Physics-Based Grasp Planning Through Clutter

Mehmet R. Dogar*, Kaijen Hsiao†, Matei Ciocarlie†, Siddhartha S. Srinivasa*
* The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
† Willow Garage, Inc., 68 Willow Road, Menlo Park, CA 94025

***Abstract*—We propose a planning method for grasping in cluttered environments, where the robot can make simultaneous contact with multiple objects, manipulating them in a deliberate and controlled fashion. This enables the robot to reach for and grasp the target while simultaneously contacting and moving aside obstacles in order to clear a desired path. We use a physics-based analysis of pushing to compute the motion of each object in the scene in response to a set of possible robot motions. In order to make the problem computationally tractable, we enable multiple simultaneous robot-object interactions, which we pre-compute and cache, but avoid object-object interactions. Tests on large sets of simulated scenes show that our planner produces more successful grasps in more complex scenes than versions that avoid any interaction with surrounding clutter. Validation on a real robot shows that our grasp evaluation method accurately predicts the outcome of a grasp, and that our approach, in conjunction with state-of-the-art object recognition tools, is applicable in real-life scenes that are highly cluttered and constrained.**

I. Introduction

Robots operating in our homes will inevitably be confronted with scenes that are congested, unorganized, and complex - or, simply put, cluttered. Consider, for example, the following representative problem: the robot must acquire an object from the back of a cluttered bookcase or fridge shelf (Fig. 1, Left). Approaching the target object from the top is impossible due to the constrained space inside the shelf; various obstacles block approaches from the front or side. Some of the obstacles are too large for the robot to grasp. How can the robot complete the task?

Planning for manipulation in clutter requires understanding the consequences of a robot's interaction with a complex scene. The most direct approach to grasping is *object-centric*, in that it separates the scene into two simple categories: the target object vs. everything else. The desired interaction with the target object is to touch it with the end-effector, instantly transitioning into a stable grasp where the object is rigidly attached to the arm. Sets of hand poses and configurations that create stable grasps for the target can be pre-computed [7], [11]. Any interaction with the rest of the scene is utterly avoided, and as such, no consequences must be predicted.

The object-centric approach works well in structured or semi-structured settings where objects are well-separated. However, unstructured and complex environments pose serious challenges: clutter often makes it impossible to guarantee avoiding the rest of the world while reaching for an object.

Recent studies address this problem by using reconfiguration planners that are able to move obstacles out of the way in order to reach the target [6], [19]. The robot uses richer semantic descriptions of a scene, differentiating between immovable and movable obstacles. However, these approaches are still object-centric, in that each individual robot action still addresses a single object, while avoiding all contact with the others. Essentially, the scene is treated like a game of chess, where the robot moves one piece at a time until the goal is achieved.

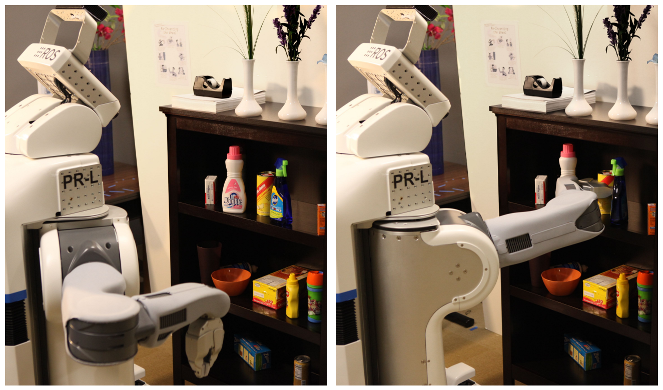

Fig. 1. Manipulation in a constrained and cluttered environment. Left: a robot tries to acquire a partly occluded object from a constrained shelf space. Right: the robot clears a path to the target by pushing obstacles in a controlled manner, and completes the grasp.

In this paper, we propose a *clutter-centric* perspective, where the fundamental action primitives enable simultaneous contact with multiple objects. For grasping, this enables the robot to reach for and grasp the target while simultaneously contacting and moving obstacles, in a controlled manner, in order to clear the desired path (Fig. 1, Right). In addition to simply closing the gripper on the target, interaction with objects in the scene is done through quasi-static pushing, recently used as the basis of the "push-grasping" [5] action. However, while push-grasping avoids all contact with any obstacles, we explicitly represent clutter and reason about how it will interact with the intended grasp trajectory. Rather than shying away from complex and sustained interactions with the world while grasping, the robot uses them to its advantage.

Instead of a static pose relative to a target object, we consider a grasp to also encapsulate a complete trajectory, possibly making and maintaining multiple contacts throughout its execution, and ending with the act of closing the robot's gripper. For predicting the outcome of an intended grasp, we use a physics-based simulation method that computes the motion of the objects in the scene in response to a set of possible robot motions. In order to make the problem computationally tractable, our method allows multiple simultaneous robot-object interactions, which can be pre-computed and cached, but avoids object-object interaction, which would have to be computed at run-time.

We compare this approach against a "static" planner that avoids all contact except for closing the gripper on the target. We also compare it against the object-centric implementation of the original push-grasp planner, which uses non-prehensile manipulation on the target but avoids all obstacles. Our results show that, for a large set of simulated cluttered scenarios, our method returns more possible grasps for a given scene, and succeeds in more scenes. We validate these findings on a real robot, a PR2, showing that the predicted outcomes of simulated tasks match the real-life results. We also show that our approach can be used in conjunction with existing object recognition tools operating on real sensor data, allowing the robots to complete grasping tasks in challenging real-life scenarios such as the above shelf.

II. Physics-Based Grasping

At its core, our approach relies on predicting the interactions between the robot's end-effector and the objects in a scene, as the robot reaches for a target. For a given trajectory of the end-effector, we would like to compute the resulting motion of all contacted objects, including the target and potential obstacles. We can predict these motions using physics-based simulations; however, these simulations are prohibitively expensive from a computational standpoint, and do not scale well with the multiple objects involved: computing contact forces between multiple rigid bodies is an NP-hard problem [1]. Additionally, we need to evaluate many candidate grasps to find one successful grasp, which further increases the computational costs.

As the general space of possible robot-scene interactions is intractable to compute, we believe that a viable path forward is to identify assumptions and constraints that allow us to study parts of this space, increasing the capabilities of our robots. As a first step, we focus on a subset of all possible hand trajectories, comprising linear motion along a pre-defined approach direction relative to the palm. If an object is contacted, this motion results in quasi-static pushing, as the hand continues to advance, with a reduced velocity in order to avoid inertial effects. The mechanics of this interaction have been previously studied [9], [12], [14], [15], and used recently in [5] with the goal of predicting the final pose of the object relative to the hand.

A second step for reducing the computational complexity of physics-based planning is to pre-compute complete object trajectories in response to hand motion along the pushing direction. For each object in our database, we compute trajectories analogous to the ones shown in Fig. 2 for many possible initial object poses relative to the hand. It is important to note that these trajectories are computed for each object in isolation, in the absence of other obstacles.

Finally, at run-time, we evaluate a potential hand trajectory by checking its effect on each object in the scene. We base our approach on this observation: we can evaluate scenarios

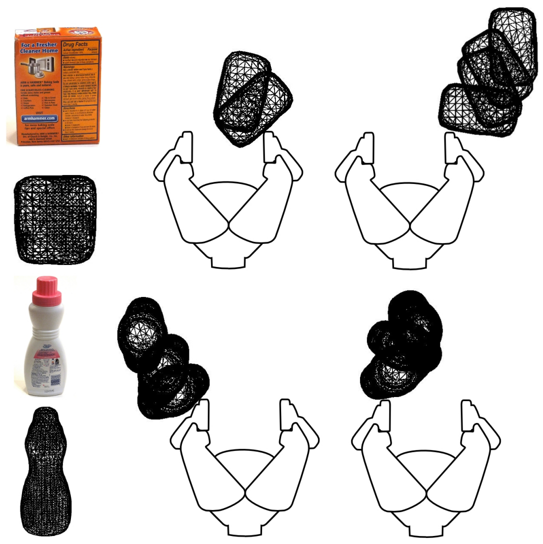
Fig. 2. Left: Example objects and their trimesh models. Right: Examples of computed trajectories of objects as the hand pushes them. The hand motion is towards the top of the page in each diagram, and is not shown.

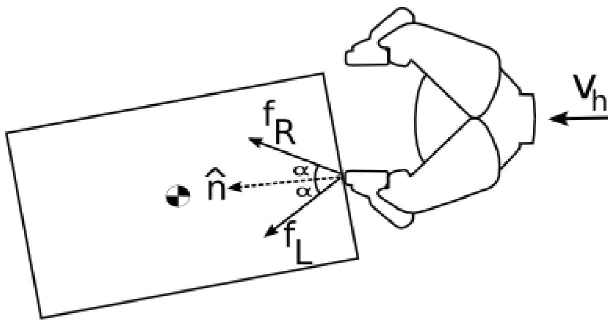

Fig. 3. Notation used for hand-object contacts. The hand's velocity is given by $\mathbf{v_h}$. The friction cone edges, $(\mathbf{f_L}, \mathbf{f_R})$, and the normal $\hat{\mathbf{n}}$ at the contact are illustrated.

in which the hand touches multiple objects as simultaneous individual interactions that are pre-computed, as long as the objects do not touch each other and do not affect the hand's trajectory. We can also detect situations where our conditions are violated, and thus avoid executing trajectories whose side-effects are not fully accounted for.

A. *Quasi-Static Pushing Analysis*

Using the notation in Fig. 3, consider a scene where the robot hand is moving in the direction $\mathbf{v_h}$ and contacts an object. We further assume that the object is resting on a planar support surface parallel to the xy plane, and that both the initial pose of the hand and its velocity are parallel to this plane. We use $\hat{\mathbf{n}}$ to show the normal to the contact, and CoM to show the center of mass of the object. The hand applies a generalized force, $\mathbf{q} = (f_x, f_y, m)$, to the object, causing it to move. Our goal is to compute the resulting generalized object velocity, $\mathbf{p} = (v_x, v_y, \omega)$, represented relative to CoM.

Given the coefficient of friction, μ_c, between the finger and the object, *Coulomb friction* restricts the tangential force f_t as a function of the normal force f_n at the contact:$f_t \leq \mu_c f_n$. It follows that the possible directions of the force, $\mathbf{f} = (f_x, f_y)$, that is applied to the object by the contact is bounded by a *friction cone* [15], defined by edges $\mathbf{f_L}$ and $\mathbf{f_R}$ making an angle $\alpha = \arctan \mu$ with the contact normal $\hat{\mathbf{n}}$. If the object is sliding on the finger as it is being moved, the force applied at the contact lies at these extreme boundaries.

If we can compute the force applied to the object by the hand, we can then convert it into a velocity for the object using the *limit surface* [8], which takes into account the contact between the object and the support surface. This contact occurs over an area rather than a point, allowing for the transmission of both forces in the xy plane and a moment around an axis perpendicular to it. Given a generalized force on the object, we find the corresponding general velocity by taking the normal to the limit surface at the point the generalized force intersects it. Our quasi-static assumption implies that the applied force always lies exactly on the limit surface.

In general, computing the limit surface of an object is not analytically solvable. Howe and Cutkosky [9], however, show that the limit surface can practically be approximated by an ellipsoid centered at the `CoM`. Given this model, we can express how limit surface relates the generalized velocity of the object to the applied generalized force as:

$$\frac{\sqrt{{v_x}^2 + {v_y}^2}}{\omega} = \frac{\sqrt{{f_x}^2 + {f_y}^2}}{m} (\frac{m_{max}}{f_{max}})^2$$

where f_{max} is the maximum force the object can apply to its support surface during sliding and m_{max} is the maximum moment the object can apply about `CoM`, which happens when the object is rotating around `CoM`. We find the ratio of f_{max} to m_{max} by:

$$f_{max} = \mu_s f_n^s$$

$$m_{max} = \mu_s \int_A |\mathbf{x}| p(\mathbf{x}) \, \mathrm{d}A$$

where μ_s is the coefficient of friction between the object and the support surface, f_n^s is the normal force that the support surface applies on the object, A is the area between the object and the support surface, $\mathrm{d}A$ is a differential element of A, $\mathbf{x}$ is the position vector of $\mathrm{d}A$, and $p(\mathbf{x})$ is the pressure at $\mathbf{x}$. Note that μ_s and the mass of the object cancel out when computing the ratio; therefore, they do not play a role in predicting the motion of the pushed object. We still need to know the pressure distribution at the contact, which we discuss in Section II-B.

Another constraint on the relation between the applied force to the object and its motion comes from the fact that the limit surface ellipsoid is a circle at the force plane: the linear velocity of the object $\mathbf{v} = (v_x, v_y)$ is always parallel to the applied force $\mathbf{f} = (f_x, f_y)$ [9].

As mentioned above, when the contact between the finger and the object is sliding, the applied force is at the friction cone edge opposing the direction of relative motion, and the two constraints listed above are sufficient to solve for the corresponding generalized velocity. When the contact is sticking and the applied force is interior to the friction cone, an extra constraint is needed. We can derive one from the fact that, by definition, in the case of sticking the contact point on the object moves with the same velocity $\mathbf{v_h}$ as its counterpart on the finger.

We determine whether the contact will be a sticking or sliding by computing the velocity vectors at the contact point that correspond to the edges of the friction cones: i.e. if the force applied to the object was at a friction cone edge, how would the contact point on the object move? Solving the above constraints for f_L and f_R defines the *motion cone* [15]. If the velocity vector at the contact point on the hand, $\mathbf{v_h}$, lies inside the motion cone, the contact will stick; otherwise, it will slide. See [9], [12] for more details.

B. *Pre-computed Object Trajectories*

We use the quasi-static contact model outlined in the previous subsection to compute *object trajectories* as the hand performs a linear pushing motion. We perform the pre-computation for each object in isolation: throughout its motion, the object's only contacts are with the hand and the support surface. During a push the hand is always parallel to the support surface at a pre-specified height (set at 9 cm for all experiments in this study), and the object is resting on the support surface. Combined with the quasi-static assumption, this means that the resulting motion of the object depends on the geometry of the hand, as well as the object's geometry, frictional and mass properties, and pose relative to the hand at the onset of pushing.

Possible initial poses of an object in the hand's coordinate frame are given by the physically plausible subset of $SE(2)$ for which the object and the hand do not interpenetrate. Furthermore, of interest to us are only the initial object poses for which the hand and the object actually make contact at some point during the hand trajectory. We refer to this set of object poses as the *contact region*:

$$C(o, \tau) = \{c \in SE(2) | \begin{smallmatrix}\text{hand contacts } o \text{ during } \tau \\ \text{if } o \text{ is initially at } c\end{smallmatrix}\}$$

where o is the object, τ is the hand trajectory, and c is the initial pose of the object in the hand's coordinate frame.

To precompute all possible motions of o in response to τ, one can discretize $C(o, \tau)$ (in this study, we used intervals of 5mm and 10 degrees for the discretization), place the object at every resulting pose, simulate the motion of the hand during τ, and record the object trajectory. The resulting set $T_{o,\tau}$ contains the trajectory of the pushed object o for each starting pose $c \in C(o, \tau)$, assuming hand trajectory τ.

In this paper, we execute linear pushing actions with different pushing distances, $\texttt{push}(d)$. We computed $T_{o,\texttt{push}(d)}$ for a very large d; in this study, we used 0.5m. Then, at planning time, we can extract the object trajectories for shorter distances from the trajectories of this long push.

The linearity of our actions simplify the computation of $T_{o,\texttt{push}(d)}$ and reduce the amount of data we need to store. We do not need to place the object at every pose in $C(o, \texttt{push}(d))$. Instead we placed the object only at the poses where the object is *initially* in contact with the hand. There will be cases where the hand contacts the object later during the push. However,

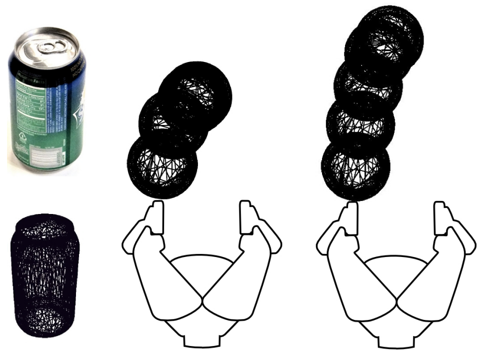

Fig. 4. Left: An example object and its trimesh model. Center: Simulated object trajectory when contact pressure distribution is concentrated at the object's `CoM`. Right: Simulated object trajectory when contact pressure distribution is concentrated at the object's periphery.

we represent these hand trajectories using their last part during which the hand is in contact with the object. The linearity of our actions implies that this last part is also a linear push, for which we already have the object trajectory data. Examples of pre-computed object trajectories can be seen in Fig. 2.

The object's mass distribution also affects the trajectory of the object, as it determines the pressure profile of the object-surface contact. However, this property is difficult to determine accurately, as is the coefficient of friction between the hand and the object. For the hand-surface contact, we assumed that the pressure distribution can take one of two different forms, one where the mass is concentrated on the periphery of the object, and one concentrated on the center of the object; Fig. 4 shows the resulting object motion for each of these two cases (each trajectory stops when the object is inside the hand). For the experiments in this study, we used a fixed value for the hand-object friction coefficient of 0.6. Generalizing these assumptions, future implementations can further discretize the space of these parameters and compute separate trajectories for each resulting point in this space.

We set the simulated robot hand at a specific configuration when we are computing the object trajectories. We would need to compute a new set of object trajectories if we were to set the hand at a different configuration, or if we were to use a different hand.

C. Grasp Evaluation in Clutter

Armed with the pre-computed data described in the previous sub-section, we are ready to evaluate grasps on complete scenes. We consider a scene to be composed of the following elements: a set of objects of known identity comprising both the target and any movable obstacles, a planar surface that the objects are resting on, and additional obstacles for which no additional semantic information is available and are thus treated as immovable.

We define a grasp as the following action: starting from a given gripper pose in the scene, the robot executes a linear gripper trajectory in the pre-defined direction normal to the palm, followed by closing the fingers. We parameterize the space of grasps by the initial pose of the gripper in the scene as well as the length of the linear motion. A successful grasp will result in the target object stably enclosed in the gripper.

Given the pose of the target object in the scene, we first generate a set of possible grasps that approach the object from different approach directions, and at different lateral offsets. The grasps in this set are evaluated in random order using the method below until a feasible grasp is found. At that point, the robot computes a collision-free motion plan to bring the gripper to the initial pose in the linear trajectory, then executes the rest of the grasp open-loop.

We present an illustration of evaluating a grasp in Fig. 5, with an environment constrained by immovable walls on three sides, and populated with three objects, the one in the middle being the target. Let us assume the grasp we evaluate has an approach direction, a, pointing upwards, and a lateral offset, l, a few centimeters to the right (relative to the target object's CoM). Our evaluation algorithm performs the following steps:
1. Compute the initial pose: We place the hand on top of the target object, pointing in the approach direction and applying the given lateral offset. We then back up (in the opposite approach direction) until the hand is collision-free to find the initial pose, p_h, for the grasp trajectory.
2. Compute pushing distance for successful grasp: Starting from p_h, we draw on our database of pre-computed object motions to see what distance d traveled by the gripper along the approach direction, if any, will bring the target object into a position where it can be grasped. In this study, we use the following heuristic: if the center of mass of the object passes behind the line connecting the fingertips of the hand, it is considered to be in a graspable pose. We found this heuristic to work well in practice for parallel grippers; for dexterous hands, checks based on force- or form-closure can be used instead.
3. Check immovable obstacles: Once we know p_h and d, we need to check if the grasp will succeed in the given environment. We first check if the hand will penetrate any non-movable objects (e.g. walls) during the grasp. If yes, we discard the grasp.
4. Check movable obstacles: We continue by identifying the movable objects the hand will make contact with during the grasp. For each of them, we use the respective pre-computed trajectories to predict motion in response to the grasp. However, since these trajectories were computed with the objects in isolation, their validity needs to be maintained. Therefore, if at any point during the grasp, pre-computed trajectories show any object colliding with an obstacle (movable or immovable), the grasp is discarded, as we cannot safely predict the resulting object motions. Fig. 6 shows modifications of the original scene where our example grasp is discarded by this process.
5. Confirm grasp for execution: Once all the tests above have passed, the grasp is deemed safe to execute.

D. Handling Uncertainty

The evaluation algorithm described above is based on knowing the relative locations of the objects in the scene.

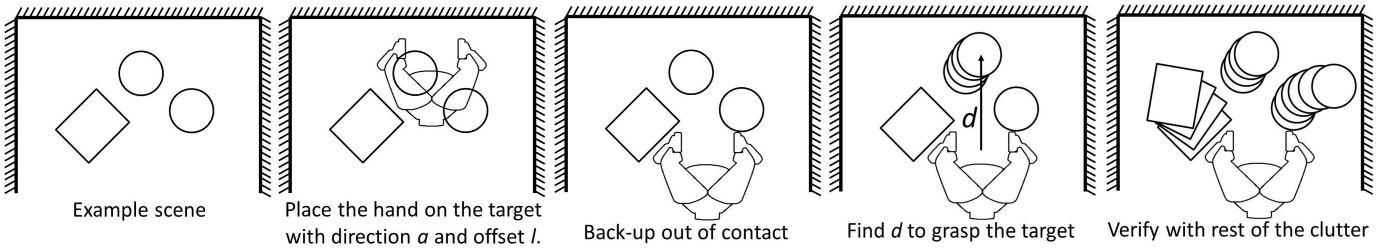
Example scene
Place the hand on the target with direction *a* and offset *l*.
Back-up out of contact
Find *d* to grasp the target
d
Verify with rest of the clutter

Fig. 5. Illustration of evaluating a grasp with a given approach direction, a, and lateral offset, l.

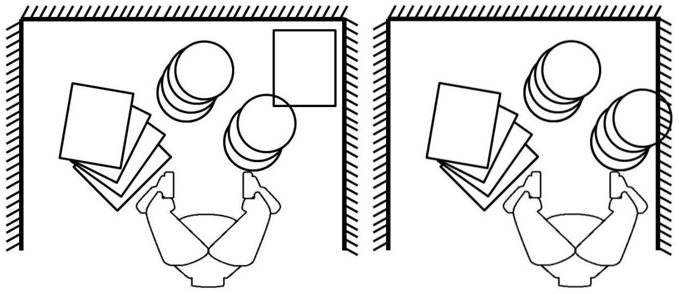

Fig. 6. Illustration of grasps that will be rejected. Left: The pushed object contacts another object. Right: The pushed object contacts a non-movable object (the side-wall).

However, when operating in unstructured environments, these poses are inevitably affected by uncertainty, due to imprecise localization, imperfect robot calibration, *etc*. Previous work has shown how pushing manipulation can be used to reduce uncertainty in the pose of the target object [5]. In our method, we use a similar principle, and extend it to surrounding obstacles.

If any of the parameters affecting the behavior of an object in the scene is uncertain, our planner will generate multiple possible samples for that object, sampling the space of possible values for the uncertain parameter. We refer to these as *uncertainty samples* of the object. We can predict how each of these uncertainty samples will react to gripper contact based on the pre-computed data described in the previous section. Our planner accepts a grasp only if it works for all samples of all objects in the environment.

For avoiding object-object collisions, this implies performing a number of collision tests on the order of $(O \times N)^2$, where O is the number of objects in the scene and N is the number of uncertainty samples for each object. In practice, however, most of the samples in the environment are not moved by the hand, and so we do not need to collision check between them.

III. Planner performance

To better understand how the clutter-grasp planner performs, we quantified its ability to plan valid grasps in randomly-generated, cluttered scenes, and compared its performance to that of two other grasp planners using geometry-only simulation. The grasp planners that we used in our experiments were as follows:

- Clutter-Grasp: The planner presented in this paper.
- Push-Grasp: This planner, similar to the one presented in [5], is constrained to moving only the target object. No surrounding object is allowed to be contacted or moved.
- Static-Grasp: This planner is not allowed to move the target object or any of the other objects in the scene; it is restricted to checking whether the object can be grasped in its current pose without collisions.

TABLE I
Planning Time Comparison

	Time per grasp evaluation (sec)	Time until first successful grasp (sec)
Clutter-Grasp	0.14	0.62
Push-Grasp	0.08	2.43
Static-Grasp	0.03	2.44

We randomly-generated 100 simulated scenes, each with 6 objects that were placed, uniformly at random, in an area of $0.4m \times 0.5m$. Scenes in which objects interpenetrated were rejected and replaced. All three planners use the same search space when planning grasps: all grasps are from the side, and the same search parameters are used for approach direction and lateral offset. Thus, the total number of evaluated grasps is the same for all three planners. For this experiment, we assume zero uncertainty; the main feature of the clutter-grasp planner over the push-grasp planner is its ability to deal with clutter, and so we are primarily concerned with the effect of increasing clutter. The clutter-grasp planner has the same ability as the push-grasp planner to deal with varying levels of uncertainty, as quantified in [5].

Fig. 7 shows how the number of grasps planned for each grasp planner changes when the 100 random scenes are sorted by scene complexity, defined here by the inverse of the number of grasps found by the static-grasp planner. The fewer grasps found by the static-grasp planner, the more cluttered and constrained the environment around the target object; the bottom images in Fig. 7 show example scenes for varying levels of scene complexity. For scenes of medium complexity, the clutter-grasp planner finds more grasps than the other two, although all three planners are able to find some valid grasps. However, for the very most cluttered scenes, only the clutter-grasp planner is able to find valid grasps that allow the robot to shove through the clutter surrounding the target object.

Planning times for the three grasp planners are shown in Table I. The clutter-grasp planner takes longer to evaluate each

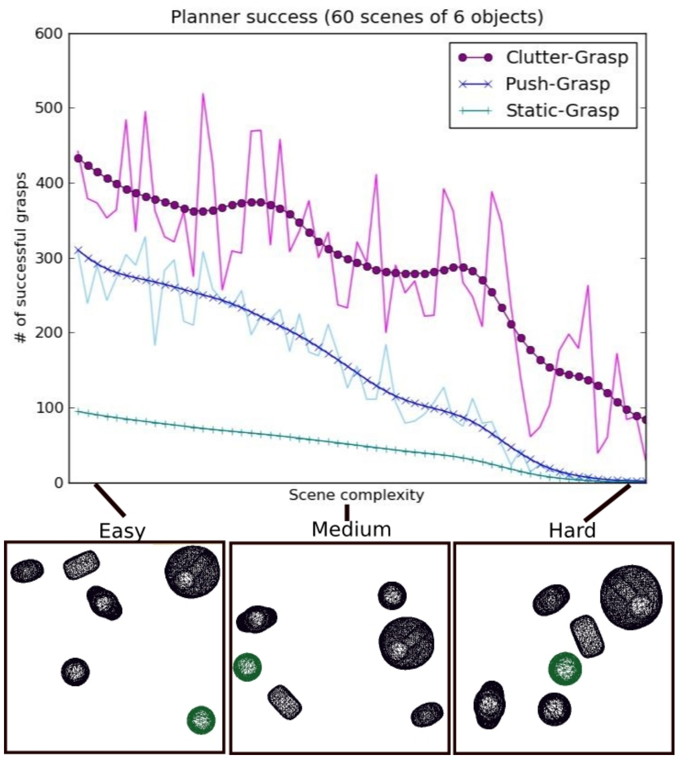

Fig. 7. Top: A comparison of number of grasps found, in randomly-generated scenes sorted by scene complexity (the 60 most complex in the set of 100 generated), for our three grasp planners. Bottom: Bird's eye view of example scenes for varying levels of complexity. The target object is shown in green.

grasp, but because a larger fraction of evaluated grasps are feasible, the average planning time until the first feasible grasp is found is lower than for the other two planners. Planning time until the first feasible grasp is found is also plotted for varying scene complexity in Fig. 8; for the most complex scenes where the push-grasp and static-grasp planners return no feasible grasps, the planning time reflects the time taken to check all grasps in the given search space and then return failure.

IV. Real Robot Experiments

We performed two sets of experiments on an actual two-armed mobile manipulator. The first set of experiments validated the grasps generated by the clutter-grasp planner in our geometry-only planning experiments by running them on a real robot; the second set of experiments tested the performance of the clutter-grasp planner using real-world uncertainty levels from an actual, state-of-the-art object recognition algorithm.

In the first set of real-world experiments, we executed grasps planned by the clutter-grasp planner on a randomly-chosen set of 10 scenes from the set of 100 randomly-generated scenes used in the geometry-only experiments in the previous section. Only scenes with clutter-grasps that were reachable and that had feasible motion plans were chosen; of the 100 total scenes, 78 had grasps within reach of the robot, and all but 6 of those had feasible motion plans for at least one grasp. Real-world objects were placed at the randomly-generated locations on

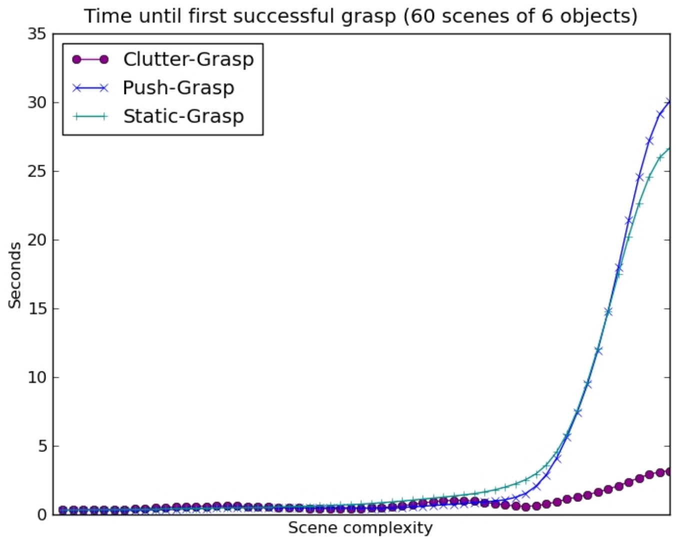

Fig. 8. Time until first grasp for all three planners, for the most complex 60 of the 100 random scenes, sorted by scene complexity.

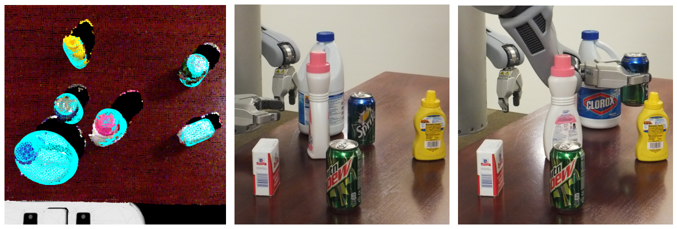

Fig. 9. An example execution of a randomly-generated scene. Left: Object meshes in the randomly-generated scene configuration, along with point cloud from a Kinect^{TM}camera. Middle: Actual scene with gripper at start of push. Right: Actual scene after object has been grasped.

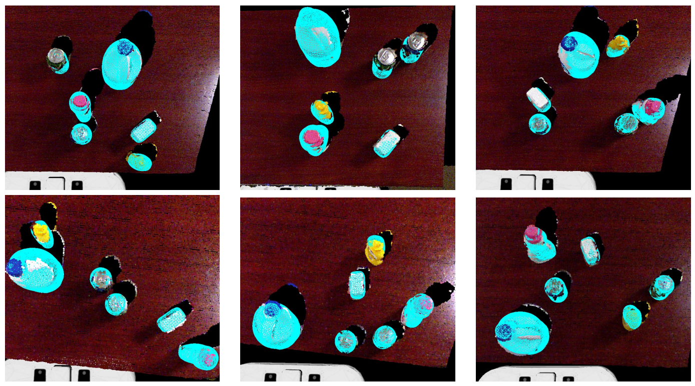

Fig. 10. More randomly-generated scenes with kinematically-feasible clutter-grasps; all attempted clutter-grasps were executed successfully.

the table by carefully aligning their locations as seen by the robot's Kinect^{TM}point cloud with the object meshes rendered at their desired locations using rviz [22], a 3D visualization tool. The resulting point clouds with aligned meshes are shown in Fig. 9 and Fig. 10, along with a sample grasp execution. For these experiments, grasps were generated for uncertainty standard deviations of 1 cm in both translation directions, and 0.1 radians in rotation, which is roughly appropriate for the accuracy of placement. 10 out of 10 grasps planned by the clutter-grasp planner succeeded in picking up the target object, while successfully moving aside obstacles as predicted.

In the second set of experiments, we used the textured object detector described in [17] to recognize objects placed on the shelf shown in Fig. 1. We only used movable objects. The resulting object poses were then fed to the clutter-grasp planner to use in planning grasps for a hand-selected target object. Scenes in which the object detector failed to recognize objects were rejected, but even when the objects are correctly identified, the poses returned by the object detector have significant amounts of pose uncertainty. The resulting object detection results are shown as red point object outlines overlaid on the images seen by the robot's Kinect^{TM}camera, along with the evaluated grasp hypotheses, in the middle images in Fig. 11 and Fig. 12. The clutter-grasp planner succeeded in picking up the target object in all five scenes shown in Fig. 11, and failed to plan grasps in the two scenes shown in Fig. 12. The two shown failures are representative of the most typical failure modes seen by the clutter-grasp planner (not including object detection failure): if objects start out in contact with each other, or if there is no way to shove obstacle objects aside without their hitting other objects or static obstacles such as shelf walls, the clutter-grasp planner will not find any grasps. Also, if there is uncertainty in the object pose, as in this case, both the push-grasp planner and the clutter-grasp planner require a clear space behind the target object so that it can be pushed into the hand; if there is no such space, then no grasps will be found.

Videos of these and other real-world examples of the clutter-grasp planner being used to grasp objects in cluttered scenes can be seen here: www.cs.cmu.edu/%7Emdogar/RSS2012/

V. Conclusion and Future Work

In this paper, we propose a *clutter-centric* perspective to grasp planning, where the action primitive enables simultaneous contact with multiple objects. The robot reaches for and grasps the target while simultaneously contacting and moving obstacles, in a controlled manner, in order to clear the desired path. Rather than shying away from complex and sustained interactions with the world while grasping, the robot uses them to its advantage. We compare our approach to other planners that avoid contact with clutter, and show that our planner is able to find more grasps in a given scene, and also that grasps that interact with the environment succeed in more scenes. Finally, we validate these findings by successfully executing such grasps on both randomly-generated scenes instantiated in real life, and also on scenes for which the input comes from existing object recognition results from real sensor data.

General physics-based analysis of robot interaction with complex scenes is still an open problem, especially when many scenarios must be evluated in a short period of time. Under a set of constraints, we have shown how we can make this problem tractable for certain scenarios. Relaxing these constraints, or removing them altogether, can further extend the range of situations a robot can handle. We outline some directions for future research below.

Object-object contacts. Our planner cannot find grasps where object-object contact will occur, because our pre-computed trajectories do not include these cases. We cannot enumerate all possible configurations of all objects and pre-compute how they move. On the other hand, simulating such interactions online is too time consuming. We believe that a grasp planner can allow object-object contact by using sensor feedback during execution to monitor the acceptability of object motions. Still, our open-loop planner will be essential to provide an initial plan of action and to work even when the sensor data flow is not robust.

Toppling. The current version of our planner assumes objects will not topple when they are pushed. Hence, during our experiments, we limited ourselves to objects that do not topple when they are pushed. A quasi-static analysis, similar to the one we use for pushing in this paper, can be made to analyze the conditions for toppling. Lynch [13] shows that the conditions for toppling can be reduced to geometric conditions, giving limits on the height the object is pushed. A similar reasoning can be incorporated into our planner, and objects that are likely to topple can be avoided by treating them as unmovable obstacles.

Jamming and contact forces. Moving rigid objects in constrained environments will eventually lead to jamming, and thus require control of contact forces. In this respect, a recent study [10] is complimentary to our work: while shoving obstacles or the target object, force monitoring can be useful to ensure that object-sliding is continuing as expected, and that objects are not stuck on unseen barriers.

Reconfiguration planning. High degrees of clutter in very constrained spaces may require clever sequencing of reconfiguration actions. Several algorithms have been proposed [2]–[4], [6], [16], [18]–[20]. The general problem is shown to be NP-hard [21]. Our clutter-grasp planner can easily be added as an additional action primitive in such planners, and by allowing several objects to be manipulated simultaneously, would hopefully reduce both the planning and execution time required.

References

[1] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10(April 1991):292–352, 1993.

[2] O. Ben-Shahar and E. Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation*, 14:549–565, 1998.

[3] O. Ben-Shahar and E. Rivlin. 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, 1998.

[4] P. Chen and Y. Hwang. Practical path planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, pages 444–449, 1991.

[5] M. Dogar and S. Srinivasa. Push-Grasping with Dexterous Hands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2123–2130, October 2010.

[6] M. Dogar and S. Srinivasa. A Framework for Push-Grasping in Clutter. In *Robotics: Science and Systems VII*, 2011.

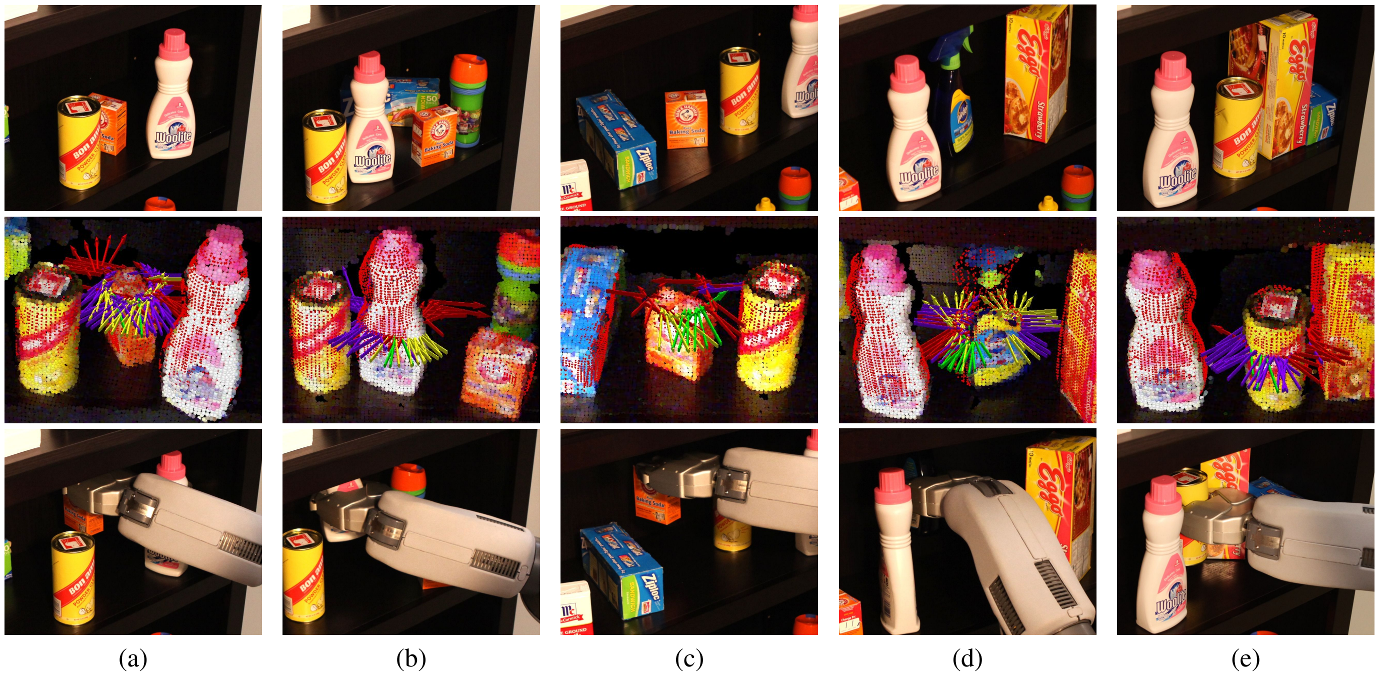
Fig. 11. A wide range of scenes and objects for which our planner succeeds by moving obstacles in a controlled manner. Each **column** shows one grasping task. For each column, **top image**: Initial scene; **middle image**: Scene as seen by the robot, showing an acquired point cloud superimposed with object recognition results (red point object outlines) and evaluated grasping directions (colored arrows); **bottom image**: Final result of successful grasp execution.

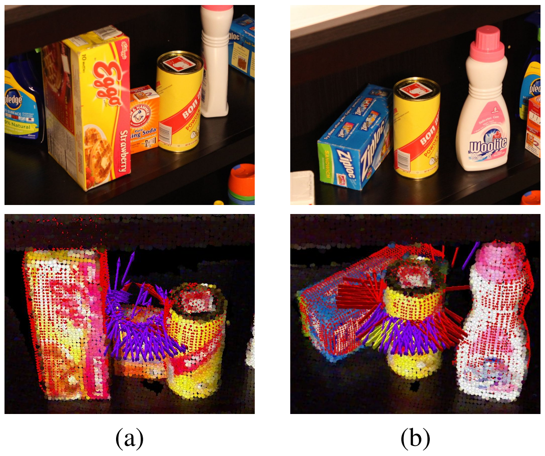
Fig. 12. Examples of cases where our planner cannot find a solution. In (a), multiple objects start out in contact with each other and object-object interaction during pushing is inevitable. In (b), insufficient space behind the target object prevents the execution of sufficiently long pushes.

[7] C. Goldfeder, M. Ciocarlie, H. Dang, and P. Allen. The columbia grasp database. In *IEEE International Conference on Robotics and Automation*, pages 1710–1716, Kobe, Japan, 05 2009.

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

[9] R. D. Howe and M. R. Cutkosky. Practical Force-Motion Models for Sliding Manipulation. *International Journal of Robotics Research*, 15(6):557–572, 1996.

[10] A. Jain, M. D. Killpack, A. Edsinger, and C. C. Kemp. Manipulation in clutter with whole-arm tactile sensing, 2011. Under review.

[11] T. Lozano-Perez and P. H. Winston. Lama: A language for automatic mechanical assembly. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 710–716, 1977.

[12] K. Lynch, H. Maekawa, and K. Tanie. Manipulation And Active Sensing By Pushing Using Tactile Feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 416–421, 1991.

[13] K. M. Lynch. Toppling Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 152–159, 1999.

[14] K. M. Lynch and M. T. Mason. Stable Pushing: Mechanics, Controllability, and Planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

[15] M. T. Mason. Mechanics and Planning of Manipulator Pushing Operations. *International Journal of Robotics Research*, 5(3):53–71, 1986.

[16] M. H. Overmars, D. Nieuwenhuisen, A. Frank, and H. Overmars. An effective framework for path planning amidst movable obstacles. In *In International Workshop on the Algorithmic Foundations of Robotics*, pages 87–102, 2006.

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, 2011.

[18] M. Stilman and J. J. Kuffner. Planning among movable obstacles with artificial constraints. In *In International Workshop on the Algorithmic Foundations of Robotics*, pages 1–20, 2006.

[19] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, pages 3327–3332, 2007.

[20] J. P. van den Berg, M. Stilman, J. Kuffner, M. C. Lin, and D. Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *In International Workshop on the Algorithmic Foundations of Robotics*, pages 599–614, 2008.

[21] G. Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 279–288, 1988.

[22] Willow Garage. rviz wiki page. http://www.ros.org/wiki/rviz, June 2012.