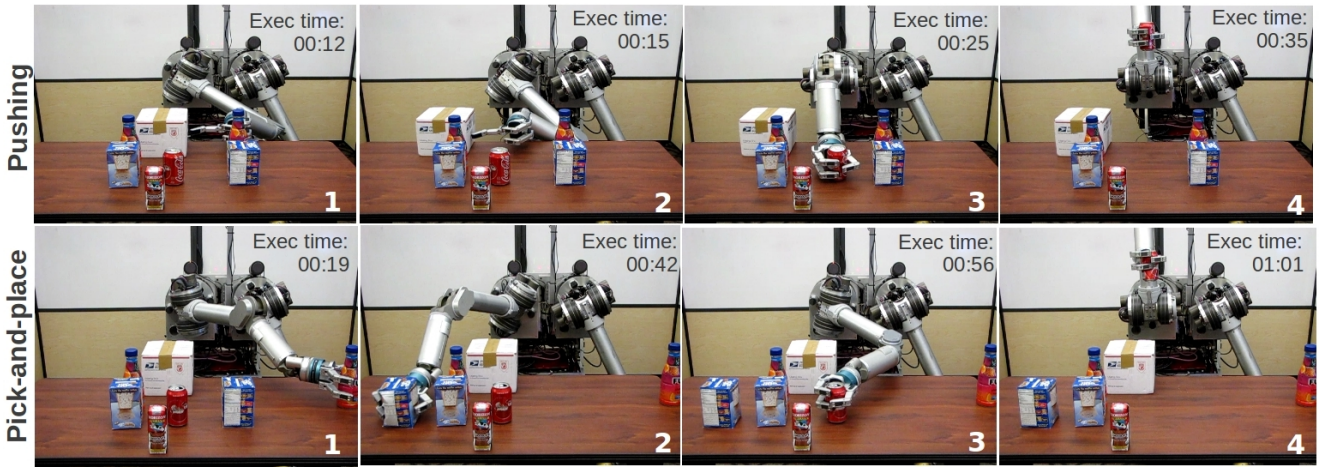


Fig. 15 The plans that the *pushing planner* and the *pick-and-place* planner generates in the same scene are presented. The pushing planner is more efficient as it is able to sweep the large box to the side. The pick-and-place plan needs to move more objects and takes more time to execute. The planning time is also more for the pick-and-place planner (27.8 sec vs. 16.6 sec) as it needs to plan more actions.

Fig. 16 An example plan to retrieve a can behind a large ungraspable box on a shelf. The robot first pushes the ungraspable large box to the side and then reaches in to get the can out.

3.2.2 Addressing uncertainty

One of the advantages of using pushing is that pushing actions can account for much higher uncertainty than direct grasping approaches. To demonstrate this we created scenes where we applied high uncertainty to the detected object poses. Fig. 2 presents an example scene. Here the objects have an uncertainty region with σ: (0.02m, 0.05rad) and $n = 20$. The pick-and-place planner fails to find a plan in this scene too, as it cannot find a way to guarantee the grasp of the objects with such high uncertainty. The pushing planner generates plans even with the high uncertainty.

4 Discussion and Future Work

We presented a framework for manipulation in clutter. This framework uses non-prehensile, as well as prehensile, actions to rearrange the environment. In this section we present a discussion around the limitations of this framework and possible extensions to it.

Conservative nature of backward chaining: Our planner starts from the goal and plans backwards. At any step we take into account all uncertainty associated with previously planned actions. This is reflected in the negative goal regions for our planner. When the uncertainty about the consequence of an action is large, this is reflected as a large negative goal region in the following planning steps. If the NGR becomes too large, the planner can run out of space to move objects to. This is a result of our planner being conservative with respect to uncertainty. In future work, we will explore the idea of risk-taking actions as a solution to this problem.

Unknown spaces: Clutter does not only block a robot's links from reaching into certain parts of the space, but it also blocks its sensors (e.g. the cameras and laser scanners). In other words clutter does not only create unreachable spaces but also creates invisible or *unknown spaces*. If the robot cannot see behind a large box, there is no way for it to know if there is another object sitting there. If the robot needs to use the space behind that box, what can it do? Our existing framework assumes that these spaces are free. This assumption does not have to be correct and can result in failures during execution. To solve this problem we can take an active approach where the robot manipulates objects to see the space behind.

Actions with Sensor Feedback: The actions presented in this paper are open-loop. To guarantee success they are conservative with respect to uncertainty. In future work we plan to use sensor feedback during pushing. One challenge when implementing this push-grasping strategy will be finding a good flow of sensor feedback. Cameras on a robot's head are usually obstructed by the robot hand during manipulation. Therefore our main focus will be using tactile sensors and force/torque sensors on the end-effector of the robot.

Other non-prehensile actions The framework we present in this paper opens up the possibility to use different non-prehensile manipulation actions as a part of the same planner. Therefore we view different non-prehensile actions, such as the ones described below, as possible primitives that can be integrated into our framework. Lynch (1999b) uses *toppling* as a manipulation primitive. Berretty et al (2001) presents an algorithm to plan a series of *pulling* actions to orient polygons. Diankov et al (2008) use *caging* to open doors as an alternative to grasping the handle rigidly. Chang et al (2010) present a system that plans to rotate an object on the support surface, before grasping it. Omrcen et al (2009) propose a method to learn the effect of pushing actions on objects and then use these actions to bring an object to the edge of a table for successful grasping.

Pushing Multiple Objects Simultaneously: When a robot rearranges clutter using our existing framework it manipulates objects one by one. If there are n objects blocking the way to achieve the primary goal, the robot executes n distinct actions to move them. If the robot could simultaneously move some of these objects out of the way, it would achieve its goal in a faster way. In future work we plan to use pushing actions that can move multiple objects simultaneously.

5 Conclusion

In this paper we presented a framework for manipulation in clutter. The framework consists of a high-level rearrangement planner, and a low-level library of non-prehensile and prehensile actions. We presented the details of how we implement a non-prehensile action, the push-grasp. We plan to extend this framework such that it can use sensor feedback, it can actively look for parts of the space that are occluded, and it can move multiple objects simultaneously in rearranging clutter.

A Computing conservative capture regions

We compute the capture region for a push-grasp using a pushing simulation. This simulation assumes that an object's limit surface can be approximated with an ellipsoid in the force-moment space as in Howe and Cutkosky (1996).

As described in §2.2.2 two parameters of this simulation affect the boundaries of the computed capture region: the pressure distribution between the object and the support surface, ρ; and the coefficient of friction between the robot finger and the object, μ_c. These values are difficult to know and we assume that our robot does not know the exact values for any object.

When we run our simulation we generate capture regions that are conservative with respect to these parameters; i.e. capture regions that work for a range of reasonable values of μ_c and ρ. For μ_c such a reasonable region is given by the values between a very small value (a very slippery contact between the robot finger and the object), and a very high value (a high friction contact). The pressure distribution ρ can take any rotationally symmetric shape, but it is limited by the boundaries of the object making contact with the support surface.

One way to achieve a conservative capture region is by discretizing the values a parameter can take, running the simulation for each of the values, and then intersecting the resulting capture regions to find a capture region that works for all the values. But for certain object shapes we can do better.

For a cylindrical object the conservative capture region can be found simply by running the simulation for specific values of μ_c and ρ. For μ_c if we choose a very high value the computed capture region will also be valid for any lower value. For ρ if we choose a pressure distribution that is completely at the periphery of the cylinder (like a rim), the capture region will be valid for any other rotationally symmetric pressure distribution that is closer to the center of the cylinder. This is equivalent to saying that, as μ_c gets smaller or as ρ gets concentrated around the center, the required pushing distance for the push-grasp will decrease. In this section we prove this claim.

In Fig. 17 we present some of the force and velocity vectors that determine how a cylinder moves under quasi-static pushing. The following notation will be used:

- $\hat{\mathbf{n}}$ is the unit normal at the contact between the finger and the object.
- $\mathbf{f_L}$ and $\mathbf{f_R}$ are the left and right edges of the friction cone. The friction cone is found by drawing the vectors that make the angle $\alpha = \arctan \mu_c$ with $\hat{\mathbf{n}}$.
- $\mathbf{f}$ is the force applied to the object by the finger. The direction of $\mathbf{f}$ is bounded by the friction cone.

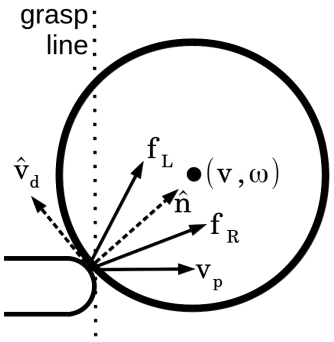


Fig. 17 Some of the force and velocity vectors we will be using in our proof. In the figure the finger is pushing the cylinder towards right.

- $\mathbf{v}$ and $\boldsymbol{\omega}$ are the linear and angular velocities of the object at its center. $\mathbf{v}$ and $\boldsymbol{\omega}$ can be found by computing the force and moment at the center of the object due to $\mathbf{f}$, finding the corresponding point on the limit surface, and taking the normal to the limit surface. Our ellipsoid approximation of the limit surface dictates that $\mathbf{v} \parallel \mathbf{f}$ (Howe and Cutkosky (1996)).
- $\mathbf{m_L}$ and $\mathbf{m_R}$ are the left and right edges of the motion cone. The edges of the motion cone are found by:
  - taking one of the edges of the friction cone, say the left edge;
  - computing the force and moment it creates at the object center;
  - using the limit surface to find the corresponding linear and angular velocity of the object, in response to this force and moment;
  - using the linear and angular velocity at the center of the object to find the velocity at the contact point. This gives the left edge of the motion cone; the right one is found by starting with the right edge of the friction cone.
- $\hat{\mathbf{v}}_\mathbf{d}$ is the unit vector pointing in the opposite direction of $\boldsymbol{\omega} \times \mathbf{r}$ where $\mathbf{r}$ is the vector from the center of the object to the contact; $\hat{\mathbf{v}}_\mathbf{d} = \frac{(\boldsymbol{\omega} \times \mathbf{r})}{|\boldsymbol{\omega} \times \mathbf{r}|}$.
- $\mathbf{v_p}$ is the velocity of the pusher/finger at the contact point. The *voting theorem* (Mason (1986)) states that $\mathbf{v_p}$ and the edges of the friction cone votes on the direction the object will rotate. For a cylinder the friction cone edges always fall on different sides of the center of the object, and $\mathbf{v_p}$ alone dictates the rotation direction; $\mathbf{v_p}.(\boldsymbol{\omega} \times \mathbf{r}) > 0$. In terms of $\hat{\mathbf{v}}_\mathbf{d}$ this means $\mathbf{v_p}.\hat{\mathbf{v}}_\mathbf{d} < 0$.
- $\mathbf{v_c}$ is the velocity of the object at the contact point; $\mathbf{v_c} = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}$.
- The grasp-line is the line at the fingertip orthogonal to the pushing direction. The push-grasp continues until the object center passes the grasp-line.

During quasi-static pushing, the contact between the finger and the object can display three different modes: *separation*, *sticking*, or *sliding*. The case of separation is trivial, in which the finger moves away from the object resulting in no motion for the object. In the case of sticking contact the contact point on the finger and the contact point on the object moves together, i.e. $\mathbf{v_p} = \mathbf{v_c}$. This happens when $\mathbf{f}$ falls inside the friction cone, and correspondingly when $\mathbf{v_c}$ falls inside the motion cone. In sliding contact the object slides on the finger as it is being pushed. In this case $\mathbf{f}$ aligns with the friction cone edge opposing the direction the object is sliding on the finger. Similarly $\mathbf{v_c}$ aligns with the motion cone edge

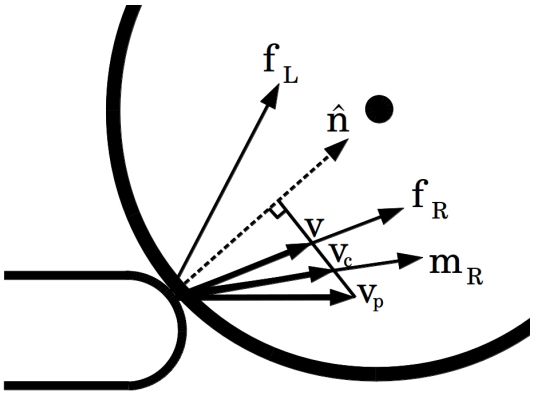


Fig. 18 The relation between $\mathbf{v}$, $\mathbf{v_p}$, and $\mathbf{v_c}$ during sliding contact.

opposing the direction the object is sliding on the finger. $\mathbf{v_p}$ is outside of the motion cone.

What follows is a series of lemmas and their proofs; which we then use to prove our main theorem.

Lemma 1 *During sticking and sliding contact* $\mathbf{v}.\hat{\mathbf{n}} = \mathbf{v_p}.\hat{\mathbf{n}}$.

Proof During sticking contact $\mathbf{v_p} = \mathbf{v_c}$, which implies

$$\mathbf{v_p}.\hat{\mathbf{n}} = \mathbf{v_c}.\hat{\mathbf{n}}$$

We know that $\mathbf{v_c} = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}$. Then,

$$\mathbf{v_p}.\hat{\mathbf{n}} = (\mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}).\hat{\mathbf{n}}$$

Since $(\boldsymbol{\omega} \times \mathbf{r}).\hat{\mathbf{n}} = 0$,

$$\mathbf{v_p}.\hat{\mathbf{n}} = \mathbf{v}.\hat{\mathbf{n}}$$

During sliding contact, the relation between $\mathbf{v_c}$ and $\mathbf{v_p}$ is given by

$$\mathbf{v_c} = \mathbf{v_p} + \mathbf{v_{slide}}$$

where $\mathbf{v_{slide}}$ is the velocity the object slides on the finger. Taking the projection of both sides along the contact normal gives

$$\mathbf{v_c}.\hat{\mathbf{n}} = (\mathbf{v_p} + \mathbf{v_{slide}}).\hat{\mathbf{n}}$$

Sliding can only happen along the contact tangent. For a cylindrical object, this means $\mathbf{v_{slide}}.\hat{\mathbf{n}} = 0$. Then,

$$\mathbf{v_p}.\hat{\mathbf{n}} = \mathbf{v_c}.\hat{\mathbf{n}}$$

This is also illustrated in Fig. 18. The rest of the proof proceeds the same as the sticking contact case. □

Lemma 2 *During sticking contact, as we change* ρ *such that it is concentrated closer to the center of the cylinder, the magnitude of the angular velocity,* $|\boldsymbol{\omega}|$*, will get larger.*

Proof As ρ concentrates at the center, the limit surface ellipsoid gets a more flattened shape at the top and bottom. This implies that for the same force and moment applied to the object, the ratio $|\boldsymbol{\omega}|/|v|$ will get larger (Howe and Cutkosky (1996)).

We can express $|\mathbf{v}|$ as,

$$|\mathbf{v}| = \sqrt{(\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d})^2 + (\mathbf{v}.\hat{\mathbf{n}})^2}$$

and using Lemma 1,

$$|\mathbf{v}| = \sqrt{(\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d})^2 + (\mathbf{v_p}.\hat{\mathbf{n}})^2} \tag{2}$$

During sticking contact $\mathbf{v_p} = \mathbf{v_c}$, hence

$$\mathbf{v_p} = \mathbf{v} + (\boldsymbol{\omega} \times \mathbf{r})$$

Since $(\boldsymbol{\omega} \times \mathbf{r}) = -|\boldsymbol{\omega}||\mathbf{r}|\hat{\mathbf{v}}_\mathbf{d}$ we have

$$\mathbf{v_p} = \mathbf{v} - |\boldsymbol{\omega}||\mathbf{r}|\hat{\mathbf{v}}_\mathbf{d}$$

Rearranging and projecting both sides onto $\hat{\mathbf{v}}_\mathbf{d}$ gives:

$$\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d} = \mathbf{v_p}.\hat{\mathbf{v}}_\mathbf{d} + |\boldsymbol{\omega}||\mathbf{r}|$$

Inserting this into Eq. 2,

$$|\mathbf{v}| = \sqrt{(\mathbf{v_p}.\hat{\mathbf{v}}_\mathbf{d} + |\boldsymbol{\omega}||\mathbf{r}|)^2 + (\mathbf{v_p}.\hat{\mathbf{n}})^2}$$

Except $|\boldsymbol{\omega}|$, the terms on the right hand side are independent of ρ. Since $\mathbf{v_p}.\hat{\mathbf{v}}_\mathbf{d} < 0$, as $|\boldsymbol{\omega}|$ increases $|\mathbf{v}|$ decreases, and vice versa. Then the only way $|\boldsymbol{\omega}|/|\mathbf{v}|$ can increase is when $|\boldsymbol{\omega}|$ increases. □

Lemma 3 *For a given configuration of the object and the finger, if we change the pressure distribution* ρ *such that it is concentrated closer to the center of the object, the change in* $\mathbf{v}$ *will have a non-negative projection on* $\hat{\mathbf{v}}_\mathbf{d}$.

Proof We will look at different contact modes separately. The separation mode is trivial. The object will not move for both values of ρ. The change in $\mathbf{v}$ will have a null projection on $\hat{\mathbf{v}}_\mathbf{d}$.

Assume that the contact mode is sliding. Then $\mathbf{f}$ will be aligned with one of the friction cone edges; let's assume $\mathbf{f_R}$ without loss of generality. Since $\mathbf{v} \parallel \mathbf{f}$, then $\mathbf{v}$ is also a vector with direction $\mathbf{f_R}$

$$\mathbf{v} = |\mathbf{v}|\hat{\mathbf{f}}_\mathbf{R}$$

where $\hat{\mathbf{f}}_\mathbf{R}$ is the unit direction along $\mathbf{f_R}$. Inserting this into the result from Lemma 1 we have

$$|\mathbf{v}|\hat{\mathbf{f}}_\mathbf{R}.\hat{\mathbf{n}} = \mathbf{v_p}.\hat{\mathbf{n}}$$

Then

$$|\mathbf{v}| = \frac{\mathbf{v_p}.\hat{\mathbf{n}}}{\hat{\mathbf{f}}_\mathbf{R}.\hat{\mathbf{n}}}$$

Multiplying both sides with $\hat{f}_R$ we have

$$\mathbf{v} = \frac{\mathbf{v_p}.\hat{\mathbf{n}}}{\hat{\mathbf{f}}_\mathbf{R}.\hat{\mathbf{n}}}\hat{\mathbf{f}}_\mathbf{R}$$

None of the terms get affected by a change in ρ, i.e. the change in $\mathbf{v}$ will have a null projection on $\hat{\mathbf{v}}_\mathbf{d}$.

As ρ concentrates at the center, $|\boldsymbol{\omega}|/|\mathbf{v}|$ will get larger. The motion cone edges will then get more and more aligned with the direction of $\boldsymbol{\omega} \times \mathbf{r}$, making the motion cone wider. At the point when the motion cone edge reaches $\mathbf{v_p}$ the contact is no more a sliding contact but a sticking one.

When the contact is sticking we have $\mathbf{v_p} = \mathbf{v_c} = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}$. Then

$$\mathbf{v} = \mathbf{v_p} - (\boldsymbol{\omega} \times \mathbf{r})$$

If we rewrite $(\boldsymbol{\omega} \times \mathbf{r})$ using the direction $\hat{\mathbf{v}}_\mathbf{d}$, we get

$$\mathbf{v} = \mathbf{v_p} + |\boldsymbol{\omega}||\mathbf{r}|\hat{\mathbf{v}}_\mathbf{d}$$

Except $|\boldsymbol{\omega}|$, the terms on the right hand side are independent of ρ. By Lemma 2, we know that as ρ concentrates around the center of the object, $|\boldsymbol{\omega}|$ increases; i.e. the change in $\mathbf{v}$ has a positive projection on $\hat{\mathbf{v}}_\mathbf{d}$. □

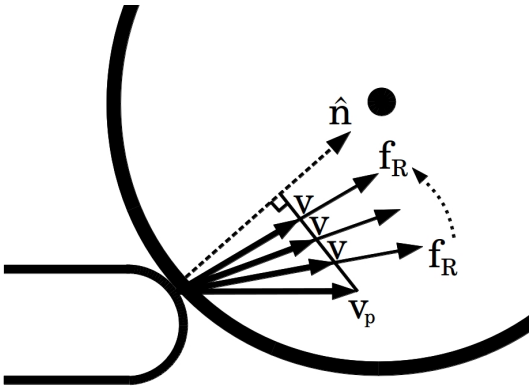


Fig. 19 $\mathbf{v}$ changes along with the edge of the friction cone as μ_c is decreased.

Now we look at the effect of μ_c, the friction coefficient between the object and the finger.

Lemma 4 *For a given configuration of the object and the finger, if we decrease the value of μ_c, the change in $\mathbf{v}$ will have a non-negative projection on $\hat{\mathbf{v}}_\mathbf{d}$.*

Proof Again we look at the two contact modes separately.

The sticking contact case is trivial. $\mathbf{f}$ is inside the friction cone. If we decrease μ_c, the friction cone will get narrower, but as long as it does not get narrow enough to leave $\mathbf{f}$ outside, the contact is still sticking. There is no effect to the velocities of the motion, including $\mathbf{v}$. The change in $\mathbf{v}$ has a null projection on $\hat{\mathbf{v}}_\mathbf{d}$.

If we continue to decrease μ_c at one point the contact will become sliding. $\mathbf{f}$ will be at the edge of the friction cone and the friction cone will get narrower as we decrease μ_c. Without loss of generality, let's assume $\mathbf{f}$ is along $\mathbf{f_R}$. Since $\mathbf{v} \parallel \mathbf{f}$, $\mathbf{v}$ will also move with $\mathbf{f_R}$. Pictorially $\mathbf{v}$ will change as in Fig. 19, resulting in a change along $\hat{\mathbf{v}}_\mathbf{d}$. Formally, in the proof of Lemma 3 we showed that during sliding contact

$$\mathbf{v} = \frac{\mathbf{v_p}.\hat{\mathbf{n}}}{\hat{\mathbf{f}}_\mathbf{R}.\hat{\mathbf{n}}}\hat{\mathbf{f}}_\mathbf{R}$$

By the definition of the friction cone we have

$$\hat{\mathbf{f}}_\mathbf{R} = \cos(\arctan\mu_c)\hat{\mathbf{n}} - \sin(\arctan\mu_c)\hat{\mathbf{v}}_\mathbf{d}$$

Replacing this into the equation above and noting that $\hat{\mathbf{v}}_\mathbf{d}.\hat{\mathbf{n}} = 0$ we have

$$\mathbf{v} = \frac{\mathbf{v_p}.\hat{\mathbf{n}}}{\cos(\arctan\mu_c)}(\cos(\arctan\mu_c)\hat{\mathbf{n}} - \sin(\arctan\mu_c)\hat{\mathbf{v}}_\mathbf{d})$$

Then we have

$$\mathbf{v} = (\mathbf{v_p}.\hat{\mathbf{n}})\hat{\mathbf{n}} - (\mathbf{v_p}.\hat{\mathbf{n}})\mu_c\hat{\mathbf{v}}_\mathbf{d}$$

Except μ_c itself, the terms on the right hand side are independent of μ_c. The contact mode requires that $\mathbf{v_p}.\hat{\mathbf{n}} > 0$. Hence, as μ_c decreases the change in $\mathbf{v}$ will be positive in the direction of $\hat{\mathbf{v}}_\mathbf{d}$.

□

Now we are ready to state and prove our main theorem.

Theorem 1 *For a cylindrical object under quasi-static pushing, where the quasi-static motion is approximated by the ellipsoid limit surface (Howe and Cutkosky (1996)), as μ_c gets smaller or as ρ gets concentrated around the center, the required pushing distance for a push-grasp will decrease or stay the same (but not increase).*

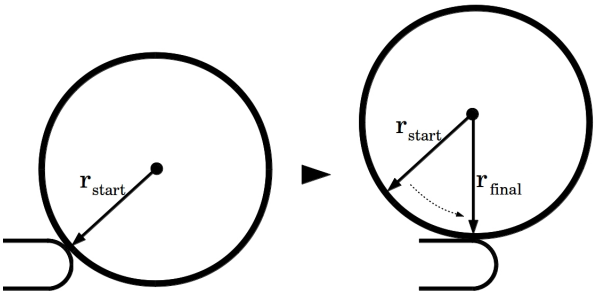


Fig. 20 Independent of μ_c and ρ, the finger and the object goes through the same set of relative configurations during the push-grasp.

Proof The push-grasp starts at a certain configuration between the finger and the object, and continues until the object's center passes the grasp-line at the fingertip and orthogonal to the pushing direction (Fig. 20). Since we assume that μ_c is uniform all around the object, we can ignore the rotation of the cylinder and simply consider its position relative to the finger. Then, independent of ρ or μ_c, the finger and the object will go through all the configurations between $\mathbf{r_{start}}$ to $\mathbf{r_{final}}$ during the push-grasp. We will show below that the velocity the object center moves towards the grasp-line never decreases as μ_c gets smaller or as ρ gets concentrated around the center.

For a given configuration of the object and the finger, the object center's velocity is given by $\mathbf{v}$ ($\boldsymbol{\omega}$ does not have an effect). We can express $\mathbf{v}$ using its components

$$\mathbf{v} = (\mathbf{v}.\hat{\mathbf{n}})\hat{\mathbf{n}} + (\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d})\hat{\mathbf{v}}_\mathbf{d}$$

Lemma 1 tells us that the component of $\mathbf{v}$ along $\hat{\mathbf{n}}$ is fixed for different ρ or μ_c:

$$\mathbf{v} = (\mathbf{v_p}.\hat{\mathbf{n}})\hat{\mathbf{n}} + (\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d})\hat{\mathbf{v}}_\mathbf{d}$$

Hence, the only change in the object center's motion happens along $\hat{\mathbf{v}}_\mathbf{d}$. Lemma 3 and 4 states that the change in $\mathbf{v}$ will be non-negative along $\hat{\mathbf{v}}_\mathbf{d}$. □

Acknowledgements Special thanks to Matt Mason, Chris Atkeson, Tomas Lozano-Perez, Charlie Kemp, Jim Rehg, Alberto Rodriguez and members of the Personal Robotics Lab at Carnegie Mellon University for insightful comments and discussions. This material is based upon work supported by DARPA-BAA-10-28, NSF-IIS-0916557, and NSF-EEC-0540865. Mehmet R. Dogar was partially supported by the Fulbright Science and Technology Fellowship.

References

Agarwal P, Latombe JC, Motwani R, Raghavan P (1997) Nonholonomic path planning for pushing a disk among obstacles. In: IEEE ICRA

Akella S, Mason MT (1998) Posing polygonal objects in the plane by pushing. International Journal of Robotics Research 17(1):70–88

Ben-Shahar O, Rivlin E (1998a) Practical pushing planning for rearrangement tasks. IEEE Transactions on Robotics and Automation 14:549–565

Ben-Shahar O, Rivlin E (1998b) To push or not to push: on the rearrangement of movable objects by a mobile robot. IEEE Transactions on Systems, Man, and Cybernetics, Part B 28(5):667–679

Berenson D, Srinivasa S, Ferguson D, Kuffner J (2009a) Manipulation Planning on Constraint Manifolds. In: IEEE ICRA

Berenson D, Srinivasa SS, Kuffner JJ (2009b) Addressing Pose Uncertainty in Manipulation Planning Using Task Space Regions. In: IEEE/RSJ International Conference on Intelligent Robots and Systems

van den Berg JP, Stilman M, Kuffner J, Lin MC, Manocha D (2008) Path planning among movable obstacles: A probabilistically complete approach. In: In International Workshop on the Algorithmic Foundations of Robotics, pp 599–614

Berretty RP, Goldberg KY, Overmars MH, van der Stappen AF (2001) Orienting parts by inside-out pulling. In: IEEE International Conference on Robotics and Automation

Brost RC (1988) Automatic grasp planning in the presence of uncertainty. International Journal of Robotics Research 7(1)

Chang LY, Srinivasa S, Pollard N (2010) Planning pre-grasp manipulation for transport tasks. In: IEEE ICRA

Chartrand G (1985) Introductory Graph Theory, New York: Dover, chap 6.3, pp 135–139

Chen P, Hwang Y (1991) Practical path planning among movable obstacles. In: IEEE International Conference on Robotics and Automation

Diankov R, Kuffner J (2008) OpenRAVE: A Planning Architecture for Autonomous Robotics. Tech. Rep. CMU-RI-TR-08-34, Robotics Institute

Diankov R, Srinivasa S, Ferguson D, Kuffner J (2008) Manipulation Planning with Caging Grasps. In: Humanoids

Fikes RE, Nilsson NJ (1971) Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3-4):189–208

Goyal S, Ruina A, Papadopoulos J (1991) Planar sliding with dry friction. Part 1. Limit surface and moment function. Wear (143):307–330

Hauser K, Ng-Thow-Hing V (2011) Randomized multi-modal motion planning for a humanoid robot manipulation task. The International Journal of Robotics Research 30(6):678–698

Howe RD, Cutkosky MR (1996) Practical Force-Motion Models for Sliding Manipulation. International Journal of Robotics Research 15(6):557–572

Kaelbling LP, Lozano-Perez T (2011) Hierarchical planning in the now. In: IEEE International Conference on Robotics and Automation

Kappler D, Chang LY, Pollard NS, Asfour T, Dillmann R (2012) Templates for pre-grasp sliding interactions. Robotics and Autonomous Systems 60:411–423

Lavalle SM, Kuffner JJ (2000) Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions

Lynch KM (1999a) Locally controllable manipulation by stable pushing. IEEE Transactions on Robotics and Automation 15(2):318 – 327

Lynch KM (1999b) Toppling Manipulation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 152–159

Lynch KM, Mason MT (1996) Stable Pushing: Mechanics, Controllability, and Planning. IJRR 15(6):533–556

Martinez M, Collet A, Srinivasa S (2010) MOPED: A Scalable and Low Latency Object Recognition and Pose Estimation System. In: IEEE ICRA

Mason MT (1986) Mechanics and Planning of Manipulator Pushing Operations. International Journal of Robotics Research 5(3):53–71

Omrcen D, Boge C, Asfour T, Ude A, Dillmann R (2009) Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In: IEEE-RAS Humanoids, pp 277 –283

Overmars MH, Nieuwenhuisen D, Nieuwenhuisen D, Frank A, Overmars H (2006) An effective framework for path planning amidst movable obstacles. In: In International Workshop on the Algorithmic Foundations of Robotics

Peshkin M, Sanderson A (1988) The motion of a pushed, sliding workpiece. IEEE Journal on Robotics and Automation 4(1):569 – 598

Srinivasa SS, Ferguson D, Helfrich CJ, Berenson D, Collet A, Diankov R, Gallagher G, Hollinger G, Kuffner J, Weghe MV (2009) HERB: a home exploring robotic butler. Autonomous Robots

Stilman M, Kuffner JJ (2006) Planning among movable obstacles with artificial constraints. In: In International Workshop on the Algorithmic Foundations of Robotics, pp 1–20

Stilman M, Schamburek JU, Kuffner J, Asfour T (2007) Manipulation planning among movable obstacles. In: IEEE International Conference on Robotics and Automation

Wilfong G (1988) Motion planning in the presence of movable obstacles. In: Proceedings of the Fourth Annual Symposium on Computational Geometry, pp 279–288

Winograd T (1971) Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Tech. Rep. MAC-TR-84, MIT