Push-Grasping with Dexterous Hands: Mechanics and a Method

Mehmet R. Dogar Siddhartha S. Srinivasa

***Abstract*— We add to a manipulator's capabilities a new primitive motion which we term a push-grasp. While significant progress has been made in robotic grasping of objects and geometric path planning for manipulation, such work treats the world and the object being grasped as immovable, often declaring failure when simple motions of the object could produce success. We analyze the mechanics of push-grasping and present a quasi-static tool that can be used both for analysis and simulation. We utilize this analysis to derive a fast, feasible motion planning algorithm that produces stable push-grasp plans for dexterous hands in the presence of object pose uncertainty and high clutter. We demonstrate our algorithm extensively in simulation and on HERB, a personal robotics platform developed at Intel Labs Pittsburgh.**

I. Introduction

Robotic grasping systems suffer from two main problems in unstructured human environments: *uncertainty* and *clutter*. Consider the task of cleaning a dining table. In such a task the robot needs to detect the objects on the table, figure out where they are, move its arm to reach the goal object, and grasp it to move it away. If there is significant sensor uncertainty, the hand could miss the goal object, or worse, collide with it in an uncontrolled way. Clutter multiplies this problem. Even with perfect sensing, it might be impossible for the hand to wrap around the object for a good grasp. With both clutter and uncertainty, the options for a direct grasp are even more restricted, and often impossible.

We address the problems for grasping in such a context. In particular, we demonstrate how the mechanics of pushing can be harnessed to provably funnel an object into a stable grasp, despite high uncertainty and clutter.

We call this capability *push-grasping*. A push-grasp aims to grasp an object by executing a pushing action and then closing the fingers. We present an example push-grasp in Fig. 1. Here, the robot sweeps a region over the table during which the bottle rolls into its hand, before closing the fingers. The large swept area ensures that the bottle is grasped even if its position is estimated with some error. The push also moves the bottle away from the nearby box, making it possible to wrap the hand around it, which would not have been possible in its original location.

Intuitively, under large uncertainty, the wider the robot opens its fingers and the longer it pushes, the larger the area it can sweep into its grasp. However, this is in direct conflict with avoiding surrounding clutter. Hence, for a successful and efficient push-grasp, we need a detailed analysis enabling the robot to decide on necessary parameters; e.g. the initial hand pose, the pushing distance, and the hand shape.

M. Dogar is with The Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA - 15213, USA. mdogar@cmu.edu

S. Srinivasa is with Intel Labs Pittsburgh, 4720 Forbes Ave., Suite 410, Pittsburgh, PA - 15213, USA siddhartha.srinivasa@intel.com

Fig. 1. An example push-grasp of an object in contact with the surrounding clutter.

In a given scene, to find the right parameters of a push-grasp efficiently, the robot must predict the consequences of the physical interaction. For this purpose, we introduce the concept of a *capture region*, the set of object poses such that a push-grasp successfully grasps it. We compute capture regions for push-grasps using a quasi-static analysis of the mechanics of pushing and a simulation based on this analysis. We show how such a precomputed capture region can be used to efficiently and accurately find the minimum pushing distance needed to grasp an object at a certain pose. Then, given a scene, we use this formalization to search over different parametrizations of a push-grasp, to find collision-free plans.

Our key contribution is the integration of a planning system based on task mechanics to the geometric planners traditionally used in grasping. We enhance the geometric planners by enabling the robot to interact with the world according to physical laws, when needed. Our planner is able to adapt to different levels of uncertainty and clutter, producing direct grasps when the uncertainty and clutter are below a certain level. Our planner infers the consequences of physical interaction with the world, in a fast and conservative manner. We do not assume specific values for the coefficient of friction between the robot and an object or the pressure distribution beneath an object, but we assume reasonable bounds for these parameters. Given our quasi-static assumption, the robot does not need to know the object masses or the coefficient of friction between the object and the supporting surface.

We demonstrate our results both in simulation and in real

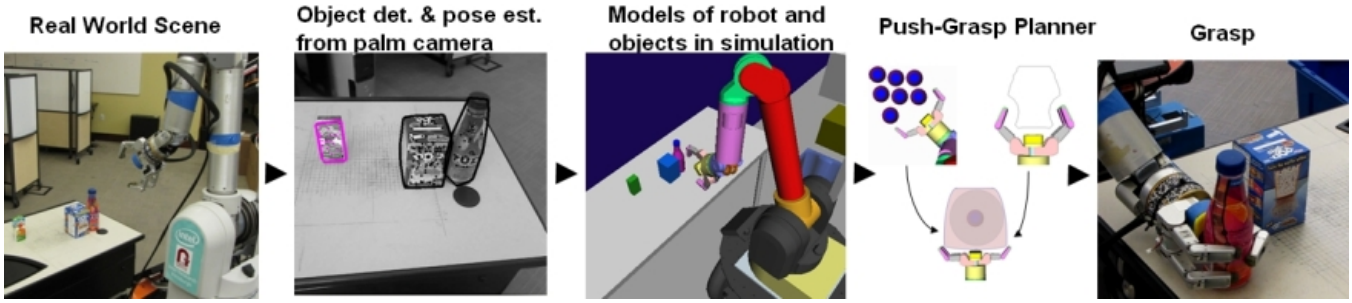


Fig. 2. Overview of our framework

world experiments with HERB, a personal robotics platform developed at Intel Labs Pittsburgh, using the framework illustrated in Fig. 2. Our robot detects and estimates the poses of the objects in the environment with a camera and a computer vision system using SIFT features. Predefined 3D models of the detected objects are inserted into the simulation environment at the estimated poses. A measure of uncertainty can be associated with each of these objects, reflecting the expected error in their estimated poses. Our planner searches for a feasible push-grasp, performing the necessary 3D collision-checking in the simulated environment. We make sure that a push-grasp is executable by the robot arm, checking if the inverse-kinematic solutions exist and are smooth. The robot also plans a collision-free arm trajectory, using an RRT-based planner, to move the hand to the initial pose of the push-grasp. After the hand reaches this initial pose, the robot executes the push-grasp.

II. Background

There is a large body of work addressing the problem of planning to grasp an object using a robot arm. The problem is usually solved by first identifying grasping goal configurations for the robot manipulator, and then planning a path from the initial configuration of the robot arm to these goal configurations. Planning algorithms like probabilistic roadmaps (PRMs) (Kavraki and Latombe [1]) or rapidly exploring random trees (RRTs) (Lavalle and Kuffner [2]) can be used in planning such a path. Using PRMs, Siméon et al. [3] propose a planner that produce transfer/transit paths separated by grasp/ungrasp actions. Berenson et al. [4] extend the RRT algorithm by integrating into the planning process, the sampling of goal configurations from a continuous range of grasping configurations for the object (called task space regions, TSRs). These planners deal with clutter through collision checking to avoid contact with the objects during the execution of a path. The grasping goal configurations are also collision checked before trying to plan to them. The objects are treated as immovable until the point that a complete grasp is acquired, and contact with any object is avoided until that point.

For our planner, there is an explicit time window, during which the robot does not have the complete grasp of the object, but is in contact with it and moving it. In our experiments, as a part of the larger framework, we also use an RRT-based planner, but we do not plan a specific grasping configuration of the hand. First, the push-grasp planner chooses an initial pose of the hand to start pushing, and then we use the RRT planner to plan to reach that initial pushing configuration. Interacting with an object prior to grasping is also studied under the name of *pre-grasp manipulation* and is shown to be useful. For instance, Chang et al. [5] use pre-grasp object rotation to increase grasping task performance and to decrease the load on the arm during the lifting action.

There have been different approaches in addressing uncertainty in object manipulation. One approach tries to come up with robust actions that work even under the given uncertainty; by computing grasps that are robust to disturbances (Nguyen [6]), or using the preimage backchaining work of Lozano-Perez et al. [7]. Uncertainty TSRs by Berenson et al. [4], also mentioned above, address uncertainty by taking different pose hypotheses for an object, and intersect the TSRs of all those pose hypotheses to find a hand pose that grasps all the hypothetical objects. The second approach to addressing uncertainty in object manipulation plans motions that actively reduce the uncertainty using the task mechanics. Brost [8] presents an algorithm that plans parallel-jaw grasping motions for polygonal objects with pose uncertainty, and coins the term *push-grasping*. The object is pushed by one plate towards the second one, and then squeezed between the two. Mason [9] investigates the mechanics and planning of pushing in object manipulation under uncertainty. One of the first planners that incorporates the mechanics of pushing was developed by Lynch and Mason [10]. This planner is able to push an object in a stable manner using edge-edge contact to a goal position, using a quasi-static analysis of the mechanics of pushing. This approach is the inspiration for our work. We use a similar analysis to Lynch and Mason [10], tailored for dexterous hands and grasping.

III. The Mechanics of Pushing

Here we review the previous work, that we directly use in implementing our pushing simulation.

When pushing an object with a robot finger, one question is whether the object will roll into or out of the hand. Mason [9] develops the *voting theorem* stating that the pushing direction and the edges of the friction cone at the contact determine the sense of rotation for a pushed body. We can use the voting theorem to immediately reject pushing if the rotation sense indicates the object will roll out of the hand.

Goyal et al. [11] show that, in the quasi-static case, the motion of a pushed object is determined by the *limit surface*. The limit surface is a three-dimensional surface in the force-torque (f_x, f_y, τ_{oz}) space. Given a point on the limit surface, the motion of the object can be computed by

taking the normal to that point on the limit surface. However, building the limit surface analytically may not be possible for the general object geometry. Also, it depends on the pressure distribution underneath the object, which is usually not known.

Howe and Cutkosky [12] show that the limit surface can be approximated by a three-dimensional ellipsoid. We use their model to simulate the motion of a pushed object. If we place the origin of our coordinate frame at the center of friction of the object this ellipsoid will be centered at the origin. 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 quasi-static assumption holds, we do not need to know the object mass or μ_s to predict the motion.

Howe and Cutkosky [12] also show that if the pressure distribution underneath an object is concentrated at the center, the object has a rotational tendency; and if it is concentrated at the periphery, the object has translational tendency. For push-grasping, rotational velocity of the pushed object is more desirable than translational velocity, as more translation of the object would require the robot to push for longer distances. Given an object, our planner assumes that the pressure distribution is at the object's periphery. This is a conservative estimate for our planner: the pushing motion that rolls this object into the hand will also roll other objects of the same shape with a different pressure distribution.

We make a similar worst-case assumption for the coefficient of friction between the robot finger and the pushed object (μ_c). A large μ_c will let the fingertip apply forces to the object along its pushing direction, causing the object to translate with the fingertip; a smaller μ_c will create a slippery contact and the object will roll in more quickly. Our planner assumes a high μ_c, such that any lower value will work.

IV. Push-Grasping

In this section, we demonstrate how the mechanics of pushing described above can be extended to produce capture regions for real-world objects with dexterous robot hands.

A. 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. 1). We parametrize (Fig. 3(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.

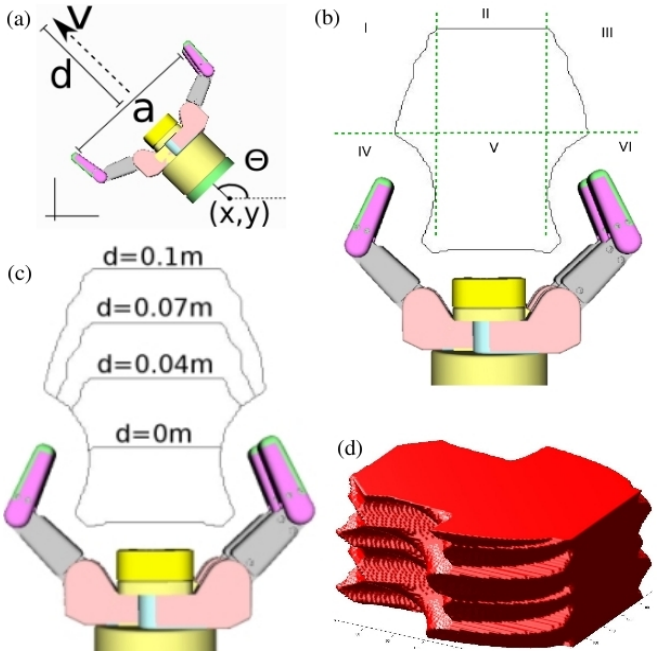


Fig. 3. *(a)* Parametrization of a push-grasp. *(b)* The capture region of a radially symmetric bottle is the area bounded by the black curve. We divided the plane into different regions using the green dashed lines. *(c)* Capture regions for push-grasps of different distances. *(d)* 3D capture region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

- 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.

B. 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 present the capture region of a juice bottle produced by our pushing simulation in Fig. 3(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.

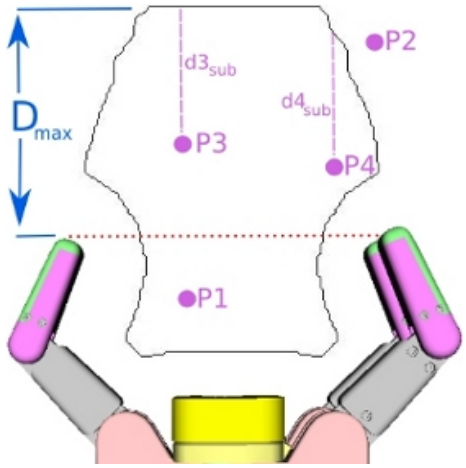


Fig. 4. Given an object pose, the minimum required pushing distance d to grasp that object can be found using a precomputed capture region of a push-grasp with pushing distance D_{max}. In the figure, $d = 0$ for P1 since it is already in the hand; P2 can not be grasped with a push shorter than D_{max} 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.

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. 3(d). We compute it by computing the two-dimensional regions of the object at different orientations. In §V, we will talk about the possibility of doing inclusion checks between capture regions and other three-dimensional regions. Fig. 3(d) shows that capture regions, in general, do not have trivial geometry; hence they are not suitable for fast inclusion checks.

C. 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 $C(G(p_h, a, d), O) = T(p_h)C(G(0_h, a, d), O)$.

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. 3(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. 3(b)) is shifted down in the vertical axis. To understand why this is the case, one can think of the last part of a long push 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. 4). The cases to handle are:

- If the object is already inside the hand (see *P1* in Fig. 4), no push is required; $d = 0m$.
- Else, if the object is outside the capture region (see *P2* in Fig. 4) 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. 4). 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.

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. 3(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.

D. Validating Capture Regions

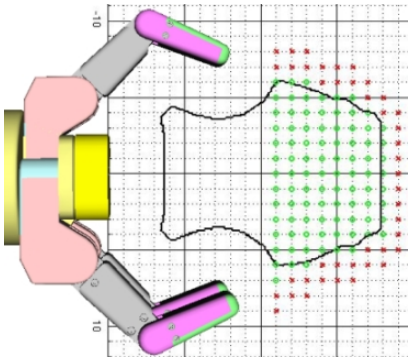
We ran 150 real robot experiments to determine if the precomputed models were good representations of the motion of a pushed object, and whether they were really conservative about which objects will roll into the hand during a push.

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. 5(b)). 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. 5(c).

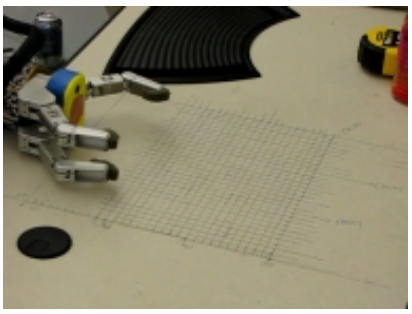
The results (Fig. 5(a)) show that, the simulated capture region is a conservative model of the real capture region.There are object poses outside the region for which the real object rolled into the hand (green circles outside the black curve); but there are no object poses inside the curve for which 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.

V. Object Pose Uncertainty

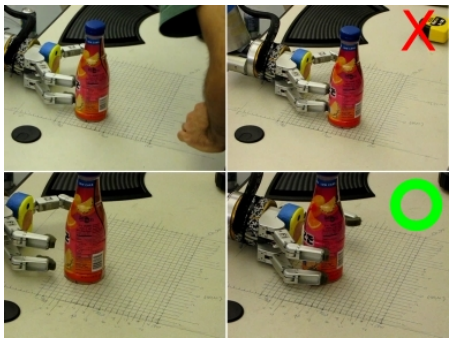
For a robot, object poses are often not exactly known. This section explains how we represent uncertainty about object poses and their relationship to capture regions.



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

A. Object Poses and Uncertainty Regions

Errors produced by a pose estimation system can usually be modeled, either analytically or by collecting statistics on deviations from ground truth. Then, given an estimate we can represent the uncertainty as a probability distribution. This distribution is six-dimensional in general, but for push-grasping we assume that the objects are on a surface and their poses are described as a three dimensional vector (x, y, θ).

Continuous probability distributions, in general, can extend to infinity. In that case we define the *uncertainty region* about an object pose to be the region bounded by a certain isocontour of the probability distribution. If the distribution is already bounded, we define it as the *uncertainty region*.

B. Overlapping Uncertainty and Capture Regions

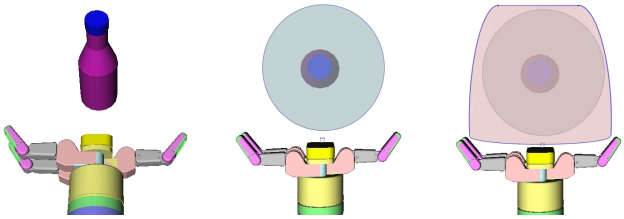
The overlap between a capture region and an uncertainty region indicates whether a push-grasp will succeed under uncertainty. To guarantee that a push-grasp will succeed it is sufficient to make sure that the uncertainty region of the goal object is included in the capture region of the push-grasp, assuming that there is no other clutter.

We illustrate this idea in Fig. 6. Here the robot detects a juice bottle (Fig. 6(a)). We illustrate the uncertainty region of the juice bottle in Fig. 6(b), and the capture region of the push-grasp in Fig. 6(c). If the uncertainty region is completely included in the capture region as in Fig. 6(c), then we can guarantee that the push-grasp will succeed.

The uncertainty and capture regions are two-dimensional in Fig. 6 only because the bottle is radially symmetric. In general, these regions are three-dimensional, nonconvex and potentially even disjoint (e.g. multimodal 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.

Using this sampling strategy, we can not guarantee the success of a push-grasp anymore, since we are not checking the inclusion of the full uncertainty region but only of samples from it. On the positive side, probabilities of poses play a role, contrary to taking all the region as a uniform distribution. The number of samples n we draw is an important parameter here that can be tuned. If n is large, we approach guaranteeing the success of a push-grasp, but we may be acting too conservatively and the planning may take longer. If n is small, the planning will be fast, but we move away from guaranteeing the success of a push-grasp.



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

Fig. 6. If the uncertainty region of an object is included in the capture region of a push-grasp, then the push-grasp will be successful.

VI. A Push-Grasp Planner

This section details our *push-grasp planner*. Given an environment, the planner computes a collision-free trajectory for the robot arm and hand that can perform the successful push-grasp of a desired object.

A. Finding a successful push-grasp

The planner searches for a push-grasp such that (i) it can grasp all the samples drawn from the uncertainty region of the goal object; (ii) the hand does not collide with any samples from the uncertainty regions of the obstacle objects; (iii) 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 planner is presented in Algorithm 1. The planner starts with loading the precomputed object capture region (line 1), and sampling from the uncertainty region of

Algorithm 1: t ← PlanPushGrasp(goalObject, obstacleObjects)

```
1  c ← goalObject.captureRegion;
2  gSamples ← Sample(goalObject, n);
3  oSamples_i ← Sample(obstacleObjects_i, n);
4  while {v, a, o} ← GetNextParam(goalObject) do
5      p ← FindInitialHandPose(goalObject, v, a, o);
6      max_d ← NULL;
7      for i ← 1 to n do
8          if IsInCaptureRegion(p, a, gSamples_i, c) then
9              d_i ← PushDistNeeded(p,a,gSamples_i,c);
10             max_d ← max(max_d, d_i);
11         else
12             max_d ← NULL;
13             break;
14         end
15     end
16     if max_d != NULL then
17         d ← max_d;
18         t ← GenerateTraj(p,a,d);
19         if CheckIK(t) and CollisionFree(t, oSamples)
           then
20             return t;
21         end
22     end
23 end
```

the goal and obstacle objects (lines 2-3). The planner loops over different parametrizations of the push-grasp relative to the goal object (line 4). For each such parametrization, first, an initial hand pose is found which is not in collision with any object samples (line 5). Here, the FindInitialHandPose function returns a hand pose ($p = (x, y, \theta)$) by first placing the hand over the goal object with direction v, offsetting in the perpendicular direction by o, and then backing up in the $-v$ direction until it is collision-free. Lines 6-15 checks whether it is possible to grasp all the goal object samples from this initial hand pose. If it is possible, the pushing distance needed is computed. The IsInCaptureRegion function returns *true* if the sample is in the capture region.

The maximum of the push distances required by all the goal samples is set as the push distance d (line 17), to ensure that all samples end up in the hand. A trajectory is generated from the initial pose p, with aperture a, and push distance d (line 18). Then we check if a smooth inverse kinematic solution for the trajectory exists, and also check for collision with the environment and the obstacle object samples.

One potential problem this planner does not deal with is the object-to-object contacts. In principle this can be handled by extending the precomputed object models with the trajectory the pushed object travels. Then, during planning we can check for collision at each point on this trajectory. This would increase the number of collision-checks needed, though. In general, the volume of space that the pushed object sweeps but the hand does not is very small. Hence object-to-object contacts are rarely a real problem, or are already handled by the hand-to-environment collision check.

TABLE I
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

VII. RESULTS

This section presents extensive experiments in simulation and on HERB to evaluate the performance of our planner. Simulation experiments are performed in OpenRAVE [13].

A. Robotic Platform

In this study we use the robotic platform HERB [14] developed at Intel Labs Pittsburgh. HERB has a 7-DoF WAM arm, and a 4-DoF Barrett hand with three fingers. A camera is attached to the palm to detect objects and estimate their poses. We use the vision system from Collet et al. [15].

B. Planner performance

We compared the performance of our 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. [4].

In §I we described how TSRs use hypotheses about object poses to address uncertainty. 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.

Table I 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 I 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 normal distributions as the uncertainty regions. For different uncertainty levels the standard deviations in object translation and rotation are: 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 I 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

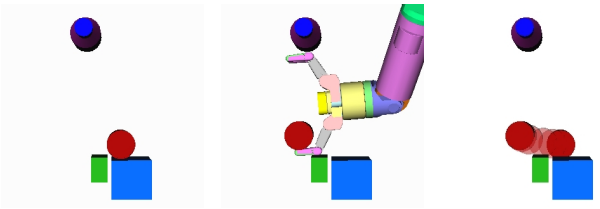


Fig. 7. A high-clutter scene where the TSR planner fails but push-grasp planner is able to find a plan.

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 I 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. 7. 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. 7, which moves the object away from the clutter first and then grasps.

The planning times also shown in Table I. The push-grasp planner takes more time than the TSR planner. This is due to the fact that it searches a larger space. It is usually able to find a plan in about ten seconds. The planning time increases to be around a minute for difficult scenes.

C. 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 artificial 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. 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 artificial uncertainty by adding noise to the positions of the objects reported by the object detection system. For Gaussian noise with $\sigma = 0.02m$, 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. This shows that with push-grasping the robot can increase its success rate in grasping objects, under high uncertainty about object pose.

Execution of some of the push-grasps can be seen in Fig. 8. Videos of our robot executing push-grasps are online at:

`http://www.cs.cmu.edu/%7Emdogar/pushgrasp`

VIII. Discussion and Future Work

This work is a starting point for a long-term study of the ways forceful interaction can be used in object manipulation. In this section we discuss the opportunities this approach offers, the challenges in realizing these, and the limitations of our current planner.

We present a framework to plan pushing actions on an object in order to increase the grasping performance under uncertainty and clutter. However, a robot can utilize pushing in many other ways for object manipulation, especially in highly cluttered scenes. It is our intention to identify different ways that pushing can be useful, and extend our planner to cover these cases. Possible extensions include:

- *Using pushing to move obstacles out of the way:* Our current planner treats obstacles as hard constraints, with which contact should be avoided at all times. In fact, it is possible that the robot pushes these objects away, either by using separate hand/finger motions, or simultaneously while it is moving towards the goal object.
- *Poking objects out of tight spots:* Our current planner uses a hand configuration where the fingers are placed symmetrically on two sides, so that the push rolls the object directly into the grasp. This may not be possible in very highly cluttered environments. An example case is when a small object is stuck between two immovable obstacles, where the robot would need to stick its fingers between the obstacles and poke the object out, before grasping it.
- *Sweep multiple objects simultaneously:* When there is very high clutter the robot can perform a large sweeping motion using its arm and hand to move many objects out of the way simultaneously.
- *Two-handed pushing:* A two-armed robot can use different pushing strategies, including using one of the hands as a barrier and using the other to push objects towards this barrier. This is similar to parallel-jaw grasping using pushing (see Brost [8]).

Our work was also inspired from the way humans grasp objects. Humans engage in pushing-like non-grasping interactions with objects continuously in everyday life, and utilize these as different strategies in manipulating objects under clutter. Identifying these strategies is important, as they can inspire the demonstration of similar capabilities on robots. We plan to conduct a study in future work, to investigate how humans use pushing actions in different settings.

Humans use continuous feedback during their interaction, and we also plan to extend our study to use a closed loop control of pushing using tactile and visual feedback.

Our current implementation computes the capture regions using a particular pushing height on the object. However, one can compute capture regions for different heights of the object, and plan to push objects at a higher or lower point. Toppling can be a problem if the object is pushed too high though, and we came across this problem with one of our objects (for an analysis of using toppling in manipulation, see Lynch [16]).

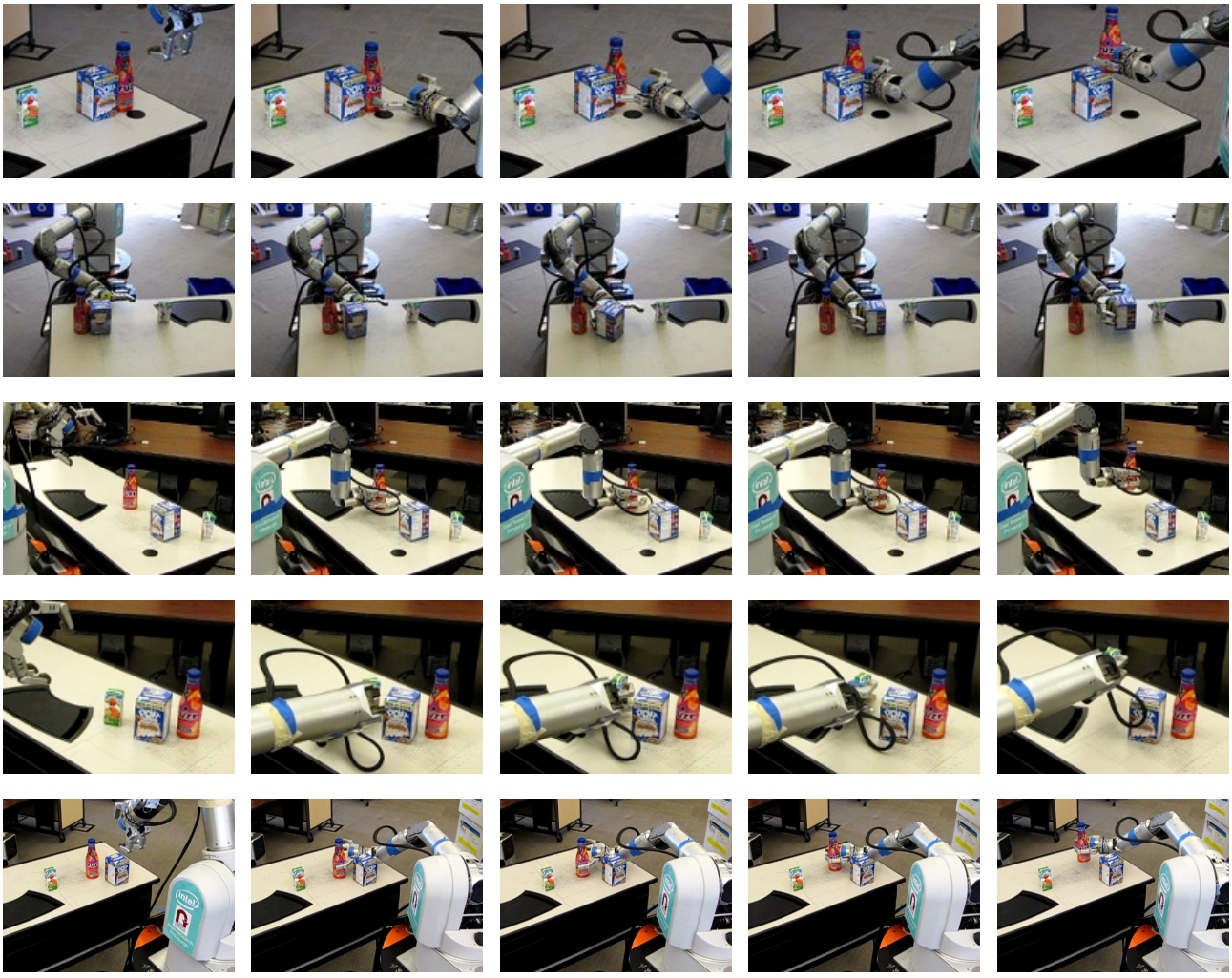


Fig. 8. Example push-grasps executed by our robot.

IX. Conclusion

In this work we enhance the use of geometric planners in object grasping by using insights from task mechanics. We present a push-grasp planner that can reduce the uncertainty about an object's pose by acting on it, and can deal with clutter by moving the object away from clutter before acquiring a complete grasp. Our planning times are reasonable, and our approach is generalizable to other robots.

X. Acknowledgments

This material is based upon work partially supported by the National Science Foundation under Grant No. EEC-0540865 and IIS-0916557. Mehmet R. Dogar is partially supported by the Fulbright Science and Technology Fellowship. Special thanks to Chris Atkeson, Charlie Kemp, Matt Mason, Jim Rehg, and members of the Personal Robotics project at Intel Labs Pittsburgh for insightful comments and discussions.

References

[1] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *ICRA*, 1994.

[2] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000.

[3] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *IJRR*, vol. 23, no. 7-8, 2004.

[4] D. Berenson, S. S. Srinivasa, and J. J. Kuffner, "Addressing Pose Uncertainty in Manipulation Planning Using Task Space Regions," in *"IROS"*, 2009.

[5] L. Y. Chang, S. Srinivasa, and N. Pollard, "Planning pre-grasp manipulation for transport tasks," in *"ICRA"*, 2010.

[6] V.-D. Nguyen, "Constructing stable grasps," *IJRR*, vol. 8, no. 1, 1989.

[7] T. Lozano-Perez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *IJRR*, vol. 3, no. 1, 1984.

[8] R. C. Brost, "Automatic grasp planning in the presence of uncertainty," *IJRR*, vol. 7, no. 1, 1988.

[9] M. T. Mason, "Mechanics and Planning of Manipulator Pushing Operations," *IJRR*, vol. 5, no. 3, pp. 53–71, 1986.

[10] K. M. Lynch and M. T. Mason, "Stable Pushing: Mechanics, Controllability, and Planning," *IJRR*, vol. 15, no. 6, pp. 533–556, 1996.

[11] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction. Part 1. Limit surface and moment function." *Wear*, no. 143, pp. 307–330, 1991.

[12] R. D. Howe and M. R. Cutkosky, "Practical Force-Motion Models for Sliding Manipulation," *IJRR*, vol. 15, no. 6, pp. 557–572, 1996.

[13] R. Diankov and J. Kuffner, "OpenRAVE: A Planning Architecture for Autonomous Robotics," Robotics Institute, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[14] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, "HERB: a home exploring robotic butler," *Autonomous Robots*, 2009.

[15] A. Collet, D. Berenson, S. Srinivasa, and D. Ferguson, "Object recognition and full pose registration from a single image for robotic manipulation," in *ICRA*, 2009.

[16] K. M. Lynch, "Toppling manipulation," in *IROS*, 1999.