Physics-Based Manipulation Planning in Cluttered Human Environments

Mehmet R. Dogar

CMU-RI-TR-13-20

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

July 2013

Thesis Committee:
Siddhartha S. Srinivasa, Chair (Carnegie Mellon University)
Matthew T. Mason (Carnegie Mellon University)
Christopher G. Atkeson (Carnegie Mellon University)
Tomas Lozano-Perez (Massachusetts Institute of Technology)

This research was supported in part by the Fulbright Science and Technology Fellowship and NSF grant No. EEC-0540865.

Contents

List of Figures

List of Tables

Abstract

This thesis presents a series of planners and algorithms for manipulation in cluttered human environments. The focus is on using physics-based predictions, particularly for pushing operations, as an effective way to address the manipulation challenges posed by these environments.

We introduce push-grasping, a physics-based action to grasp an object first by pushing it and then closing the fingers. We analyze the mechanics of push-grasping and demonstrate its effectiveness under clutter and object pose uncertainty. We integrate a planning system based on push-grasping to the geometric planners traditionally used in grasping. We then show that a similar approach can be used to perform manipulation with environmental contact in cluttered environments. We present a planner where the robot can simultaneously push multiple obstacles out of the way while grasping an object through clutter.

In the second part of this thesis we focus on planning a sequence of actions to manipulate clutter. We present a planning framework to rearrange clutter using prehensile and non-prehensile primitives. We show that our planner succeeds in environments where planners which only use prehensile primitives fail. We then explore the problem of manipulating clutter to search for a hidden object. We formulate the problem as minimizing the expected time to find the target, present two algorithms, and analyze their complexity and optimality.

Acknowledgments

I would like to thank my advisor, Sidd Srinivasa, for his guidance, support, and friendship. I learned so much from you and had so much fun.

I would like to thank Chris Atkeson, who was my co-advisor for the first year, was my committee member, and was always there whenever I needed advice. Thanks to my committee members Matt Mason and Tomas Lozano-Perez for their invaluable comments and guidance.

I would like to thank Jean Harpley and Suzanne Lyons-Muth for making everything so much easier for me with whatever official business I needed to go through.

I had the opportunity to collaborate with great people: Kaijen Hsiao, Matei Ciocarlie, Michael Koval, Abhijeet Tallavajhula, and Bryan Hood. Thank you all for the discussions, the hard work, and the fun.

The Personal Robotics Lab was such a great place to work at. Thanks to those who made it so: Mike Vande Weghe, Dmitry Berenson, Alvaro Collet-Romea, Anca Dragan, Shervin Javdani, Kyle Strabala, Chris Dellin, Mike Dawson-Haggerty, Jen King, Liz Cha, and Pras Velagapudi. Oh, and HERB of course.

Thanks to my family, Ayce, Ikbal, Mustafa, Rukzan, and Mete for their infinite support.

Finally, thanks to Nursel for always being there for me. This would not have been possible without you.

Part I

Introduction

Chapter 1

Introduction

There are striking differences between the way humans and current robots manipulate objects. One difference is in the variety of actions used. The list of actions that we humans use to throw, pour, blow, cut, bend, push, pull, and play with the objects around us is nearly endless (Fig. 1.1). Current robots working in human environments, however, manipulate objects almost exclusively through pick-and-place actions. As a consequence, robots are also limited in the variety of tasks that they can perform.

Robots are limited to pick-and-place actions because they use motion planners which are agnostic to physics. Pick-and-place actions do not require physics models to predict how the manipulated object moves: it is rigidly attached to the hand. However, complex manipulation skills require complex physics-based models to predict how the world behaves. For example, to push a heavy piece of furniture out of the way, a robot needs a physics model that predicts how the furniture will move.

In this thesis we investigate methods to use realistic physics models in manipulation planning. We develop planners that enable robots to physically interact with the environment. We focus on manipulation tasks in human environments with *clutter* and *uncertainty*. Some of these tasks are impossible to perform using only pick-and-place actions.

Within this framework we address several key problems in manipulation.

Environmental Contact: Planning for manipulation in clutter requires understanding the consequences of a robot's interaction with a complex scene. Consider, for example, the scene in Fig. 1.2-a where the robot needs to grasp the circular object. The object is surrounded by others and is impossible to grasp directly without environmental contact. Physics-based predictions enable 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.

Rearrangement: In scenes of high clutter, in order to reach a goal it may be necessary to plan a sequence of manipulation actions that move multiple objects; i.e. plan a rear-

Figure 1.1: Humans use a wide variety of actions to manipulate their environment.

rangement of the scene. Cluttered human environments include a wide variety of objects, and hence different manipulation operations may be required to perform the rearrangement. Consider the example in Fig. 1.2-b, where some of the objects are too large to be graspable. Physics-based actions, such as pushing, make the rearrangement possible.

Searching for an Object: Imagine looking for the salt shaker in a kitchen cabinet. Upon opening the cabinet, you are greeted with a cluttered view of jars, cans, and boxes—but no salt shaker. It must be hidden near the back of the cabinet, completely obscured by the clutter. Robots in human environments should be able to manipulate clutter to search for an object in an efficient way. This requires reasoning about the rearrangement of the environment, as well as the effects of clutter on perception (Fig. 1.2-c).

Uncertainty: In a grasping task, the robot needs to detect the objects, figure out where they are, move its arm to reach one of them, and grasp it to move it away. If there is significant sensor uncertainty, the hand could miss the object, or worse, collide with it in an uncontrolled way. Pushing actions can be useful in planning manipulator operations that are robust to such uncertainty (Fig. 1.2-d).

In approaching these problems we use a physics-based analysis to predict how objects move when they are pushed. We integrate these predictions into a planning algorithms which produce pushing actions as well as pick-and-place actions.

1.1 Main Themes

Our exploration of physics-based manipulation in human environments leads to a number of new opportunities and challenges. These constitute the main themes which underlie this thesis. In this section we present these themes.

1. *In human environments, physical predictions are particularly useful in addressing clutter and uncertainty.*

   Cluttered human environments vary widely in terms of the number of objects in a given scene and the properties of each object, such as pose, shape, weight, material, and rigidity. It is impossible to design a one-for-all manipulation primitive. Physical

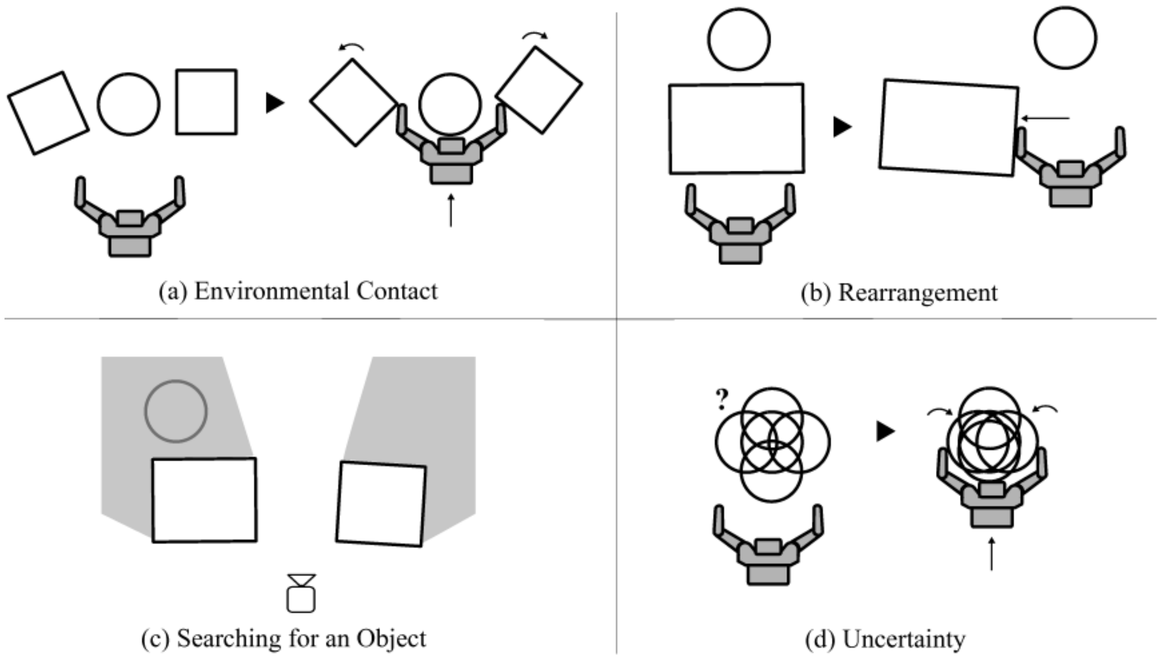


Figure 1.2: Four problems robots need to address in human environments with clutter and uncertainty.

predictions enable a larger variety of manipulation operations, increasing the performance of robotic manipulation in cluttered human environments.

In this thesis we use pushing predictions to create new manipulation primitives. First, this enables robots to perform manipulation simultaneously on multiple objects, which is very useful if contacting objects one-by-one is not efficient or feasible. Second, pushing enables robots to manipulate ungraspable (e.g. very large or heavy) objects. Third, the uncertainty in the pose and motion of objects can be accounted for by planning pushing primitives which funnel uncertainty without the use of a sensor.

2. *Physical predictions come at a high computational cost. Pre-computing primitive-based structures can lead to fast plans.*

   The computational cost of making accurate physical predictions is high. Manipulation planners need to consider alternative cases, running simulations many times, which results in long planning times. We propose a solution to this problem based on pre-computing and caching the physical interactions between the robot manipulator and an object for specific primitives.

   The pre-computation approach does come with limitations. First, the robot needs to have prior information about the possible scenes it will encounter. For example, we assume that there is a predefined set of objects. Second, pre-computation becomes infeasible as the number of scenes which need to be considered separately increases. For example, if a primitive's effects depend on the interaction of objects between each other, it may not be feasible to precompute these interactions as it would be impossible to enumerate all contact configurations of a multi-object system.

3. *Making physical predictions require knowledge about the physical properties of objects. It is important to choose the right physics model for a given primitive, extracting the important object parameters.*

   The physical properties a physics model needs may include friction coefficients, pressure distributions, inertial parameters, elasticity, etc. It is not always reasonable to expect a robot to know all these properties for a given object.

   For our pushing predictions we use the quasi-static physics model. This means we assume that frictions are high enough to dissipate energy quickly and the accelerations can be neglected. This approximation works very well for the tasks that we look at: pushing objects at lower speeds on surfaces with moderate friction. This simple model also reduces the number of parameters the robot needs to know about an object.

   Another approach we take is to relieve the robot from the burden of predicting the physical properties exactly. Instead, we model the robot's uncertainty about the physical properties of objects. Our planners produce actions which take this uncertainty into account.

4. *Physical predictions can be an extra source of uncertainty. Still, conservative planning can produce robust actions.*

   There may be a significant degree of uncertainty associated with a robot's predictions about the world state after executing a physics-based action. This may be due to (i) the initial uncertainty about the poses of objects, (ii) the robot's lack of knowledge about the physical properties of the objects, and (iii) the fact that simple physics models are approximations of the real phenomena.

   Plans based on inaccurate predictions run the risk of failure. One solution to this problem is to plan conservatively, producing actions which guarantee success given the initial uncertainty of the state and the uncertainty about the physical properties of the objects. In this thesis, we take this approach and show that it can produce robust and efficient manipulation actions. There are, however, limitations with this approach too: First, if the uncertainties are large, the planner can get too constrained, failing to generate solutions; second, this approach assumes that the underlying physics model is perfect, i.e. it ignores the third source of uncertainty we mentioned above.

As a solution to this problem, we also explore the design of closed loop physics-based actions. This approach places less constraints on the planner, potentially enabling it to produce plans in more cases. This approach also addresses the concern that the underlying physics model may not be accurate.

5. *Manipulation in clutter requires reasoning about manipulation and perception simultaneously.*

   Clutter makes certain parts of the environment unreachable for the robot manipulator. Similarly, clutter makes parts of the environment unreachable for the robot's sensors. A robot needs to reason about both constraints for efficient manipulation in cluttered environments.

   We explore the problem of searching for an object in such a framework. We model the scene as a graph where both manipulation and perception constraints are represented as connections. In human environments, this graph is usually divided into different connected components, e.g. in the form of objects in separate shelves, or different clusters of objects on a table. We devise search strategies which takes advantage of this structure of human environments.

1.2 Contributions

In this section we present a list of our contributions.

- Formulation of the *push-grasp primitive* which aims to grasp an object by executing a pushing action and then closing the fingers. This primitive harnesses the mechanics of pushing to funnel an object into a stable grasp, despite high uncertainty and clutter (Dogar and Srinivasa, 2010, 2012).

- Formulation of the *capture region* as a representation of the physical interaction between the robot hand and an object during a push-grasp. A capture region is the set of initial 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 (Dogar and Srinivasa, 2010, 2012).

- A *push-grasp planner* which uses conservative capture regions to generate actions in a given scene with object pose uncertainty. We show how capture regions can be used to efficiently and accurately find the minimum pushing distance needed to grasp an object with pose uncertainty. We then formulate a search over different parametrizations of a push-grasp to find kinematically feasible plans (Dogar and Srinivasa, 2010, 2012).

- A proof showing that certain strategies of estimating the object physical properties result in conservative capture regions (Dogar and Srinivasa, 2011).
- An approach to plan *grasps through clutter* where the fundamental action primitives enable simultaneous contact with multiple objects. 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 (Dogar et al., 2012).
- A *rearrangement planner* which plans sequences of actions to rearrange clutter in manipulation tasks. The planner is not restricted to pick-and-place operations and can accommodate other non-prehensile actions and object pose uncertainty (Dogar and Srinivasa, 2011, 2012).
- A *formal description of the object search by manipulation problem* by defining the expected time to find the target as a relevant optimization criterion and the concept of accessibility and visibility relations (Dogar et al., 2013).
- Proposing two different *algorithms for object search*. We prove that given an appropriate definition of utility, the *greedy approach* to removing objects is optimal under a set of conditions, and provide insight into when it is suboptimal. We introduce an alternative algorithm, called the *connected components algorithm*, and present a partial proof that it is optimal under all situations along with empirical data to back that claim (Dogar et al., 2013).
- A *particle filtering formulation* for estimating the pose of an object during pushing actions. This approach uses physics-based predictions as the motion model, and feedback from contact sensors as the observation model (Koval et al., 2013).
- Implementation of the aforementioned algorithms and planners on real robot platforms such as HERB (Srinivasa et al., 2012) and the PR2 (Dogar et al., 2012).

1.3 Related Work

State-of-the-art robotic systems which address manipulation in human environments (e.g. Srinivasa et al. (2009), Ciocarlie et al. (2010), Beetz et al. (2011)) take a similar approach to planning. This approach uses a geometric model of the environment and plans a sequence of pick-and-place actions to achieve a certain goal. There is a large body of work addressing the problem of planning to pick up an object using a robot manipulator given a geometric environment model. The problem is often solved in the configuration space (Lozano-Pérez and Wesley, 1979, Lozano-Pérez, 1983) by planning a path from the initial configuration of the robot arm to a goal configuration. The configuration space is usually very high-dimensional, leading to the use of probabilistic search algorithms like probabilistic roadmaps (PRMs) (Kavraki and Latombe, 1994) or rapidly exploring random trees (RRTs) (Lavalle and Kuffner, 2000). These algorithms can find solutions in high-dimensional spaces in a fast and probabilistically complete way. This approach deals with clutter through collision checking to avoid contact with the objects during the execution of a path. 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. Using PRMs, Siméon et al. (2004) propose such a planner which identifies robot motion as either *transfer* (i.e. motion with an object rigidly grasped) or *transit* (i.e. motion where no object is moved), where the two phases are separated by grasp/ungrasp actions.

In this setting grasping is often modeled as a static pose of the hand relative to the object. Such hand poses are computed based on where the hand contacts the object when the fingers close. Grasps are evaluated based on different quality metrics (Suárez et al., 2006, Miller and Allen, 2004), such as a force-closure of the object (Nguyen, 1989).

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 such manipulation actions.

Probably the most studied method of non-prehensile manipulation is pushing. Mason (1986) presents an analysis of the mechanics of pushing in the quasi-static case. The *voting theorem* is given, which states that the sense of rotation of an object pushed with point contact can be found by a voting between the edges of the friction cone and the direction of the motion of the contact point on the pusher. Mason also coins the term *push-grasp*, where one finger pushes the object for a time, and then the second finger squeezes the object for a grasp. This non-prehensile grasping primitive is robust to rotational uncertainty.

Goyal et al. (1991) 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 space. Given a point on the limit surface, the motion of the object can be computed by taking the normal to that point. However, building the limit surface analytically may not be possible for general object geometry. Also, it depends on the pressure distribution supporting the object on the surface, which is usually not known. Howe and Cutkosky (1996) show that the limit surface can be approximated by a three-dimensional ellipsoid.

The pressure distribution supporting the object is often not known. This requires any pushing based primitive to be robust to changes in the pressure distribution. Peshkin and Sanderson (1988) show how to find the locus of centers of rotation of an object for all possible pressure distributions. Brost (1988) presents an algorithm that generates a series of parallel-jaw grasping actions to bring a polygonal object to known orientation. The series of actions are robust to uncertainties in the object's initial location.

Lynch and Mason (1996) discuss a variety of issues regarding the mechanics, control, and planning of pushing. They show that the stable pushing system is subject to nonholonomic constraints. Then they show the conditions for the system to be *controllable* and *small-time locally controllable*. They show that the system of pushing with a point contact is controllable and small-time locally controllable (as long as the slider is not a frictionless disk). Then they show that stable pushing with line contact is controllable for common cases, and small-time locally controllable for some polygons (Lynch characterizes the classes of such polygons in Lynch (1999b)). Lastly, they show how to compute center of rotations for the stable push with line contact, and they present a planner based on Dijkstra's algorithm. Other similar pushing planners include Akella and Mason (1998) and Agarwal et al. (1997).

Other non-prehensile operations have been explored as well. Lynch (1999a) analyzes *toppling*. 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. This increases grasping task performance and decreases the load on the arm during the lifting action. Other recent work on non-prehensile manipulation include Omrcen et al. (2009), Kappler et al. (2010), Cosgun et al. (2011), Lau et al. (2011), Kappler et al. (2012), Zito et al. (2012).

In this thesis one of our key contributions is the integration of a planning system based on non-prehensile actions with 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. With this framework, we approach a number of key problem domains in robotic manipulation in human environments. In the next few sections we review the related work in each of these domains.

1.3.1 Manipulation under Uncertainty

There have been different approaches in addressing uncertainty in object manipulation. In one approach manipulation is sensorless, but the goal is achieved through actions which funnel the uncertainty to the goal state(s). Erdmann and Mason (1988) present a planner which can orient an object on a tray through a sequence of tilting operations without the use of any sensors. Brost (1988) presents an algorithm that plans parallel-jaw grasping motions for polygonal objects with pose uncertainty. Similar strategies have been extensively used in designing manipulation primitives which work even under significant uncertainty. Recent examples include the planner from Berenson et al. (2009b) which plans static grasps given an uncertainty distribution of an object. Stulp et al. (2011) presents a system to learn grasping actions which succeed under uncertainty.

Another strategy involves using sensor feedback, especially contact feedback, during manipulation to reduce uncertainty. An early example is the work by Lozano-Perez et al. (1984) which proposes *pre-image backchaining* for the automatic generation of actively compliant actions that lead to successful execution of manipulation tasks under sensing and control uncertainty. Brost and Christiansen (1996) describe how to assign probabilities to such actions. Lynch et al. (1992) describes a pushing control system which uses tactile feedback and physics-based predictions about how the pushed object moves. More recently, Hsiao et al. (2007) casts the problem as a partially observable Markov decision process (POMDP, Kaelbling et al. (1998)) and investigates strategies to generate robust policies efficiently. Platt et al. (2010) present an approach to solve the POMDP problem quickly by assuming that the maximum likelihood observations will always be obtained and by re-planning when necessary. Platt et al. (2011, 2012) also propose an approach to represent non-gaussian belief states which enables fast planning of information gethering actions.

There is also a significant body of work which simply aims to localize an object through contact feedback. Jia and Erdmann (1999) uses the physics of pushing to build analytical state estimators to track the pose of the object from contact positions on the hand. Zhang and Trinkle (2012) use particle filters (Thrun et al., 2005) for object pose estimation during manipulation with stochastic motion and observation models. Recent work uses probabilistic methods for the tactile localization of *immovable* objects (Javdani et al., 2013, Petrovskaya and Khatib, 2011). These systems try to produce a minimum number of distinct touch actions that provide information about the object pose.

1.3.2 Manipulation with Environmental Contact

Environmental contact can be dealt with in two basic ways: making the manipulator compliant and/or taking advantage of the environmental compliance.

One approach to making the manipulator compliant is by designing passive mechanical compliance. Pratt and Williamson (1995) analyzes the use of series elastic actuators. Using force feedback, manipulators can also display active compliance, implemented in software as a control loop (Whitney, 1977, Mason, 1981). A recent work by Jain et al. (2013) applies a force-control approach to manipulation in cluttered environments.

The problem of planning in domains with environmental compliance has also been studied. Rodriguez et al. (2006) presents an extension of the RRT algorithm which works in deformable environments, and Frank et al. (2011) presents a system which uses a PRM formulation to solve a similar problem.

Environmental contact is also important for robot locomotion, as a robot's feet make and lose contact with the ground. Among the different trajectory optimization approaches taken to solve this problem, Erez and Todorov (2012) and Posa and Tedrake (2013) formulate contact in a general way which allows their methods to be applied to certain manipulation problems as well.

1.3.3 Rearrangement Planning

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 the blocks-world problem (Winograd, 1971). STRIPS (Fikes and Nilsson, 1971) is an early well-known planner to solve this problem. In robotics, the problem is named *planning among movable obstacles* (Stilman and Kuffner, 2006). 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, 1998a, Chen and Hwang, 1991, van den Berg et al., 2008). These approaches are not practical for a manipulator arm with high degrees of freedom. Another group of planners are based on a search over different orderings to move the obstacle objects in the environment (Ben-Shahar and Rivlin, 1998b, 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). This planner 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 which uses pre-image backchaining to identify obstacles by computing the swept volumes of future actions. This planner can perform rearrangement in the presence of uncertainty.

1.3.4 Object Search

Traditionally, the problem of searching for an object in an environment has been treated as an instance of geometric sensor placement (de Berg et al., 2008). Ye and Tsotsos (1995, 1999), Shubina and Tsotsos (2010) use a mobile robot with an active camera and formulate an optimization problem where the goal is to maximize the probability of detecting the target object with minimal cost. This sensor-based approach also leads to strategies which takes into account the properties of specific object detection algorithms (Sjo et al., 2009, Ma et al., 2011, Anand et al., 2013).

Recent work discusses the *object search by manipulation* problem, where the environment is configured so that a previously hidden object becomes visible. Gupta and Sukhatme (2012) present a planner which selects the action maximizing a k-step information gain. Kaelbling and Lozano-Perez (2012) present a general hierarchical planning framework, and use object-search as one example problem the planner can solve. In human environments, one can associate different probabilities with a target object being hidden in different regions. Wong et al. (2013) present a framework to learn and generate such probabilities, and to use them for object search planning.

1.4 Roadmap

Chapter 2 introduces the push-grasping primitive and how it is used for physics-based grasping under uncertainty. Chapter 3 presents our planner for manipulation with environmental contact. Chapter 4 presents our rearrangement planner. Chapter 5 introduces our formulation of the object search by manipulation problem. Chapter 6 and Chapter 7 presents future work and preliminary results involving physics-based manipulation with feedback. Chapter A presents our proof showing how to compute conservative capture regions. Chapter B presents a partial proof of optimality for the algorithm presented in Chapter 5.

1.5 Publication note

Most of Chapter 2 appeared in Dogar and Srinivasa (2010) and Dogar and Srinivasa (2012). Most of Chapter 3 appeared in Dogar et al. (2012) and is joint work with Kaijen Hsiao and Matei Ciocarlie. Most of Chapter 4 appeared in Dogar and Srinivasa (2011) and Dogar and Srinivasa (2012). Most of Chapter 5 appeared in Dogar et al. (2013) and is joint work with Michael Koval and Abhijeet Tallavajhula. Most of Chapter 6 appeared in Koval et al. (2013) and is joint work with Michael Koval, Michael being the principal investigator.

Part II

Physics-Based Grasping under Uncertainty and Clutter

Chapter 2

Physics-Based Grasping under Object Pose Uncertainty

In this section, we demonstrate how the mechanics of pushing can be harnessed to 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. 2.1. 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.

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

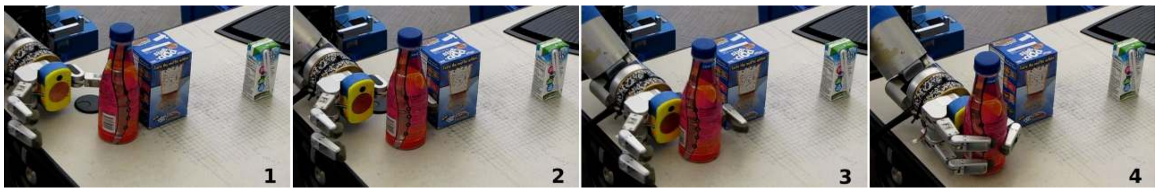


Figure 2.1: An example push-grasp of an object in contact with the surrounding clutter.

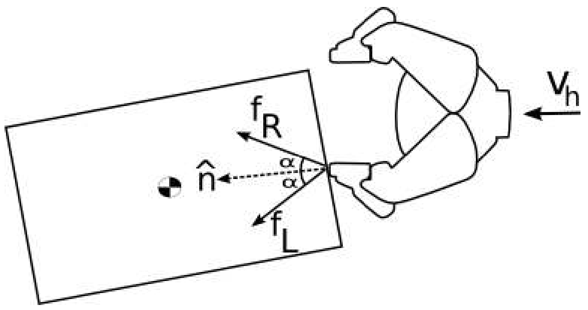


Figure 2.2: Notation used for hand-object contacts. The hand's velocity is given by $\mathbf{v_h}$. The friction cone edges, $(\mathbf{f_L}, \mathbf{f_R})$, and the normal $\hat{\mathbf{n}}$ at the contact are illustrated.

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.

2.1 The Mechanics of Pushing

When pushing an object with a robot finger, one question is whether the object will roll into or out of the hand. Mason (1986) 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. In order to predict the instantaneous velocity of a pushed object in the quasi-static case, we can use the *limit surface* (Goyal et al., 1991).

Using the notation in Fig. 2.2, 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* (Mason, 1986), 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* (Goyal et al., 1991), 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 (1996), however, show that the limit surface can practically be approximated by an ellipsoid centered at the `CoM` assuming the object's pressure distribution displays periodic rotational symmetry. Given this model, we can express how limit surface relates the generalized velocity of the object to the applied generalized force as:

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

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:

$$f_{max} = \mu_s f_n^s \tag{2.1}$$

$$m_{max} = \mu_s \int_A |\mathbf{x}| p(\mathbf{x}) \, \mathrm{d}A \tag{2.2}$$

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 cancels out when computing the ratio. Similarly, the mass of the object is part of the normal force f_n^s in Eq. (2.1), and also part of the $p(\mathbf{x})$ in Eq. (2.2) such that it can be taken out of the integral since we assume that the pressure distribution displays periodic rotational symmetry. 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.Section 3.1.1.

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 $\mathbf{f} = (f_x, f_y)$ (Howe and Cutkosky, 1996).

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* (Mason, 1986). 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. Since μ_c determines the shape of this friction cone, it is another object property which affects the motion of the object. See Lynch et al. (1992), Howe and Cutkosky (1996) for more details.

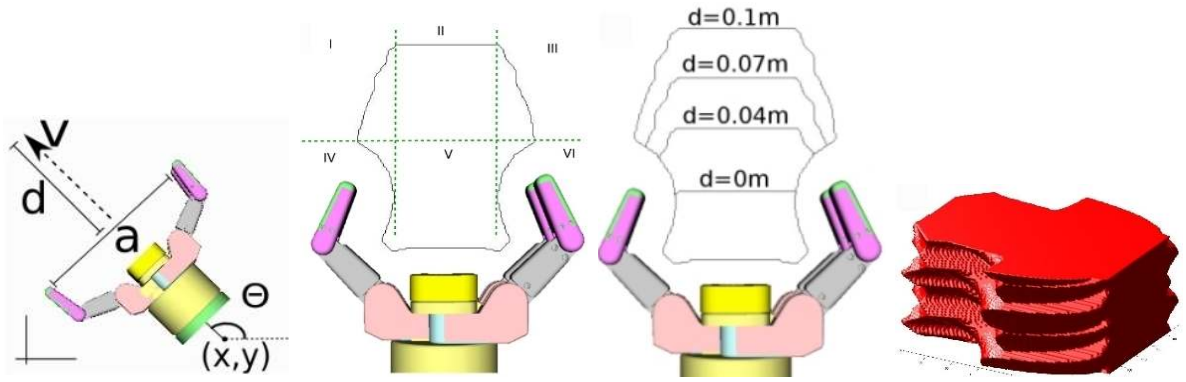
2.2 Push-Grasping

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.

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. 2.1). We parametrize (Fig. 2.3(a)) the push-grasp $G(p_h, a, d)$ by:

- The *initial pose* $p_h = (x, y, \theta)$ 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.



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

Figure 2.3: (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 with horizontal dimensions of $4cm \times 6cm$.

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

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 present the capture region of a juice bottle produced by our pushing simulation in Fig. 2.3(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. 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. 2.3(d). We compute it by computing the two-dimensional capture regions of the object at different orientations.

The left and right sides of the capture region are no longer symmetric in (x, y, θ). The ramps on the right side "ramps up" since the objects, as they are pushed, rotate positively there and jump to the higher level. The ramps on the left side "ramps down" as the objects there are rotating in the negative direction as they are pushed.

As we noted Section 2.1, the shape, the pressure distribution supporting the object on the surface and the friction between the robot finger and the object, μ_c, affect the quasi-static motion of the object. We assume that the robot knows the shape of objects. We compute capture regions conservatively with respect to the other 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 is presented in Appendix A.

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

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. 2.3(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. 2.3(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. 2.4). The cases to handle are:

- If the object is already inside the hand (see *P1* in Fig. 2.4), no push is required;

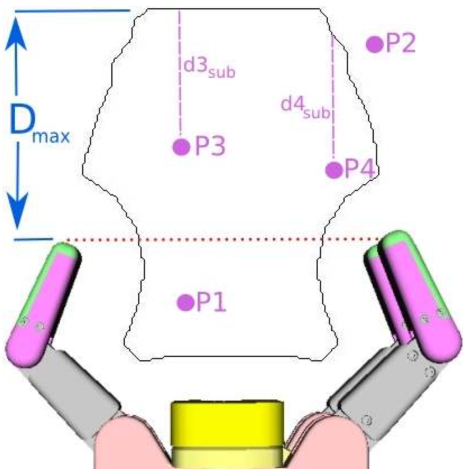


Figure 2.4: Given an object pose, the minimum required pushing distance d to grasp that object can be found using a precomputed capture region of a push-grasp with pushing distance D_{max}. In the figure, $d = 0$ for P1 since it is already in the hand; P2 can not be grasped with a push shorter than D_{max} 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.

$d = 0m$.

- Else, if the object is outside the capture region (see *P2* in Fig. 2.4) there is no way to grasp it with a push shorter than D_{max}. Reject this object.

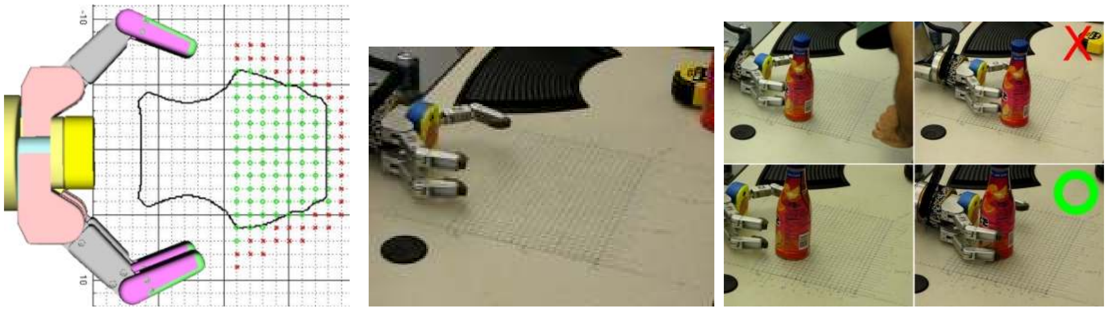
- Else, the minimum pushing distance required can be found by using the formula

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

  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. 2.4). 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. 2.3(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.



(a) (b) (c)

Figure 2.5: Capture region generated with our push-grasping simulation and validated by robot experiments. (a) Simulation and real-world experiments. Green circles: real world successes; red crosses: real world failures. (b) Push-grasping validation setup. 150 validation tests were performed in total. (c) Two example cases where the push fails (top row), and succeeds (bottom row).

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. 2.5b). 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. 2.5c.

The results (Fig. 2.5a) 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.

One can also see that the experiment results are not perfectly symmetric for the left and right fingers, whereas the simulated result is. This is due to the fact that the physics of the real fingers are not exactly same: the exact joint angles, the stiffnesses, the geometry of the padding at the fingertips differ between fingers.

2.3 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.

2.3.1 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*.

2.3.2 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. 2.6. Here the robot detects a juice bottle (Fig. 2.6a). We illustrate the uncertainty region of the juice bottle in Fig. 2.6b, and the capture region of the push-grasp in Fig. 2.6c. If the uncertainty region is completely included in the capture region as in Fig. 2.6c, then we can guarantee that the push-grasp will succeed.

The uncertainty and capture regions are two-dimensional in Fig. 2.6 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

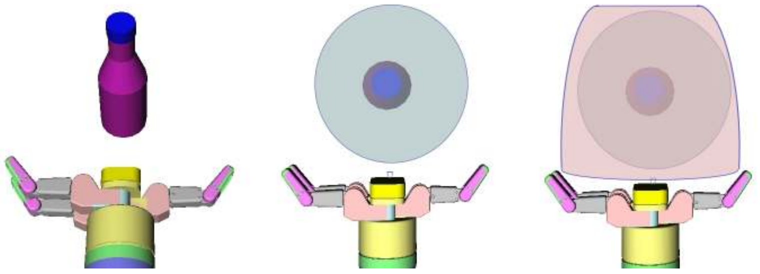

(a) Detected object (b) Uncertainty region (c) Capture region

Figure 2.6: If the uncertainty region of an object is included in the capture region of a push-grasp, then the push-grasp will be successful.

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.

2.4 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.

2.4.1 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.

Algorithm 1: t ← PlanPushGrasp(goalObject, obstacleObjects)

```
c ← goalObject.captureRegion;
gSamples ← Sample(goalObject, n);
$\text{oSamples}_i$ ← Sample($\text{obstacleObjects}_i$, n);
while {v, a, o} ← GetNextParam(goalObject) do
    p ← FindInitialHandPose(goalObject, v, a, o);
    $\max_d$ ← NULL;
    for $i$ ← 1 to n do
        if IsInCaptureRegion(p, a, $\text{gSamples}_i$, c) then
            $\text{d}_i$ ← PushDistNeeded(p,a,$\text{gSamples}_i$,c);
            $\max_d$ ← max($\max_d$, $\text{d}_i$);
        else
            $\max_d$ ← NULL;
            break;
    if $\max_d$ != NULL then
        d ← $\max_d$;
        t ← GenerateTraj(p,a,d);
        if CheckIK(t) and CollisionFree(t, oSamples) then
            return t;
```

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 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,

Table 2.1: Planner Performance.

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

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.

2.5 Results

This section presents extensive experiments in simulation and on HERB to evaluate the performance of our planner. Simulation experiments are performed in OpenRAVE (Diankov and Kuffner, 2008).

2.5.1 Robotic Platform

HERB has a 7-DoF WAM arm, and a 4-DoF Barrett hand with three fingers. We use the vision system from Martinez et al. (2010) to estimate object poses.

2.5.2 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. (2009b). Uncertainty TSRs try to generate static hand poses to grasp an object with given pose hypotheses representing 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.

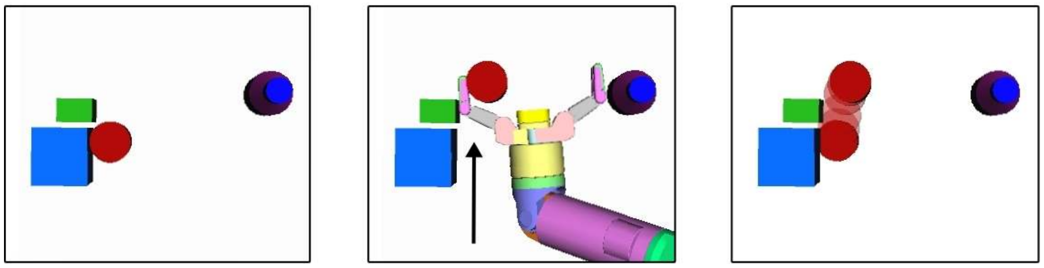


Figure 2.7: A high-clutter scene where the TSR planner fails but push-grasp planner is able to find a plan.

Table 2.1 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 2.1 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 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 2.1 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 2.1 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. 2.7. 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. 2.7, which moves the object away from the clutter first and then grasps.

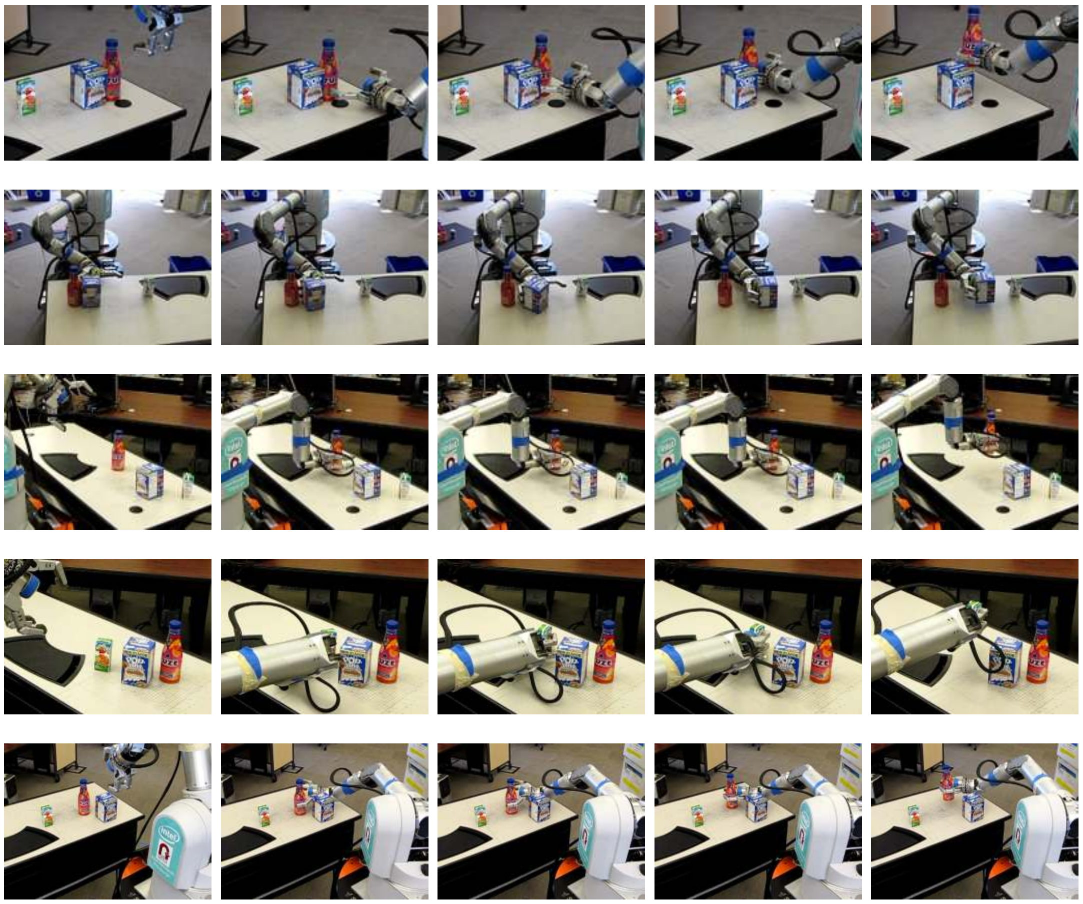


Figure 2.8: Example push-grasps executed by our robot.

The planning times also shown in Table 2.1. 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 scenes with higher uncertainty or clutter, due to the larger number of push-grasps the planner needs to evaluate in these scenes.

2.5.3 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 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. The 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. The push-grasp planner was not able to find a plan when the amount of clutter around the goal object was even denser.

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. 2.8. Videos of our robot executing push-grasps are online at: www.cs.cmu.edu/˜mdogar/pushgrasp

2.6 Discussion

In this work we present a push-grasp planner that can reduce the uncertainty about an object's pose by acting on it. We pre-compute capture regions, and execute push-grasps as open-loop actions. Open-loop actions have advantages: they do not depend on the existence of specific sensors, and their execution is fast as they do not require an expensive sensor processing step in their control loop.

However, planning open-loop actions have limitations as well. These actions try to address all the initial uncertainty in the system during planning time. When the uncertainty is large, this leads to failure in finding solutions and long planning times.

Another drawback of planning open loop actions is that, for successful execution, it requires accurate physical predictions. Such high-accuracy predictions are expensive to make. This forces us to precompute the physics-based interactions. However, precomputation can quickly become a limiting factor. It does not work for novel objects, which are an important part of human environments. Even for known objects, it quickly becomes difficult to precompute all the useful interactions between the robot hand and the object.

One solution to both problems is to integrate sensor-feedback to the push-grasping process. We present future work and preliminary results along this direction in Chapter 6.

Chapter 3

Grasping through Clutter with Environmental Contact

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. 3.1, 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 (Miller and Allen, 2004, Berenson et al., 2007, Goldfeder et al., 2009). 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.

In this section, 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. 3.1, Right). In

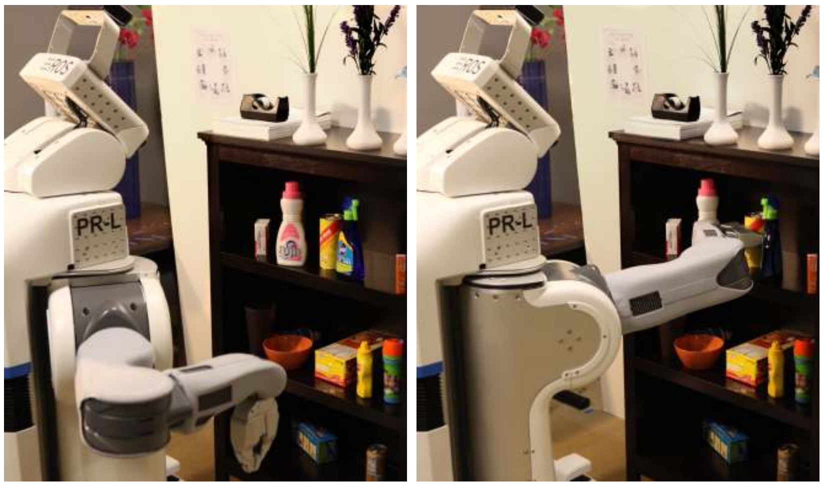


Figure 3.1: Manipulation in a constrained and cluttered environment. Left: a robot tries to acquire a partly occluded object from a constrained shelf space. Right: the robot clears a path to the target by pushing obstacles in a controlled manner, and completes the grasp.

addition to simply closing the gripper on the target, interaction with objects in the scene is done through quasi-static pushing, similar to push-grasping (Chapter 2). However, while push-grasping avoids all contact with any obstacles, we now 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.

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.

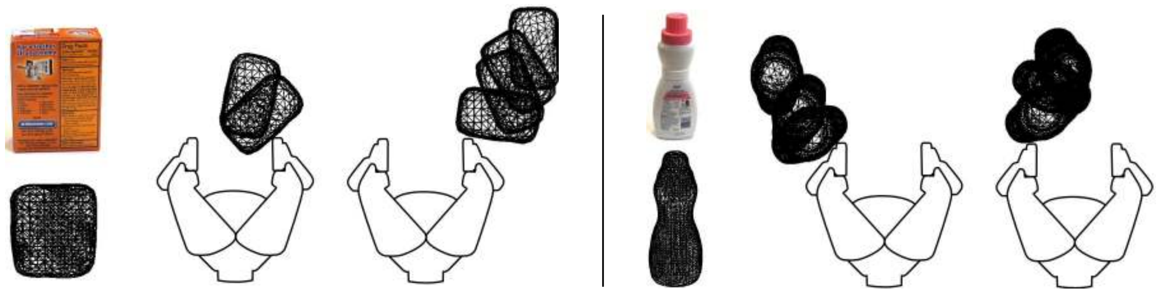


Figure 3.2: Example objects, their trimesh models, and computed trajectories of objects as the hand pushes them. The hand motion is towards the top of the page in each diagram, and is not shown.

3.1 Physics-Based Grasping in Clutter

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 (Baraff, 1993). 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 low velocity in order to avoid inertial effects.

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. 3.2 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 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.

3.1.1 Pre-Computed Object Trajectories

We use the quasi-static contact model outlined in Section 2.1 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, 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*:

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

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 work, 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, we represent these hand trajectories using their last part during which the hand

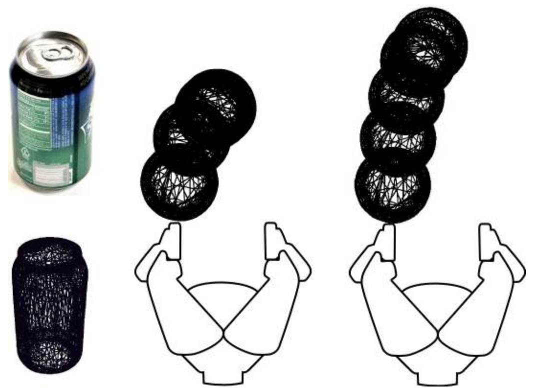


Figure 3.3: Left: An example object and its trimesh model. Center: Simulated object trajectory when contact pressure distribution is concentrated at the object's `CoM`. Right: Simulated object trajectory when contact pressure distribution is concentrated at the object's periphery.

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. 3.2.

The object's pressure distribution also affects the trajectory of the object. 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. 3.3 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 preshape 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 preshape, or if we were to use a different hand.

3.1.2 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.

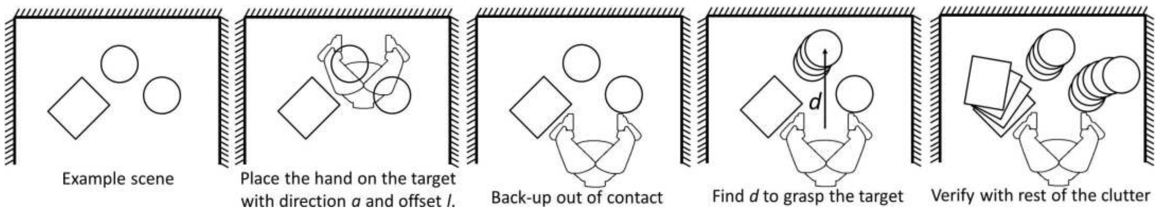

Example scene
Place the hand on the target with direction *a* and offset *l*.
Back-up out of contact
Find *d* to grasp the target
Verify with rest of the clutter

Figure 3.4: Illustration of evaluating a grasp with a given approach direction, a, and lateral offset, l.

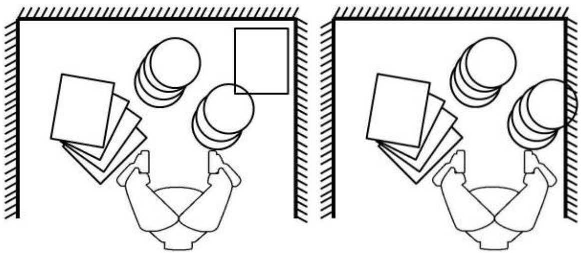


Figure 3.5: Illustration of grasps that will be rejected. Left: The pushed object contacts another object. Right: The pushed object contacts a non-movable object (the side-wall).

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. 3.4, 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. 3.5 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.

3.1.3 Handling Uncertainty

The evaluation algorithm described above is based on knowing the relative locations of the objects in the scene. However, when operating in unstructured environments, these poses are inevitably affected by uncertainty, due to imprecise localization, imperfect robot calibration, *etc.*

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.

3.2 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 chapter using precomputed object trajectories for a pushing height of 9 cm.
- Push-Grasp: This planner, similar to the one presented in Chapter 2, 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.

We randomly-generated 100 simulated scenes, each with 6 objects (1 target and 5 obstacle 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.

Fig. 3.6 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. 3.6 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 3.1. The clutter-grasp planner takes longer to evaluate each grasp, but because a larger fraction of evaluated grasps are feasible, the average planning time until the first feasible grasp is found is lower

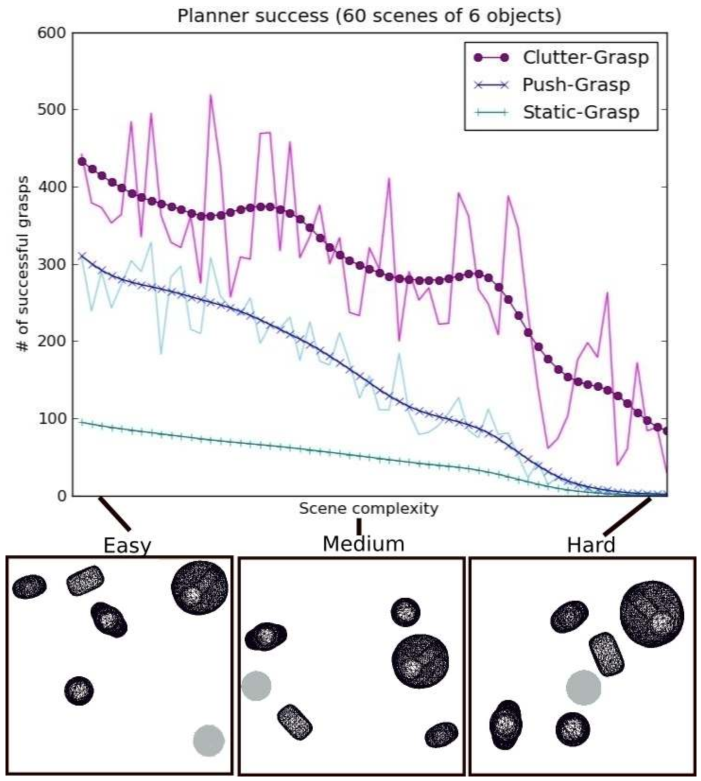


Figure 3.6: Top: A comparison of number of grasps found, in randomly-generated scenes sorted by scene complexity (the 60 most complex in the set of 100 generated), for our three grasp planners. Bottom: Bird's eye view of example scenes for varying levels of complexity. The target object is shown in light gray.

Table 3.1: Planning Times for Grasping in Clutter

	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

than for the other two planners. Planning time until the first feasible grasp is found is also plotted for varying scene complexity in Fig. 3.7; 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.

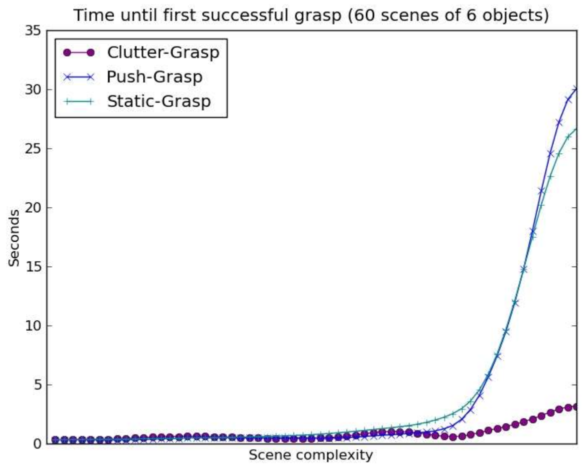


Figure 3.7: Time until first grasp for all three planners, for the most complex 60 of the 100 random scenes, sorted by scene complexity.

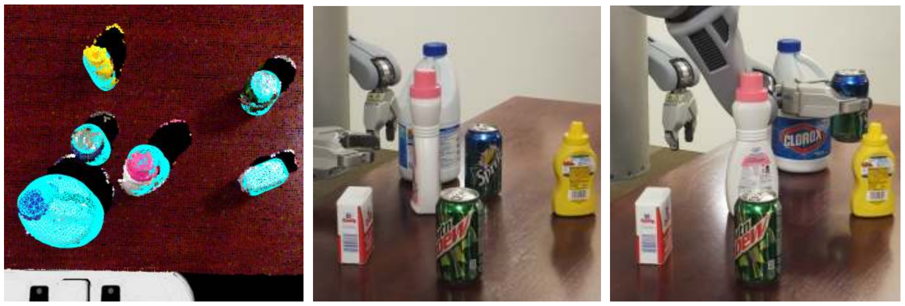


Figure 3.8: An example execution of a randomly-generated scene. Left: Object meshes in the randomly-generated scene configuration, along with point cloud from a Kinect^{TM}camera. Middle: Actual scene with gripper at start of push. Right: Actual scene after object has been grasped.

3.3 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

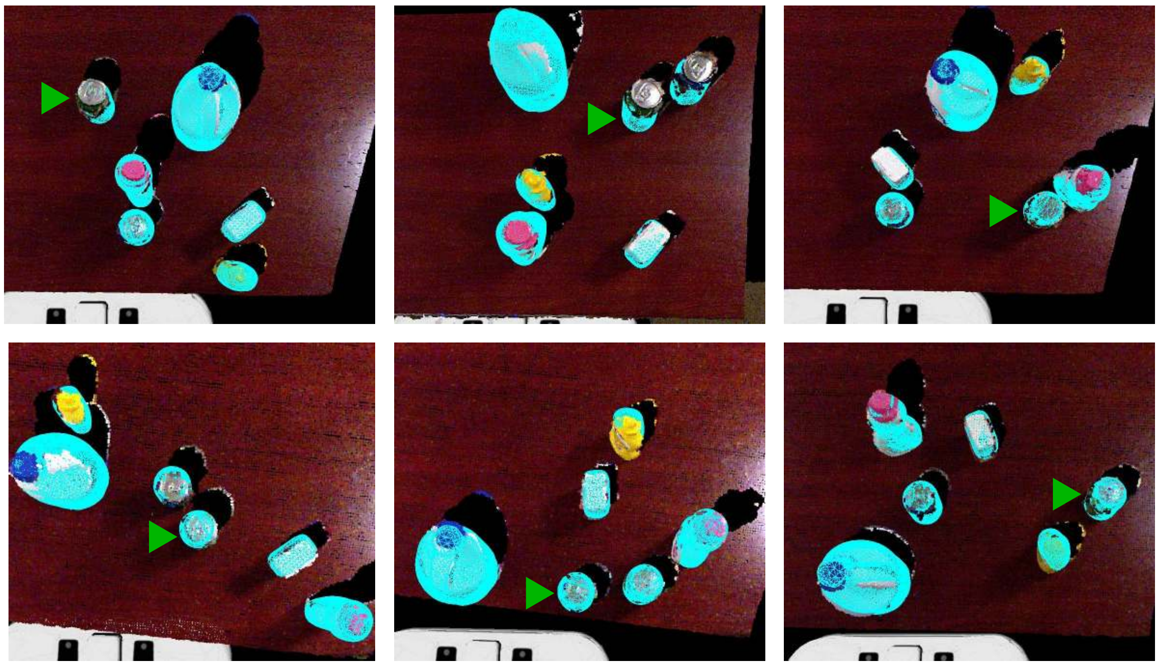


Figure 3.9: More randomly-generated scenes with kinematically-feasible clutter-grasps. The target objects are marked with green triangles. All attempted clutter-grasps were executed successfully.

motion plans for at least one grasp. Real-world objects were placed at the randomly-generated locations on the table by carefully aligning their locations as seen by the robot's Kinect$^{\mathrm{TM}}$point cloud with the object meshes rendered at their desired locations using rviz (Willow Garage, 2012), a 3D visualization tool. The resulting point clouds with aligned meshes are shown in Fig. 3.8 and Fig. 3.9, 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 Rublee et al. (2011) to recognize objects placed on the shelf shown in Fig. 3.1. 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 point cloud object outlines overlaid on the images seen by the robot's Kinect$^{\mathrm{TM}}$camera, along with the evaluated grasp hypotheses, in

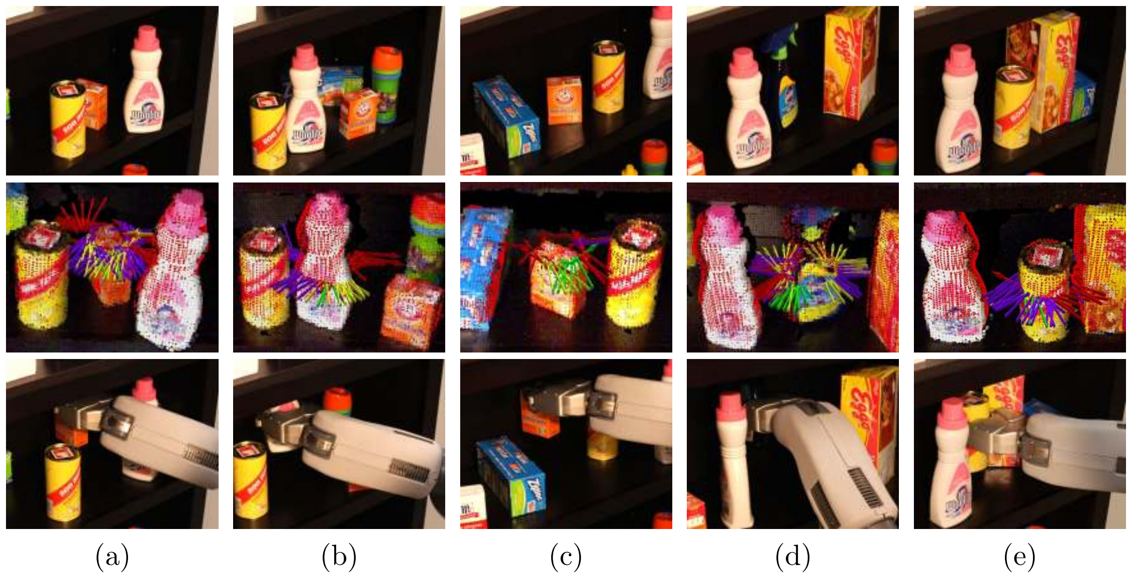


Figure 3.10: A wide range of scenes and objects for which our planner succeeds by moving obstacles in a controlled manner. Each **column** shows one grasping task. For each column, **top image**: Initial scene; **middle image**: Scene as seen by the robot, showing an acquired point cloud superimposed with object recognition results (red point object outlines) and evaluated grasping directions (arrows); **bottom image**: Final result of successful grasp execution.

the middle images in Fig. 3.10 and Fig. 3.11. The clutter-grasp planner succeeded in picking up the target object in all five scenes shown in Fig. 3.10, and failed to plan grasps in the two scenes shown in Fig. 3.11. 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/˜mdogar/RSS2012

3.4 Discussion

In this section 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

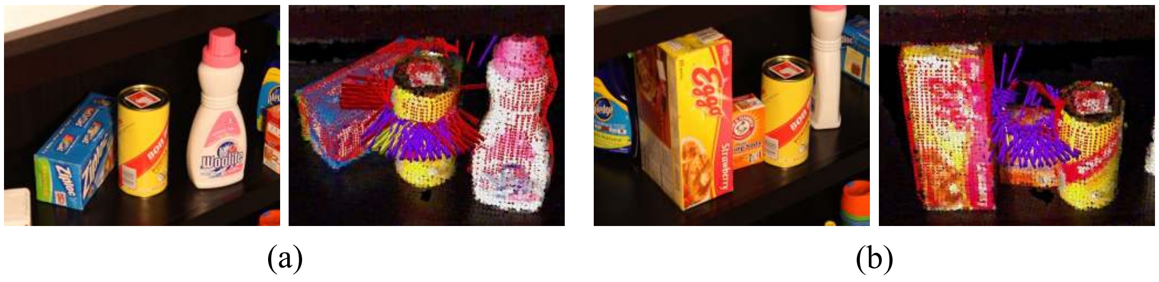


Figure 3.11: Examples of cases where our planner cannot find a solution. In (a), multiple objects start out in contact with each other and object-object interaction during pushing is inevitable. In (b), insufficient space behind the target object prevents the execution of sufficiently long pushes.

manner, in order to clear the desired path.

Our approach has two important limitations. First, it does not address object-object contacts. We believe fast but approximate physics predictions can be used with sensor feedback to overcome this problem.

Second, the existing framework limits robot-object interactions to happen only at the end-effector. Extending this to the whole manipulator is a major and exciting challenge for us.

Part III

Manipulating Clutter

Chapter 4

Rearrangement Planning with Pushing Actions

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.

The list of primitives that we use to move, slide, push, pull and play with the objects around us is nearly endless. But they share common themes. We are fearless to *rearrange 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.

How can we enable our robots to fearlessly rearrange the clutter around them while maintaining provable guarantees on the consequences of their actions? How can we do this in reasonable time? We would rather not have our robot stare at the fridge for 20 minutes planning intricate moves. Finally, can we demonstrate that these human-inspired actions do work on our robots with their limited sensing and actuation abilities? These are the research questions we wish to address.

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 accommodate other 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,

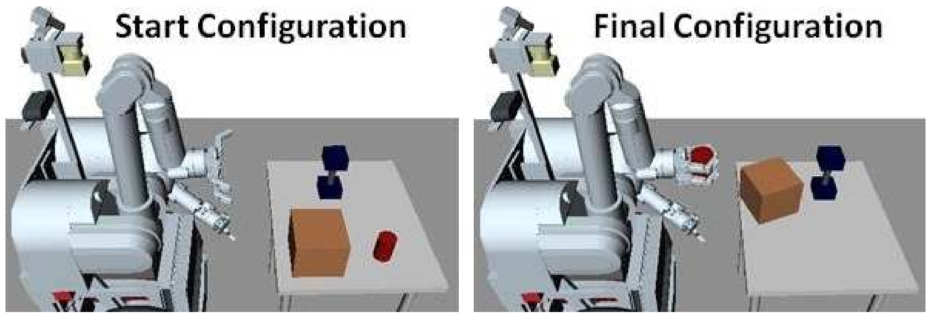


Figure 4.1: An example scene. The robot's task is picking up the cylindrical can. The robot rearranges the clutter around the goal object and achieves the goal in the final configuration. The robot executes the series of actions shown in Fig. 4.2. We present the planning process in Fig. 4.4.

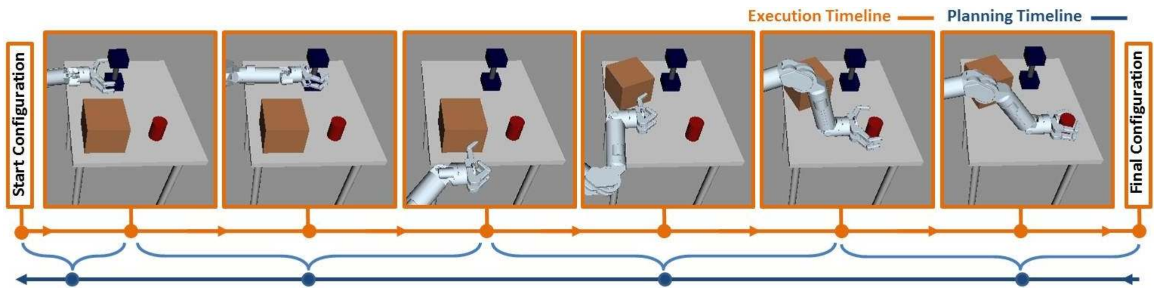


Figure 4.2: We show the snapshots of the planned actions in the order they are executed. The execution timeline goes from left to right. Each dot on the execution timeline corresponds to a snapshot. Planning goes from right to left. Each dot on the planning timeline corresponds to a planning step. The connections to the execution timeline shows the robot motions planned in a planning step. Details of this planning process are in Fig. 4.4.

heavy ungraspable objects in the environment. We also show that our planner is robust to uncertainty.

4.1 Planning Framework

We present an open-loop planner that rearranges the clutter around a goal object. This requires manipulating multiple objects in the scene. The planner decides which objects to move and the order to move them, decides where to move them, chooses the manipulation 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 following example (Fig. 4.1). The robot's task is picking up the can. There are two other objects on the table: a box which is too large to be grasped, and the dumbbell which is too heavy to be lifted.

The sequence of robot actions shown in Fig. 4.2 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 can to grasp and move it to the goal position.

Fig. 4.2 also shows that the actions to move objects are planned backwards in time. We visualize part of this planning process in Fig. 4.4. In each planning step we move a single object and plan two arm trajectories. The first one (e.g. *Push-grasp* and *Sweep* in Fig. 4.4) is to manipulate the object. The second one (*GoTo* in Fig. 4.4) 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 Section 4.2. We discuss a number of questions below to explain the planning process and then present the algorithm in Section 4.1.5.

4.1.1 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. 4.4). During this grasp, both the robot and the 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 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. 4.4, the box is added to `move`.

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

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

We compute the volume of space an object occupies by taking into account the pose uncertainty (Section 4.1.2).

In subsequent planning steps (e.g. Step 2 in Fig. 4.4) the planner searches for actions that move the objects in `move`. The robot and the manipulated object are again allowed to penetrate other movable objects' spaces, and penetrated objects are added 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 do backward planning to identify the objects to move.

Allowing the planner to 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 Section 4.2.

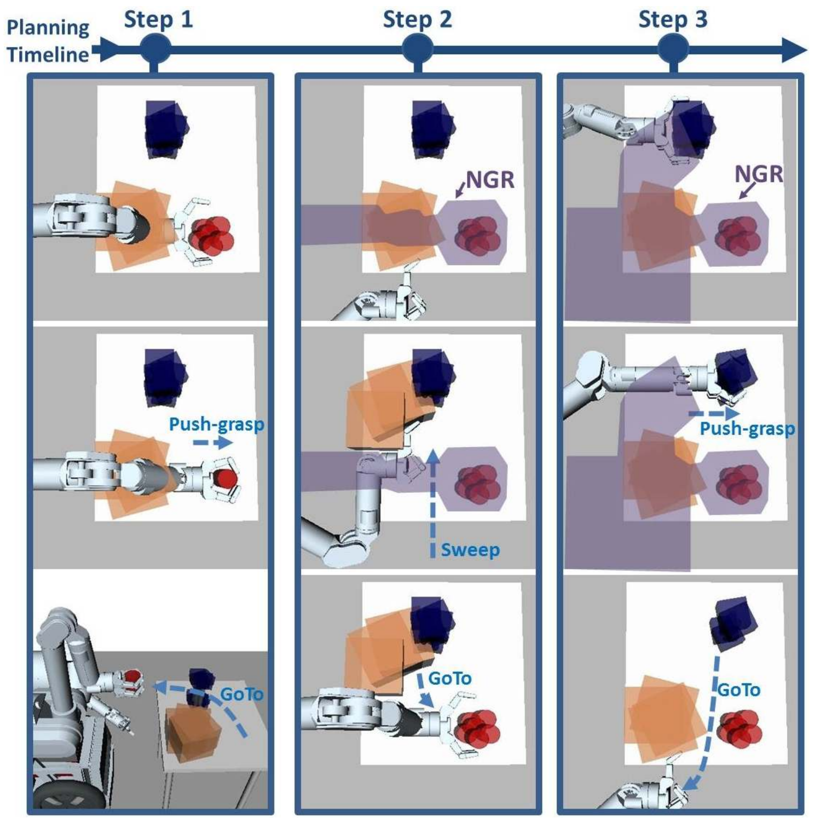


Figure 4.4: The planning timeline. Three snapshots are shown for each planning step. The planner plans two consecutive arm motions at each step, from the first snapshot to the second snapshot, and from the second snapshot to the third snapshot. These motions are represented by dashed lines. The regions marked as *NGR* show the *negative goal regions* (Section 4.1.4). The object pose uncertainty is represented using a collection of samples of the objects.

We also restrict the plans we generate 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. 4.4 in Step 2 the `avoid` list includes the red can, in Step 3 it includes the can and the box.

4.1.2 How to address uncertainty?

Robots can detect and estimate the poses of objects with a perception system (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. Fig. 4.4 visualizes the pose uncertainty using copies of the object at different poses.

In this chapter we use the word *region* to refer to a subset of the configuration space of a body. We define the *uncertainty region* of an object $\mathtt{o}$ at time t as the set of poses it can be in with probability larger than ϵ:

$$U(\mathtt{o}, t) = \{q \in \mathbb{SE}(3) | \mathtt{o} \text{ is at } q \text{ at time } t \text{ with prob.} > \epsilon\}$$

The manipulation actions change the uncertainty of an object. This is represented as a trajectory ν:

$$\nu : [0,1] \rightarrow \mathcal{R}$$

where $\mathcal{R}$ is the set of all subsets of $\mathbb{SE}(3)$. We call ν the *evolution of the uncertainty region* of that object.

In the rest of this chapter, we will drop the time argument to U and use $U(\mathtt{o})$ to stand for the initial uncertainty region (i.e. the uncertainty region before manipulation) of the object $\mathtt{o}$. We will use ν to refer to the uncertainty region as the object is being manipulated, and specifically $\nu[1]$ to refer to the final uncertainty region of the object after manipulation. We get $U(\mathtt{o})$ by modeling the error profile of our perception system. Each manipulation action outputs ν. Section 4.2 describes how this is computed for our actions.

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 ν. 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, **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 too; e.g. **Volume**$(\mathtt{o}, \nu)$ gives the volume of space swept by the uncertainty of the object during its manipulation.

4.1.3 How to move an object?

The traditional manipulation planning algorithms assume two types of actions: *Transfer* and *Transit* (Siméon et al., 2004, Stilman and Kuffner, 2006) or *Manipulation* and *Navigation* (Stilman et al., 2007). Transit does not manipulate any objects, Transfer manipulates only an already rigidly grasped object. Our algorithm lifts this assumption and opens the way for non-prehensile actions. At each planning step, our planner searches over a set of possible actions in its action library. For example in Step 1 of Fig. 4.4 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. We will describe the details of specific actions we use (e.g. push-grasp and sweep) in Section 4.2. Below we present the general properties an action should have so that it can be used by our planner.

In grasp based planners robot manipulation actions are simply represented by a trajectory of the robot arm: $\tau : [0,1] \to \mathcal{C}$ 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 `a` in our framework takes as an input the object to be moved `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 ν:

$$(\tau, \nu) \leftarrow a(\texttt{o}, G, avoidVol) \tag{4.1}$$

The returned values τ and ν must satisfy:

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

If the action cannot produce such a τ and ν, 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 the end of one object manipulation action to the start of other.

4.1.4 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 can in the example above. It is the goal configuration passed into the planner, e.g. the final configuration in Fig. 4.1. But for subsequent objects, the planner does not have a direct goal. Instead the object (e.g. the box in Step 2 of Fig. 4.4) needs to be moved out of a certain volume of space in order to make the previously planned actions (Step 1 in Fig. 4.4) feasible. We call this volume of space the *negative goal region* (NGR) at that step (shown as a purple region in Fig. 4.4) [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: $G \leftarrow \mathbf{StablePoses}(\texttt{o}) - NGR$.

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

[1]Note that the NGR has a 3D volume in space. In Fig. 4.4 it is shown as a 2D region for clarity of visualization.

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:

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

where τ is the planned robot trajectory, `o` is the manipulated object, and ν is the evolution of the uncertainty region of the object at that planning step.

4.1.5 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 $q_{\texttt{goal}} \in \mathbb{SE}(3)$. The planning process is initiated with the call:

$$\texttt{plan} \leftarrow \textbf{Reconfigure}(\texttt{goal}, \{q_{\texttt{goal}}\}, \{\}, \{\}, *)$$

The $*$ here means that the final configuration of the arm does not matter as long as the object is moved to $q_{\texttt{goal}}$.

Each recursive call to **Reconfigure** is a planning step (Alg. 2). 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 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.

4.2 Action Library

In this section we describe the actions implemented in our action library. The generic interface for actions is given in Eq. (4.1). There are four actions we implemented.

Algorithm 2: `plan` $\leftarrow$ **Reconfigure**($\mathtt{o}, G, NGR, \mathtt{move}, \mathtt{avoid}, r^{t+2}$)

```
1  repeat
2      a ← next action from action library
3      $avoidVol \leftarrow \sum_{\mathtt{i} \in \mathtt{avoid}} \textbf{Volume}(\mathtt{i}, U(\mathtt{i}))$
4      $(\tau_1, \nu) \leftarrow \mathtt{a}(\mathtt{o}, G, avoidVol)$
5      if $\tau_1$ is empty then
6          Continue at line 2
7      $\tau_2 \leftarrow \text{GoTo}(\tau_1[1], r^{t+2}, avoidVol + \textbf{Volume}(\mathtt{o}, \nu[1]))$
8      if $\tau_2$ is empty then
9          Continue at line 2
10     $\tau \leftarrow \tau_1 + \tau_2$
11     $vol \leftarrow \textbf{Volume}(\mathtt{robot}, \tau) + \textbf{Volume}(\mathtt{o}, \nu)$
12     $\mathtt{move_{next}} \leftarrow \mathtt{move} + \textbf{FindPenetrated}(vol, \mathtt{obj})$
13     if $\mathtt{move_{next}}$ is empty then
14         return $\{\tau\}$
15     $NGR_{next} \leftarrow NGR + vol$
16     $\mathtt{avoid_{next}} \leftarrow \mathtt{avoid} + \{\mathtt{o}\}$
17     foreach $\mathtt{i} \in \mathtt{move_{next}}$ do
18         $\mathtt{plan} \leftarrow \textbf{Reconfigure}(\mathtt{i}, \textbf{StablePoses}(\mathtt{i}) - NGR_{next}, NGR_{next}, \mathtt{move_{next}} -$
           $\{\mathtt{i}\}, \mathtt{avoid_{next}}, \tau[0])$
19         if plan is not empty then
20             return $\mathtt{plan} + \{\tau\}$
21 until all actions in action library are tried
22 return empty
```

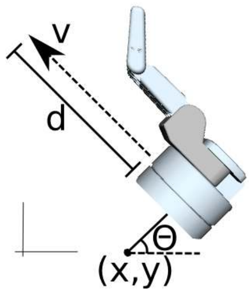
- *Push-grasp*: Grasp objects even when they have large initial uncertainty regions.

- *Sweep*: Push objects with the outer side of the hand. Useful to move large objects.

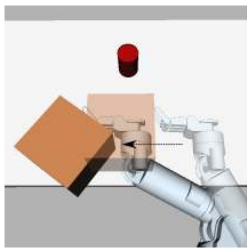
- *GoTo*: Moves from a robot configuration to another.

- *PickUp*: Combination of Push-grasp and GoTo. Used to grasp an object and move it to somewhere else by picking it up.

Each action can be parametrized in different ways in a given environment. For example the robot can Push-grasp an object by pushing in different directions. An action searches over its parameter space to find valid robot and object trajectories. In this section we specify these parameters for each action and present the way the search is done. We also explain how we compute the evolution of the uncertainty region.



(a) Parametrization


(b) Example sweep

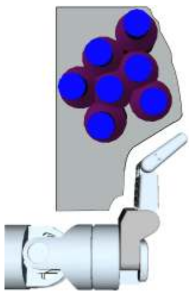

(c) Capture Region

Figure 4.5: (a) Sweep is parametrized by the initial hand pose and pushing distance. (b) Sweeping can move objects that are too large to be grasped. (c) The capture region of the sweep action for the a cylindrically symmetric bottle.

4.2.1 Push-Grasp

This action is presented in detail in Chapter 2. Here we explain how it is used in the context of rearrangement planning.

While a push-grasp we compute avoids the objects in `avoid` it is allowed to penetrate the space of other movable objects as explained in Section 4.1.1. But we try to minimize the number of such objects to get more efficient plans. Therefore we compute a heuristic value for the 36 different directions to push-grasp the object. We rotate the robot hand around the goal object and check the number of objects it collides with. We prefer directions v with a smaller number of colliding objects.

We also use the capture region to represent the evolution of the uncertainty region, ν. As the push proceeds, the top part of the capture region shrinks towards the hand and the resulting uncertainty region is captured inside the hand (Fig. 2.3(c)). Since the object cannot escape out of the capture region during the push-grasp, the uncertainty during the action can be conservatively estimated using the shrinked capture region at every discrete step. These series of capture regions can be used to represent ν. **Volume** operator samples poses from a capture region to compute the total volume.

4.2.2 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. 4.5b). Similar to Push-grasp, we parametrize a Sweep by $S(p_h, d)$; the hand pose and the pushing distance (Fig. 4.5a).

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. 4.5c) we use the pushing simulation 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. 4.5c).

We cannot know the exact location of the object after the sweep because a sweep action does not have a particular minimum pushing distance. We know that the object ends up inside the hand at the end of a push-grasp, and the uncertainty is very small. However, for sweeping this uncertainty can be large. We approximate the evolution of the uncertainty region of sweep by using samples from two different regions. The first region is object's initial uncertainty region. Until the sweeping hand makes a contact with a sample from this region that sample is included in ν. The second region is around the sweeping surface of the hand representing all possible poses of the object in contact with the hand surface.

4.2.3 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. We use the Constrained Bi-directional RRT planner (CBiRRT) (Berenson et al., 2009a) to implement this action.

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.

4.2.4 PickUp

In highly cluttered environments, moving objects locally may not be possible because all the immediate space is occupied. In such cases, picking up an obstacle object and moving it to some other surface may be desirable. We implement this action in our planner as the PickUp action. PickUp is also useful to move the original goal object of the plan to the final goal configuration. We implement PickUp as a Push-grasp followed by a GoTo.

4.3 Implementation and Results

4.3.1 Implementation

We implemented the planner on our robot HERB. We conducted simulation experiments using OpenRAVE. We created scenes in simulation and in real world. The robot's goal was

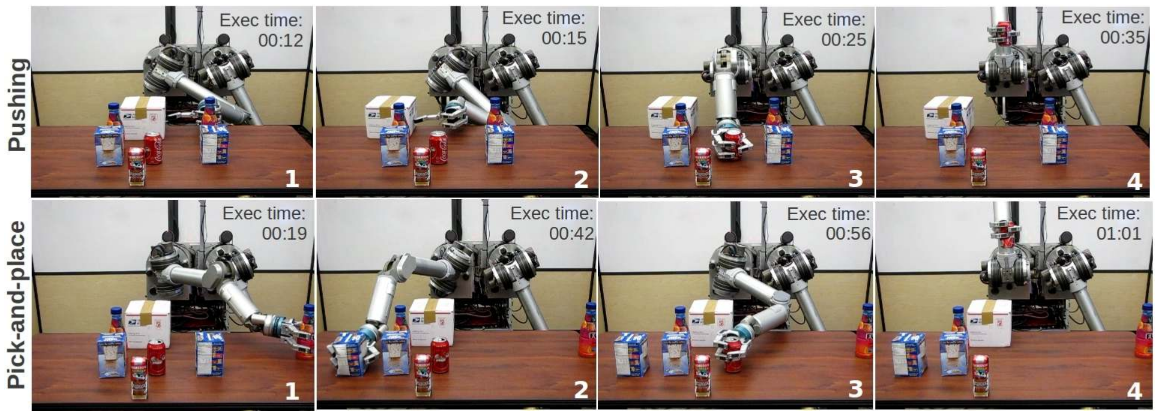


Figure 4.6: The plans that the *pushing planner* and the *pick-and-place* planner generates in the same scene are presented. The pushing planner is more efficient as it is able to sweep the large box to the side. The pick-and-place plan needs to move more objects and takes more time to execute. The planning time is also more for the pick-and-place planner (27.8 sec vs. 16.6 sec) as it needs to plan more actions.

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

4.3.2 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. 4.6. 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-

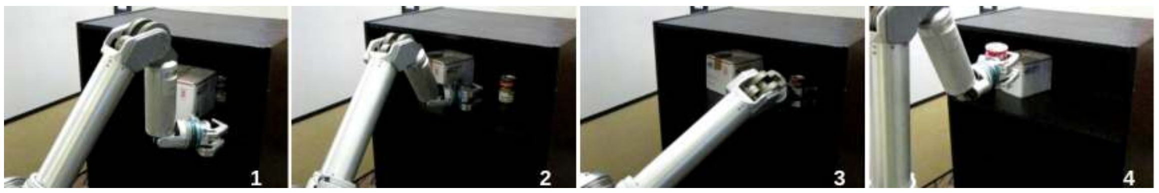


Figure 4.7: An example plan to retrieve a can hidden behind a large box on a shelf. The pick-and-place planner fails to find a plan in this scene. But pushing planner finds the presented plan. Planning time is 51.2 sec.

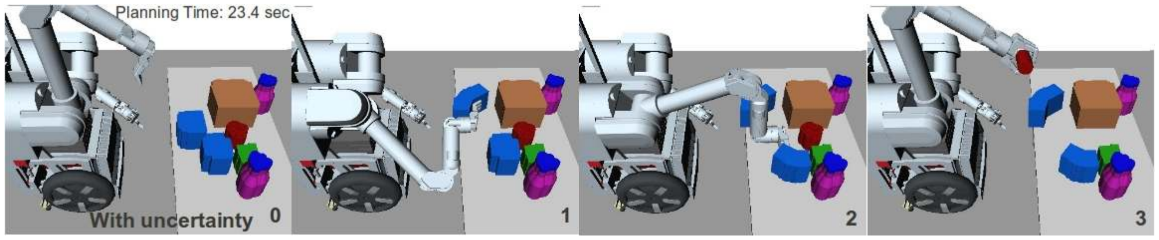


Figure 4.8: An example plan under high uncertainty and clutter. The pick-and-place planner fails to find a plan in this scene, as it cannot find a feasible grasp of the objects with such high uncertainty. Pushing planner succeeds in generating a plan, presented above.

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. 4.7 presents a scene where the pick-and-place planner fails. The pushing planner generates a plan and is presented in the figure.

4.3.3 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. 4.8 presents an example scene. Here the objects have an uncertainty region which is a Gaussian with $\sigma_{x,y} = 2cm$ for translation and $\sigma_\theta = 0.05rad$ for the rotation of the object. 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.

Table 4.1: Planning Times for Rearrangement Planning

	Total	GT	PU	SW	PG
Pushing	25.86	10.92	6.76	6.08	1.92
Pick-and-Place	12.52	6.54	5.98	-	-

4.3.4 Effect on planning time

We also conducted experiments to see the effect of adding the pushing actions to the pick-and-place planner in cases where both planners would work. We created five random scenes with different graspable objects and generated plans using the pushing and pick-and-place planner in these scenes. We ran each planner three times for each scene, due to the random components of our GoTo and PickUp actions. The average planning time for each planner is shown in Table 4.1 in seconds. The division of this time to each action is also shown (GT: GoTo, PU: pick-up, SW: sweep, PG: push-grasp).

On average, the pushing planner takes two times the time the pick-and-place planner takes. This is due to a variety of reasons. First, our implementation gives priority to pushing actions Push-grasp and Sweep before trying PickUp. An ordering where the PickUp comes first will generate results similar to the pick-and-place planner. The second reason is the large uncertainty region the Sweep action generates. This usually causes more objects to be moved, which is reflected in the higher time spent on the GoTo action.

4.4 Discussion

An important idea in the rearrangement planning algorithm is planning manipulator motions which are allowed to violate constraints (i.e. go through some objects): this is *the* method through which objects are identified to be moved out of the way. Currently this is achieved using an RRT implementation which is allowed to go through *any* of the objects in the environment if a free-space trajectory cannot be found. Most of the time this leads to spurious rearrangement actions, and making the problem harder than it really is, runs the risk of causing the rearrangement planner to fail.

Ideally, one would want a motion planner which goes through only the absolutely necessary objects for the rearrangement plan. An approximation would be to use a motion planner which minimizes the number of constraints violated. Hauser (2012) presents a motion planner which can produce paths violating the minimum number of constraints. Our current effort includes integrating this planner into our rearrangement planner to produce shorter rearrangement plans.

Another limitation of our planner is due to planning backwards in time. 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 runs out of space to move objects to and fails. This is a result of our planner being conservative with respect to uncertainty. A solution may be to interleave the execution of actions with re-inspection of the environment. In Chapter 7 we present our ideas about this for future work.

Chapter 5

Manipulating Clutter for Object Search

Imagine looking for the salt shaker in a kitchen cabinet. Upon opening the cabinet, you are greeted with a cluttered view of jars, cans, and boxes—but no salt shaker. It must be hidden near the back of the cabinet, completely obscured by the clutter. You start searching for it by pushing some objects out of the way and moving others to the counter until, eventually, you reveal your target.

Humans frequently manipulate their environment when searching for objects. If robotic manipulators are to be successful in human environments, they require a similar capability of searching for objects by removing the clutter that is in the way. In this context, clutter removal serves two purposes. First, removing clutter is necessary to gain visibility of the target. Second, it is necessary to gain access to objects that would be otherwise inaccessible.

Prior work has addressed the issues of interacting with objects to gain visibility and accessibility as separate problems. Work on the *sensor placement* (Espinoza et al., 2011) and *search by navigation* (Ye and Tsotsos, 1995, 1999, Shubina and Tsotsos, 2010, Sjo et al., 2009, Ma et al., 2011, Anand et al., 2013) problems focuses on moving the sensor to gain visibility.

Conversely, the *rearrangement planning* approach which presented an example of in Chapter 4 problems focus on moving objects to grant the manipulator access to previously-inaccessible configurations. These approaches would be effective at gaining access to the salt shaker once its pose is known, but are incapable of planning before the target is visually revealed.

Fig. 5.2 shows a scene in which both situations occur. In this figure, HERB is searching for the white battery pack hidden on a cluttered table. HERB uses its camera to detect and localize objects. As Fig. 5.2-Top shows, HERB is initially unable to detect the battery pack because it is occluded by the blue Pop-Tart box. From HERB's perspective, the

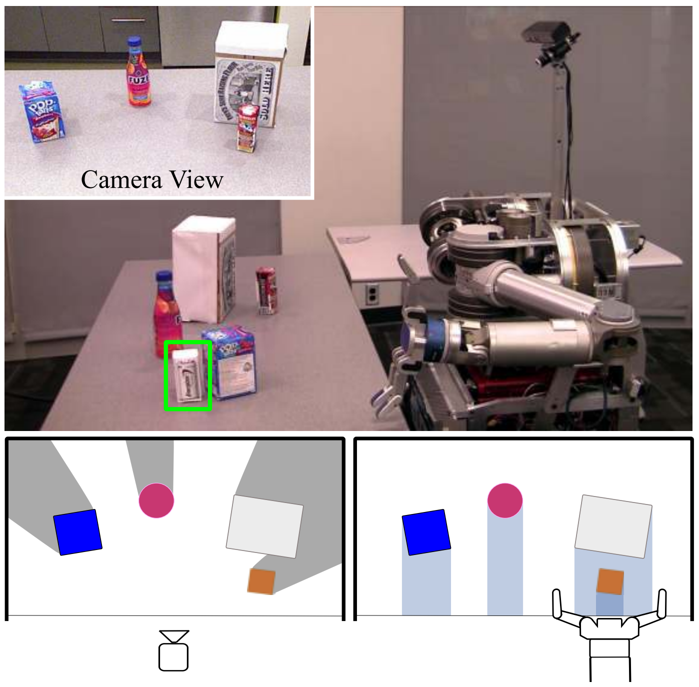


Figure 5.2: An example of the object search problem on a real robot. The robot is searching for a target object (highlighted by the bounding box) on the table, but its view is occluded (drawn as gray regions) by other objects. The robot must remove these objects to search for the target. Objects may block the robot's access to other objects.

battery pack could be hiding in any of the occluded regions shown in Fig. 5.2-Left. With no additional knowledge about the location of the target, HERB must sequentially remove objects from the scene subject to the physical limitations of its manipulator until the target is revealed. For example, Fig. 5.2-Right shows that HERB is unable to grasp the large white box without first moving the brown juicebox out of the way.

In this section, we formally describe the object search by manipulation problem by defining the *expected time to find the target* as a relevant optimization criterion and the concept of *accessibility* and *visibility* relations. armed with these definitions, we are able to propose and analyze algorithms for object search by manipulation. We make the following theoretical contributions:

Greedy is sometimes optimal: We prove that, under an appropriate definition of utility, the greedy approach to removing objects is optimal under a set of conditions, and provide insight into when it is suboptimal (Section 5.2).

The connected components algorithm: We introduce an alternative algorithm, called the *connected components algorithm*, which takes advantage of the structure of the scene to approach polynomial time complexity on some scenes (Section 5.4). Our extensive experiments show that this algorithm produces optimal plans under all situations, and we present a partial proof of optimality.

Finally, we demonstrate both algorithms on our robot HERB (Section 5.5.1 and Section 5.5.2) and provide extensive experiments that confirm the algorithms' theoretical properties (Section 5.5).

5.1 Formulation

We start with a scene $\mathcal{S}$ that is comprised of a known, static world populated with the set of movable objects $\mathcal{O}$, each of which has known geometry and pose.

A robot perceives the scene with its sensors and has partial knowledge of the objects that the scene contains. To the robot, the scene is comprised of the set of visible objects $\mathcal{O}_{seen} \subset \mathcal{O}$ and the volume of space V that is occluded to its sensors. In the object search problem, the occluded volume hides a target object $\texttt{target} \in \mathcal{O}$ with known geometry, but unknown pose. For the remainder of this section, we study a specific variant of the problem in which the target is the only hidden object, i.e. $\mathcal{O} = \mathcal{O}_{seen} \cup \{\texttt{target}\}$. We discuss the presence of other hidden objects in Section 5.7.

The robot searches for the target by removing objects from $\mathcal{O}_{seen}$ until the target is revealed to its sensors. We define the order in which objects are removed from the scene as an *arrangement*.

Definition 1 (Arrangement)**.** *An arrangement of the set of objects o is a bijection $\mathcal{A}_o : \{1, \ldots, |o|\} \to o$ where $\mathcal{A}_o(i)$ is the i^{th} object removed.*

Additionally, we define $\mathcal{A}_o(i, j)$ as the sequence of the i^{th} through the j^{th} objects removed by arrangement $\mathcal{A}_o$.

Given an arrangement $\mathcal{A}_o$ that reveals the target, the expected time to find the target is

$$E(\mathcal{A}_o) = \sum_{i=1}^{|o|} P_{\mathcal{A}_o(i)} \cdot T_{\mathcal{A}_o(1,i)} \tag{5.1}$$

where $P_{\mathcal{A}_o(i)}$ is the probability that the target will be revealed after removing object $\mathcal{A}_o(i)$ and $T_{\mathcal{A}_o(1,i)}$ is the time to move all objects up to and including $\mathcal{A}_o(i)$.

Our goal is to find the arrangement $\mathcal{A}^*_{\mathcal{O}_{seen}}$ that minimizes $E(\mathcal{A}^*_{\mathcal{O}_{seen}})$; i.e. reveals the target as quickly as possible.

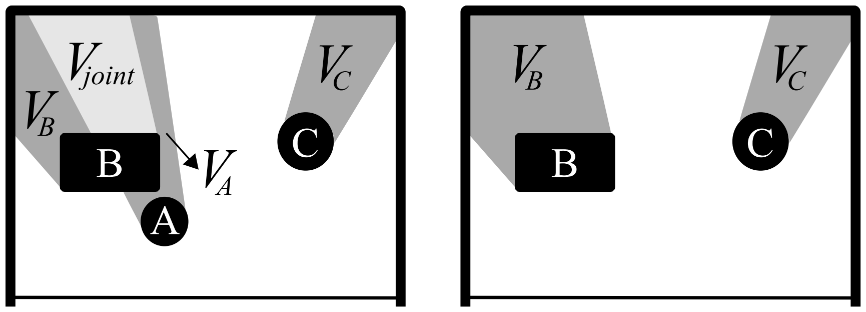
5.1.1 Visibility

When the robot removes a set of objects from the scene it reveals the volume behind those objects.

Definition 2 (Revealed Volume)**.** *The volume of space V_o revealed by removing objects $o \subseteq \mathcal{O}_{seen}$ from scene $\mathcal{S}$.*

In Fig. 5.3a we show the revealed volumes of objects in an example scene[1]. V_{joint} is

[1]We use two-dimensional examples, e.g. Fig. 5.3, throughout the section for clarity of illustration. Our actual formulation and implementation uses complete three-dimensional models of the scene, objects, and volumes.



(a) Initial Scene (b) `A` Removed

Figure 5.3: (a) An example of a scene containing a joint occlusion. Occlusions are drawn as dark gray and the joint occlusions as light gray. (b) The scene after `A` is removed.

jointly occluded by object `A` and `B`, and is *not* included in either $V_{\mathtt{A}}$ or $V_{\mathtt{B}}$. This is because V_{joint} will not be revealed if only `A` or only `B` is removed from the scene.

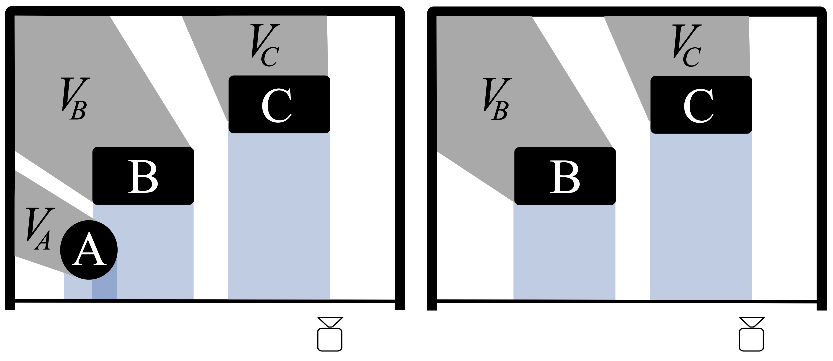
The volume that is revealed by removing an object may change as the scene changes. In Fig. 5.3b we show $V_{\mathtt{B}}$ after `A` is removed from the scene in Fig. 5.3a. Since `A` is no longer in the scene, $V_{\mathtt{B}}$ now includes V_{joint}. Similarly, $V_{\mathtt{A}}$ would expand to include V_{joint} if `B` was the first object removed from the scene. Regardless of the order in which `A` and `B` are removed, the revealed volume of $\{\mathtt{A},\mathtt{B}\}$ is $V_{\mathtt{A},\mathtt{B}} = V_{\mathtt{A}}+V_{\mathtt{B}}+V_{joint}$. In the most general case, an arbitrary number of objects can jointly occlude a volume. In that case, the volume would be revealed only after all of the occluding objects are removed from the scene.

We compute the probability that removing an object `A` will reveal the `target` using the revealed volume:

$$P_{\mathtt{A}} = \frac{V_{\mathtt{A}}}{V_{\mathcal{O}_{seen}}} \tag{5.2}$$

We compute the revealed volume in the configuration space (C-space) of the target object. In our implementation we assume that the target rests stably on the workspace; i.e. the target's pose can be represented by $(x, y, \theta) \in SE(2)$. We discretize the target's C-space to generate a set of candidate poses and estimate V_o as the number of `target` poses that become visible when o is removed.

Our framework supports any deterministic visibility criterion. However, partial views of objects are often hard to detect. Therefore, our implementation uses a conservative condition: we consider the target at a certain pose visible if and only if the sensor can see it completely. This is implemented by sampling points on the target, raytracing from the sensor to each point, and verifying that no ray is occluded.



(a) Initial Scene (b) `A` Removed

Figure 5.4: A scene where the greedy algorithm performs suboptimally due to an accessibility constraint.

5.1.2 Accessibility

The manipulator uses a motion planner to grasp an object and remove it from the scene. To achieve this, the object must be *accessible* to the manipulator. Accessibility is blocked by other visible objects, and also by the occluded volume, which the manipulator is forbidden to enter.

Definition 3 (Accessibility Constraint)**.** *There is an* accessibility constraint *from an object* `A` *to object* `B` *if* `A` *must be removed for the manipulator to access* `B`*.*

Any arrangement of objects in a scene must respect the objects' accessibility constraints. For example, in Fig. 5.2-Right, the access to the big box is blocked by the smaller box in front of it.

We identify the accessibility constraints using a motion planner, which returns a manipulator trajectory for each object in the scene. The manipulator trajectory for an object sweeps a certain volume in the space (illustrated as light blue regions in Fig. 5.2). Objects that penetrate the swept volume result in accessibility constraints. Additionally, objects for which the occluded volume penetrates the swept volume also result in accessibility constraints.

We also use the manipulator trajectory for an object `A` to compute $T_{\mathtt{A}}$ by estimating the time necessary to execute the trajectory on the robot. Since there is only a single action for each object, $T_{\mathtt{A}}$ is constant for a given scene and does not depend on the sequence in which objects are removed.

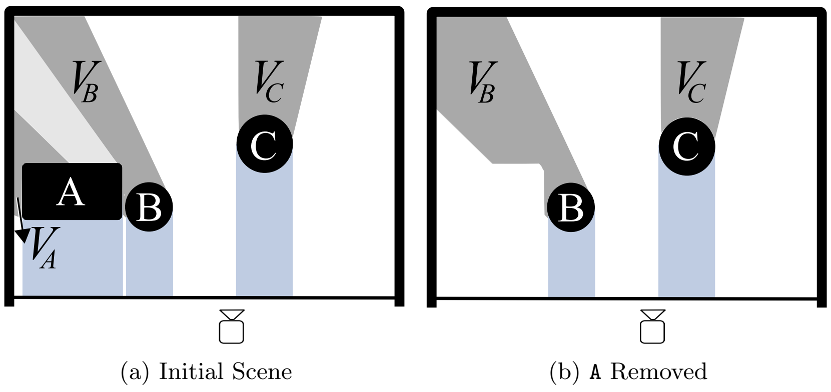


Figure 5.5: A scene where the greedy algorithm performs suboptimally due to a visibility constraint.

5.2 Utility and Greedy Search

In this section, we discuss a greedy approach to solving the object search by manipulation problem.

While the overall goal is to minimize the amount of time it takes to find the target, a greedy approach requires a utility function to maximize at every step. The faster the robot reveals large volumes, the sooner it will find the target. Using this intuition, we define the utility of an object similar to the utility measures defined for sensor placement (Ye and Tsotsos, 1995, Espinoza et al., 2011).

Definition 4 (Utility)**.** *The* utility *of an object* `A` *is given by*

$$U(\mathtt{A}) = \frac{V_{\mathtt{A}}}{T_{\mathtt{A}}}$$

This measure naturally lends itself to greedy search. A greedy algorithm for our problem ranks the accessible objects in the scene based on their utility and the removes highest utility object. This results in a new scene, whereby the algorithm repeats until the target is revealed. In the worst case, this continues until all objects are removed.

Unsurprisingly, it is easy to create situations where greedy search is suboptimal. Consider the scene in Fig. 5.4. In this scene, $V_{\mathtt{B}} \gg V_{\mathtt{C}} > V_{\mathtt{A}}$. For the sake of simplicity we assume that the time to move each object is similar, hence $U(\mathtt{C}) > U(\mathtt{A})$. As `B` is not accessible, the greedy algorithm compares $U(\mathtt{A})$ and $U(\mathtt{C})$ and chooses to move `C` first, producing the final arrangement $\mathtt{C} \rightarrow \mathtt{A} \rightarrow \mathtt{B}$. However, moving the lower utility `A` first is the optimal choice because it reveals $V_{\mathtt{B}}$ faster (Fig. 5.4b), and gives the optimal arrangement $\mathtt{A} \rightarrow \mathtt{B} \rightarrow \mathtt{C}$. It is easy to see that greedy can be made arbitrarily suboptimal by adding more and more objects with utility $U(\mathtt{C})$ to the scene.

We present a second example of greedy's suboptimality in Fig. 5.5. In this scene, all objects are accessible, $V_{\mathtt{C}} > V_{\mathtt{A}}$, and $V_{\mathtt{C}} > V_{\mathtt{B}}$. The greedy algorithm inspects the utilities and moves C first. However, there is a large volume jointly occluded by A and B, such that when either A or B is removed, the volume revealed by the second object significantly increases. We illustrate this in Fig. 5.5b with A is removed. Hence, the optimal arrangement is $\mathtt{A} \to \mathtt{B} \to \mathtt{C}$ because it quickly reveals the large volume jointly occluded by A and B.

The examples in Fig. 5.4 and Fig. 5.5 may suggest a k-step lookahead planner for optimality. However, the problem is fundamental: one can create scenes where arbitrarily many objects jointly occlude large volumes, or where arbitrarily many objects block the accessibility to an object that hides a large volume behind it.

Surprisingly, however, it is possible to create nontrivial scenes where greedy search is *optimal*. We define the requirements of such scenes in the following theorem.

Theorem 5.2.1. *In a scene where all objects are accessible and no volume is jointly occluded, a planner that is greedy over utility minimizes the expected time to find the target.*

Proof. Suppose that $\mathcal{A}^*$ is a minimum expected time (i.e. optimal) arrangement. For any i, $1 \leq i < |\mathcal{O}_{seen}|$, we can create a new arrangement, $\mathcal{A}$, such that the i^{th} and $(i+1)^{\text{th}}$ objects are swapped; i.e. $\mathcal{A}(i) = \mathcal{A}^*(i+1)$ and $\mathcal{A}(i+1) = \mathcal{A}^*(i)$. $\mathcal{A}$ must be a valid arrangement because all objects are accessible.

No volume is jointly occluded, so the revealed volume of all objects will stay the same after the swap; i.e. $V_{\mathcal{A}^*(i)} = V_{\mathcal{A}(i+1)}$ and $V_{\mathcal{A}^*(i+1)} = V_{\mathcal{A}(i)}$. Since the rest of the two arrangements are also identical, using Eq. (5.1) and Eq. (5.2), we can compute the difference between $E(\mathcal{A})$ and $E(\mathcal{A}^*)$ to be:

$$E(\mathcal{A}) - E(\mathcal{A}^*) = V_{\mathcal{A}^*(i)} \cdot T_{\mathcal{A}^*(i+1)} - V_{\mathcal{A}^*(i+1)} \cdot T_{\mathcal{A}^*(i)}. \tag{5.3}$$

$E(\mathcal{A}^*)$ is optimal, therefore $E(\mathcal{A}) - E(\mathcal{A}^*) \geq 0$ and

$$\frac{V_{\mathcal{A}^*(i)}}{T_{\mathcal{A}^*(i)}} \geq \frac{V_{\mathcal{A}^*(i+1)}}{T_{\mathcal{A}^*(i+1)}},$$

which is simply $U(\mathcal{A}^*(i)) \geq U(\mathcal{A}^*(i+1))$. Hence, the optimal arrangement consists of objects sorted in weakly-descending order by their utilities.

There can be more than one weakly-descending ordering of the objects if multiple objects have the same utility. To see that all weakly-descending orderings are optimal, the same reasoning can be used to show that swapping two objects of the same utility does not change the expected time of an arrangement. □

The greedy algorithm is incredibly efficient in terms of computational complexity. At each step, the algorithm finds the accessible object with maximum utility in linear time.

In a scene of n objects, this results in a total computational complexity of $O(n^2)$. We show in Section 5.4 that the worst-case complexity of the optimal search is $O(n^2 2^n)$. The theorem, however, shows that there are scenes in which greedy is optimal. We shall show in Section 5.5 that these scenes do occur surprisingly regularly even with randomly generated object poses. However, as we have shown above, the greedy algorithm can also produce arbitrarily suboptimal results.

In the next section we present an algorithm based on A-Star search, which is always optimal but has exponential computational complexity. Then, in Section 5.4 we present a new algorithm which approaches the polynomial complexity of the greedy algorithm, yet maintains optimality in the general case as shown by our empirical evaluations in Section 5.5.

5.3 A-Star Search Algorithm

We use A-Star search to find the optimal arrangement in a scene. A-Star works with a directed-acyclic-graph structure where the nodes are the set of remaining objects in the scene. Neighbors are scenes with one accessible object removed. Assume the partial arrangement $\mathcal{A}$ of k objects in the scene reaches a node in the search graph. The cost-to-come is

$$f = \sum_{i=1}^{k} \left(\frac{V_{\mathcal{A}(i)}}{V_{\mathcal{O}_{seen}}} \right) T_{\mathcal{A}(1,i)}$$

The cost-to-go can be approximated as

$$g = \left(\frac{V_{\mathcal{O}_{seen}} - V_{\mathcal{A}(1,k)}}{V_{\mathcal{O}_{seen}}} \right) (T_{\mathcal{A}(1,k)} + \min_{a \in \{\mathcal{O}_{seen} \setminus \mathcal{A}(1,k)\}} (T_a))$$

The cost-to-go heuristic optimistically reasons that in the $(k+1)^{\text{th}}$ action, all the remaining occluded volume will be revealed. Among the remaining objects, $\mathcal{O}_{seen} \setminus \mathcal{A}(1,k)$, we find the object that can be removed with the minimum time and use its time as the time of the $(k+1)^{\text{th}}$ action. The heuristic is admissible as it underestimates the time to find the target.

The A-Star search produces the optimal arrangement. However, running it on a large scene is intractable due to its high computational complexity. A-Star must search over a graph containing up to 2^n nodes and $O(n^2)$ edges, resulting in a worst-case complexity of $O(n^2 2^n)$.

5.4 Connected Components Algorithm

The structure of the object search problem becomes more clear once we represent the visibility and accessibility constraints of a scene as a graph. Each node of this graph corresponds to an object in the scene. There is an edge between the nodes `A` and `B` if:

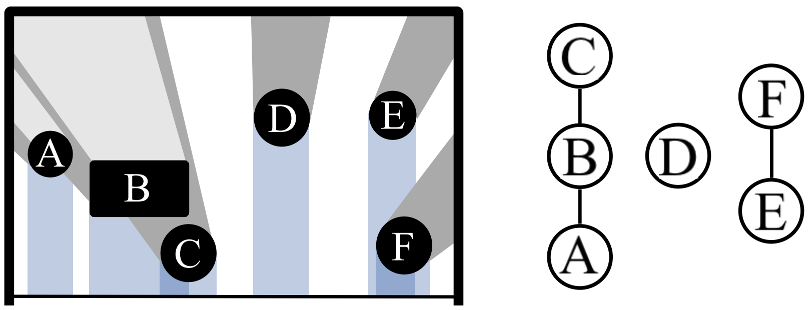


Figure 5.6: Left: An example scene. Volumes occluded by a single object are shown in dark gray, joint occlusions are shown in light gray, and swept volumes are shown in light blue. Right: The corresponding graph with three connected components.

Algorithm 3: Object Search With Connected Components

```
1  $\{c^1, c^2, ..., c^m\} \leftarrow$ FindConnectedComponents
2  foreach connected component $c^i$ do
3      $\mathcal{A}^*_{c^i} \leftarrow$ AStar($c^i$)
4  $\mathcal{A}^*_{\mathcal{O}_{seen}} \leftarrow [\,]$
5  repeat
6      bag $\leftarrow \emptyset$
7      foreach component arrangement $\mathcal{A}^*_{c^i}$ do
8          for $j \leftarrow 1$ to $|c^i|$ do
9              bag.Add( $\mathcal{A}_{c^i}(1, j)$ )
10     seq $\leftarrow \arg\max_{\mathcal{A} \in \texttt{bag}} U(\mathcal{A})$
11     Add seq to the end of $\mathcal{A}^*_{\mathcal{O}_{seen}}$
12     Remove seq from the $\mathcal{A}^*_{c^i}$ it belongs
13 until all objects are in the plan
14 return $\mathcal{A}^*_{\mathcal{O}_{seen}}$
```

- `A` is blocking the access to `B`, or vice versa; or
- `A` and `B` are jointly occluding a non-zero volume.

An example scene and the corresponding graph is in Fig. 5.6.

We can divide the constraint graph into *connected components*. A connected component of the graph is a subgraph such that there exists a path between any two nodes in the subgraph (Hopcroft and Tarjan, 1973). For example, there are three connected components in Fig. 5.6: {`A`, `B`, `C`}, {`D`}, and {`E`, `F`}.

A key insight is that the objects in a connected component do not affect the utility of the objects in another connected component. Hence, we can perform an optimal search, e.g. using A-Star, to solve the arrangement problem for a connected component independently and then merge the solutions to produce a complete arrangement of the scene.

It is non-trivial to merge arrangements of multiple connected components. The complete plan may switch from one connected component to the other and then switch back to a previous component. Our algorithm provides an efficient greedy way to perform this merge.

The examples in Fig. 5.4 and Fig. 5.5 show that the utility of a single object is not informative enough to achieve general optimality with a greedy algorithm. Instead, we consider the utility of removing multiple objects from the scene.

Definition 5 (Collective Utility). *The* collective utility *of a set of objects o is given by*

$$U(o) = \frac{V_o}{T_o}$$

A general greedy approach which considers the collective utility of all possible sequences of all sizes in the scene would quickly become infeasible as the number of such sequences is $O(|o|!)$. In our case, we take advantage of the fact that we have optimal plans for each connected component in which the objects are already sorted. We then need to compute collective utilities of only the prefixes (i.e. the first k objects where k ranges from 1 to the size of the connected component) of these optimal sequences.

We present our algorithm in Alg. 3 that uses the collective utility of sequences from connected components to generate an arrangement of the complete scene. It first identifies the connected components in the scene (Line 1). Then it finds the optimal arrangement internal to a connected component using A-Star search (Line 3). It then merges these arrangements iteratively by finding the maximum utility[2] prefixes of the optimal arrangements of the connected components.

In Section 5.5 we show that Alg. 3 generates the optimal result in all scenes we tried it on and it uses a fraction of the time A-Star requires on the complete scene. We present a partial proof of our algorithm's optimality in the appendix.

5.4.1 Complexity of the Connected Components Algorithm

The connected components algorithm divides the set of objects into smaller sets, runs A-Star on each connected component, and then merges the plans for each component. If the scene has no constraints, then there is one object per connected component and this algorithm reduces to the greedy algorithm. Conversely, if the constraint graph is connected, this algorithm is equivalent to running A-Star on the full scene. Therefore, the performance of this algorithm ranges from $O(n^2)$, the performance of the greedy algorithm, to $O(n^2 2^n)$, the performance of A-Star, depending upon the size of the connected components. Geometric limitations put an upper bound on the number of accessibility and joint occlusion constraints that are possible in a given scene, so it is unlikely that any scene will exercise the worst

[2]In the rare event that that multiple sequences share the maximum utility, the algorithm breaks the tie by choosing the sequence with the maximum utility prefix recursively.

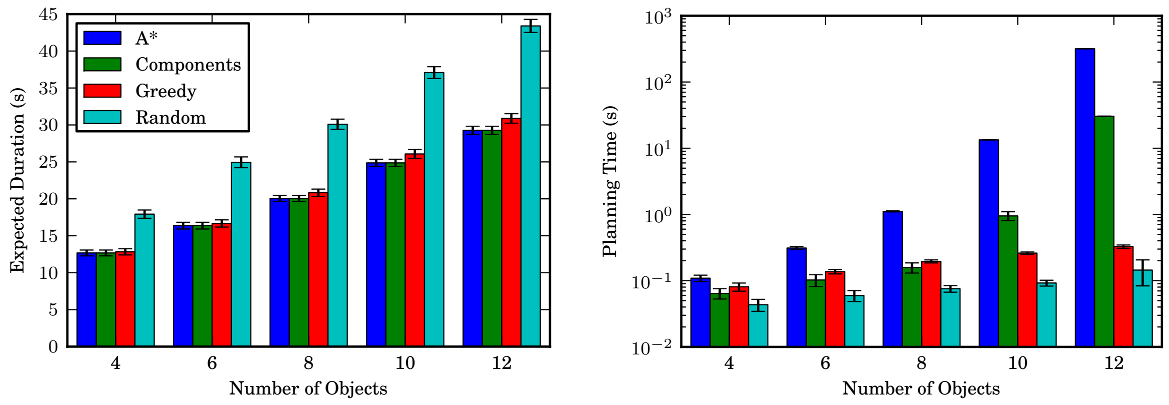


Figure 5.7: Performance of the random, greedy, A-Star, and connected component planners as a function of number of objects. All results are averaged over approximately 400 random scenes and are plotted with their 95% confidence interval. The planning times are presented in log-scale, where the confidence intervals are also plotted as log-scale relative errors (Baird, 1995).

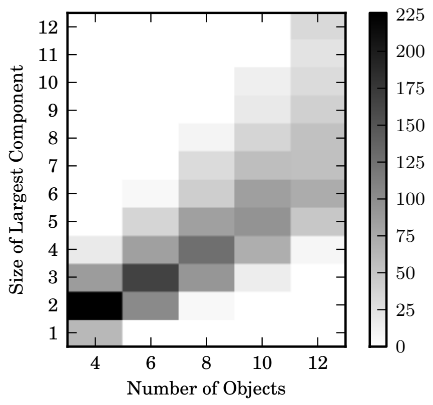
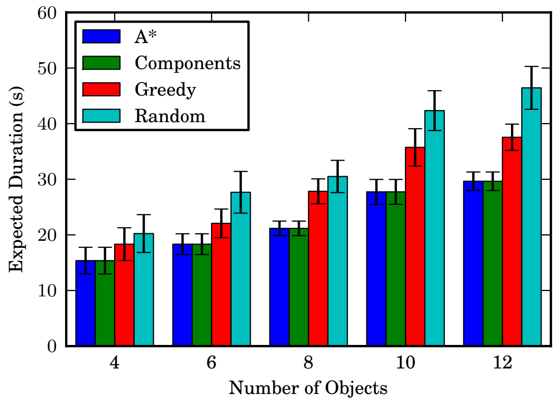


Figure 5.9: The relationship between scene size and the size of the largest connected component is plotted as a two-dimensional histogram.

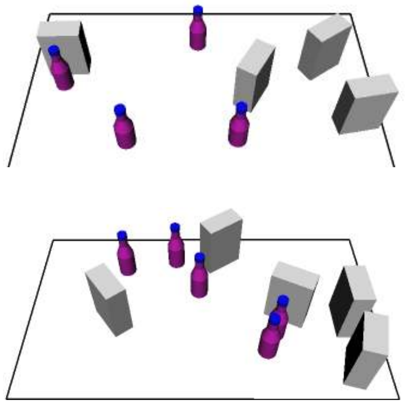
case performance. These performance gains will be most significant on large scenes in which objects are spatially partitioned, e.g. on different shelves in a fridge, but will be modest on small, densely packed scenes.

5.5 Experiments and Results

We investigated the performance of the different algorithms through extensive experiments in simulation and on a real robot. We implemented the greedy, A-Star, and connected components algorithms in OpenRAVE (Diankov and Kuffner, 2008). We also implemented a baseline algorithm which randomly picks an accessible object and removes it from the scene. We evaluated these algorithms on randomly generated scenes. Each scene contained



(a) Expected Search Duration



(b) Example Scenes

Figure 5.10: (a) 95th percentile of expected time to find the target (b) Two example scenes where greedy performed poorly. The black lines denote the workspace boundary.

n objects—half juice bottles and half large boxes—that were uniformly distributed over a wide 1.4×0.8 m workspace. None of the generated scenes contained hidden objects and the planner used a motion planner based on the capabilities of a simple manipulator. The manipulator was only capable of moving straight, parallel to the table and at a constant speed of 0.1 m/s. Visibility was simulated using the pinhole camera model under the conservative assumption that an object is visible if and only if it is completely unoccluded.

We present results from scenes with 4, 6, 8, 10, and 12 objects in Fig. 5.7 along with the 95% confidence intervals. We conducted approximately 400 simulations for each different number of objects, resulting a in total of 2000 different scenes. The data in Fig. 5.7a shows that the greedy algorithm becomes increasingly suboptimal as the number of objects increases. All three algorithms significantly outperform the random algorithm, which serves as a rough upper bound for the expected search duration. Unfortunately, the optimality of A-Star comes with the cost of exponential complexity in the number of objects. This complexity causes the planning time of A-Star to dominate the other planning times shown in Fig. 5.7b (note the logarithmic scale).

While still optimal in all 2000 scenes, the connected components algorithm achieves much lower planning times than A-Star. By running A-Star on smaller subproblems, the connected components algorithm is exponential in the size of the largest connected component, k, instead of the size of the entire scene. Fig. 5.9 shows that $k \approx n/2$ for $n \leq 8$ and increases when $n = 10$, causing the large increase in planning time between $n = 8$ and $n = 10$ in Fig. 5.7b. With fixed computational resources, these results show that the connected components algorithm is capable of solving most scenes of size $2n$ in the amount of time it would take A-Star to solve a scene of size n. For sparse scenes, the connected components algorithm achieves optimality with planning times that are comparable those of the greedy algorithm.

One surprising results of our experiments is that, while greedy is not optimal in the general case, it does remarkably well on average. We found that in 50% of the 2000 different scenes, the greedy algorithm produced the optimal sequence. Our explanation for greedy's performance is that the geometry of our workspace enforces a tradeoff between the volume occluded by an object and the number of objects that block its accessibility: For an object to occlude a large volume it must be near the front of the workspace, which makes it unlikely that multiple objects can be placed in front of it.

To see the greedy's worst-case behavior, we plotted the expected time to find the target for the 5% of scenes where greedy performed worst in Fig. 5.10a. Across all the scenes, the worst performance was 2.04 times the expected duration of the optimal sequence. We show two example scenes where greedy performs poorly in Fig. 5.10b. Both scenes include small bottles blocking access to large boxes. There is very little volume hidden behind the bottles, so the boxes are—suboptimally—removed late in the plan.

5.5.1 Real Robot Implementation

We implemented the greedy and connected components algorithms on our robot HERB. We used HERB's camera and the MOPED (Martinez et al., 2010) system to detect and locate objects in the scene. We present an example scene where HERB successfully found the target object using the greedy algorithm in Fig. 5.11. In this scene the target object, a battery pack, is hidden behind the large box, which also occludes the largest volume. Since the large box is inaccessible, the greedy planner compares the utilities of the other three objects, and removes the largest utility object at each step. Even though the large box is hiding a large volume, the greedy planner removes it last, resulting in a long task completion time.

In Fig. 5.12 the scene is the same but HERB uses the connected components algorithm. There are three connected components in this scene $\{\texttt{BlueBox}\}$, $\{\texttt{Bottle}\}$, and $\{\texttt{LargeBox}, \texttt{SmallBox}\}$. The connected components algorithm considers the collective utilities of multiple objects from each connected component, including both $U(\texttt{SmallBox})$ and $U(\texttt{SmallBox}, \texttt{LargeBox})$. The utility of `SmallBox` is very small compared with the other immediately accessible objects, but combined with the `LargeBox`, their utility is large enough that the algorithm removes `SmallBox` as the first object. It then removes the large box and finds the target object. We present the actual footage of these experiments at
http://youtu.be/i06GBj1iDOo

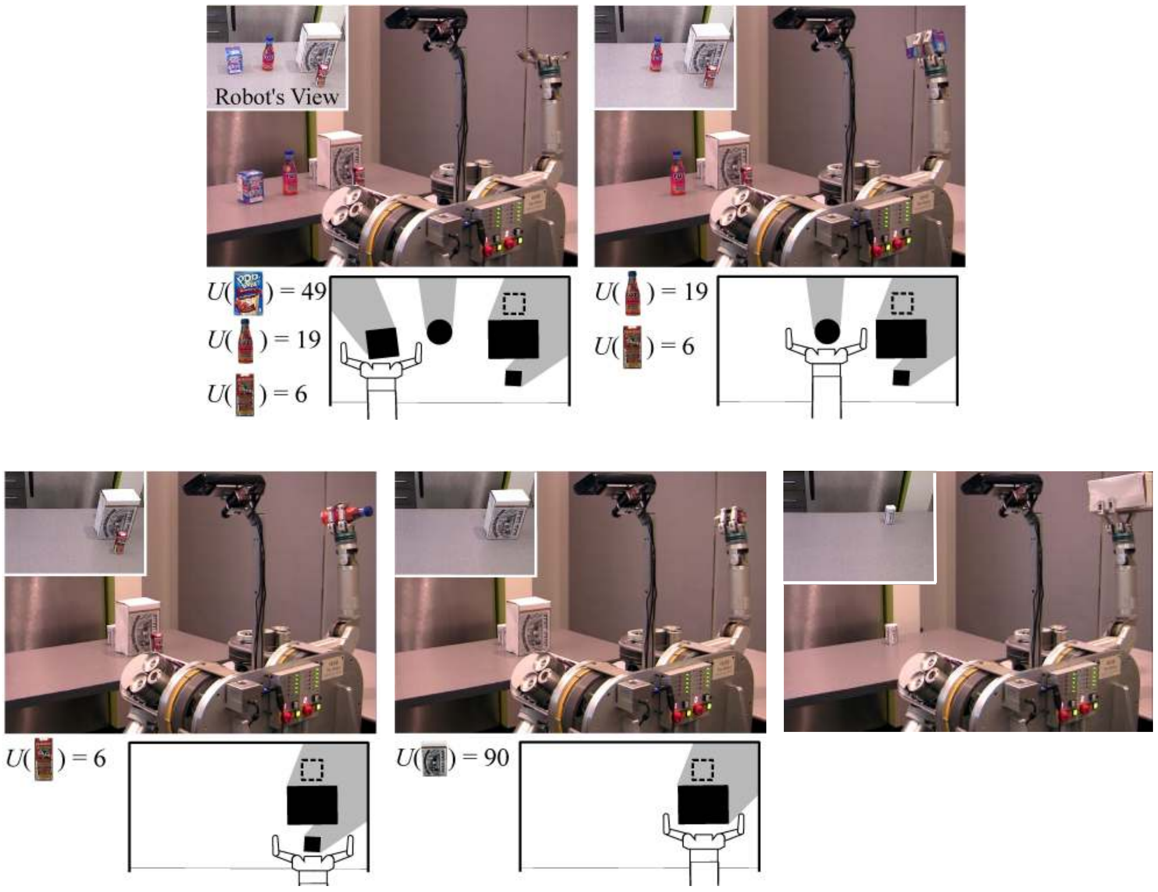


Figure 5.11: Greedy planner. We present the utility of all accessible objects at each step. The pose of the target (unknown to the robot) is marked with dashed lines in the illustration.

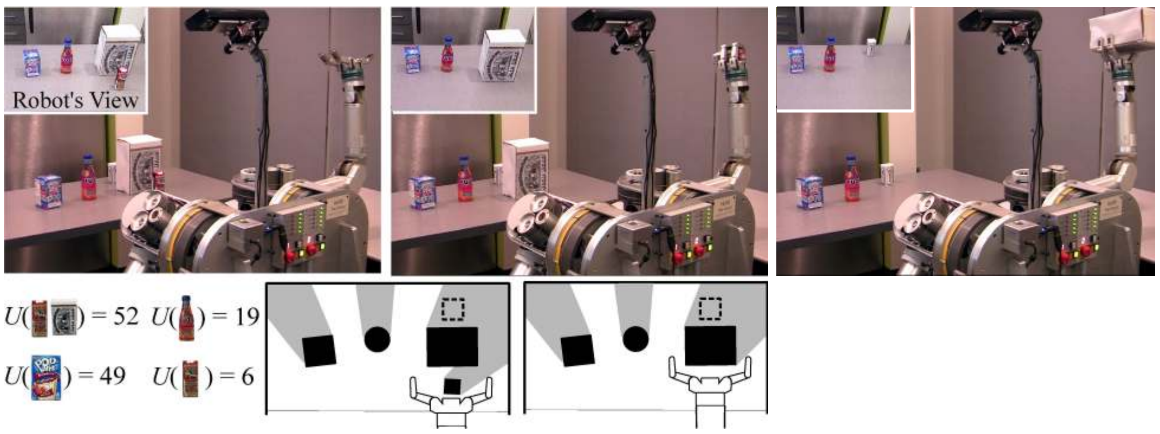


Figure 5.12: Connected-components planner. The utilities of all prefixes from each connected component are presented at each step.

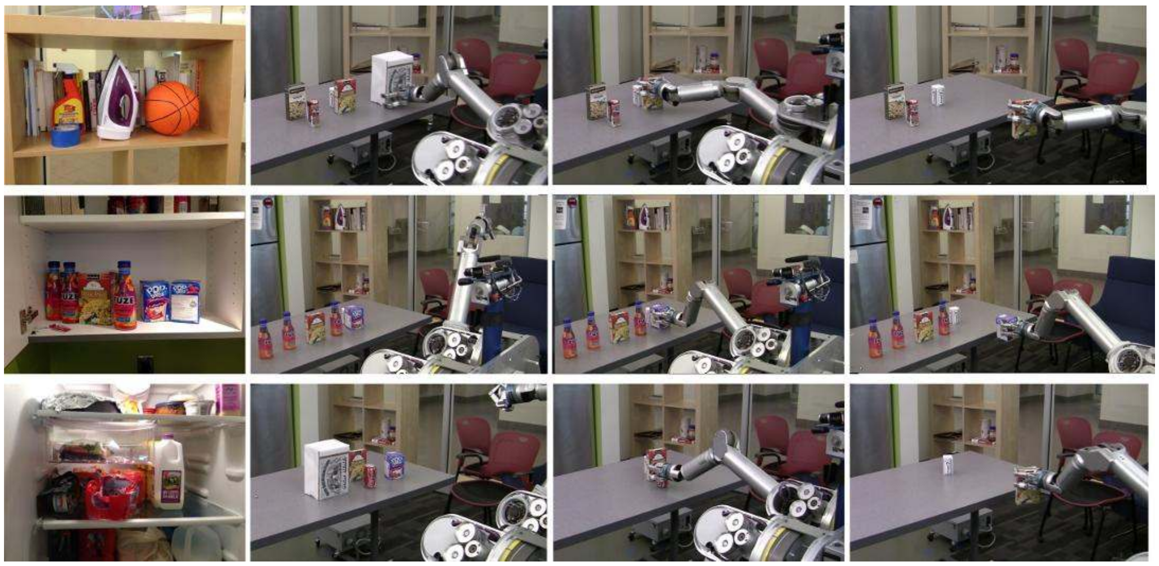


Figure 5.13: Example executions on scenes inspired by real human environments. Scenes are inspired from a cluttered human shelf (top), a cabinet (center), and a fridge (bottom).

Table 5.1: Object Search Planning and Execution Times

	Total Time	Planning	Execution
Shelf	132.7s	16.1s	116.6s
Cabinet	94.6s	26.1s	68.5s
Fridge	242.0s	16.7s	225.3

5.5.2 Performance in Human Environments

Objects are not distributed randomly in real human environments: they display a structure specific to human clutter. We conducted a simple evaluation of our planner by creating scenes which are similar in structure to human clutter.

For this evaluation we identified three different places where a robot might need to search for an object by manipulation: a bookshelf, a cabinet, and a fridge. We captured images of the natural clutter in these environments in our lab. We display these images as the leftmost column in Fig. 5.13.

Limitations of our robot's perception system and the difference between the sizes of a human arm/hand and our robot's manipulator prevented us from running our planner directly on these scenes. Therefore, we constructed new scenes that are scaled up to the dimensions of HERB's manipulator and consist of objects that our perception system can reliably detect. We attempted to faithfully mimic the relative size and configuration of objects in the original scenes as much as possible. We hid the target object randomly in the occluded portion of the table.

We present snapshots from our robot's search for the target as the rows of Fig. 5.13. All plans were automatically generated by the connected components algorithm and HERB successfully found the target in all three scenes. Table 5.1 shows the planning time, execution time, and the total time it took the robot to find the object. Note that execution time is the dominating factor, emphasizing the importance of generating short plans when searching for objects with a real robot.

5.6 Planning to Place Objects

The robot must place an object down before picking up another one. If the robot is allowed to place the object at a new pose where it creates new visibility or accessibility relations, then the object search problem becomes a version of *rearrangement planning*. Here, we avoid this complexity by placing objects only at poses that do not create new accessibility or visibility relations.

Our formulation requires us to compute the time it takes to manipulate an object before we decide the arrangement. We use fixed placement poses on a nearby empty surface to satisfy this constraint. We found this to be a reasonable strategy in practice: Even when the robot is working in a densely crowded cabinet shelf, there is usually a nearby counter or another shelf to place objects on.

However, one can also *re-use* the explored space to place objects: after an object is picked up and the robot sees the volume behind that object, the planner can safely use this volume. In particular, for an arrangement $\mathcal{A}_{\mathcal{O}_{seen}}$, object $\mathcal{A}_{\mathcal{O}_{seen}}(i)$ can be placed where:

- it avoids penetrating $V_{\mathcal{A}_{\mathcal{O}_{seen}}(i+1,|\mathcal{O}_{seen}|)}$,
- it avoids occluding $V_{\mathcal{A}_{\mathcal{O}_{seen}}(i+1,|\mathcal{O}_{seen}|)}$,
- it avoids blocking access to the objects $\mathcal{A}_{\mathcal{O}_{seen}}(i+1, |\mathcal{O}_{seen}|)$,
- it avoids colliding the placement poses of the objects $\mathcal{A}_{\mathcal{O}_{seen}}(1, i-1)$.

In Fig. 5.14 we illustrate each of these constraints and the remaining feasible placement poses for an object.

Surprisingly, certain scene structures lead to very simple and fixed placement strategies. For example, in a scene where there are no accessibility relations and no joint occlusions (where the greedy algorithm is optimal), an object can be placed where it was picked up: the robot lifts an object, looks behind it, and places it back. This strategy respects the constraints listed above.

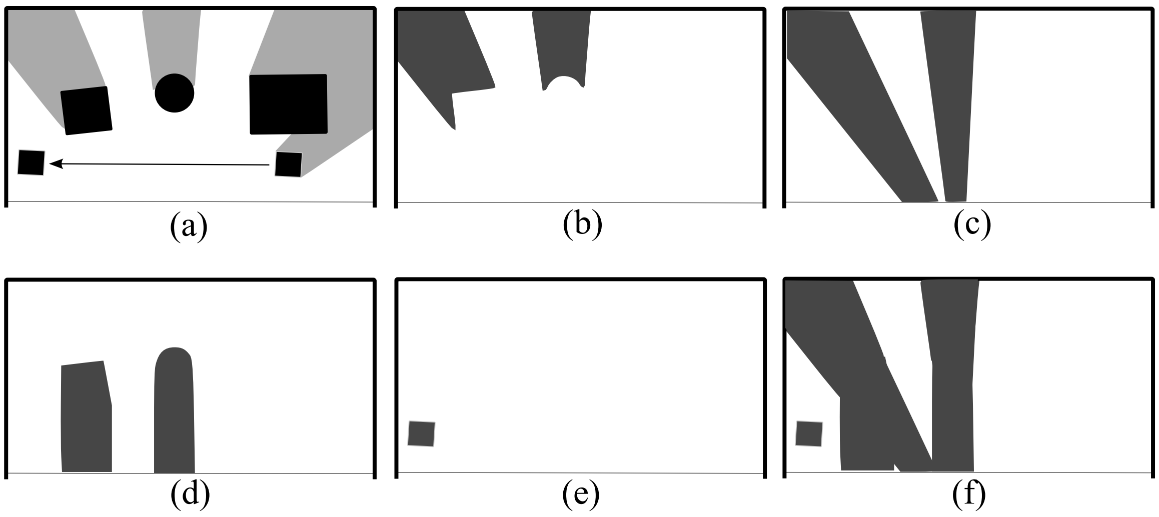


Figure 5.14: An example illustrating placement constraints. (a) In this scene the small box is moved to the left and then the large box is picked up. Now the planner must place the large box. The large box *cannot* be placed where (b) it will penetrate the volume which is not explored yet; (c) it will occlude the volume which is not explored yet; (d) it will block access to the objects which are not moved yet; (e) it will collide with the new poses of the objects which are already moved. (f) The combined placement constraints for the large box.

5.7 Replanning for Hidden Objects

All of the algorithms described above can be easily generalized to handle environments that contain hidden objects in addition to the target. Objects must be smaller than the target object or there is danger of the arm colliding with an hidden object while searching for the target. If this condition holds, then one can simply re-execute the planner on the remaining objects whenever an hidden object is revealed. This strategy is optimal given the available information if there is no a priori information about the type, number, or location of the hidden objects. If there are k hidden objects, then this replanning strategy multiplies the total planning time of an optimal algorithm by a factor of $O(k)$. In the case of the greedy or random algorithm, the replanning adds $O(k)$ overhead from reevaluating visibility after each object is revealed.

Fig. 5.15 shows an example of replanning on a scene containing six objects. Two objects, shown as semi-transparent in the figure, are initially hidden and are revealed once the occluding objects are removed. The robot begins by executing the connected components planner on a scene containing the four visible objects. After executing the first two actions in that plan, the robot detects that a new object has been revealed and replans for the remaining objects. In this case, the optimal ordering is unchanged and the newly-revealed

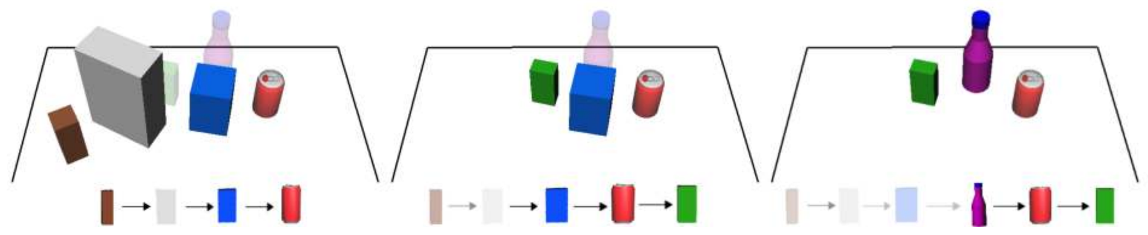


Figure 5.15: Example of replanning on a scene with two hidden objects. Each replanning stage is shown as a separate frame along with the corresponding plan. Hidden objects are shown as semi-transparent and the workspace bounds are indicated by a black line.

object is simply appended to the existing plan. After executing another action, the second hidden object is revealed and the robot must replan a second time. Order of the optimal sequence is changed by the addition of the hidden object and it is suboptimal to continue executing the previous plan.

5.8 Discussion

Finally, we discuss some of the limitations of our approach to the object search problem.

Perception Model. Our current implementation assumes that an object can be detected by the robot only if the object is fully visible. Our general framework is not tied to this assumption and can in fact accommodate any other deterministic perception model. However, real perception systems often do not behave deterministically because of unmodeled conditions such as the lighting of the environment. This may require an object search algorithm to consider the possibility that the perception system does not detect an object even if it is "out there".

Complex Motion Planners. Within this planner we use a straight-path reaching for objects, which is conceptually simple: there is a single action for an object. We are excited about studying how a more complex motion planner, e.g. one returning a minimum-constraint violation trajectory Hauser (2012), can be integrated into our system.

Sensor Planning. Aside from reaching to objects, our robot does not move its base. By combining the ability of searching by manipulation with sensor planning, the robot could find targets quicker. Sensor planning would include working with multiple camera poses and planning for the base when searching for a target in a larger environment.

Part IV

Future Work and Conclusion

Chapter 6

Pushing with Contact Feedback

The actions we presented so far are open-loop. In this section we layout our ideas on how feedback can be used during the execution of low-level actions. In particular, we will discuss how feedback can be used for push-grasping.

The first question we ask is: given a push-grasp for which the capture region includes the complete uncertainty, would feedback be useful at all? The answer is yes. Take the example in Fig. 6.1. The hand is located in front of a bottle, which has a large pose uncertainty. The robot needs a large pushing distance to capture all the uncertainty (Fig. 6.1(a)). Now, let's say, as the robot moves it feels that its fingertip contacts the bottle (Fig. 6.1(b)). Using this information the robot can shrink the uncertainty region of the bottle (Fig. 6.1(c)). Now the robot needs to execute a much shorter push-grasp.

The example above shows that even with our current approach to push-grasping, we can gain by using feedback. But finding push-grasps which cover the whole uncertainty region are costly. It takes more and more time as there is increasing clutter and uncertainty in the scene. We can see this effect on the lower-right quarter of Table 2.1. To speed up the search, we can relax the constraint which requires the capture region to include all uncertainty at the beginning. We can start executing a push-grasp and then use feedback to increase the probability of success.

One aspect of the problem is the continuous estimation of the object pose using contact feedback, and the other aspect is adapting the push-grasp online. We started exploring the former, using probabilistic methods like particle filters for object pose estimation with stochastic motion and observation models.

We observed that conventional particle filters (Thrun et al., 2005, Zhang and Trinkle, 2012) suffer from a startling problem in contact manipulation: *they systematically perform worse as sensor resolution or sensor update rate increases.* The problem arises because contact sensing is highly *discriminative* between contact and no-contact states: if a particle (i.e. a hypothesized object pose) is infinitesimally close to the robot hand but not touching

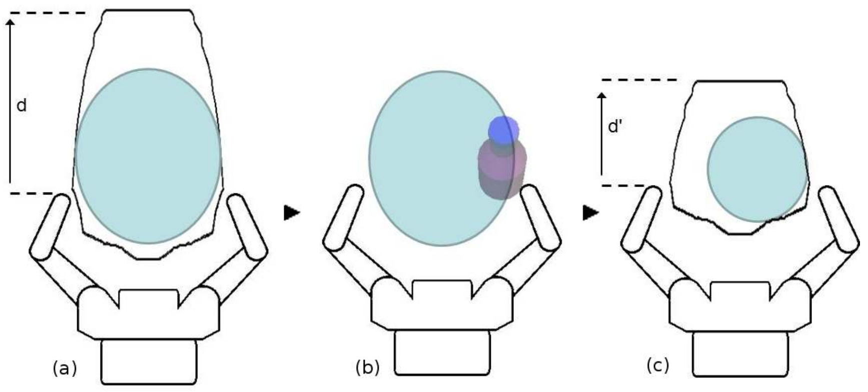


Figure 6.1: (a) A large uncertainty region for a bottle is shown in light blue. It is completely in the capture region of pushing distance d. (b) The hand moves forward and contacts the bottle. (c) Based on the feedback the uncertainty region of the bottle changes. Now the required pushing distance to capture all the uncertainty, d', is much smaller.

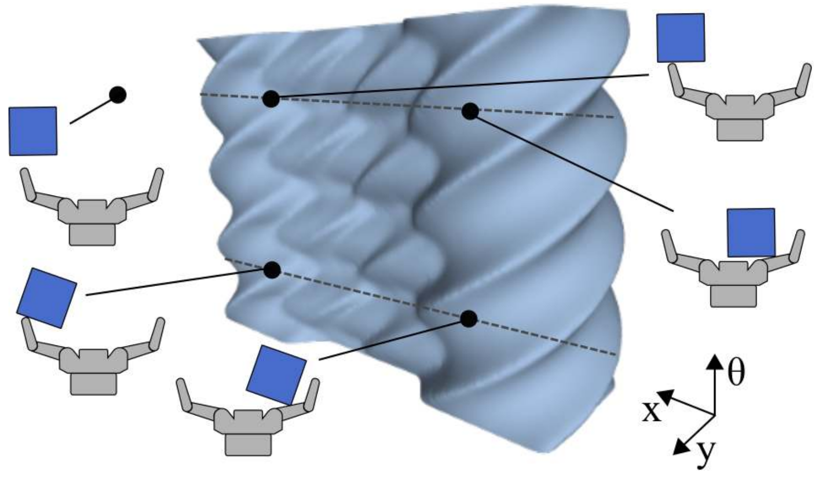


Figure 6.3: The contact states constitute a lower-dimensional manifold in the object's state space.

it, then contact sensors will not discriminate between it and another particle which is much farther away from the hand. Topologically, the observation space of contact sensors constitute a lower dimensional manifold in the state space of the pose of the object (Fig. 6.3). In practice, particles sampled from the state space during contact will have very low probability of falling into the observation space which will result in particle deprivation in the vicinity of the correct state (Fig. 6.5). This results in *particle starvation.* Artificially introducing noise into the observation model sidesteps this problem but comes at the expense of losing precious information.

We recently developed the Manifold Particle Filter (MPF) to address this problem. MPF estimates state on multiple manifolds of possibly different dimensions.

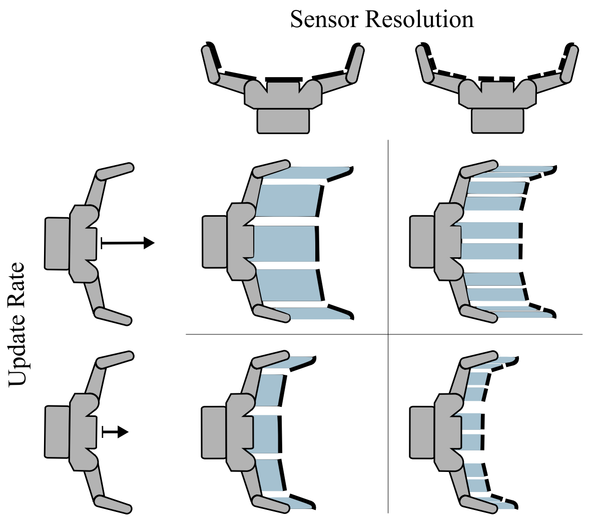


Figure 6.5: The swept volume of contact sensors shrinks as the update rate or resolution of the sensors increases.

6.1 Manifold Particle Filter for Pose Estimation during Contact Manipulation

Let x be the state of a dynamical system which evolves under actions u and provides observations z. The state estimation problem addresses the computation of the *belief* which is a probability distribution over the state space

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t}) \tag{6.1}$$

given the past prior actions $u_{1:t}$ and observations $z_{1:t}$.

In our problem, the state is the pose $x \in SE(2)$ of the manipulated object (Fig. 6.6a). Actions are motions of the hand, given by the velocity twist $u \in se(2)$. During contact, the object moves with a velocity $f_\Phi(x, u)$ where the function f encodes the physics of the object motion in response to pushing actions (Fig. 6.6b). The parameter Φ includes environmental properties such as the coefficient of friction between the object and the underlying surface, the coefficient of friction between the robot hand and the object, the mass distribution of the object, and the pressure distribution of the object.

Contact sensors provide observations z about where the object touches the hand during manipulation. In Fig. 6.6c we illustrate an example distribution of nine contact sensors (shown as bold line segments) on the hand.

Suppose we divide a state space S into m disjoint components $M = \{M_i\}_1^m$, where $M_1, ..., M_{m-1}$ are manifolds and $M_m = S - \cup_{i=1}^{m-1} M_i$ is the remaining free space. Then we can write the belief in this space as:

$$bel(x_t) = \sum_i^m bel(x_t | M_i) \Pr(x_t \in M_i) \tag{6.2}$$

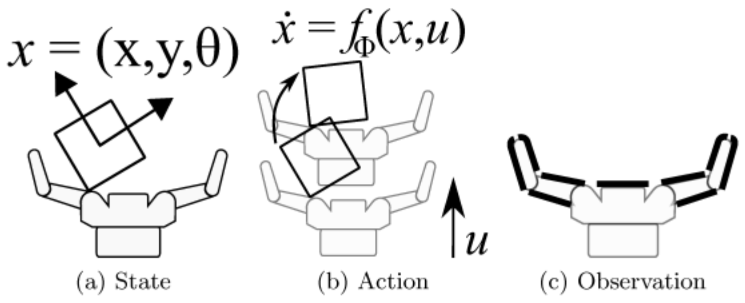


Figure 6.6: Examples illustrating the (a) state, (b) action, and (c) observation for the state estimation for contact manipulation problem.

Our algorithm approximates this belief using particles. For each particle we first choose which manifold to sample from according to $\Pr(x_t \in M_i)$. Then, we sample the particle from that manifold according to $bel(x_t|M_i)$.

Ideally, we would compute $\Pr(x_t \in M_i)$ as

$$\Pr(x_t \in M_i) = \int_{M_i} bel(x_t)\, dx_t. \tag{6.3}$$

Unfortunately, computing this integral requires knowing $bel(x_t)$, which is precisely the distribution that we are trying to estimate.

Instead, we approximate $\Pr(x_t \in M_i)$. Using the previous belief state to compute $\int_{M_i} bel(x_{t-1})\, dx_{t-1}$ might seem like a good approximation, but in fact it does *not* work. To see why, consider an update step at which the filter receives an observation which suggests the state to be on a manifold for the first time. If we use the previous belief, it will indicate a low probability for the state to be on the manifold and we will not pick that manifold for sampling. Even if we keep receiving observations that suggest the manifold, we will never be able to place particles on it since we will always be using the belief from the previous step.

Hence we approximate Eq. (6.3) using only the most recent observation

$$\Pr(x_t \in M_i) \approx \int_{M_i} \frac{p(z_t|x_t, u_t)}{\pi(z_t|u_t)}\, dx_t, \tag{6.4}$$

where $\pi(z_t|u_t) = \int_{x_t} p(z_t|x_t, u_t)\, dx_t$ is the prior probability of receiving observation z_t. This is not, in general, equivalent to Eq. (6.3). However, Eq. (6.4) is a good approximation when $p(z_t|x_t, u_t)$ accurately discriminates between the manifolds. In the limit, when observations perfectly discriminate between manifolds, $p(z_t|x_t, u_t)$ becomes binary and this sampling technique introduces no bias. This approximation performs well for our purposes since contact sensors accurately discriminate between contact and no-contact.

Finally, we sample a particle $x_t^{[i]}$ according to the belief distribution on the chosen manifold $bel(x_t|M_i)$. Our key insight is that we can apply a different sampling technique

Algorithm 4: Manifold Particle Filter

```
1  $X_{t-1}$, previous particles
2  $X_t$, particles sampled from $bel(x_t)$
3  $X_t \leftarrow \emptyset$
4  for $i = 1, \ldots, |X_{t-1}|$ do
5      Sample $M_i \sim \Pr(x_t \in M_i)$
6      if $M_i \neq M_m$ then
7          Sample $x_t^{[i]} \sim \frac{p(z_t|x_t,u_t)}{\pi(z_t|u_t)}$
8          $w_t^{[i]} = \pi(z_t|u_t) \cdot \text{EstimateDensity}(X_{t-1}, x_t^{[i]})$
9      else
10         $x_t^{[i]}, w_t^{[i]} \leftarrow \text{ConventionalSampling}(X_{t-1}, u_t, z_t)$
11     $X_t \leftarrow \{x_t^{[i]}\} \cup X_t$
12 $X_t \leftarrow \text{Resample}(X_t)$
```

for each manifold that is specifically designed to take advantage of the structure of M_i. In the case of the free space M_m, we can sample $x_t^{[i]}$ from S using the conventional technique and reject any $x_t^{[i]} \in \cup_{i=1}^{m-1} M_i$.

For the contact manifold, we importance sample using the *dual proposal distribution* (Thrun et al., 2000) which samples from the observation model and computes importance weights using the motion model:

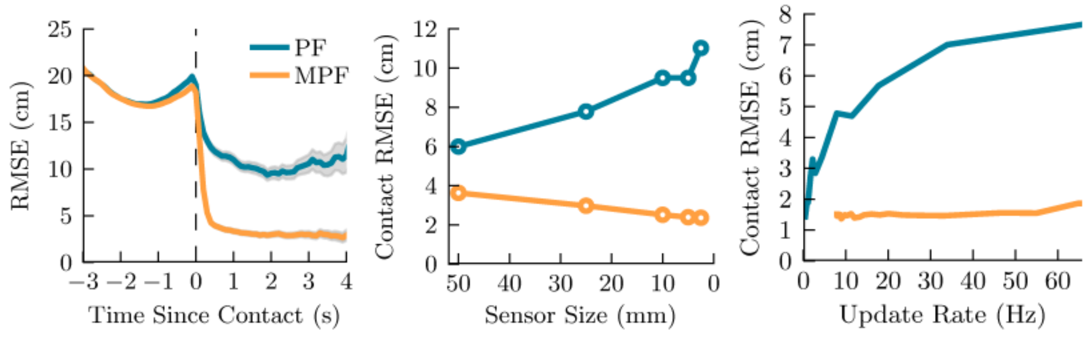
- *Sampling from the observation model*: Since the contact manifold is a two-dimensional manifold embedded in $SE(2)$, we can approximate it using a relatively small set of pre-computed hand-relative object poses. For each pre-computed sample, we compute its probability using the current action and observation. Then, we sample $x_t^{[i]}$ from this weighted set.

- *Computing importance weights using the motion model*: We estimate the importance weights of particles by applying a density estimation algorithm. First, we forward-simulate the set of particles X_{t-1} to time t with the motion model. Next, we use *kernel density estimation* to approximate $\int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1}$ from the forward-simulated samples.

We present the MPF algorithm in Alg. 4.

We evaluated the regular particle filter (PF) and the Manifold Particle Filter (MPF) on HERB and in simulation. In simulation we compare the performance of the algorithms with the following metrics:

- *Root-mean-square error* (RMSE) between the particles and the true state.



(a) RMSE (b) Sensor Resolution (c) Update Rate

Figure 6.7: (a) RMSE of the PF and MPF plotted over time for 500 particles. Contact occurs at $t = 0$ and is denoted by the dashed line. (b) Effect of sensor resolution on RMSE. The MPF monotonically decreases in RMSE as resolution increases. Conversely, the accuracy of the PF degrades with high resolution sensors. (c) Real-time performance of the filters. The MPF achieves lower RMSE than the PF given fixed computational resources. Additionally, the MPF achieves acceptable error with real-time update rates of 60+ Hz. The gray shaded region in (a) denotes the 95% confidence interval. This is omitted from (b), and (c) because the confidence intervals are of a negligible size.

- *Update rate* (UR), the maximum rate at which particle filter updates step can be executed.

These metrics are have traditionally been used in the localization literature (Thrun et al., 2000, Montemerlo et al., 2003).

Our experiments show that:

- MPF achieves significantly lower error than the PF (Fig. 6.7a);
- MPF performance improves with sensor resolution and it outperforms the PF at all sensor resolutions (Fig. 6.7b);
- MPF performance improves with higher update rates and is fast enough to be used real-time (Fig. 6.7c).

For real experiments on HERB, we used the BarrettHand's finger strain gauges to detect binary contact on the distal links and a wrist-mounted force/torque sensor to detect binary contact on the rest of the hand at approximately 10 Hz. We present two examples in Fig. 6.8. In Fig. 6.8-Left, the object initially contacts the palm and settles into persistent contact with the hand. The palm has a large swept volume, so both the PF (Fig. 6.8-Top-Left) and MPF (Fig. 6.8-Bottom-Left) successfully tracks the state. In Fig. 6.8-Right, the object contacts the hand's left distal link and slips off the finger during the duration of the push. In this case, the distal link has a small swept volume and the PF incorrectly converged to

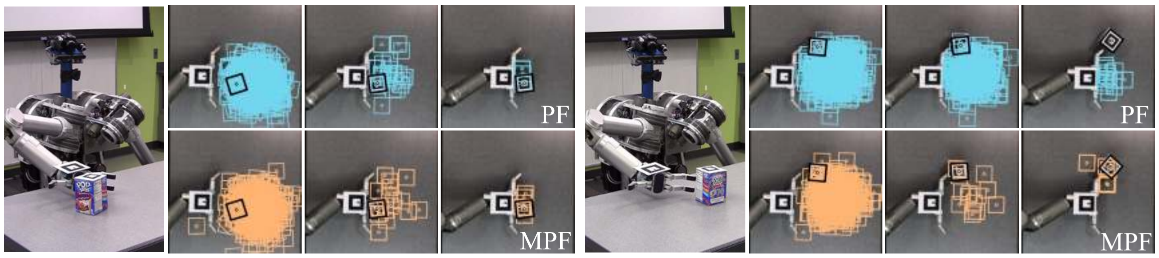


Figure 6.8: HERB pushing a box across the table. The top and bottom rows show the belief as estimated by the PF (top, light blue) and the MPF (bottom, light orange) during different stages of manipulation. The PF and MPF perform similarly when the object contacts the large palm (left), but the MPF outperforms the PF when contact occurs with the small distal links (right).

the palm. As expected, the MPF successfully tracks the state through the duration of the contact.

Chapter 7

Interleaving Rearrangement Planning with Feedback

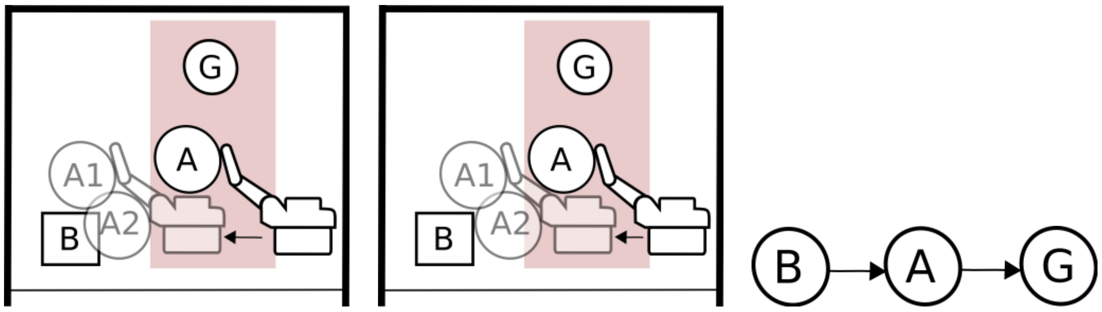
The rearrangement framework we presented in Chapter 4 does not use sensor feedback once the execution starts. The downside is that, to guarantee success the planner needs to be very conservative.

Fig. 7.1 explains what we mean by being conservative. The robot's goal is to grasp 'G' and the planner identifies that it needs to use the light red region to be able to do that. This region becomes a negative goal region (NGR) for object 'A'. Now the robot's goal is to move 'A' completely out of the NGR. Let's assume the planner chooses to push 'A' left. We have uncertainty about where it will end up after a push. For the specific push in the figure, this uncertainty is shown as hypotheses 'A1' and 'A2'. There are two things to notice:

- If the robot plans a shorter push it cannot guarantee that the object will leave the NGR. This is illustrated in Fig. 7.1(b), where sample 'A1' is completely out of the NGR, but 'A2' is not.

- If the robot plans the longer push it needs to move object 'B' out of the way, before it can push 'A'. This is illustrated in Fig. 7.1(a) where 'A2' penetrates the space of 'B'. Again there is a good chance that 'A' will not collide 'B' after this push ('A1' does not penetrate 'B'), but to guarantee that this will not happen, 'B' needs to be moved.

Since our planner is conservative it first pushes 'B' out of the way and then moves 'A' with the long push. If the robot pushes 'A' without moving 'B', there is a possibility that 'A' can move out of NGR and does not collide with 'B'; but our planner does not take that risk.



(a) Long Push (b) Short Push (c) Object dependency graph

Figure 7.1: Robot wants to grasp ‘G’ and need to use the space shown in light red. This area is the NGR for ‘A’ and it needs to be pushed out of the NGR. There is uncertainty about the motion of ‘A’, shown as two hypotheses about the final pose: ‘A1’ and ’A2’. (a) After a long push ‘A’ is guaranteed to leave the NGR, since both ‘A1’ and ‘A2’ are out of it. But ‘A2’ suggests that there is possibility ‘A’ collides with ‘B’. (b) After a shorter push ‘A2’ does not completely leave the NGR. ‘A’ is not guaranteed to leave the NGR.

Here is the question we ask: After executing an action on an object, if the robot has the chance to look at the scene, get sensor feedback and see the effect of its action, how would this be useful for our framework? Below we look at three attempts of formulating how to use feedback.

Attempt 1: Use the existing planner to produce a plan. During execution, get sensor feedback after every manipulation action and see if any steps of the plan can be skipped. We quickly see that this strategy cannot skip any steps. To see why, it is useful to represent a plan as a graph of object dependencies, which gives us a directed acyclic graph. In this graph each node corresponds to an object. The sink node is the goal object. An edge $X \to Y$ means X must be moved to free the space that Y, with its uncertainty, will use. Fig. 7.1(c) shows the graph for the simple example from Fig. 7.1(a). During execution, the robot moves an object that is a leaf node (i.e. one that has no incoming edges) and removes it from the graph; then it keeps doing the same until the goal object. For an edge $X \to Y$, if an oracle tells us the exact motion of Y, there is a possibility that the edge $X \to Y$ becomes unnecessary. For example, in Fig. 7.1(a) if the oracle tells us that ‘A’ will end up in ‘A1’ we can skip moving ‘B’. But we never have such a case during execution of a plan because all we move are leaf nodes at any given state of the graph.

Attempt 2: Use the existing planner to produce a plan. Execute the first action, get feedback, replan. We see that this attempt also fails in producing a plan shorter than the one generated by the original planner with no feedback. This follows from the fact that we have a planner that plans backwards: the object to be moved first is planned for last. The uncertainty of this object does not constrain any other planning step but the last.

Hence, resolving this uncertainty does not affect the rest of the planning process. For example in Fig. 7.1 the planner first plans for object ‘G’. The volume of space needed by this motion generates constraints for the planning step for ‘A’. The spaces needed for ‘G’ and ‘A’ generate constraints when planning for ‘B’. When the robot moves ‘B’ away, gets feedback to identify where it moved, and replans, this does not change any of the constraints for the planning of the motion of ‘G’ and ‘A’.

The two attempts above show that it is not straightforward to integrate feedback into our planner. This does not mean it is impossible though. Consider this plan for Fig. 7.1:

1. Execute the shorter push on ‘A’ (Fig. 7.1(b)), and get feedback to see where it ended up.
2. If it is ‘A1’, then go and grasp ‘G’ and stop. If it is not ‘A1’, then move ‘B’ out of the way.
3. Push ‘A’ further.
4. Grasp ‘G’ and stop.

To produce such a plan, we need to drop the approach of being conservative.

Attempt 3: Do not always be conservative. For this approach we change our planner slightly. Our planner will now accept an action if its probability to move an object out of the NGR is larger than or equal to P, where P is a parameter of the planner. Then, during execution the object will not move out of the NGR with probability $1.0 - P$, and we need to use sensor feedback to check if this happens. If it does then we need to call the planner again in the new scene. The planner can reuse cached information when the planner is called again during execution. For the example in Fig. 7.1 this planner will first generate the plan “Execute short push on ‘A’, then grasp ‘G”’. After the robot executes the short push on ‘A’, it checks to see if it is still inside the NGR. If so, the planner is called again. The planner reuses the plan to grasp ‘G’. It generates new actions to move object ‘B’ out of the way, and push ‘A’ more to move it out of the NGR.

In future work, we hope to explore this approach to develop a rearrangement planning framework which can effectively use feedback between actions.

Chapter 8

Conclusion

In this thesis we presented a series of planners and algorithms for manipulation in cluttered human environments. We focused on using physics-based predictions, particularly for pushing operations, as an effective way to address the manipulation challenges posed by these environments.

Our work in this thesis leads to the following lessons for us:

- Our work on push-grasping (Chapter 2), on grasping through clutter (Chapter 3), and on rearrangement planning using non-prehensile actions (Chapter 4) show that *physical predictions has the potential to dramatically improve a robot's manipulation skills in cluttered and uncertain human environments.*

  In this thesis we have taken a relatively small step by exploring the potential of pushing actions, but we hope we have convincingly made the point that robots needs to move beyond pick-and-place for skillfull manipulation in human environments.

- We showed that *even though accurate physical predictions are computationally expensive, precomputing primitive-based structures makes fast physics-based planning possible.* The capture region of a push-grasp is a good example of such structures.

  We have however also pointed out the limitations of the precomputation approach, especially when the desired interactions involve object-object contact.

- We showed that *choosing the right physics model for a physics-based primitive and analyzing it is important to minimize and identify the physical properties of objects the robot needs to know.*

  In our work we used the quasi-static model, identified the object properties we need to estimate, and developed planners which can deal with uncertainties in the values of these object properties.

- We showed that *physics-based actions can reduce uncertainty (e.g. push-grasping), as well as increase uncertainty (e.g. grasping through clutter).*

  We proposed conservative planning as a solution to uncertainty for open-loop actions. We showed that this approach leads to robust and efficient planning. However, we also showed that there are limitations to the conservative planning approach: if the uncertainties become too large, the planners can get too constrained, failing to generate solutions.

- We showed that *manipulation in clutter requires reasoning about manipulation and perception simultaneously.*

  In Chapter 5 we presented an algorithm to manipulate clutter for searching an object. We showed that we can take advantage of the structure of everyday environments — e.g. the visibility and accessibility relations between objects on different shelves of a fridge — to produce fast but effective solutions to clutter manipulation problems.

- We proposed *using sensor feedback during physics-based actions as a remedy to many of the limitations of our approach.*

  One important feedback source is the contact between the robot and the object during manipulation, and in Chapter 6 we presented preliminary results about a system which can use contact feedback to estimate the pose of a pushed object.

We believe extending the skill set of robots with physics-based primitives will get them one step closer to being a part of everyday life in human environments.

Appendices

Appendix 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 Section 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

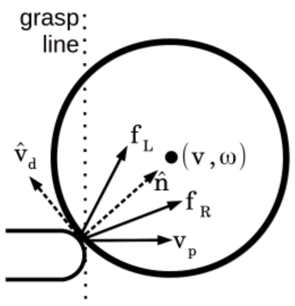


Figure A.1: Some of the force and velocity vectors we will be using in our proof. In the figure the finger is pushing the cylinder towards right.

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. A.1 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.
- $\mathbf{v}$ 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.

- $\mathbf{\hat{v}_d}$ 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; $\mathbf{\hat{v}_d} = \frac{(\boldsymbol{\omega}\times\mathbf{r})}{|\boldsymbol{\omega}\times\mathbf{r}|}$.
- $\mathbf{v_p}$ 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 $\mathbf{\hat{v}_d}$ this means $\mathbf{v_p}.\mathbf{\hat{v}_d} < 0$.
- $\mathbf{v_c}$ 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 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 A.0.1. *During sticking and sliding contact* $\mathbf{v}.\mathbf{\hat{n}} = \mathbf{v_p}.\mathbf{\hat{n}}$.

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

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

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

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

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

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

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

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

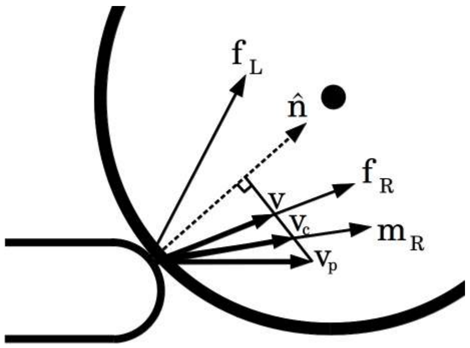


Figure A.2: The relation between $\mathbf{v}$, $\mathbf{v_p}$, and $\mathbf{v_c}$ during sliding contact.

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. A.2. The rest of the proof proceeds the same as the sticking contact case. □

Lemma A.0.2. *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 A.0.1,

$$|\mathbf{v}| = \sqrt{(\mathbf{v}.\hat{\mathbf{v}}_\mathbf{d})^2 + (\mathbf{v_p}.\hat{\mathbf{n}})^2} \tag{A.1}$$

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 $\mathbf{\hat{v}_d}$ gives:

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

Inserting this into Eq. A.1,

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

Except $|\boldsymbol{\omega}|$, the terms on the right hand side are independent of ρ. Since $\mathbf{v_p}.\mathbf{\hat{v}_d} < 0$, 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 A.0.3. *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 $\mathbf{\hat{v}_d}$.*

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 $\mathbf{\hat{v}_d}$.

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 $\mathbf{f_R}$

$$\mathbf{v} = |\mathbf{v}|\mathbf{\hat{f}_R}$$

where $\mathbf{\hat{f}_R}$ is the unit direction along $\mathbf{f_R}$. Inserting this into the result from Lemma A.0.1 we have

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

Then

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

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

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

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

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 A.0.2, 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 $\hat{\mathbf{v}}_\mathbf{d}$. □

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

Lemma A.0.4. *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. A.3, resulting in a change along $\hat{\mathbf{v}}_\mathbf{d}$. Formally, in the proof of Lemma A.0.3 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 $\hat{\mathbf{v}}_\mathbf{d}$.

□

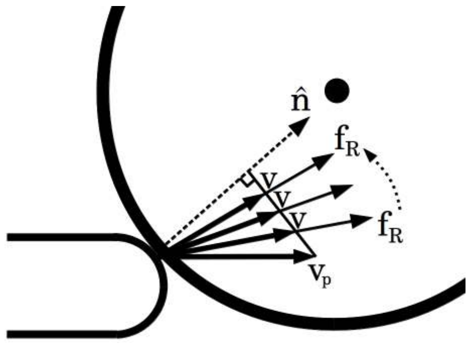


Figure A.3: $\mathbf{v}$ changes along with the edge of the friction cone as μ_c is decreased.

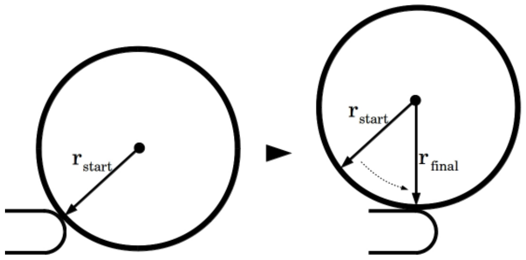


Figure A.4: Independent of μ_c and ρ, the finger and the object goes through the same set of relative configurations during the push-grasp.

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

Theorem A.0.1. *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).*

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. A.4). 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 A.0.1 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 A.0.3 and A.0.4 states that the change in $\mathbf{v}$ will be non-negative along $\hat{\mathbf{v}}_\mathbf{d}$. $\square$

Appendix B

Optimality of the Connected Components Algorithm for Object Search

We present a partial proof of optimality for Alg. 3.

We state a property of the collective utility as a lemma.

Lemma B.0.5. *Given an arrangement $\mathcal{A}_o$,*

$$U(\mathcal{A}_o(1,|o|)) \geq U(\mathcal{A}_o(1,k)) \rightarrow U(\mathcal{A}_o(k+1,|o|)) \qquad \geq U(\mathcal{A}_o(1,|o|))$$

In other words, if the utility of the complete arrangement is larger than the utility of the first k objects, then the utility of the last $|o|-k$ objects must be larger than the utility of the complete arrangement.

Proof. We are given that

$$\frac{V_{\mathcal{A}_o(1,k)} + V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(1,k)} + T_{\mathcal{A}_o(k+1,|o|)}} \geq \frac{V_{\mathcal{A}_o(1,k)}}{T_{\mathcal{A}_o(1,k)}}$$

Rearranging yields

$$V_{\mathcal{A}_o(k+1,|o|)} \cdot T_{\mathcal{A}_o(1,k)} \geq V_{\mathcal{A}_o(1,k)} \cdot T_{\mathcal{A}_o(k+1,|o|)}$$

Adding $V_{\mathcal{A}_o(k+1,|o|)} \cdot T_{\mathcal{A}_o(k+1,|o|)}$ to both sides and rearranging, we get

$$\frac{V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(k+1,|o|)}} \geq \frac{V_{\mathcal{A}_o(1,k)} + V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(1,k)} + T_{\mathcal{A}_o(k+1,|o|)}}$$

□

Theorem B.0.2. *Given an optimal arrangement of a scene $\mathcal{A}^*$, for any two adjacent sequence of objects in the arrangement $\mathcal{A}^*(i,j)$ and $\mathcal{A}^*(j+1,k)$, where $i \leq j < k$, if there are neither accessibility constraints nor joint occlusions between the objects in the two sequences (i.e. if the sequences are from different connected components), then the utility of the former sequence is greater than or equal to the utility of the latter sequence: $U(\mathcal{A}^*(i,j)) \geq U(\mathcal{A}^*(j+1,k))$.*

Proof. The proof proceeds similar to the proof of Theorem 5.2.1. We create a new arrangement $\mathcal{A}$ that is identical to $\mathcal{A}^*$ except that the two adjacent sequences are swapped:
$\mathcal{A}(i, i+k-j) = \mathcal{A}^*(j+1,k)$ and
$\mathcal{A}(i+k-j+1,k) = \mathcal{A}^*(i,j)$. $\mathcal{A}$ must be a valid arrangement since we are given that no object in $\mathcal{A}^*(i,j)$ is blocking access to $\mathcal{A}^*(j+1,k)$. Then we can compute the difference $E(\mathcal{A}) - E(\mathcal{A}^*)$ to be:

$$\sum_{l=i}^{j} \left(\frac{V_{\mathcal{A}^*(l)}}{V_{\mathcal{O}_{seen}}} \cdot T_{\mathcal{A}^*(j+1,k)} \right) - \sum_{l=j+1}^{k} \left(\frac{V_{\mathcal{A}^*(l)}}{V_{\mathcal{O}_{seen}}} \cdot T_{\mathcal{A}^*(i,j)} \right)$$

Since $\mathcal{A}^*$ is optimal, $E(\mathcal{A}) - E(\mathcal{A}^*) \geq 0$. After canceling out the common terms and rearranging, we are left with

$$\frac{\sum_{l=i}^{j} V_{\mathcal{A}^*(l)}}{T_{\mathcal{A}^*(i,j)}} \geq \frac{\sum_{l=j+1}^{k} V_{\mathcal{A}^*(l)}}{T_{\mathcal{A}^*(j+1,k)}}$$

Simply, $U(\mathcal{A}^*(i,j)) \geq U(\mathcal{A}^*(j+1,k))$. □

We state a lemma and leave its proof to future work.

Lemma B.0.6. *The relative ordering of objects in the optimal arrangement of a connected component will be preserved in the optimal ordering for the complete scene. Formally, if $\mathcal{A}_c^*$ is the optimal arrangement for a connected component c, and $\mathcal{A}_o^*$ is the optimal arrangement of o, such that $c \subseteq o$, then*

$$i < j \rightarrow \mathcal{A}_o^{*-1}(\mathcal{A}_c^*(i)) < \mathcal{A}_o^{*-1}(\mathcal{A}_c^*(j))$$

where $1 \leq i,j \leq |c|$, and $\mathcal{A}_o^{*-1}$ returns the index of an object in the arrangement $\mathcal{A}_o^*$.

Finally we can prove that the connected components algorithm is optimal.

Theorem B.0.3. *Let's say we are given m connected components of a set of objects, o, and we are also given an optimal arrangement for each connected component $\mathcal{A}_{c^i}$ for $i = 1, \ldots, m$. Let's say we computed the utility of all sequences of objects in the form $\mathcal{A}_{c^i}(1,j)$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, |c^i|$, and found $\mathcal{A}_{c^*}(1,j^*)$ to have the maximum utility. Then an optimal arrangement for o starts with $\mathcal{A}_{c^*}(1,j^*)$.*

Proof. Assume that the optimal arrangement $\mathcal{A}_o^*$ does not start with $\mathcal{A}_{c^*}(1, j^*)$. We will prove that this is not possible.

Given an arrangement of o, we can view it as a series of partitions, where each partition consists of a contiguous sequence of objects from the same connected component. Due to Lemma B.0.6, each such partition in $\mathcal{A}_o^*$ can be represented as subsequences of the connected component arrangements $\mathcal{A}_{c^i}$. In particular, we are interested in two partitions of the optimal arrangement of o:

$$\mathcal{A}_o^* = \left[\mathcal{A}_{c'}(1, j') \ldots \mathcal{A}_{c^*}(k, l) \ldots\right]$$

where c' is one of the connected components, and $1 \leq j' \leq |c'|$. $\mathcal{A}_{c^*}(k, l)$ is the partition that includes the object $\mathcal{A}_{c^*}(j^*)$, hence $k \leq j^* \leq l$. We know that $\mathcal{A}_{c^*}(1, j^*)$ has the maximum utility of all the sequences in the form $\mathcal{A}_{c^i}(1, j)$ where c^i is any connected component and $j = 1, \ldots, |c^i|$. Then,

$$U(\mathcal{A}_{c^*}(1, j^*)) > U(\mathcal{A}_{c^*}(1, k-1)) \tag{B.1}$$

and also

$$U(\mathcal{A}_{c^*}(1, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{B.2}$$

Using Lemma B.0.5 and Eq. (B.1), we get

$$U(\mathcal{A}_{c^*}(k, j^*)) > U(\mathcal{A}_{c^*}(1, j^*))$$

Then from Eq. (B.2),

$$U(\mathcal{A}_{c^*}(k, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{B.3}$$

Considering the utilities of all the partitions in $\mathcal{A}_o^*$ up to $\mathcal{A}_{c^*}(k, l)$, we know that they should be ordered in decreasing order of utility and be larger than $\mathcal{A}_{c^*}(k, j^*)$ (Theorem B.0.2):

$$U(\mathcal{A}_{c'}(1, j')) > \ldots > U(\mathcal{A}_{c^*}(k, j^*))$$

which contradicts Eq. (B.3). $\square$

References

Pankaj Agarwal, Jean-Claude Latombe, Rajeev Motwani, and Prabhakar Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *IEEE International Conference on Robotics and Automation*, 1997.

Srinivas Akella and Matthew T. Mason. Posing polygonal objects in the plane by pushing. *International Journal of Robotics Research*, 17(1):70–88, 1998.

Abhishek Anand, Hema Swetha Koppula, Thorsten Joachims, and Ashutosh Saxena. Contextually guided semantic labeling and search for three-dimensional point clouds. *International Journal of Robotics Research*, 32(1):19–34, 2013.

David C. Baird. *Experimentation: An introduction to measurement theory and experiment design*. Prentice-Hall (Englewood Cliffs, NJ), 1995.

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

Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, L Mosenlechner, Dejan Pangercic, T Ruhr, and Moritz Tenorth. Robotic roommates making pancakes. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 529–536. IEEE, 2011.

Ohad Ben-Shahar and Ehud 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, 1998a.

Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation*, 14:549–565, 1998b.

Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, 2007.

Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, and James Kuffner. Manipulation Planning on Constraint Manifolds. In *IEEE International Conference on Robotics and Automation*, 2009a.

Dmitry Berenson, Siddhartha S Srinivasa, and James J Kuffner. Addressing Pose Uncertainty in Manipulation Planning Using Task Space Regions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009b.

Robert-Paul Berretty, Kenneth Y. Goldberg, Mark H. Overmars, and A. Frank van der Stappen. Orienting parts by inside-out pulling. In *IEEE International Conference on Robotics and Automation*, 2001.

Randy C. Brost. Automatic grasp planning in the presence of uncertainty. *International Journal of Robotics Research*, 7(1), 1988.

Randy C Brost and Alan D Christiansen. Probabilistic analysis of manipulation tasks: A conceptual framework. *The International journal of robotics research*, 15(1):1–23, 1996.

Lillian Y. Chang, Siddhartha Srinivasa, and Nancy Pollard. Planning pre-grasp manipulation for transport tasks. In *IEEE International Conference on Robotics and Automation*, 2010.

Gary Chartrand. *Introductory Graph Theory*, chapter 6.3, pages 135–139. New York: Dover, 1985.

Pang C. Chen and Yong K. Hwang. Practical path planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, 1991.

Matei Ciocarlie, Kaijen Hsiao, E Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan A Sucan. Towards reliable grasping and manipulation in household environments. In *Intl. Symposium on Experimental Robotics*, pages 1–12, 2010.

A. Cosgun, T. Hermans, V. Emeli, and M. Stilman. Push Planning for Object Placement on Cluttered Table Surfaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008. ISBN 9783540779735.

Rosen Diankov and James Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, 2008.

Rosen Diankov, Siddhartha Srinivasa, David Ferguson, and James Kuffner. Manipulation Planning with Caging Grasps. In *Humanoids*, 2008.

Mehmet Dogar and Siddhartha Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. *Proceedings of*, 2010.

Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and Systems VII*, 2011.

Mehmet Dogar and Siddhartha Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, pages 1–20, 2012.

Mehmet Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and Systems VIII*, 2012.

Mehmet Dogar, Michael Koval, Abhijeet Tallavajhula, and Siddhartha Srinivasa. Object search by manipulation. In *IEEE International Conference on Robotics and Automation*, 2013.

Michael A Erdmann and Matthew T Mason. An exploration of sensorless manipulation. *Robotics and Automation, IEEE Journal of*, 4(4):369–379, 1988.

Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4914–4919. IEEE, 2012.

Judith Espinoza, Alejandro Sarmiento, Rafael Murrieta-Cid, and Seth Hutchinson. Motion Planning Strategy for Finding an Object with a Mobile Manipulator in Three-Dimensional Environments. *Advanced Robotics*, 2011.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

Barbara Frank, Cyrill Stachniss, Nichola Abdo, and Wolfram Burgard. Using gaussian process regression for efficient motion planning in environments with deformable objects. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter Allen. The columbia grasp database. In *IEEE International Conference on Robotics and Automation*, pages 1710–1716, 2009.

Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction. Part 1. Limit surface and moment function. *Wear*, (143):307–330, 1991.

Megha Gupta and Gaurav Sukhatme. Interactive perception in clutter. In *The RSS 2012 Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments*, 2012.

Kris Hauser. The minimum constraint removal problem with three robotics applications. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6), 1973.

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

Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Grasping pomdps. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4685–4692. IEEE, 2007.

Advait Jain, Marc Daniel Killpack, Aaron Edsinger, and Charlie Kemp. Reaching in clutter with whole-arm tactile sensing. *The International Journal of Robotics Research*, 2013.

Shervin Javdani, Matthew Klingensmith, J. Andrew (Drew) Bagnell, Nancy Pollard, and Siddhartha Srinivasa. Efficient touch based localization through submodularity. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

Yan-bin Jia and Michael Erdmann. Pose and motion from contact. *International Journal of Robotics Research*, 1999.

Leslie Pack Kaelbling and Tomas Lozano-Perez. Hierarchical planning in the now. In *IEEE International Conference on Robotics and Automation*, 2011.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

L.P. Kaelbling and T. Lozano-Perez. Unifying perception, estimation and action for mobile manipulation via belief space planning. In *IEEE International Conference on Robotics and Automation*, 2012.

Daniel Kappler, LillianY. Chang, Markus Przybylski, Nancy Pollard, Tamim Asfour, and Rudiger Dillmann. Representation of Pre-Grasp Strategies for Object Manipulation. In *IEEE-RAS Humanoids*, 2010.

Daniel Kappler, Lillian Y. Chang, Nancy S. Pollard, Tamim Asfour, and Rdiger Dillmann. Templates for pre-grasp sliding interactions. *Robotics and Autonomous Systems*, 60:411–423, 2012.

Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation*, 1994.

Michael Koval, Mehmet Dogar, Nancy Pollard, and Siddhartha Srinivasa. Pose estimation for contact manipulation with manifold particle filters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013. Submitted.

Manfred Lau, Jun Mitani, and Takeo Igarashi. Automatic learning of pushing strategy for delivery of irregular-shaped objects. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3733–3738. IEEE, 2011.

Steven M. Lavalle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 2000.

Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, 100(2):108–120, 1983.

Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

Tomas Lozano-Perez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

Kevin M. Lynch. Toppling Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 152–159, 1999a.

Kevin M. Lynch. Locally controllable manipulation by stable pushing. *IEEE Transactions on Robotics and Automation*, 15(2):318 – 327, 1999b.

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

Kevin M. Lynch, Hitoshi Maekawa, and Kazuo Tanie. Manipulation and active sensing by pushing using tactile feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

Jeremy Ma, Timothy H Chung, and Joel Burdick. A probabilistic framework for object search with 6-dof pose estimation. *International Journal of Robotics Research*, 30(10): 1209–1228, 2011.

Manuel Martinez, Alvaro Collet, and Siddhartha Srinivasa. MOPED: A Scalable and Low Latency Object Recognition and Pose Estimation System. In *IEEE International Conference on Robotics and Automation*, 2010.

Matthew T Mason. Compliance and force control for computer controlled manipulators. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(6):418–432, 1981.

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

Andrew T. Miller and Peter K. Allen. Graspit! A Versatile Simulator for Robotic Grasping. *IEEE RAM*, 11(4):110 – 122, 2004.

Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *International Joint Conference on Artificial Intelligence*, 2003.

Van-Duc Nguyen. Constructing stable grasps. *International Journal of Robotics Research*, 8(1), 1989.

Damir Omrcen, Christian Boge, Tamim Asfour, Ales Ude, and Rudiger Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *IEEE-RAS Humanoids*, pages 277 –283, 2009.

Mark H. Overmars, Dennis Nieuwenhuisen, Dennis 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*, 2006.

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

Anna Petrovskaya and Oussama Khatib. Global localization of objects via touch. *Robotics, IEEE Transactions on*, 27(3):569–585, 2011.

Robert Platt, Russ Tedrake, Leslie Kaelbling, and Tomás Lozano-Pérez. Belief space planning assuming maximum likelihood observations. *Robotics: Science and Systems VII*, 2010.

Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. Efficient planning in non-gaussian belief spaces and its application to robot grasping. In *International Symposium on Robotics Research*, 2011.

Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. Non-gaussian belief space planning: Correctness and complexity. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4711–4717. IEEE, 2012.

Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer, 2013.

Gill A Pratt and Matthew M Williamson. Series elastic actuators. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 399–406. IEEE, 1995.

Samuel Rodriguez, Jyh-Ming Lien, and Nancy M Amato. Planning motion in completely deformable environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2466–2471. IEEE, 2006.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, 2011.

Ksenia Shubina and John Tsotsos. Visual search for an object in a 3d environment using a mobile robot. *Computer Vision and Image Understanding*, pages 535–547, 2010.

Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research*, 23(7-8), 2004.

Kristoffer Sjo, Dorian Lopez, Ana Paul, Patric Jensfelt, and Danica Kragic. Object search and localization for an indoor mobile robot. *Journal of Computing and Information Technology*, pages 67–80, 2009.

Siddhartha Srinivasa, Dmitry Berenson, Maya Cakmak, Alvaro Collet Romea, Mehmet Dogar, Anca Dragan, Ross Alan Knepper, Tim D Niemueller, Kyle Strabala, J Michael Vandeweghe, and Julius Ziegler. Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, 100(8):1–19, 2012.

Siddhartha S. Srinivasa, Dave Ferguson, Casey J. Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. HERB: a home exploring robotic butler. *Autonomous Robots*, 2009.

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

Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, 2007.

Freek Stulp, Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning to grasp under uncertainty. In *IEEE International Conference on Robotics and Automation*, 2011.

Raúl Suárez, Jordi Cornellà, and Máximo Roa Garzón. *Grasp quality measures.* Institut d'Organització i Control de Sistemes Industrials, 2006.

Sebastian Thrun, Dieter Fox, and Wolfram Burgard. Monte Carlo localization with mixture proposal distribution. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics.* MIT Press, 2005.

Jur P. van den Berg, Mike Stilman, James Kuffner, Ming C. Lin, and Dinesh Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *In International Workshop on the Algorithmic Foundations of Robotics*, pages 599–614, 2008.

Daniel E Whitney. Force feedback control of manipulator fine motions. *ASME J. of Dynamic Systems, Measurement and Control*, 98:91–97, 1977.

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

Willow Garage. rviz wiki page. http://www.ros.org/wiki/rviz, 2012.

Terry Winograd. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Technical Report MAC-TR-84, MIT, 1971.

Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *IEEE Conference on Robotics and Automation*, 2013.

Yiming Ye and John Tsotsos. Where to look next in 3d object search. In *IEEE International Symposium on Computer Vision*, 1995.

Yiming Ye and John Tsotsos. Sensor planning for 3d object search. *Computer Vision and Image Understanding*, 73(2):145–168, 1999.

Li Zhang and Jeffrey C. Trinkle. The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing. In *IEEE International Conference on Robotics and Automation*, 2012.

Claudio Zito, Rustam Stolkin, Marek Kopicki, and Jeremy L Wyatt. Two-level rrt planning for robotic push manipulation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 678–685. IEEE, 2012.