Robust Rearrangement Planning using Nonprehensile Interaction

Jennifer E. King

December 15, 2016

CMU-RI-TR-16-65
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Siddhartha S. Srinivasa, CMU RI
Matthew T. Mason, CMU RI
Maxim Likhachev, CMU RI
David Hsu, National University of Singapore
Terrence W. Fong, NASA Ames Research Center

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

Abstract

As we work to move robots out of factories and into human environments, we must empower robots to interact freely in unstructured, cluttered spaces. Humans do this easily, using diverse, whole-arm, nonprehensile actions such as pushing or pulling in everyday tasks. These interaction strategies make difficult tasks easier and impossible tasks possible.

In this thesis, we aim to enable robots with similar capabilities. In particular, we formulate methods for planning robust open-loop trajectories that solve the *rearrangement planning* problem using nonprehensile interactions. In these problems, a robot must plan in a cluttered environment, reasoning about moving multiple objects in order to achieve a goal.

The problem is difficult because we must plan in continuous, high-dimensional state and action spaces. Additionally, during planning we must respect the physical constraints induced by the nonprehensile interaction between the robot and the objects in the scene.

Our key insight is that by embedding physics models directly into our planners we can naturally produce solutions that use nonprehensile interactions such as pushing. This also allows us to easily generate plans that exhibit full arm manipulation and simultaneous object interaction without the need for programmer defined high-level primitives that specifically encode this interaction. We show that by generating these diverse actions, we are able to find solutions for motion planning problems in highly cluttered, unstructured environments.

In the first part of this thesis we formulate the rearrangement planning problem as a classical motion planning problem. We show that we can embed physics simulators into randomized planners. We propose methods for reducing the search space and speeding planning time in order to make the planners useful in real-world scenarios.

The second part of the thesis tackles the imperfect and imprecise worlds that reflect the true reality for robots working in human environments. We pose the rearrangement planning under uncertainty problem as an instance of *conformant probabilistic planning* and offer methods for solving the problem.

We demonstrate the effectiveness of our algorithms on two platforms: the home care robot HERB and the NASA rover K-Rex. We demonstrate expanded autonomous capability on HERB, allowing him to work better in high clutter, completing previously infeasible tasks and speeding feasible task execution. In addition, we show these planners increase autonomy for the NASA rover K-Rex by allowing the rover to actively interact with the environment.

Acknowledgments

First and foremost, I would like to thank my advisor, Sidd Srinivasa, for his countless hours of discussion and whiteboard sessions. I have learned so much from him about robotics, critical thinking and clear presentation. His lessons translate beyond academics and I will always feel grateful that he took a chance on me. I will also always have a special love for Palatino font.

I am very fortunate to have such a respected and distinguished committee: Matt Mason, Max Likhachev, David Hsu and Terry Fong. Their guidance and feedback throughout my thesis work was invaluable and I am grateful for the opportunity to interact with and learn from each one of them.

Throughout my work on this degree, I have had several external collaborators that each uniquely shaped my journey. I would like to offer special thanks to Terry Fong and the members of the Intelligent Robotics Group at NASA Ames Research Laboratories, especially Vytas Sunspiral and Michael Furlong, for all their help with the KRex rover. In addition, thank you to the visiting researchers in our lab – Joshua Haustein, Marco Cognetti, Carolina Loureiro and Stefania Pellegrinelli.

My experience throughout my Ph.D. was made infinitely better by the members of the Personal Robotics Laboratory - Mehmet, Anca, Koval, Dellin, Pras, Shushman, Gilwoo, Shervin, Liz, Laura, Aaron Walsman, Aaron Johnson, Matt, Stefanos, Clint, Henny, Rosario, Shen, Ariana, Oren, Mike0, Mike1, Rachel and Vinitha. The motivation and inspiration you each provided in immeasurable. But beyond that, you made even the most stressful times fun. Also, thank you to Jean Harpley, Keyla Cook and Suzanne Lyons Muth, without whom the Robotics Institute and the Personal Robotics Lab would not have functioned.

I am so very fortunate to be surrounded by an incredible set of friends and family. My parents, Eileen and Dennis, have been my biggest supporters throughout life. They taught me to work hard and persevere, both through their words and example. They gave me the confidence and encouragement to achieve anything. And they never let me quit. For that I am so thankful. Thank you to my sister, Annie, and brother-in-law, Roy, for the endless encouragement and for being the best siblings I could ever ask for. Finally, to Anne and Marin. Thank you for your constant and consistent support and love and for always being there to celebrate my successes and pick me up after failures. You made this possible.

The material in this dissertation was supported by funding from the NASA Space Technology Research Fellowship program (award NNX13AL61H), the National Science Foundation IIS (award 1409003), Toyota Motor Engineering and Manufacturing (TEMA) and the Office of Naval Research.

Contents

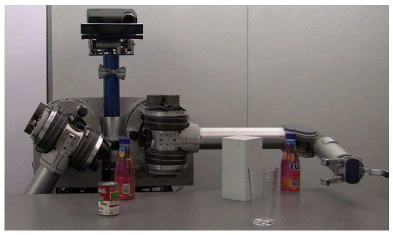
Introduction

Recent years have seen an increased emphasis on robots performing autonomous manipulation tasks. These tasks require robots to actively interact with and change their environment. Factory robots were the first to perform such tasks. These robots work in highly structured worlds, with fully known environments and very focused tasking. As we move robotic technology forward, we must advance their capabilities to allow them to work in unstructured and cluttered environments. These advancements will allow us to transition robots from their highly specific and ordered workspaces to complex and often disorganized human environments.

Most commonly used autonomous manipulators rely solely on the ability to pick-and-place objects, carefully moving one object at a time. Humans use a much more diverse suite of actions to accomplish everyday tasks. Consider grabbing an item from the back of a cluttered pantry. You may push aside items using your elbow, forearm and the back of your hand, while simultaneously caging the coveted item in your palm and dragging it to the front to grab it. We rely on whole arm and whole body nonprehensile interactions such as pushing or pulling in order to accomplish even basic tasks.

To transition robots into human environments we must empower them with these same strategies. In this thesis, we develop planners that generate open-loop strategies using nonprehensile interaction to solve the *rearrangement planning* problem. In these problems, a robot must manipulate several objects in clutter in order to achieve a goal. Our planners must be capable of generating solutions that allow simultaneous object interaction and whole arm manipulation. We believe these properties to be critical to efficiently working in clutter. Additionally, we require our planners quickly generate solutions robust to uncertainty in the planning environment.

Developing planners that exhibit these behaviors poses a number of challenges we must consider. We characterize these challenges and present a set of insights that inform our approach. From these insights, we develop a suite of planners that generate robust solutions to rearrangement planning problems that exhibit diverse whole arm


(a) HERB manipulating objects on a table

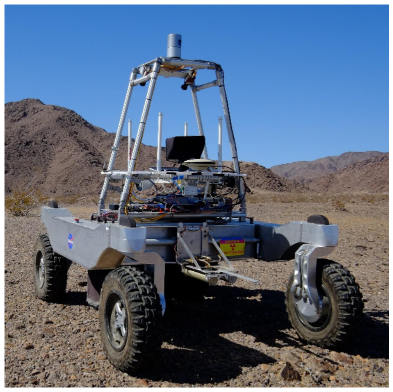

(b) K-Rex

Figure 1: (a) The HERB personal home robot. (b) The K-Rex lunar explorer. These robots will be used to demonstrate the effectiveness of our algorithms.

interactions.

Challenge 1: *Integrating nonprehensile interactions.* Nonprehensile interactions have been shown to be critical for pre-grasp manipulation [29, 65], large object manipulation [33] and simultaneous object interaction [48, 64]. In addition, they can make manipulation possible for robots not traditionally designed for interaction tasks. Consider the K-Rex robot from Fig.1b. Unless extra payload explicitely designed for performing manipulation tasks is added to the rover, this robot must rely on interactions such as pushing and toppling in order to manipulate the environment.

Purely geometric planners struggle to reason about nonprehensile interaction because objects do not move rigidly with the robot. Instead, the motion of objects evolves under non-holonomic constraints that represent the physics of the environment and the contact between robot and objects. Our planning algorithms must respect these motion constraints.

Challenge 2: *Fast planning in high-dimensional state and action spaces.* Our goal is to solve problems that require contact and interaction with objects in the environment. This interaction changes the planning environment. Because of this, we must track the objects the robot interacts with in our state during planning. This leads to a search space size linear in the number of objects the manipulator can move. In addition, we wish to plan motions for a high degree-of-freedom (DOF) robot such as the HERB robot [115] (Fig.1a). Our planning algorithms must be capable of quickly planning in high dimensional state and action spaces.

Challenge 3: *Planning robust trajectories.* We require our planners generate open-loop trajectories that are robust to uncertainty in object pose, physics modeling, and trajectory execution. This is particularly hard for planning with pushing interactions. The contact between robot and objects causes physics to evolve in complex, non-linear ways and quickly leads to multimodal and non-smooth distributions. Consider Fig.2. Here the initial uncertainty in the object pose appears Gaussian, but after a single push the distribution has sharp edges and is multimodal and unstructured. Our planners must be capable of handing complex evolution of uncertainty in order to produce trajectories robust to noise in object pose, action execution and modeling of physical interactions.

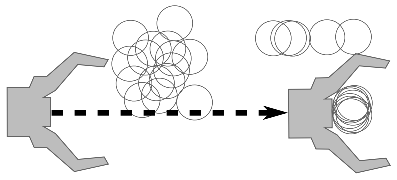


Figure 2: Contact between the robot and object quickly leads to multimodal and non-smooth distributions.

Approach

This thesis describes a suite of planners that address these three challenges. In Part I of the thesis, we describe a set of deterministic planners that solve rearrangement planning under the assumption of perfect knowledge of the world and perfect modeling of robot motions and interactions. These planners address the first two challenges. We then augment these planners to allow us to address the third challenge and handle the uncertainties prevalent when executing tasks in the real world in Part II.

Randomized Rearrangement Planning in Deterministic Worlds

Randomized planners such as the Probabilistic Roadmap (PRM) [62] or Rapidly Exploring Random Tree (RRT) [76] have been shown to work well for high-dimensional state and action spaces. These planners typically rely on being able to quickly solve the two-point boundary value problem (BVP) to connect two states in state space. The use of pushing actions introduces non-holonomic constraints in the planning problem that make solving the two-point BVP difficult. In particular, the motion of the pushed object is directly governed by the physics of the contact between robot and object. To empower our planner to reason about object motion, we *embed a physics model into the core of a kinodynamic randomized planner* [77]. This allows us eliminate the need for simplifying assumption about object geometry and robot-object interaction prevalent in prior work [5, 15]. In this thesis we demonstrate the use of both an analytic physics model with closed form representation and commercial physics models such as Box2D [3].

We first formulate the problem under the quasistatic assumption (Ch.3). This increases tractability by allowing us to eliminate the need to consider velocities in our search space, instead planning only in configuration space. While this limits our solution space, the quasistatic assumption applies in many manipulation applications [6, 7, 8, 24, 34, 89, 97, 129]. We show that we can reliably produce solutions for multiple tasks including pushing objects to goal locations and clearing areas of clutter.

Even in this reduced state space the use of physics models within our planner introduces extra computational complexity that slows the search. Additionally, our formulation in Ch.3 lacks goal-directed actions that explicitly aim to create the contact with objects critical to rearrangement tasks. This further slows planning time. In Ch.4 we demonstrate two techniques to regain speed. First, we show that we can increase the *quantity* of searched space by employing parallelization during tree growth. Second, we increase the *quality* of the search by improving the actions the planner considers. In particular, we show our framework is amenable to including motion primitives prevalent in prior works [15, 36, 94]. By coupling these primitives with our embedded physics model we strengthen their applicability, allowing them to demonstrate whole arm interaction with multiple objects simultaneously. We demonstrate that *allowing the planner to consider both low-level robot motions and higher level object relative primitives improves planning times* and produces a powerful planner able to solve many problems.

Limiting the search to quasistatic interactions allows us to manage the complexity of the search by planning only in configuration space. However, embedding physics models allows us to model dynamic interactions such as striking an object and letting it slide. Naive integration of these dynamic interactions into our planner doubles the search space by requiring incorporation of velocities into the state. This can have a crippling effect on the search time for our planner. We observe that for our problems the absence of any external forces other than gravity causes a manipulated object to eventually come to rest due to friction. In Ch.5, we show that we can use this observation to avoid the increased planning complexity by *considering only dynamic actions that lead to statically stable states*, i.e. we require all objects in the scene to come to rest before the robot executes a new action. This increases the space of problems our planner can handle with only minor penalties to planning time.

Robust Rearrangement Planning

The planners we present in Ch.3-Ch.5 allow us to solve rearrangement problems under perfect knowledge of the planning environment. Such perfect conditions rarely exist. In the second half of the thesis we construct planners that consider the uncertainties prevalent in the real world. In Ch.8 we incorporate uncertainty by exploiting the fact that each call to a randomized planner will generate a fundamentally different trajectory and each generated trajectory varies in its likelihood to achieve the goal when executed in uncertain environments. We formulate rearrangement planning under uncertainty as a *trajectory selection* problem. We use the planners developed in Part I to generate several candidate trajectories. Then we describe a bandit style algorithm to efficiently evaluate these candidates and select the most robust.

Framing the problem as a trajectory selection problem allows us to use our planners with no modification and deal with uncertainty entirely as a post-processing step. While attractive, it does not allow us to make decisions during planning. Prior work [34] has shown pushing interactions can be inherently uncertainty reducing. In Ch.9 we provide a set of *metrics that allow us to characterize the performance of individual actions under uncertainty* and use this to identify actions like the one pictured in Fig.3. We integrate these metrics into our randomized planning framework and show these allow us to produce more robust plans by handling uncertainty at plan time.

This augmentation to our planner allows us to characterize performance of individual actions but does not consider the evolution of uncertainty throughout sequences of actions. In Ch.10 we frame our problem as an instance of an Unobservable Markov Decision Process (UMDP). The nonprehensile interactions we consider lead to non-Gaussian and non-smooth distributions (Fig.2) that have no closed form representation. We leverage the physics model used during planning to perform Monte Carlo simulations of action sequences. This allows us to model the complicated evolution of uncertainty induced by the nonprehensile interaction. We show we can extend Monte Carlo algorithms for solving Markov Decision Precesses to the UMDP domain and use fast heuristic planners to quickly evaluate the robustness of action sequences. This allows us to produce trajectories that perform well under real world uncertainties.

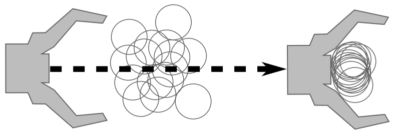


Figure 3: Non-prehensile interactions like pushing can be inherent uncertainty reducing. Here a simple push collapsed initial uncertainty in the pose of the circular object.

Contributions

In summary, we propose the following contributions in this thesis:

1. A kinodynamic randomized planner capable of solving complex rearrangement problems using nonprehensile interactions such as pushing.
2. A set of metrics that identify and measure the robustness of individual actions and full rearrangement plans to uncertainties prevalent at execution time.
3. A set of methods that use Monte Carlo simulations to estimate these metrics, allowing us to identify and generate robust open-loop rearrangement plans.
4. Experimental validation of our developed planners against state-of-the-art baseline approaches on multiple platforms.

We note that this thesis does not offer a single “golden” solution to the rearrangement planning problem. The planners provided here offer trade-offs between implementation complexity, planning time and robustness of the solution (Fig.4). Throughout the thesis we demonstrate these trade-offs on a consistent set of problems for both the HERB robot [115] (Fig.1a) performing household tasks and the KRex lunar rover (Fig.1b) performing tasks in outdoor environments. For each robot, we provide results and anlysis from simulations and real-world experiments. We leave it to the reader to decide the best solutions given the constraints of their individual problem.

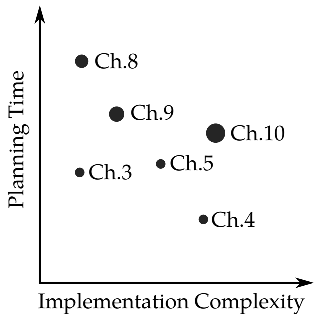


Figure 4: Trade-offs of the proposed planners in this thesis. Larger diameter points indicate greater robustness to uncertainty. Each of the proposed planners demonstrates trade-offs between implementation complexity, planning time and robustness to uncertainty.

Part I

Rearrangement Planning in Deterministic Environments

1 Related Work

This work lies at the intersection of *planning among movable obstacles* and *nonprehensile manipulation*. We provide an overview of related work in each of these individual fields.

1.1 Planning Among Movable Obstacles

We wish to perform tasks that require the robot to work in clutter. To achieve such tasks robots must be able to reason about moving multiple objects. Wilfong [125] was the first to show that problems of this type are NP-hard. However, it has been shown that imposing simplifying assumptions can make the problem tractable.

One of the first formalizations of this class of planning problems was the Navigation Among Movable Obstacles (NAMO) problem [116, 118] (Fig.1.1). Here, the robot is tasked with planning to navigate from a start configuration to a goal among several movable obstacles. These problems differ from prior navigation problems in that the robot is not forced to adapt a plan to the environment but can instead conform the environment to the robot's goal. In other words, the robot can reason about changing the environment through contact to facilitate goal achievement.

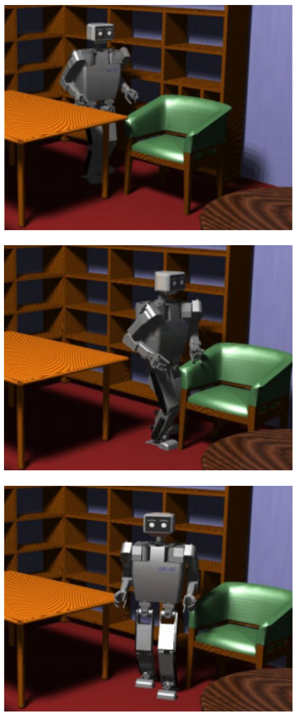


Figure 1.1: An example solution to a Navigation Among Movable Obstacles problem [116]

The rearrangement planning problems we consider are an extension of NAMO into the manipulation domain. Here the robot is tasked with reasoning about the displacement of multiple objects in order to achieve a manipulation goal. Initial work in the field focused on the use of pick-and-place actions to solve the problem [96, 119]. While effective, the class of solvable problems was limited to containing only graspable items. Other work [18, 35, 36] showed that empowering the robot to consider nonprehensile actions in addition to pick-and-place actions broadened the functionality of the robot, allowing robots to solve scenes with items too large or heavy for the robot to grasp.

The NAMO problem and the rearrangement problem are example problems in the general domain of *planning among movable obstacles*. Solutions to the problem can generally be grouped into three categories: (1) forward search over the free space, (2) forward search over the state space or (3) backchaining. In the following sections we summarize the three methods and their applicability to our domain.

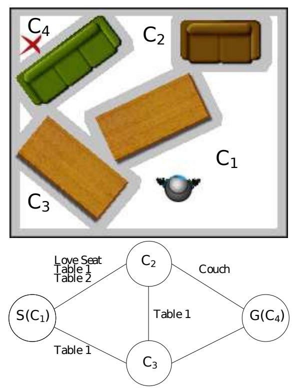


Figure 1.2: An example of rearrangement planning using free space search ([116]). Here the planner constructs a graph of free space components and the objects that form the borders.

Free Space Search

We first define state space in the context of the planning among movable obstacles problems. A *state* is defined as the joint states of the robot and all the movable objects. Under this definition, the size of the state space increases linearly with the number of movable objects in the scene. To avoid searching this high-dimensional space, one approach to the planning among movable obstacles problem is to track the reachable free space. This is the workspace reachable by the robot without moving objects. The planning problem can then be framed as a search for a sequence of actions that place the robot and the goal in the same free space component. Planners which reason over this space [116, 123] often use a hierarchical structure. Here a high-level planner reasons about connecting disjoint regions, while a low-level planner is used to move the robot within a single free space component.

State Space Search

In practice, tracking the free space is often difficult. Alternatively, it is often favorable to plan in state space and use heuristics and problem decomposition to limit the searched region of space.

When performing a forward search over state space, the problem is framed as a search for any sequence of actions or controllers that moves from a start state to any state that represents a goal. Many works have explored modified version of traditional navigation and manipulation planning algorithms. For example, Ben-Shahar and Rivlin [18] perform forward search using a hill-climbing method to traverse a cost-function. This method suffers from two significant drawbacks. First, the cost function represents cost to achieve the goal. Building such a function requires reverse simulation of pushing actions. This is difficult in the general case.

Second, such a cost function relies on a fully specified goal. Often in rearrangement planning problems, the goal is underspecified. In these cases, the goal represents a region of the joint state space rather than a single point. For example, we consider problems that specify the final location for only one of the movable objects. This induces an infinite set of goals with all other movable objects in any location in free space.

Other solutions follow the terminology for manipulation planning introduced by Simeon et. al. [112] and structure the problem as reasoning over two types of action classes:

- *Transit* - The robot moves on its own, without making contact with any movable objects
- *Transfer* - The robot manipulates one or more movable objects

Then the problem becomes searching for a sequence of transit and transfer actions that lead to a goal state. Such problems can be solved using search algorithms suitable for high dimensional problems, such as the RRT. Tractibility is maintained by limiting search to alternating sequences of transit and transfer actions [96].

Further tractibility of the problem can be gained by decomposing the full problem into a sequence of subgoals involving subsets of objects [17, 31, 72, 99]. However, such decomposition explicitly forbids multi-object contact, eliminating a large set of feasible solutions.

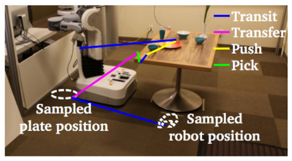


Figure 1.3: An example of a state space search ([15]). Here the planner looks for a sequence of transit and transfer actions that move through the state space to achieve the goal.

Perhaps the work closest to our is the DARRT planner [15]. Here the planner uses an RRT to reason over a set of high-level primitives targeted at a single object. To maintain tractability, the authors exploit the fact that the constraints on most motion, including nonprehensile motion, limit the application of many primitives to a much lower-dimensional space that the full configuration space. We use similar ideas in our formulation but remove the reliance on primitives, reducing programmer burden and allowing multi-object contact and whole arm interaction.

Backchaining

Most of the planners presented in the previous two sections provide methods for solving the planning among movable obstacles problem through forward reasoning. In other words, starting from the initial configuration, the planners search for a sequence of objects to move to achieve a goal configuration. Several works [35, 71, 119] have solved the rearrangement planning problem using a backchaining technique [41, 56, 85, 129]. The planners begin reasoning from the goal, selecting an action and computing the set of objects that must be moved in order to execute that action. This method is applied recursively, building a list and ordering of objects to move in the process. The planners reason over a predefined set of action primitives and attempt to minimize the number of actions required to solve the problem. To reduce the search space the planners rely on the assumption of monotonicity [35, 117, 119] - each object can be moved at most once. In practice, this assumption can be quite limiting. Recent work has proposed methods to eliminate this assumption [71],

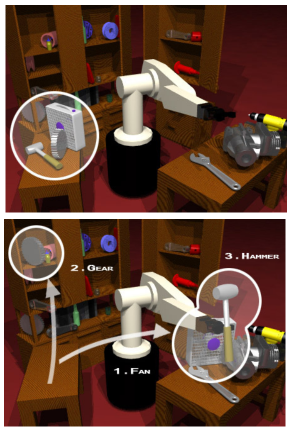


Figure 1.4: An example of a solution to rearrangement planning obtained by backchaining([119]). The goal is to grab the hammer. To do this, the planner must move the gear. To move the gear, the fan must be moved.

	Free space search	Forward state-space search	Backchaining	Single object interaction	Monotone plans	Contact primitives	Fully specified goal	Nonprehensile interaction
[9]		X		X		X		
[15]		X		X		X		X
[17]		X		X	X		X	X
[18]		X		X				X
[31]		X				X		X
[35]			X	X	X	X		X
[36]			X	X	X	X		X
[72]		X		X		X	X	
[71]			X	X		X	X	
[94]		X		X			X	X
[96]		X		X				X
[99]		X		X		X	X	
[112]		X		X			X	X
[116]	X			X		X		X
[117]			X	X	X	X		
[119]			X	X	X	X		
[123]	X			X		X		

Table 1.1: Planning among movable objects

freeing the planners to interact with objects multiple times. We follow this lead: our planning framework allows the robot to repeatedly make and break contact with objects.

Tab.1.1 provides a summary of the works cited in this section and their techniques and assumptions.

1.2 *Nonprehensile Manipulation*

Many definitions of nonprehensile have been proposed in the literature [5, 88, 128]. We will use the simple definition offered by Mason [92]: nonprehensile manipulation is manipulation without grasping.

One of the first uses of nonprehensile manipulation was as a strategy for reducing uncertainty in object pose for parts alignment in manufacturing [7, 21, 23, 42, 40, 46, 54, 103, 128, 129]. These systems used vibration, pushing, rolling and sliding to reliably position and orient parts.

Peshkin [103] noted that design of these parts-feeding and planning robot motion strategy can be considered duals to one another: in a parts-feeder the workpieces move and interact with stationary elements of the machine, while in a robot motion plan the moving

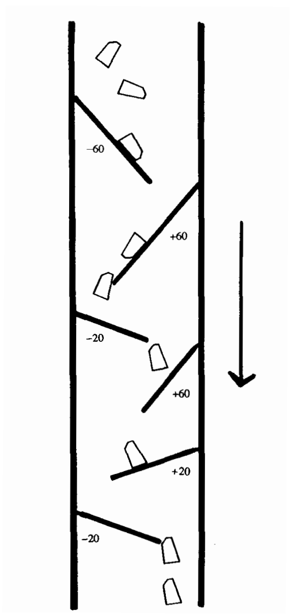


Figure 1.5: Example parts feeder using stationary fences to align objects [103]

robot interacts with stationary parts. Studying parts feeding greatly improved the understanding of nonprehensile interaction and created a baseline for the strategies used by robotic manipulators today.

We focus on the most studied form of nonprehensile manipulation by robots: pushing. Many early works analyzed the motion of a pushed object [86, 84, 91, 87, 102]. The analytical models developed in these works are the foundation for many applications of manipulation by pushing, including ours. Perhaps one of the earliest formulations of a planner using nonprehensile actions was offered by Akella and Mason [8]. They introduce the *Planar Pose Problem*: given a polygonal object on a horizontal table with known start pose (position and orientation), find a sequence of pushing actions to move the object to a goal pose.

The use of pushing imposes constraints not present in traditional manipulation planning algorithms. In particular, the set of forces that can be imparted on the object are limited by the geometric relationship between the robot and the object. This means, for example, that once an object is pushed in one direction, the action cannot be reversed by simply reversing the trajectory of the robot. Additionally, the motion of the pushed object is constrained to the support surface. The existence of these constraints must be incorporated into the motion planning problem.

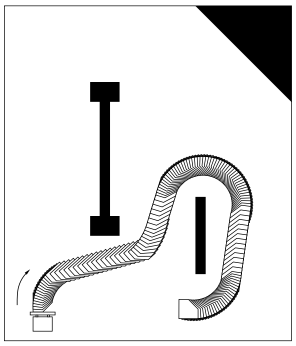


Figure 1.6: An example plan for a pushed object treated as a nonholonomically constrained vehicle [87]

Similar to the planning among movable objects problem, many works have employed traditional motion and maniplation planning techniques to solve the planar pose problem. For example, it has been shown that an object moved by stable pushing with line contact behaves the same as a nonholonomically constrained vehicle [86, 87]. This allows for the extension of planners originally derived for planning for nonholonomic vehicles [14, 79] to the domain of pushing [65, 87].

Alternatively, algorithms common to manipulation planning such as the Probabilistic Roadmap (PRM) [61] or the Rapidly Exploring Random Tree (RRT) [76] have been adapted to handle pushing constraints [5, 94, 127].

Finally, some works have used trajectory optimization techniques to solve the problem [8, 88] by formulating the nonprehensile interaction as a set of constraints on the problem.

We draw inspiration and guidance from many of these works. We employ an RRT with a quasistatic pushing model, but extend beyond the planer pose problem to consider contacts with multiple objects by robots with complex geometries.

2 The Rearrangement Planning Problem

$\mathcal{R}$	Robot
X^R	Robot state space
$\mathcal{M}$	Movable objects
X^i	Movable i state space
$\mathcal{O}$	Static obstacles
X	Planning state space
X_{free}	Free state space
X_G	Goal region
$\mathcal{U}$	Control space
ξ	Feasible trajectory
π	Control sequence

Table 2.1: Rearrangement planning terminology

Assume we have a robot, $\mathcal{R}$, endowed with state space X^R. The robot is working in a bounded world populated with a set, $\mathcal{M}$, of objects that the robot is allowed to manipulate. Each object is endowed with state space X^i for $i = 1 \dots m$. Additionally, there is a set, $\mathcal{O}$, of obstacles which the robot is forbidden to contact. Fig.2.1 depicts each of these sets.

We define the state space of the planner X as the Cartesian product space of the state spaces of the robot and objects: $X = X^R \times X^1 \times \dots \times X^m$. We define a state $x \in X$ by $x = \left(x^R, x^1, \dots, x^m\right), x^R \in X^R, x^i \in X^i \; \forall i$.

We define the free state space $X_{free} \subseteq X$ as the set of all states where the robot and objects are not contacting the obstacles and are not penetrating themselves or each other. Note that this allows *contact* between robot and movable objects, which is critical for manipulation.

We consider pushing interactions. Thus, the state x evolves non-linearly based on the physics of the manipulation, i.e. the motion of the objects is governed by the contact between the objects and the robot. We describe this evolution as a non-holonomic constraint:

$$\dot{x} = f(x, u) \tag{2.1}$$

where $u \in \mathcal{U}$ is an instantaneous control input. The function f encodes the physics of the environment.

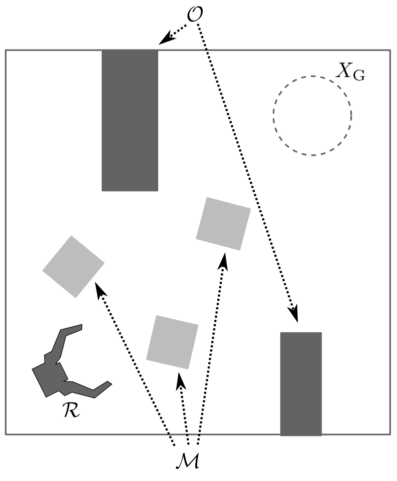


Figure 2.1: All objects in the movable set ($\mathcal{M}$) can be moved to achieve the goal. Objects in the obstacle set ($\mathcal{O}$) must be avoided.

In most manipulation problems, the goal is often under-specified. We define a goal region $X_G \subseteq X_{free}$ as the set of states with the relevant subspace of the state meeting the specification. For example, in [116] only the robot's goal is specified. To represent this goal, we can denote the robot's goal as $x_G \in X^R$. Then we define X_G as the set of all states with the robot in state x_G. In many other problems [33] the task is to move a specific object to a specific place (or a set of places). In these problems, we denote this object as the *goal object*

$G \in \mathcal{M}$ with its state space X^G and its goal as the set $\mathcal{G} \subseteq X^G$. We define $X_{\mathrm{G}} \subseteq X_{free}$ as the set of all states with the goal object in $\mathcal{G}$.

The task of the rearrangement planning problem is to find a feasible trajectory $\xi : \mathbb{R}^{\geq 0} \rightarrow X_{free}$ starting from a state $\xi(0) \in X_{free}$ and ending in a goal region $\xi(T) \in X_{\mathrm{G}} \subseteq X_{free}$ at some time $T \geq 0$. A path ξ is feasible if there exists a mapping $\pi : \mathbb{R}^{\geq 0} \rightarrow \mathcal{U}$ such that $\dot{\xi}(t) = f(\xi(t), \pi(t))$ for all $t \geq 0$. This requirement ensures we can satisfy the constraint f while following ξ by executing the controls dictated by π. Tab.2.1 provides a summary of the terminology introduced in this section.

3
Quasistatic Rearrangement Planning

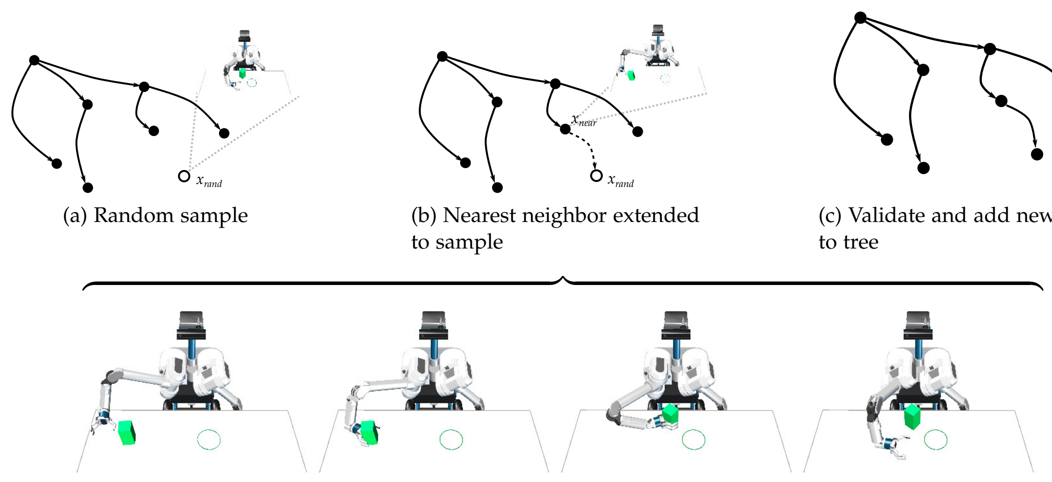
This chapter is adapted from King et al. [64].

We utilize a Rapidly Exploring Random Tree (RRT) [76] to solve the rearrangement problem. The basic RRT algorithm iteratively builds a tree with nodes representing states in X_{free} and edges representing actions or motions of the system through X_{free}. Tree building proceeds in four steps: (1) sample a random state $x_{rand} \in X_{free}$ (Fig.3.1a), (2) locate the nearest node in the tree x_{near} under a distance metric, (3) select a control, $u \in \mathcal{U}$ that minimizes distance from x_{near} to x_{rand} while remaining in X_{free} (Fig.3.1b), (4) add x_{new}, the state reached by applying u, and the edge connecting x_{near} to x_{new} to the tree (Fig.3.1c). The algorithm iterates until a node is added to the tree that represents a goal state $x \in X_G$. RRTs have been shown to be well suited for planning in high dimensional state spaces with non-holonomic constraints, making them an ideal fit for our problem.

Because we must plan in the joint configuration space of the robot and objects, selecting the control u that exactly minimizes the distance from x_{near} to x_{rand} is as difficult as solving the full problem. For example, consider the extension in Fig.3.1. To transition from x_{near} to x_{rand}, we must first find a collision free path for the manipulator from its configuration in x_{near} to a location near the object. Then we must find a path that pushes the object to its new location. Finally, we must generate a collision free path to move the manipulator to its configuration in x_{rand}. As the number of objects in the scene increases, the complexity of finding these sequences that connect two states in X_{free} grows exponentially.

As suggested by Lavalle [77] a useful alternative is to use a discrete time approximation to Eq.(2.1) to forward propagate all possible controls and select the best using a distance metric defined on the state space. In particular, we define an action set $\mathcal{A} : \mathcal{U} \times \mathbb{R}^{\geq 0}$ where $a = (u, d) \in \mathcal{A}$ describes a control, u, and associated duration, d, to apply the control. Then we use a transition function $\Gamma : X \times \mathcal{A} \rightarrow X$ to approximate our non-holonomic constraint.

Our control space $\mathcal{U}$ is continuous, rendering full enumeration of



(a) Random sample
(b) Nearest neighbor extended to sample
(c) Validate and add new edge to tree

Figure 3.1: The basic RRT algorithm. For rearrangement planning, solving for optimal sequence of controls that connect x_{near} and x_{rand} is as difficult as solving the full problem.

the action set infeasible. Instead we approximate it by sampling k actions, forward propagating each under Γ and selecting the best from this discrete set. During forward propagation, we apply a physics model to properly capture motion of the robot and objects. Alg.1 shows the basic implementation. In the following sections we detail important components of the algorithm.

3.1 *Configuration Sampling*

We wish to sample a state $x \in X_{free}$ (Alg.1- Line 3). We can sample from any distribution, as long as we can guarantee that we will densely sample from the space X_{free}. In practice, we sample the robot and all objects from the uniform distribution.

We use a modified rejection sampling to ensure the sampled configuration is valid. We discard states that have robot-obstacle or object-obstacle contact. For sampled states that have object-object or robot-object penetration, we perturb one object in a direction selected uniformly at random until the two components are in non-penetrating contact. If the resulting state is invalid, i.e. the object is in contact with an obstacle or outside of the world bounds, it is discarded. This allows us to sample from the measure-zero set of states with objects and robot in contact.

3.2 *Distance Metric*

Defining the distance between two states $x_1, x_2 \in X_{free}$ is difficult. Prior work [15] denotes the correct distance metric is the length of the shortest path traveled by the robot that moves each movable object

Algorithm 1 Kinodynamic RRT

```
1: $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2: while not ContainsGoal($T$) do
3:     $x_{rand} \leftarrow$ SampleConfiguration()
4:     $x_{near} \leftarrow$ Nearest($T$, $x_{rand}$)
5:     for $i = 1 \dots k$ do
6:         $(u_i, d_i) \leftarrow$ SampleAction()
7:         $(x_i, d_i) \leftarrow$ ConstrainedPropagate($x_{near}, u_i, d_i$)
8:     $i^* = \mathrm{argmin}_i$ Dist$(x_i, x_{rand})$
9:     $T$.nodes $\cup \{x_{i^*}\}$
10:    $T$.edges $\cup \{((x_{near}, x_{i^*}), u_{i^*}, d_{i^*})\}$
11: $path \leftarrow$ ExtractPath($T$)
```

from its configuration in x_1 to its configuration in x_2. Computing this distance exactly is intractable. Even approximating this distance is as difficult as solving the rearrangement problem.

Instead, we use a weighted Euclidean metric.

$$\texttt{Dist}(x_1, x_2) = w_R \|x_1^R - x_2^R\| + \sum_{i=1}^{m} w_i \|x_1^i - x_2^i\| \qquad (3.1)$$

where $x_1 = (x_1^R, x_1^1, \dots x_1^m)$, $x_2 = (x_2^R, x_2^1, \dots x_2^m)$ and $w_R, w_1, \dots w_m \in \mathbb{R}$.

The distance metric serves two purposes in the algorithm. First, it is used to define the nearest neighbor in the tree prior to tree extension (Alg.1-Line 4). Second, it is used to select the best control during propagation (Alg.1-Line 8).

3.3 *Action Space*

We follow the ideas of Simeon et al. [112] and describe feasible plans as combinations of two types of actions: *transit* and *transfer*. We define *transit* actions as those where the robot moves without pushing any movable objects. *Transfer* actions are those where one or more movable objects are contacted during execution of the action.

Transfer actions are critical in rearrangement planning. We focus our action selection such that we remain in areas of the robot configuration space where we are likely to generate *transfer* actions. In particular, we project all actions to a constraint manifold parallel to the pushing support surface (e.g. table). This projection limits the action space to the set of motions where the end-effector of the robot moves along the manifold.

We use the `ConstrainedPropagate` function shown in Alg.2. This constrained extension behaves similar to that described in Beren-

Algorithm 2 The constrained physics propagation function.

```
Require: A step size $\Delta t$
1: function ConstrainedPropagate($x$,$u$,$d$)
2:     $t \leftarrow 0$
3:     $q \leftarrow$ ExtractManipConfiguration($x$)
4:     while $t < d$ do
5:         $q_{new} \leftarrow$ Project($q + \Delta t u$)
6:         $u_{new} \leftarrow q_{new} - q$
7:         $x_{new} \leftarrow$ PhysicsPropagate($x$, $u_{new}$)
8:         if not Valid($x_{new}$) then
9:             break
10:        $(t, x, q) \leftarrow (t + \Delta t, x_{new}, q_{new})$
11:    return $(x, t)$
```

son, et al. [20]. During extension, we apply the input control, u, for a small time step, then project the resulting configuration back to our constraint (Alg.2-Line 5). If successful, we generate a new control, u_{new}, that moves directly along the constraint to this projected point (Alg.2-Line 6). This new control is pushed through our physics model (Alg.2-Line 7). The process is repeated for the entire sample duration d or until an invalid state is encountered.

3.4 *Quasistatic Pushing Model*

We use a quasistatic planar pushing model with Coulomb friction [87] to perform the physics propagation(Alg.2-Line 7). In this model, we assume pushing motions are slow enough that inertial forces are negligible. In other words, objects only move when pushed by the robot. Objects stop immediately when forces cease to be imparted on the object.

We assume friction between the object and underlying surface is finite, the pressure distribution between the object and the surface is known and finite, and friction between the object and manipulator is known. Under these assumptions, we can analytically derive the nonlinear motion of an object when pushed by the manipulator [45, 55].

Using a quasistatic model of interaction allows us to plan on a lower dimensional manifold, $C = \{x = (q, \dot{q}) \in X_{free} | \dot{q} = \mathbf{0}\}$, that represents joint configuration space rather than joint state space, i.e. we do not have to include object velocities in our planning state. Additionally, restricting to pushing in the plane allows $X^i = SE(2)$ for all i, i.e. we can represent the state of movable objects by (x, y, θ).

3.5 Experiments and Results

We implement the algorithm by extending the Open Motion Planning Library (OMPL) framework [120]. We test our algorithm for three goals: 1. Push an object to a goal region, 2. Move the manipulator to a goal region, 3. Clear a region of all objects. In the following sections, we present results and analysis from simulation experiments for each of these three goals. We then present results from execution of the planned paths in the real world.

3.5.1 Push object

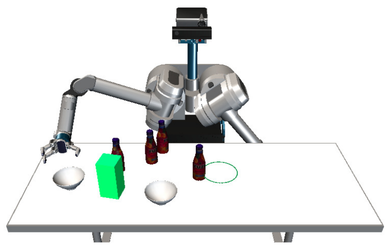


Figure 3.2: An example task. HERB must push the green box along the table into the region denoted by the green circle.

In our first set of experiments, we task our robot HERB [115] with pushing an object (denoted "goal object") on a table to a goal region of radius 0.1 m using the 7-DOF left arm. We test our planner using a dataset consisting of 7 randomly generated scenes with between 1 and 7 movable objects in the robot's reachable workspace. Fig.3.2 shows an example scene. In each scene, we use the same goal object. The starting pose of the goal object is randomly selected. The goal region is placed in the same location across all scenes. We run each experiment 50 times, giving us a total of 350 trials. A trial is considered successful if a solution is found within 300 seconds.

We constrain the end-effector to move in the xy-plane parallel to the table. This allows us to define our action space as the space of feasible velocities for the end-effector. Actions are uniformly sampled from a bounded range. The `Project` function takes the sampled end-effector velocity and generates an updated pose using the Jacobian pseudoinverse:

$$q_{new} = q + \Delta t(J^{\dagger}(q)a + h(q)) \tag{3.2}$$

where q is the current arm configuration, a is the sampled end-effector velocity and $h : \mathbb{R}^7 \rightarrow \mathbb{R}^7$ is a function that samples the nullspace.

Comparison with baseline planners

We denote our planner Physics Constrained RRT (PC RRT) and compare its performance against two baseline planners. The first planner (denoted Static RRT in all results) only allows the robot to push the goal object. All other movable objects are treated as static obstacles. Comparing with this planning scheme allows us to explore our first hypothesis:

H.1 Allowing the planner to move clutter increases success rate and decreases plan time.

H.1 is motivated by two purposes. First, previous work has demonstrated that allowing the manipulator to move clutter increases the number of problems that can be solved [33, 48]. We verify our planner is consistent with these results. Second, we ensure that the extra time required to propagate with the physics model is not so large that the planner can no longer generate feasible solutions in a reasonable amount of time.

We also compare our planner to an implementation of DARRT [15]. Following DARRT, we define three primitives:

1. **Transit** - Move the manipulator from one pose to another via a straight line in configuration space. The motion must be free of collision with any static or movable object in the space.

2. **RRTTransit** - Move the manipulator from one configuration to another by planning using the RRT-Connect algorithm [73]. The motion must be free of collision with any static or movable object in the scene. The planner is run for 5 seconds before the primitive is considered failed.

3. **Push** - Push (or pull) an object along a straight line from the start pose to the goal pose of the object. The object motion is modeled as a rigid connection with the hand. Again, the motion must be free of collision with any static objects and all movable objects other than the one being moved.

At each iteration, DARRT chooses to either sample a new pose for the manipulator or a new pose for a single movable object. In our implementation, we sample these options with equal probability.

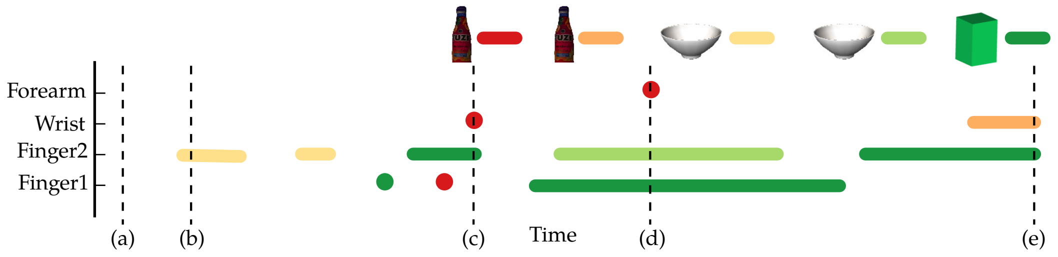
If the manipulator is sampled, then the planner attempts to apply the **Transit** or **RRTTransit** primitive. If any of the objects are sampled, the **RRTTransit** primitive is first applied to move the manipulator to make contact between the end-effector (hand) and object. Then, the **Push** primitive is applied to relocate the object to its desired position.

Comparing with this existing state-of-the-art rearrangement planner allows us to explore a second hypothesis:

H.2 Our algorithm increases success rate and decreases planning time when compared to existing primitive based solutions.

Parameter selection

We identify three parameters that can affect performance of the algorithm. The first is p_{goal}, the probability that the sample state



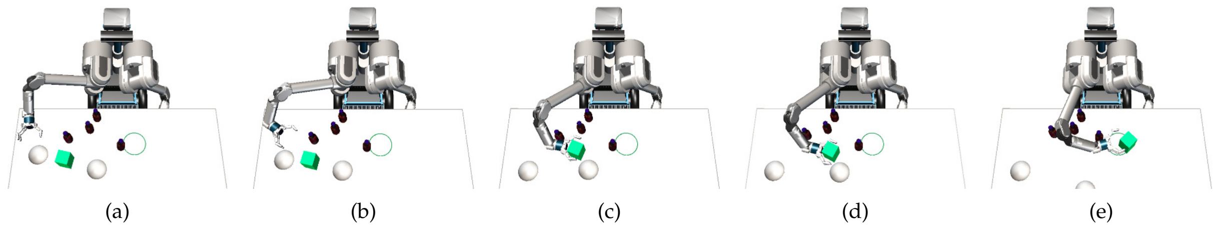


Figure 3.4: In this scene the robot uses the whole arm to manipulate objects to achieve the goal. At several time points ((c), (d), (e)), the robot moves multiple objects simultaneously.

during tree extension is a goal state. We set this value to 0.2 for all three planners. This indicates that approximately 20% of extensions will attempt to connect to a goal state.

Next we select bounds on the control space used for sampling actions at each extension. As described earlier, the control space defines the set of planer twists to apply to the end-effector. These twists are then transformed to velocities for the arm using the Jacobian pseudo-inverse (Eq.(3.2)). While we have hardware limits that define maximum velocities for motions in the full configuration space of the robot, it is difficult to transform these to a set of limits on the end-effector velocities that are guaranteed to produce feasible velocities when transformed through the Jacobian pseudo-inverse from *all* configurations without being overly conservative. Instead, we select end-effector velocity bounds that conform to the full configuration space velocity bounds in *most* configurations. Then, we add a check during tree extension that invalidates any control that produces a full arm velocity that violates bounds. The bounds we select are 0.5 m/s for the Δx and Δy components of the twist and 1.0 rad/s for $\Delta\theta$.

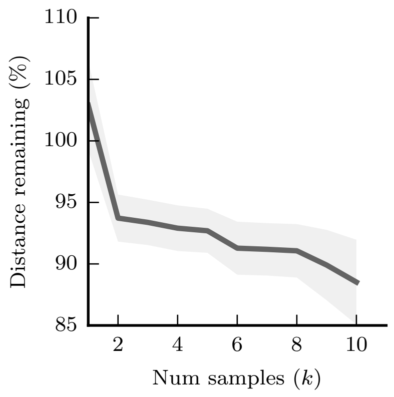


Figure 3.3: Average distance to sample across 50 trials selecting random points.

Finally, we must select k, the number of actions sampled during each extension. We make this selection empirically by analyzing the performance of the extension as k is increased. We run 50 extensions between pairs of randomly generated start and target configurations on a scene with a single object. We try each extension with $k = 1 \ldots 10$ randomly sampled controls and record the distance from the end of the extension to the target for each value of k. Fig.3.3 shows the result. As can be seen, for values of $k \geq 3$ the performance gain is minimal. Thus, we run all experiments with $k = 3$.

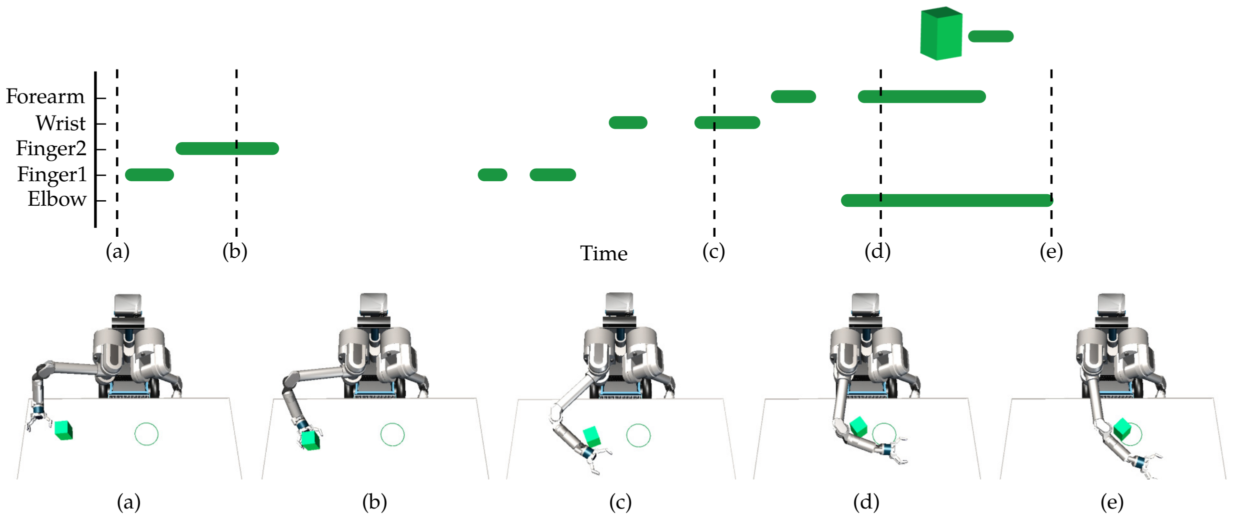


Figure 3.7: In low clutter scenes, the planner still is able to use the whole arm to manipulate the object.

Statistical analysis

Fig.3.5 shows the success rate of the three planners as a function of planning time. Fig.3.6 compares the average plan times across the three planners. For these results, failed planning calls where assigned a duration of 300 s (the planning time limit).

Given a budget of 300 s, our planner improves overall success rate by 15% and decreases plan time by 27 s on average when compared to the Static RRT. We run an analysis of variance (ANOVA) using planner (PC RRT, Static RRT, DARRT) as an independent variable. The results show a significant main effect for both success rate ($F(2, 1046) = 13.45, p < 0.000001$) and plan time ($F(2, 1046) = 7.114, p < 0.001$). Tukey HSD post-hoc analysis reveals the PCRRT differs significantly from the Static RRT in success rate ($p < 0.001$) and plan time ($p < 0.02$). These results support **H.1**: **Allowing the planner to move clutter increases success rate and decreases plan time.**

Fig.3.5 shows that our planner also outperforms the DARRT planner for any time budget greater than 15 s. Overall, our planner solves 73% of scenes while the DARRT planner is only able to solve 57%. Additionally, our planner decreases plan time by 35 s on average. Again, Tukey HSD post-hoc analysis reveals the PCRRT differs significantly from DARRT in success rate ($p < 0.0001$) and plan time ($p < 0.001$). These results support **H.2**: **Our algorithm increases success rate and decreases planning time when compared to existing primitive based solutions.**

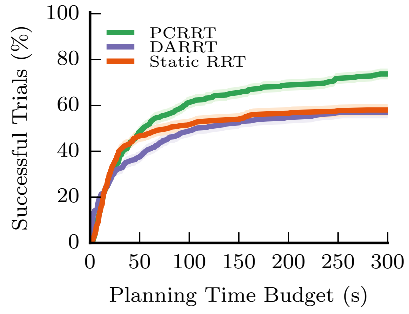


Figure 3.5: The success rate of our algorithm compared with the Static RRT and DARRT. The graph depicts the expected success each algorithm achieves given any time budget up to 300 s.

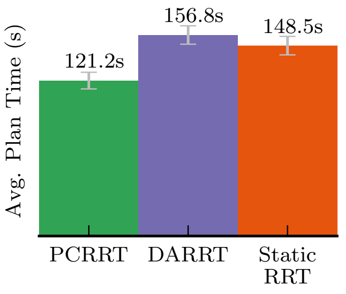


Figure 3.6: The average planning time across all 350 trials for each planner. Failed trials were assigned the max planning time (300s). As can be seen, the PCRRT reduces planning time.

Qualitative analysis

A deeper look at the solutions our planner achieves provides some

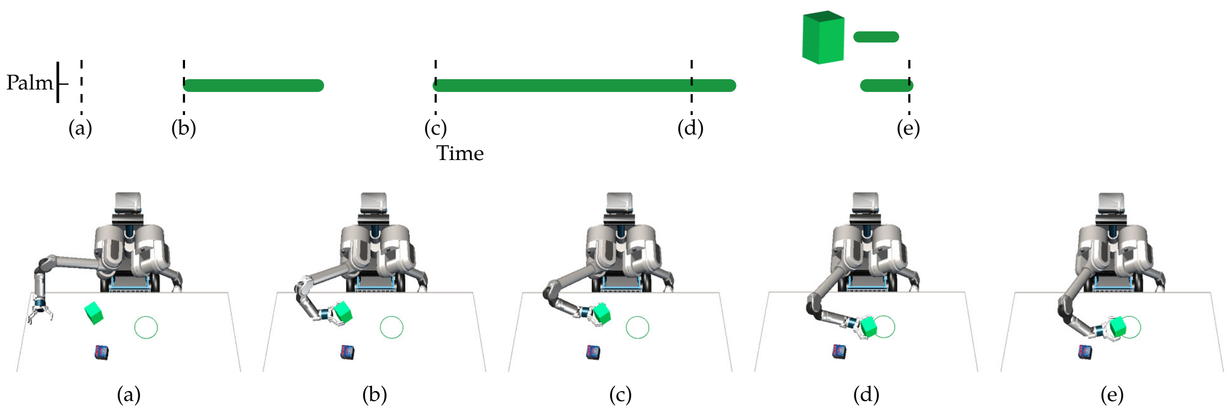


Figure 3.8: In low clutter scenes, the DARRT planner can often solve the problem trivially by applying a single primitive.

insight on the DARRT comparison. Fig.3.4 shows an example solution for a high clutter scene. The solution found by our planner relies heavily on interaction with the full arm and moving multiple object simultaneously. Even in lower clutter scenes such as the one depicted in Fig.3.7, the planner relies on pushing with multiple parts of the manipulator.

Typical primitives-based planners, including our DARRT implementation, allow only interaction with a single object at a time, and restrict that interaction to contact using the end-effector only. This restricts the solutions the planner can consider, causing high rates of failure and longer plan times in scenes with more than just a few items.

However, if we consider scenes with lower clutter, as in Fig.3.8, primitive-based planners such as DARRT work well. In Fig.3.8, the goal object can be moved directly to the goal region, without the need to clear clutter. The **Push** primitive can solve this trivially, allowing for very low planning times. Fig.3.9 shows the success rate as a function of plan time for only this scene. As can be seen, DARRT often finds solutions faster.

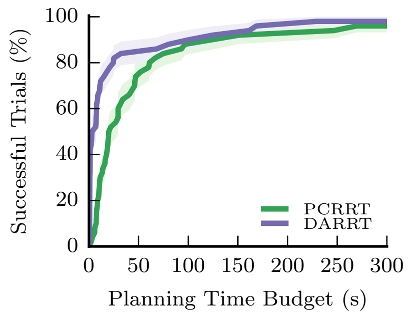


Figure 3.9: Success rate as a function of plan time for the scene from Fig.3.8. In low clutter scenes, primitive based planners such as DARRT outperform our planner because a single primitive can trivially solve the problem.

This highlights a fundamental trade-off between our approach and planners similar to DARRT. The effectiveness of primitive based planners is heavily influenced by the richness of the underlying primitive set. For example, were we to augment our DARRT implementation with additional primitives to move the base or perform pick-and-place it is likely the planner performance would improve. Conversely, our planner is able to achieve good performance sampling from a very basic action set, but trades efficiency on scenes that could be solved with a single primitive.

3.5.2 *Clear region*

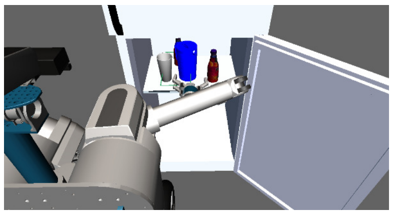


Figure 3.10: An example clearance task. HERB must move the 3 objects in the green rectangle outside the rectangle in order to make space for placing a new item into the refrigerator.

In our next set of experiments, we task HERB to clear space on a shelf in a refrigerator. Fig.3.10 shows an example scene. HERB must

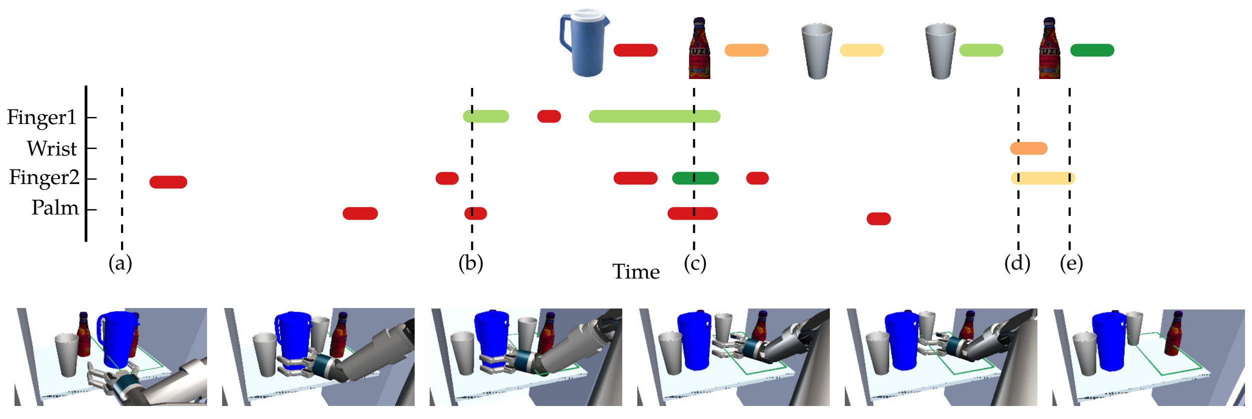


Figure 3.11: In small cluttered space like the shelf of a refrigerator, empowering the planner to allow the robot to touch multiple objects simultaneously and use the whole arm for interaction is critical to finding solutions.

move all objects outside of the green rectangle. The number of objects on the shelf ranges from 3 to 5 objects. The robot can make contact with any items of the shelf, but cannot make contact with the shelf itself or the refrigerator. Similar to the previous set of experiments, we constrain end-effector of the robot to move in the xy-plane parallel to the shelf and use the Jacobian pseudo-inverse to transform end-effector motions to full arm motions.

Fig.3.11 shows an example solution for the task. The result highlights the usefulness of allowing the planner to contact multiple objects simultaneously and use the entire arm. The tight cluttered space makes it nearly impossible to interact with a single object at a time. The DARRT planner struggles to find solutions to these problems. The **Push** primitive fails often because the fingers and arm collide with several objects in the scene while executing even small pushes. Our planner also is slower to solve problems in these small crowded spaces (Fig.3.12) but is eventually able to solve 78% of trials.

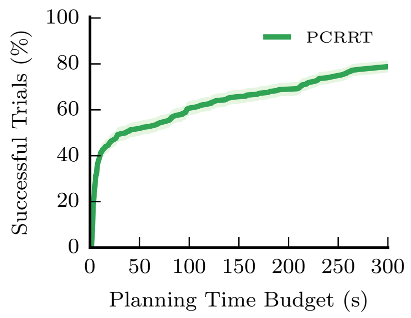


Figure 3.12: The success rate of the PCRRT for the clearance tasks

3.5.3 *Real robot experiments*

Finally, we test that the trajectories we generate are able to be executed with some success on the HERB robot. For this, we return to the task of pushing an object on a table from Sec.3.5.1. We generate trajectories for four scenes with varying amounts of clutter. We measure initial locations of objects and the robot relative to the table. This eliminates the need for a full perception system, and reduces the inaccuracies in initial pose estimates. For each of the four scenes, we generate a trajectory and run it open loop on the robot 10 times. Fig.3.13 depict an execution of a single trial for two of the four scenes. For the 10 trials for each scene, we record success or failure based only on whether the goal object ended in the goal region. Tab.3.1 shows the result.

Scene 1	Scene 2	Scene 3	Scene 4
10/10	4/10	7/10	10/10

Table 3.1: Success counts for trajectories executed on the real robot.

With these tests we are only verifying that our physics model is

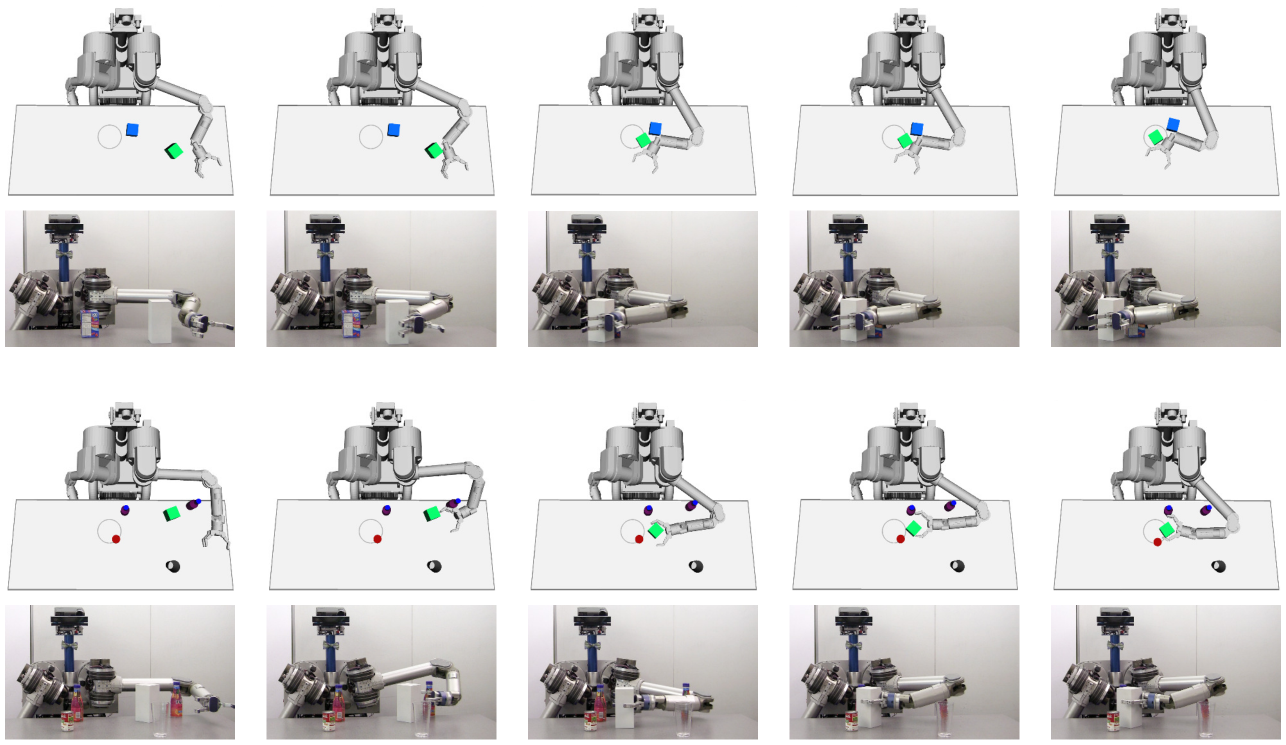


Figure 3.13: Real robot experiments. *Top* Scene 1: The robot is able to achieve the goal in all 10 executions of the trajectory. *Bottom* Scene 3. The robot failed to achieve the goal in 3 of 10 attempts at executing the trajectory. These failures were due to small uncertainties in robot motion and initial object pose that led the robot to push the one of the bottles off of the edge of the table.

good enough to achieve some success. Our results are encouraging– we are able to successfully achieve the goal many times. However, it is clear that our model is not a perfect description of the physical world. Inaccuracies in the physical model of objects, motion of the robot and even small inaccuracies in the initial pose of the objects account for the failures, particularly in Scene 2. In Part II, we will present methods for accounting for these uncertainties.

3.6 *Summary and Discussion*

In this chapter, we show we can use a randomized kinodynamic motion planner to solve rearrangement planning problems. By embedding a physics model into the planner we are able to generate trajectories with full arm manipulation and simultaneous object interaction. Careful selection of this model allows us to reduce our state and action space, making the search feasible. Our experiments show that this solution allows us to solve more problems than primitive based approaches.

We note that while our planner does not require definition of any motion primitives, it is inclusive of them. In particular, if a set of primitives exists, it can be included in the action space. In fact, the DARRT results on scenes with few objects demonstrate that including such primitives may in fact be very beneficial to overall planning time. In the next chapter, we explore this idea in more detail. Additionally, our planner can itself be used as a primitive in a hierarchical planner such as [16, 31]. While this thesis does not explore this idea in detail, we do provide some initial thoughts and further discussion in Ch.11.

4
Improving Randomized Rearrangement Planning

In this chapter, we present methods to improve upon the basic implementation described in Ch.3. In particular we focus on improving the speed and efficiency in generating solutions in Sec.4.2 and Sec.4.3 and improving the quality of the final result in Sec.4.4.

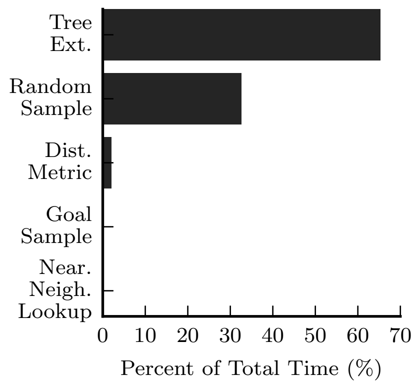


Figure 4.1: A breakdown of the time spent by the planner.

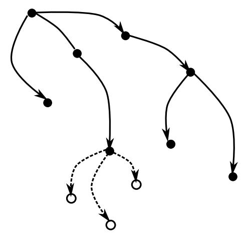
4.1 *Timing Analysis*

Fig.4.1 shows a breakdown of the plan time across five time intensive components of the algorithm: tree extension, random state sampling, nearest neighbor look-up, distance metric computation and goal state sampling. These metrics were collected from all trials for the HERB object pushing task in Sec.3.5.1. As can be seen, tree extension dominates plan time. This is not particularly surprising: extension is often expensive because it requires collision checking a full action, often at high resolution. For our planner, the expense of tree extension is compounded by the need to run a physics model for every action. Additionally, for each tree extension, we run the physics model k times, one for each sampled action. We can improve on our current implementation in two ways. First, we can improve the *quantity* of extensions per second by parallelization. Second, we can improve the *quality* of each extension by sampling from a more relevant action set. We explore both of these methods in the following two sections.

4.2 *Parallelization*

We improve the *quantity* of extensions per second by parallelization. In particular, we parallelize the k rollouts required at each extension, allowing us to test many sampled controls simultaneously. Additionally, we parallelize tree building, allowing us to grow the tree in many directions simultaneously.

First, we use parallelization to speed plan time for a single extension of the tree. In particular, each of the k rollouts performed during an extension is independent given the start state and a dedicated instance of the physics simulator. If we can run separate instances of the physics model simultaneously, we can perform rollouts of our k action samples in parallel (Fig.4.2a). This allows us to significantly reduce the time for each extension.


(a) k rollouts performed simultaneously during an extension

In addition to parallelizing action rollouts for a single extension, we take advantage of the large body of literature focused on parallelizing randomized planners [26, 28, 58] in order to enable parallelizing multiple extensions simultaneously (Fig.4.2b). As a result, we can grow the tree from multiple nodes at the same time. To do this, we use a pool of m *extension threads*. Once a node in the tree is selected for extension, the work is handed to an *extension thread*. The *extension thread* performs the physics rollouts and selects the action to add to the tree, then reports this result back to the main thread for insertion into the tree. Offloading the extension frees the main thread to continue generating random samples and selecting nodes for extension. The result is faster growth of the tree, leading to more exploration of the state space in a given planning time budget.

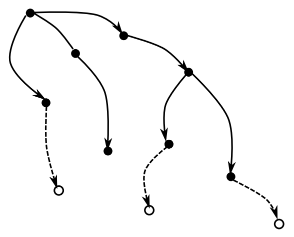

(b) m extensions performed simultaneously

Figure 4.2: Parallelized tree search.

4.2.1 *Analysis*

We first analyze the performance improvements achieved by parallelizing the k rollouts performed during a single extension of the tree. We generate a dataset of 200 extensions. Then, we run each extension 10 times with $k = \{1, \ldots, 8\}$ sampled controls (experiments executed on a machine with 8 available cores). We record the total time to perform each extension for each value of k when we parallelize the k rollouts vs. when we perform the simulations in serial (non-parallelized).

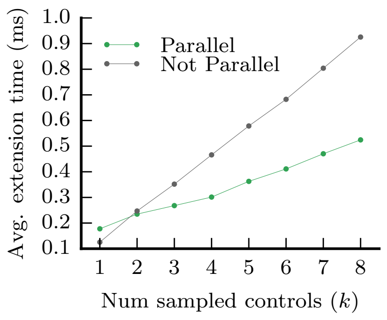


Figure 4.3: Average time for a single tree extension. As we increase the number of controls considered at each extension (k) the improvements from parallelization increase.

Fig.4.3 compares the time to perform an extension for each value of k. For small values of k ($k = 1, 2$) the overhead of performing the parallelization outweighs any performance gains. However, as we increase k, the positive effects of parallelization become more prevalent. When $k = 8$, performing extensions using parallelization leads to a 44% reduction in time per tree extension.

Next we examine the performance improvements achieved by parallelizing tree extension. We run the parallelized planner for a short duration ($t = 10\,\text{s}$) using $m = \{1, \ldots, 8\}$ threads for growing the tree and record the size of the tree after 10 s of planning. We run 30 trials for each value of m, ensuring that each trial runs for the full $t = 10\,\text{s}$, i.e. if the planner finds a solution in less that 10 s, the trial data is discarded and the trial is restarted.

Fig.4.4 shows the average tree size across all trials. For $m = \{1, \ldots, 5\}$ we see significantly improved tree size by increasing the number of threads simultaneously extending the tree. However, for $m > 5$ the performance improvements disappear. At this point, the bottleneck becomes random sample generation, i.e. the main thread cannot select nodes for extension fast enough to keep all worker threads running.

Finally, we test the affect of each of these two methods on the full test dataset for the HERB pushing task from Sec.3.5.1. We test the default planner that did not use any parallelization (**Baseline**) against a planner that parallelizes rollouts during extension (**Extensions**, $k = 8$) and a planner that parallelizes tree growth (**Tree growth**, $m = 5$). Fig.4.5 shows the results. Parallelizing extensions does not demonstrate a significant improvement however it also does not demonstrate decreased performance, despite the fact that we rollout 5 more samples per extension compared to results from Sec.3.5.1. Parallelizing tree building does lead to significantly higher overall success because we explore the state space faster.

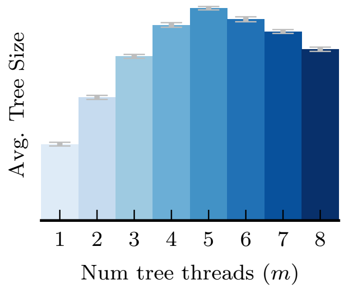


Figure 4.4: Average tree size after 10 s of tree growth. Increasing the available threads for simultaneous tree growth m increases the number of extensions we can consider, growing the tree larger faster.

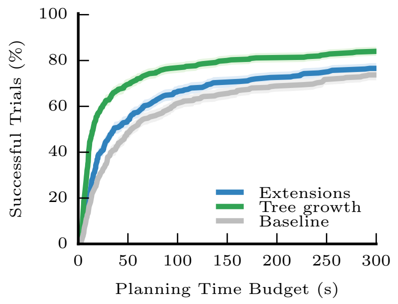


Figure 4.5: Average plan time across successful trials

4.3 *Object-centric Action Spaces*

This section is adapted from King et al. [63].

In this section, we improve the *quality* of each extension by biasing our control sampling to regions of control space likely to create extensions toward our sampled state. This will lead to the need for fewer overall extensions, reducing plan time.

The planner presented in Ch.3 samples lower-level primitives that describe only robot motions during tree extension. These primitives have no object-relevant intent or explicit object interaction. The use of these *robot-centric* motions contrasts methods from previous works that have solved rearrangement planning using high-level primitives that describe purely *object-centric* actions [15, 36, 113, 116, 119]. In these works, the *object-centric* actions guide the planners to perform specific actions on specific objects. Such actions can be highly effective – allowing the planner to make large advancements towards the goal. However, the use of only these actions limits the types of solutions generated by the planner. In particular, these interactions usually involve only contact with a single part of the robot, i.e. the end-effector, and often forbid simultaneous object contact.

The use of *robot-centric* motions in our planner relaxes the restrictions imposed by using *object-centric* actions, allowing the planner to generate solutions that exhibit whole arm interaction and simultaneous object contact. However, the planner suffers from long plan times due to the lack of goal directed motions available to the planner.

Our insight is that both types of actions are critical to generating expressive solutions quickly. By integrating the two action types, we can use the freedom of interaction fundamental to the *robot-centric* actions while still allowing for the goal oriented growth central to the *object-centric* methods. In this section, we first provide an overview of *object-centric* based methods, then describe a modification to our planner that allows for incorporating such actions.

4.3.1 *Using high-level primitives*

During tree extension, the algorithm described in Ch.3 samples k low-level *robot-centric* actions, forward propagates each and selects the one that extends the tree nearest a desired point in configuration space. This is necessary because solving the two-point boundary value problem (BVP) is difficult. However, when selecting actions uniformly at random, we often do not make significant progress towards the sampled configuration. Consider the simple scene in Fig.4.6-*top*. Fig.4.6-*bottom* shows the percent reduction in distance to the sampled configuration as we increase k (averaged across 100 random extensions for the scene) using our method from Ch.3 (purple line). Even with $k = 10$ samples, we only reduce the distance to the sample configuration by 5% on average. This can be particularly detrimental when our sampled configuration is a goal configuration, because our tree fails to grow towards the goal.

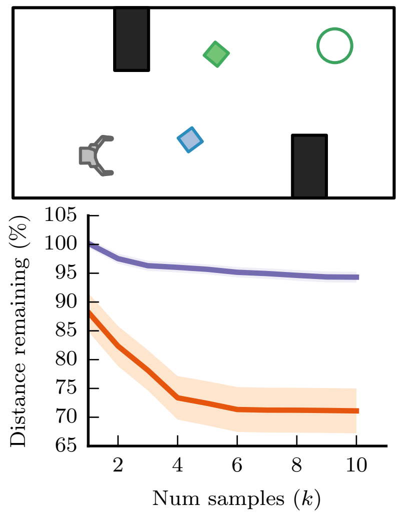


Figure 4.6: Improvement in distance to a sampled configuration as we increase k, the number of sampled controls on each extension of the tree. Using object-centric actions (—) improves the extension quality over using only robot-centric actions (—).

Algorithm 3 Kinodynamic RRT using motion primitives

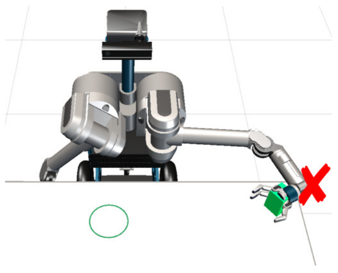
```
1:  $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2:  while not ContainsGoal($T$) do
3:      $x_{rand} \leftarrow$ SampleConfiguration()
4:      $x_{near} \leftarrow$ Nearest($T$, $x_{rand}$)
5:      $a_1, \dots, a_j \leftarrow$ GetPrimitiveSequence()
6:      $x_{new} \leftarrow$ PrimitivePropagate($x_{near}, (a_1 ... a_j)$)
7:      if Valid($(x_{near}, x_{new}), (a_1, \dots, a_j)$) then
8:          $T.\texttt{nodes} \cup \{x_{new}\}$
9:          $T.\texttt{edges} \cup \{((x_{near}, x_{new}), (a_1, \dots, a_j))\}$
10: $path \leftarrow$ ExtractPath($T$)
```

An alternate method employed in [15, 113] is to use a set of *object-centric* primitives capable of solving the two-point BVP in a lower dimensional subspace. For example, a "push-object" primitive would be capable of providing a sequence of actions the moves a single object from a start configuration to a sampled configuration. Alg.3 shows the integration of primitives into the RRT.

This method is attractive because it can allow large extensions of the tree, and the sampling method is highly connected to tree growth. Fig.4.6-*bottom* (orange) shows the reduction in distance to the sampled state on an extension is much better when using these *object-centric* primitives. This is particularly useful when the sample is a goal state: it allows the tree to grow to the goal.

However, the reliance on *object-centric* actions to generate all object motion is detrimental in two ways. First, the actions are limited in their expressiveness. In particular, contact is restricted to only interactions between the manipulator and the single object targeted by the action. The `PrimitivePropagate` function (Alg.3-line 6) explicitly prohibits contact with other movable objects or obstacles in the scene. This prevents simultaneous object interactions, eliminating many feasible solutions when the robot is working in clutter.

Second, and possibly more important, this method is susceptible to failure if the primitive cannot be successfully applied. Consider the example in Fig.4.7a. An example primitive may be to move the hand near the box with the palm facing in the direction of the desired push, then push the box in the direction of its sampled location. The box is near the edge of the reachable workspace of the manipulator. As a result, all attempts at applying the high-level action will fail because the robot cannot reach the desired pose relative to the box. Even more problematic, a solution to the scene cannot be found given the current action space. To generate a solution, the programmer must define alternative or more flexible primitives (Fig.4.7b).


(a) Desired end-effector pose for a "push-object" primitive is not within the reachable workspace of the robot

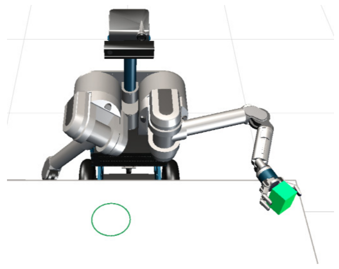

(b) An alternative achievable end-effector pose that cages and pulls the object.

Figure 4.7: An example failed "push-object" primitive. The desired end-effector pose is not reachable. An alternate primitive that cages and pulls the object must be defined for the planner to find a solution in *object-centric* primitive based approaches.

4.3.2 *Hybrid approach*

Alg.4 shows a modified algorithm that allows for the freedom of interaction fundamental to our *robot-centric* method while still allowing for the goal oriented growth central to the *object-centric* methods. Like in our original solution, at each tree extension the best of k possible actions is selected. However, each candidate i expresses a sequence of actions, A_i. With some probability, p_{rand}, the sequence A_i contains a single action $a = (u, d)$ drawn uniformly at random from the space of feasible robot motions. With probability $1 - p_{rand}$, A_i contains a sequence of actions, $\{a_1, \ldots, a_j\}$, with noise applied to the primitive parameters. In all cases, the sampled action sequence A_i is propagated through the physics model and the sequence is truncated at the first infeasible state encountered, i.e. collision with a static obstacle.

This solution is attractive because it combines the strengths of our original algorithm with the strengths of high-level primitive based methods. Incorporating the physics model into the propagation removes the restriction that *object-centric* primitives can only allow interaction between the manipulator and the object the primitive is defined on. Instead, any unintended contact with other objects in the

Algorithm 4 Kinodynamic RRT using hybrid action sampling

```
1: $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2: while not ContainsGoal($T$) do
3:     $x_{rand} \leftarrow$ SampleConfiguration()
4:     $x_{near} \leftarrow$ Nearest($T$, $x_{rand}$)
5:     for $i = 1 \ldots k$ do
6:         $r \leftarrow$ Uniform01()
7:         if $r < p_{rand}$ then
8:             $A_i \leftarrow$ SampleUniformAction()
9:         else
10:            $A_i \leftarrow$ SamplePrimitiveSequence()
11:        $(x_i, A_i) \leftarrow$ PhysicsPropagate($x_{near}, A_i$)
12:    $i^* = \text{argmin}_i$ Dist($x_i, x_{rand}$)
13:    if Valid($(x_{near}, x_{i^*}), A_{i^*}$) then
14:        $T$.nodes $\cup \{x_{i^*}\}$
15:        $T$.edges $\cup \{((x_{near}, x_{i^*}), A_{i^*})\}$
16: $path \leftarrow$ ExtractPath($T$)
```

scene can be modeled. Often, this unintended contact is not detrimental to overall goal achievement and should be allowed. Sampling random actions with some probability allows the planner to generate actions that move an object when all primitives targeted at the object would fail (i.e. the example in Fig.4.7).

4.3.3 *Experiments and Results*

Again, we implement the planner described in this section in the OMPL framework. We test three versions of our updated planner. First, we set $p_{rand} = 0$. This forces the planner to always sample *object-centric* actions. We denote this planner **object-centric** in all results. Second, we set $p_{rand} = 1$. This forces the planner to always sample *robot-centric* actions. We note this is exactly the planner originally presented in Ch.3. We denote these results **robot-centric** in this section. Finally, we set $p_{rand} = 0.5$. This allows the planner to choose *object-centric* or *robot-centric* actions with equal probability. We denote this planner as **hybrid** in all results. In the next section, we discuss alternate values for p_{rand} and their impact on the results.

Using the planner, we execute a set of experiments to test the following hypotheses:

H.1 Propagating **object-centric** primitives through the physics model during tree extension improves the performance of planners that purely rely on these primitives by allowing the primitives to exhibit simultaneous object contact and full arm interaction.

H.2 The **hybrid** planner achieves higher success rate and faster plan times than the **object-centric** or **robot-centric** planners.

We test the hypotheses across multiple tasks in scenarios for two robots: the KRex rover and HERB.

HERB Experiments

We first test **H.1** using the same dataset described in Sec.3.5.1. Here, we use 7 scenes that require our robot HERB to push an object across a table from its starting location to a goal region. To test the hypothesis we compare the **object-centric** planner to our implementation of the DARRT planner described in Sec.3.5.

Object-centric primitives

We define a *push* primitive similar to the DARRT primitive defined in the experiments in Ch.3. The *push* primitive pushes an object along the straight line connecting a start and goal configuration for the object. The primitive returns a set of actions that first move the end-effector to a position and orientation "behind" the object, then move the end-effector straight along the line, pushing the object. The motion of the end-effector is confined to the plane parallel to the table surface during the entire primitive.

This primitive differs from the DARRT primitive in two ways. First, we allow and model contact between the robot and all other movable objects in the scene during execution of the primitive. Second, we only allow the robot to transit to the object via a straight line in workspace. We note that this is more restrictive than the DARRT primitive that can use a motion planner to plan motions of the arm out of the plane.

By using these primitives within the **object-centric** planner ($p_{rand} = 0.0$), we create a planner very similar to DARRT. The key difference is the use of the physics model to propagate primitives and model contact between all movable objects in the scene and the full arm. As a result, by comparing these two planners we are able to isolate and understand the effect of allowing these new interactions.

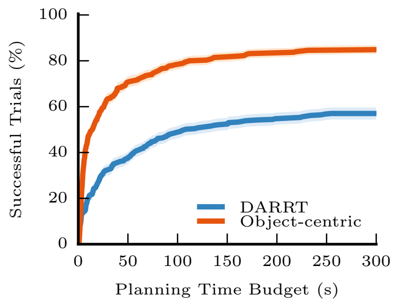
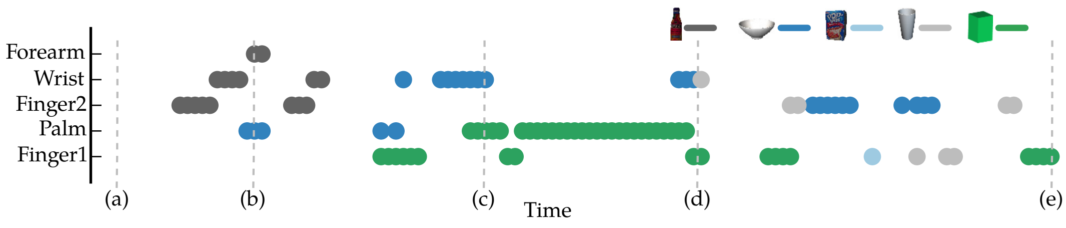


Figure 4.8: Success rate as a function of plan time for HERB pushing objects on a table.

Quantitative results

Fig.4.8 shows the success rate as a function of plan time across all 7 scenes for the object-centric planner and the DARRT planner.



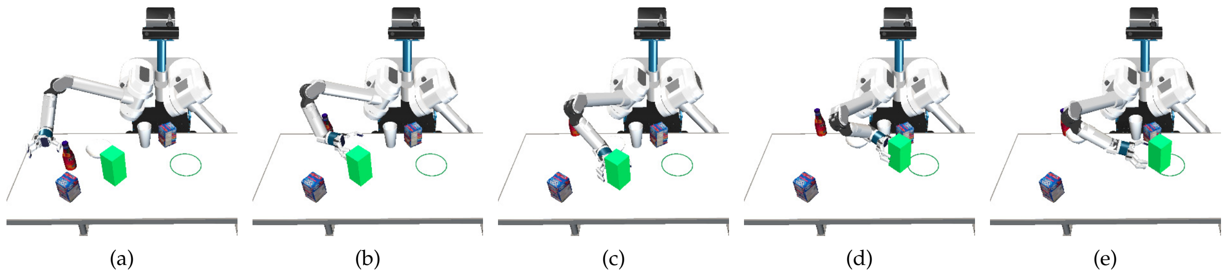


Figure 4.9: An example object-centric solution. The push primitive is used to move the bowl in (b) and the box in (c)-(e). Other objects are moved through incidental contact during execution of push or transit primitives.

The **object-centric** planner is able to solve 84% of problems while the DARRT planner solves only 57%. This confirms our hypothesis: **Propagating object-centric primitives through the physics model during tree extension improves the performance of planners that purely rely on these primitives by allowing the primitives to exhibit simultaneous object contact and full arm interaction.**

Qualitative analysis

Analyzing breakdown of the success rate of the planners as a function of the amount of clutter in the scenes reveals much of the performance improvements come from higher clutter scenes.

Fig.4.9 shows an example object-centric solution from a scene with 6 movables. For this scene, DARRT found solutions in 24% of trials while the **object-centric** planner solved 98% of trials. The *object-centric* solution uses the push primitive to move the bowl (Fig.4.9b) and the green box (Fig.4.9c, Fig.4.9d). The bottle, glass and blue box get moved through incidental contact during other push or transit primitives. The contact occurs using several parts of the arm including the wrist, forearm and back of the hand. These contacts are not typically modeled in primitive based approaches. The ability to use the whole arm and make simultaneous contact with objects allows the transit and push primitives to succeed when they would otherwise fail if restricted to hand-object interaction.

KRex Experiments

We next test **H.2** using the mobile manipulator KRex (Fig.4.10). The robot behaves as a steered car. For this robot, $X^R = SE(2)$ and a control $u = (v, \delta) \in \mathcal{U}$ describes the forward velocity and steering an-

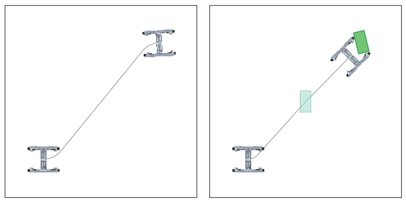
Figure 4.10: The K-Rex lunar explorer.

gle applied to the robot. The robot interacts with objects in the plane. We use Box2d [3] as our physics model to forward propagate all actions. As in the previous chapter, we use objects that are quasistatic in nature, allowing us to represent the state of objects by only their configurations. As a result $X^i = SE(2)$ for $i = 1 \ldots m$.

We define the following primitive set for the robot:

***Transit**:* The transit primitive moves the robot from a start to a goal configuration in $SE(2)$ by finding the shortest length Dubins curves [39] connecting two configurations (Fig.4.11a).

Push The push primitive pushes an object along a straight line connecting a start and goal configuration for the object. The primitive returns a set of actions that first move the robot to a position and orientation "behind" the object, then drives the robot straight along the ray, pushing the object (Fig.4.11b).


(a) Transit primitive (b) Push primitive
Figure 4.11: Two primitives defined for KRex. Dubins paths are used to generate paths between two poses in $SE(2)$.

When sampling random actions, we sample forward velocity from the range $[-0.2, 0.2]$ m/s and duration from the range 0.5 s to 5.0 s. We sample steering angle from the range $[-0.5, 0.5]$ rad. We also use these velocity and steering angle bounds when generating Dubins curves. The values are imposed by the physical limitations of the vehicle.

Tasks

We test the updated planner across three tasks for KRex:

***Traversal**:* The rover must drive from a start configuration to a goal region with 10 cm radius. We test this task across 6 scenes.

***Push object**:* The rover must push a box from a start configuration to a goal region with 20 cm radius. We test this task across 5 scenes.

***Clearance**:* The rover must clear a region of all items. We test this task across 7 scenes. The size of the clearance region varies across scenes.

For each task, we run each of the three planners 50 times for each scene and record results. The scenes in each task contain a varying number of movable objects and static obstacles. Examples of scenes for each task are presented in the following sections.

Quantitative results

Fig.4.12 compares the success rate of the three approaches across all three tasks. The hybrid approach consistently outperforms both the robot-centric and object-centric planners. For the traversal task,

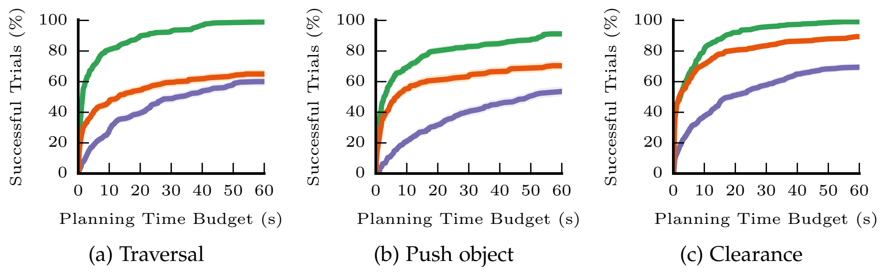


Figure 4.12: The success rate as a function of plan time for each of the three KRex tasks. The hybrid planner (—) outperforms both the object-centric (— and robot-centric (—) planners across all tasks.

the hybrid approach is able to solve 99% of trials, outperforming both the object-centric (65%) and robot-centric (60%) methods. In addition, it finds solutions much faster, solving over 75% of problems in less than 10 s. Statistical analysis reveals a significant main effect for plan time ($F(2, 897) = 121.2, p < 0.0001$) and success ($F(2, 897) = 84.61, p < 0.0001$).

Similar results are revealed with statistical analysis of the push object task (plan time - $F(2, 747) = 75.4(p < 0.0001)$, success rate - $F(2, 747) = 49.32(p < 0.0001)$) and the clearance task (plan time - $F(2, 946) = 49.86(p < 0.00001)$, success rate $F(2, 946) = 39.17(p < 0.00001)$).

Tukey HSD post-hoc analysis shows the hybrid approach differs significantly from the robot-centric and object-centric approaches in success rate ($p < 0.0001$) and plan time ($p < 0.0001$) on all three tasks.

This analysis supports our **H.2**: **The hybrid planner achieves higher success rate and faster plan times that the object-centric or robot-centric planners**.

Qualitative analysis

While we have shown that in general the hybrid planner solves problems more effectively than using only robot-centric or object-centric actions, a deeper look at solutions to individual scenes provides further insight into these gains.

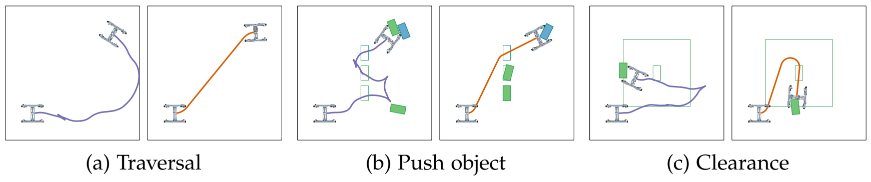


Figure 4.13: A robot-centric (—) and object-centric (—) solution easy scenes for each task. *Left*: The robot must traverse through an empty environment to a goal region in the top right corner. *Middle*: The robot must push the blue box into a goal region in the top right corner. All green boxes are movable objects. *Right* The robot must push the green box from its start pose in the middle of the environment to any pose outside the region outlined in green.

Fig.4.13 depicts a pure robot-centric and pure object-centric solution for a single scene for each of the three tasks. These scenes are all "easy": they can be solved by a single primitive. The traversal task (Fig.4.13a) requires KRex to drive from a start pose in the bottom left corner of the world to any pose in a 10 cm radius goal region in the top right corner. In this object and obstacle free world, a single transit primitive can be used to find a direct path (Fig.4.13a-right). Conversely, several small robot motions must be used to solve the task when using the planner from Ch.3 (Fig.4.13a-left). As a result, the object-centric planner is able to find solutions for this scene much more efficiently than the robot-centric planner (Tab.4.1). The hybrid planner can also find these single-primitive solutions, allowing it to solve the scene just as quickly as the object-centric. Similar results can be seen for the push object task (Fig.4.13b) and the clearance task (Fig.4.13c).

		Robot-centric	Hybrid	Object-centric
Traversal	Success(%)	76	100	100
	Plan time(s)	21.82 ± 23.92	0.13 ± 0.13	0.54 ± 0.51
Push	Success(%)	78	100	100
	Plan time(s)	19.3 ± 16.6	0.75 ± 0.75	1.28 ± 1.39
Clearance	Success(%)	100	100	100
	Plan time(s)	1.17 ± 1.39	0.08 ± 0.03	0.10 ± 0.03

Table 4.1: Success rate and plan time for easy scenes. Plan times report mean and $\pm$ one standard deviation.

However, when we look at higher clutter scenes we begin to see the advantage of the hybrid planner. Consider the solution to the pushing task in Fig.4.14 found by the hybrid planner. Transit and pushing primitives are used to make large motions through the state space that connect points. However, the importance of robot-centric motions is highlighted in Fig.4.14b- Fig.4.14d. The push primitive executed in Fig.4.14b led the robot to push a movable object (green box) near an obstacle (gray box). The robot cannot advance forward without pushing the movable into the obstacle. As a result, both the transit and push primitives fail. A robot-centric action must be applied to back the robot away (Fig.4.14c), freeing the robot to apply another push primitive on a different movable (Fig.4.14d). A similar sequence of motions can be seen in Fig.4.14d- Fig.4.14f.

The traversal task in Fig.4.15 demonstrates similar use of robot-centric actions to move away from obstacles (Fig.4.15c) and boundaries (Fig.4.15e). The need for the robot-centric motion is due to the use of Dubins curves to instantiate the object-centric primitives. The set of primitives does not fully span the space motions the robot can achieve – neither the transit or push primitive can reverse the robot. By sampling low-level motions uniformly from $\mathcal{U}$ the robot-centric

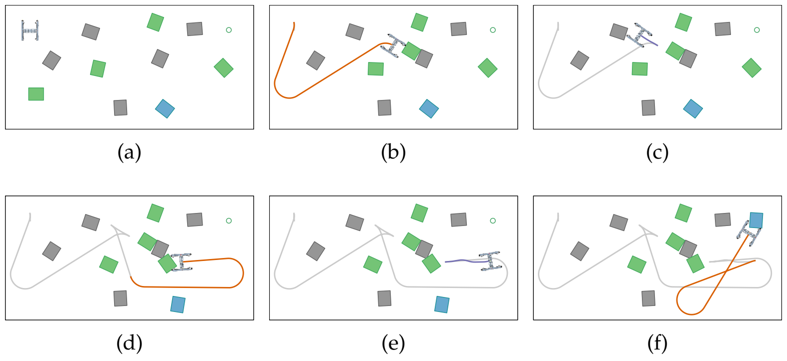


Figure 4.14: A single solution to a complex pushing task. Here the robot uses object-centric (—) motions to make large advancements through the state space. Robot-centric motions (—) are used to back the robot away from obstacles.

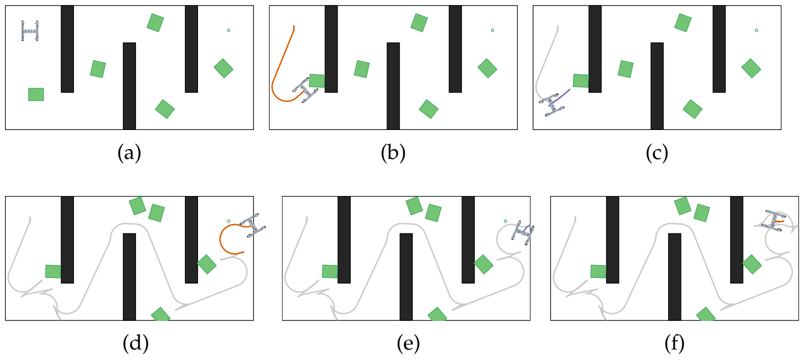


Figure 4.15: A single solution to a complex pushing task. Here the robot uses object-centric (—) motions to make large advancements through the state space. Robot-centric motions (—) are used to back the robot away from obstacles.

motions can "fill the gaps" left by object-centric primitives.

Real robot experiments

In addition to our simulation results, we test the ability of the robot to successfully execute plans generated for the pushing and clearance tasks. These tests were executed on the KRex rover at the NASA Ames Research Center.

We first tasked KRex with pushing a box from a start pose into a goal region with 50 cm radius. We used the same setup as shown in Fig.4.13b and tested three scenarios, one for each box. In scenario 1, KRex was tasked with pushing the left-most box from its start pose to the goal region. In scenario 2, KRex was asked to push the center box and in scenario 3, KRex was asked to push the right-most box. The goal location and start position of KRex were the same across scenarios. We used the hybrid planner to generate a trajectory for each scenario. We then executed each trajectory on the robot 10 times. Objects were measured and placed in the correct starting location. This eliminated the errors and complexity that the use of a perception system introduces. After executing a trajectory, the distance from the center of the box to the center of the goal region was measured and recorded.

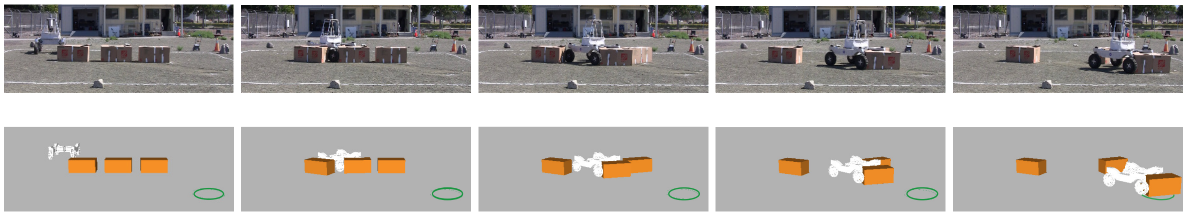


Figure 4.16: Execution of the object pushing task on the KRex rover

Scenario 1	Scenario 2	Scenario 3
54.3 ± 4.2	48.7 ± 13.3	79.5 ± 6.5

Table 4.2: Average distance from goal center (cm)

Fig.4.16 compares the planned execution with the true execution of a single solution to scenario 2. Encouragingly, the actual trajectory closely follows the planned trajectory. Tab.4.2 shows the average distance of the final pose of the box to the center of the goal region. In scenario 2, the goal was achieved in all but 1 trial.

Scenario 1 and scenario 3 performed worse. In each of these scenarios, the final pose of the object stopped short of the goal several times. These failures can be attributed to two sources of error. The first is in the execution of the robot motion. For our HERB experiments in Ch.3, the robot used a position controller to track the desired robot trajectory. Given a sufficiently small stepsize between trajectory points, the deviations between actual robot motion and our modeled robot motion were minimal. In these KRex experiments, the robot used a velocity controller to follow the planned actions. This type of control is much more susceptible to integration errors, i.e. small inaccuracies in modeling early motion can accumulate to large inaccuracies in final robot position. As a result, the accuracy of our modeled robot motion was lower.

The second source of error is in modeling of the physical environment. The box that KRex pushed was modeled as a rigid object, though in reality there is some compliance in the interaction. This compliance leads the box to move less than modeled. In addition, the pushing surfaces was composed of crushed gravel. While we account for this when selecting coefficients of friction, we do not explicitly model the gravel in the physics simulator. Thus effects such as piling of the stones as the box is pushed are unmodeled. These phenomenon prevent the box from moving as modeled.

While the goal failed to be achieved many times in scenario 1 and scenario 3, the overall deviation of the final pose of the box across trials was less than 10 cm. This indicates that the trajectories themselves are stable and repeatable.

Next we tasked KRex with a set of four clearance tasks. Here, we asked KRex to push a box from its start pose to any pose outside a designated clearance region. Fig.4.17 shows an example solution for one scenario.

For this task, we performed five trials for each of the four scenar-

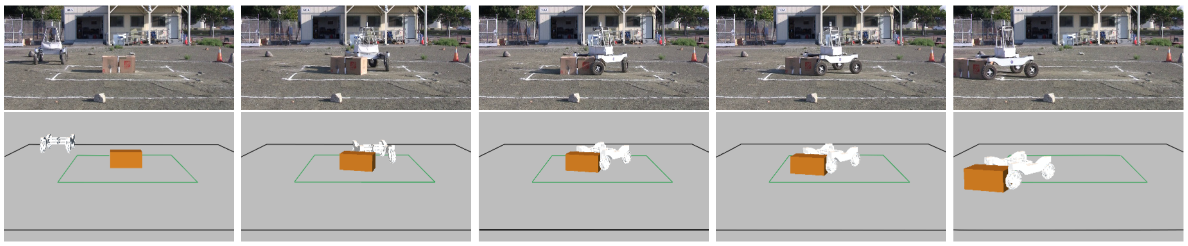


Figure 4.17: Execution of the clearance pushing task on the KRex rover

ios and simply recorded success or failure based on whether the final pose of the box was outside the clearance region. Tab.4.3 shows the result. Scenario 1 proved the least robust. However, in each of the 3 failures, the box was less that 5 cm from the edge of the goal region. Overall, this task exhibited more success than the object push task. We believe this is due to the much larger goal region inherent to the task. While the motion of the box may not be perfectly modeled, the errors are much more likely to lead to alternate states that still lie inside the goal region. This is evident when comparing the planned vs. actual final configuration in Fig.4.17. These achieved final configuration (*top*) differs from the planned final configuration (*bottom*) but is still in the goal region.

Scenario 1	Scenario 2	Scenario 3	Scenario 4
2/5	5/5	5/5	5/5

Table 4.3: Success rate of the clearance task.

Effect of p_{rand}

In our KRex and HERB experiments, we used a hybrid planner with $p_{rand} = 0.5$ to compare against an object-centric ($p_{rand} = 0$) and robot-centric planner ($p_{rand} = 1$). Fig.4.18 examines the success rate of the planner for different values of p_{rand} for all 7 HERB scenes and all 5 KRex traversal scenes. As we decrease p_{rand} we allow the planner to consider object-centric actions more often. The plot shows two interesting trends. First, for both HERB and KRex, the use of any object-centric primitives ($p_{rand} < 1$) leads to improved success rate. For the HERB scenes, p_{rand} seems to have little effect beyond this point. However, for KRex, the use of both object-centric and robot-centric primitives ($0 < p_{rand} < 1$) proves beneficial.

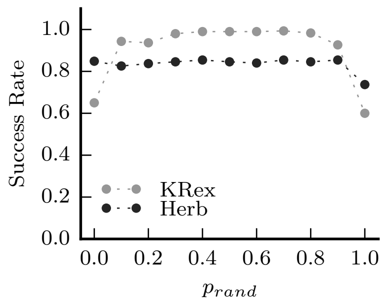
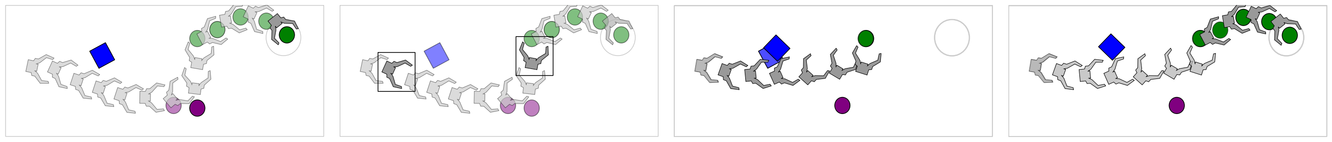


Figure 4.18: The success rate as a function of p_{rand}, the percentage of time robot-centric actions are used to grow the tree.

We believe these seemingly conflicting results can be explained by the construction of the object-centric primitive set for each robot. As mentioned previously, the KRex primitive set fails to span the space of feasible actions for KRex, namely they do not include reverse motions for the robot. Robot-centric motions are important because they include capabilities not expressed in the object-centric primitive set.

Conversely, the object-centric primitives for HERB do span the space of feasible actions: the transit primitive can move the robot in any direction. By empowering the object-centric primitives to



(a) The initial path. (b) The two manipulator states selected for connection. (c) The new states generated for the shortcut. (d) The new states added after the shortcut.

Figure 4.19: The shortcutting algorithm. For simplicity, we depict only the motion of the end-effector. However, the whole arm is considered during planning. In (d) the motion of the manipulator and green object remain unchanged along the remainder of the path. However, states are updated to reflect the new location of the blue box and purple circle.

allow simultaneous object contact and full arm interaction, we have improved the strength and applicability of the primitive, rendering the robot-centric actions less necessary for this task.

4.4 *Path Shortcutting*

Finally, in this section we improve the quality of the output of the planners. The use of a randomized planner leads to paths that often contain unnecessary movements of the manipulator. Several post-processing techniques to smooth these paths have been introduced in the literature. One commonly used technique [43, 51, 108, 110] is to randomly select two points from the path and attempt to connect them. If successful, intermediate points between the two selected points are discarded from the path, and the new connection is added.

Use of this technique typically requires solving the two-point boundary value problem to generate the new edge. As described in Ch.3, this is non-trivial for our problem. Instead we employ a slightly altered technique illustrated in Fig.4.19.

Given a trajectory produced by our planner, we first randomly select two points in the trajectory (Fig.4.19b). We then attempt to generate a new action to connect only the robot configuration in these two states. If such an action is generated, we forward propagate it using our physics model, and record the new intermediate states (Fig.4.19c).

We note that the ending state of the shortcut action could differ from the sampled state in the configuration of one or more movable objects. For example, in Fig.4.19c the final state differs in the configuration of both the blue box and the purple circle. Because of this, we must forward propagate all remaining actions in the trajectory and ensure the goal is still achieved (Fig.4.19d). If successful, the updated path is accepted and the algorithm iterates.

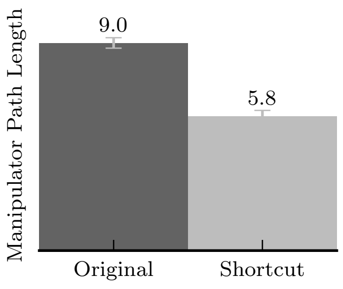


Figure 4.20: Results from shortcutting for 15s.

4.4.1 *Analysis*

To examine the performance of the shortcutting algorithm, we return to the results provided in Sec.3.5.1. We run the shortcut algorithm

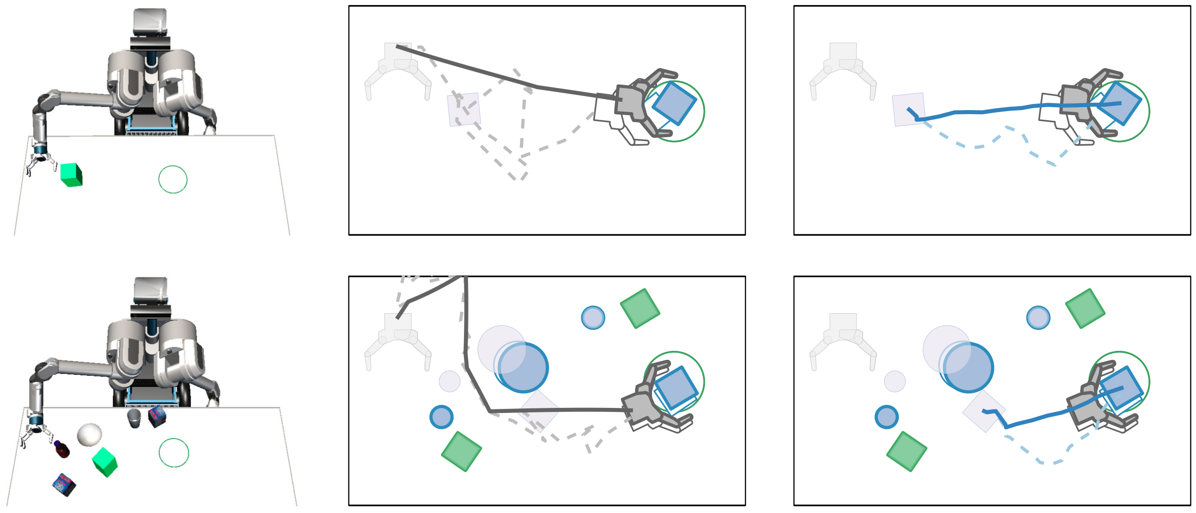

(a) Original scene (b) Manipulator path (c) Object path

Figure 4.21: An example shortcut. (b) The original path (dotted) contains unnecessary extra movements removed in the shortcut path (solid). This leads to a new final pose of the robot and object (filled) when compared to the original (outline). (c) The shortcut also often leads to reduced motion of the object.

for a maximum of 15 s on each of the 258 successfully generated paths. Fig.4.20 shows a 35% reduction in manipulator path length after shortcutting. A paired t-test comparing manipulator path length before and after shortcutting shows a significant main effect ($t = 17.34, p < 0.00001$).

Fig.4.21 shows two example shortcutting results for the scenes in Fig.4.21a. In Fig.4.21b we render the original path of the manipulator (dotted) and the new path of the manipulator (solid). As can be seen, the new path is more direct, cutting out motions unnecessary for goal achievement. In both scenes, the shortcut path leads to a slightly modified final state (original final state shown in outline). This new state is also a goal, allowing the shortcut to be accepted. Note that we render only the path of the end-effector for visual simplicity, though the shortcutting is done in the full configuration space.

In Fig.4.21c we render the new path of the goal object. Though reducing the distance this object is moved is not part of the shortcut objective, the reduction of robot motions leads to a reduction of motion of the object in many cases.

4.5 *Summary and Discussion*

In this chapter we presented three improvements to the randomized rearrangement planner presented in Ch.3. We improved the *quality* of each extension of the tree by integrating *object-centric* high level primitives into our action set. By empowering these primitives to allow simultaneous object contact and full arm interaction, we improved their applicability. In addition, we showed that these primitives alone are not always strong enough to solve rearrangement problems. By supplementing the *object-centric* primitives with low-level *robot-centric* primitives we were able to improve overall performance of the planning framework.

In addition to the improved *quality* of each extension, we also increased the *quantity* of extensions through parallelization. We showed that for our problem parallelizing tree growth can speed planning.

Finally, we proposed a shortcutting technique that allows us to improve the quality of solutions our randomized planner generates. While our inability to exactly solve the two-point BVP prevents us from directly using traditional shortcutting techniques, we showed we are able to take advantage of the ability to solve the two-point BVP in the lower dimensional subspace containing only the robot in order to find feasible shortcuts.

The combination of these techniques improve the speed and quality of our planner. However, our experiments on KRex indicate that while these plans may be feasible, uncertainties in the real world cause them to fail to be properly executed. The use of the KRex robot introduced new uncertainties in the control and environment modeling that are less prevalent in HERB's well modeled indoor environments. We must account for these additional uncertainties as we formulate methods to create more robust output from the planner in Part II.

5
Rearrangement Planning with Dynamic Interactions

This chapter is adapted from Haustein et al. [52] and contains work completed in collaboration with Joshua Haustein.

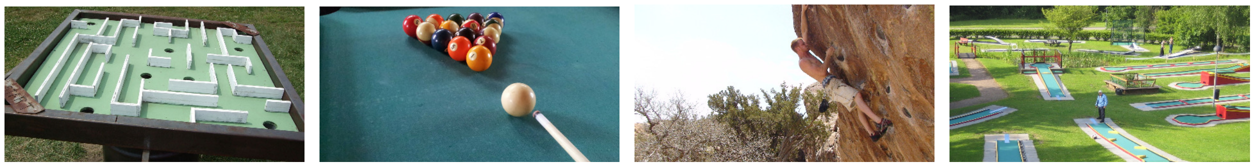
The solution presented in Ch.3 reduced the search space by assuming all interactions between robot and objects are quasistatic in nature. This allows us to plan in configuration space, removing the need to track velocities of objects in the scene. However, the use of the quasistatic assumption limits the applicability of the planner to scenes with objects that move quasistatically. In particular, the planner cannot be used to solve planning queries with objects that easily slide or roll. In this chapter we introduce a technique that eliminates the need for the quasistatic assumption while still maintaining tractability in the search.

5.1 Incorporating Dynamic Interactions

The recent increasing availability of fast dynamic rigid body physics models [1, 2, 3, 4] allows modeling dynamic interactions between a robot and the objects in its environment in planning algorithms [126, 127]. However, directly incorporating these models into our algorithm incurs the costly consequence of searching in the full joint state space containing both the configuration and velocity of each object and the robot. This doubles the dimension of the search presented in previous chapters.

Our insight is that we can choose dynamic actions that result in statically stable states, e.g. the environment comes to rest after each action. This allows us to keep the search in the lower dimensional configuration space. Consider a bartender sliding a beer bottle to a customer or a soccer player kicking a ball. In both cases, in absence of any external forces other than gravity, the manipulated object eventually comes to rest due to friction. In many activities humans follow a similar approach by transitioning dynamically from one static state into another static state (Fig.5.1). We modify our planner to harness these same approaches to interaction.

We incorporate dynamic actions by formulating the rearrangement



(a) Labyrinth maze (b) Pool billard (c) Bouldering (d) Mini golf

Figure 5.1: Examples for dynamic actions between statically stable states utilized by humans.

problem as a search for dynamic transitions between statically stable states in the joint configuration space. We then incorporate a dynamic physics engine into our kinodynamic RRT to model dynamic motions such as a ball rolling. To guarantee statically stable states, we require the environment come to rest after a duration T_{max}. In the following section, we detail the changes to our algorithm required and present results from the updated planner.

5.2 *Terminology and Assumptions*

We denote the manifold $C = \{x = (q, \dot{q}) \in X_{free} | \dot{q} = \mathbf{0}\}$ as the environment's statically stable free state space, where every movable object and the robot are at rest. As pointed out in Sec.3.4, this manifold represents the free configuration space of the robot and movable objects. Let C^R and C^i denote the statically stable free state space for the robot and all movables objects $i = \{1 \dots m\}$.

We assume our start state $x_0 \in C$, i.e. the robot and all objects start at rest. Additionally we assume the goal region $\mathcal{G} \subseteq C$. The task of our planning problems remains unchanged from Ch.2: we wish to find a feasible trajectory, ξ, through X_{free} that begins at x_0 and ends in $\mathcal{G}$.

5.3 *Planner Updates*

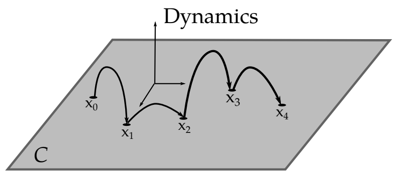


Figure 5.2: Dynamic transitions between statically stable states in C

In Ch.3 we structured the problem as a search for actions $a \in \mathcal{A}$ that connect states in X_{free}. We assumed Γ approximated a quasistatic model of physics, ensuring $x_{t+1} = \Gamma(x_t, a) \in C$. In other words, the quasistatic assumption ensured that action a connected two states in the statically stable free space. In this section, we allow Γ to approximate a fully dynamic model of physics. We modify our definition of an action to include a duration, d_{rest}, to apply the null control to our robot, i.e. the duration to keep the robot at rest but allow movable objects to continue rolling or sliding. Then, we require that given a state $x_t \in C$ and an action $a \in \mathcal{A}$, $x_{t+1} = \Gamma(x_t, a) \in C$ (Fig.5.2).

To ensure this property holds during planning, we update the `ConstrainedPropagate` function from Alg.2. Alg.5 shows the updated function (denoted `DynamicConstrainedPropagate`). During exten-

Algorithm 5 The dynamic constrained propagation function. After applying the manipulator action for the desired duration, the physics propagate is called for an additional duration with no manipulator movement. All objects must come to rest by the end of this additional time.

```
Require: A step size $\Delta t$
1:  function DynamicConstrainedPropagate($x$,$u$,$d$,$d_{rest}$)
2:      $t \leftarrow 0$
3:      $q \leftarrow$ ExtractManipConfiguration($x$)
4:      $\hat{x} \leftarrow$ AddVelocities($x$)
5:      while $t < d + d_{rest}$ do
6:          if $t > d$ then
7:              $u \leftarrow \emptyset$
8:          $q_{new} \leftarrow$ Project($q + \Delta t u$)
9:          $u_{new} \leftarrow q_{new} - q$
10:         $\hat{x}_{new} \leftarrow$ PhysicsPropagate($\hat{x}$, $u_{new}$)
11:         if not Valid($\hat{x}_{new}$) then
12:             break
13:         $(t, \hat{x}, q) \leftarrow (t + \Delta t, \hat{x}_{new}, q_{new})$
14:     if ObjectsAtRest($\hat{x}$) then
15:         $x \leftarrow$ RemoveVelocities($\hat{x}$)
16:         return $(x, t)$
17:     else
18:         return $(\emptyset, 0)$
```

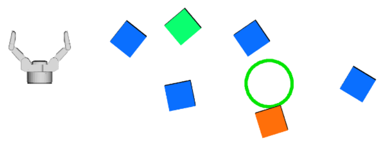
sion, the state is expanded to include velocities. The `AddVelocities` functions initializes velocities of all objects to 0 (Alg.5-Line 4). After the robot action has been applied for the desired duration d, the physics model continues to be applied for an additional duration, d_{rest}, with no robot movement (Alg.5-Line 7). Before returning, the return state is checked to ensure all objects in the environment are stopped (Alg.5-Line 14). Only extensions resulting in these states are considered valid.

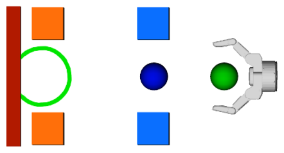
5.4 *Experiments and Results*

In these experiments we use Box2D [3] as our physics model. Due to the nature of this model, all experiments are run in a 2D workspace. The state space for each movable object is $X^i = SE(2) \times se(2)$, the set of poses and twists in the plane. The robot is only the hand (end-effector) of HERB, which is assumed to move holonomically in the plane. Therefore $X^R = SE(2) \times se(2)$.

The action space $\mathcal{A} = se(2) \times \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, is the set of twists, consisting of translational and rotation velocities to be applied for a duration, d. As described in Sec.5.3, the third element of an action describes the duration, d_{rest}, for the robot to wait while objects come to rest. We set d_{rest} to be a constant value, T_{max}, throughout a single planning call, i.e. every action has the same value for d_{rest}. We model the dynamics of the robot leading to achievement of a desired twist as a ramp profile. In other words, we assume max acceleration is applied to achieve the twist, then max deceleration is applied to bring the robot back to rest at the end of the duration d. In these experiments we set $k = 10$.

We test 12 different scenes with varying numbers of movable objects and obstacles. Fig.5.3 depicts 3 of the scenes. We allow and model contact between movable objects and obstacles. However, we do not allow the robot to contact any obstacles. We task the robot with pushing a single object into a circular goal region. We run each planner 110 times on each scene. A run is considered a success if the planner finds a solution within a planning time of 60 s.


(a) Scene 1


(b) Scene 2

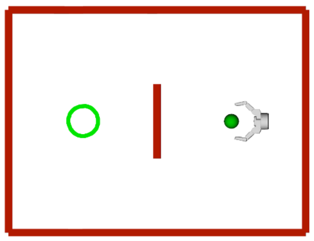

(c) Scene 3

Figure 5.3: Three example scenes. In all three scenes, the robot must move the green object into the goal region denoted by a green circle. Blue objects are movable. Red and orange objects are obstacles.

Comparison with baseline planners

We run our planner (denoted **Semi-dynamic** in all results) for different choices of $T_{max} = 0\,\text{s}, 1\,\text{s}, 8\,\text{s}, 20\,\text{s}, 60\,\text{s}$. This allows us to test our first hypothesis:

H.1 Increasing T_{max} leads to higher success rates for the **semi-dynamic** planner.

H.1 is motivated by the fact that larger values of T_{max} allow a greater variety of actions. This is because the larger wait time increases the likelihood of the environment coming to rest.

We compare against a baseline planner that plans in the full state space, X_{free} (denoted **Dynamic** in all results). This planner uses the original propagate function from Alg.2 but uses the dynamic physics model during the `PhysicsPropagate` step. As a result, this planner searches across the full free space, X_{free}, rather than the manifold C. This allows us to examine the following hypothesis:

H.2 Given a restricted planning time budget, the **semi-dynamic** planner achieves higher success rate than the **dynamic** planner.

H.2 is motivated by the fact that the **semi-dynamic** planner plans in $C \subset X_{free}$ while the dynamic planner searches the larger X_{free}. Given a restricted time budget, we expect the semi-dynamic planner to find a solution faster as long as a semi-dynamic solution exists, i.e. there is enough friction in the environment to eventually bring all objects to rest within T_{max}.

Generating full arm trajectories

Algorithm 6 Jacobian pseudo-inverse trajectory conversion

```
Require: An end-effector trajectory $\xi$ of duration $T$, a mapping $\pi$, a
    step size $\Delta t$
1: $\xi' \leftarrow$ InitTrajectory()
2: $\pi' \leftarrow$ InitActionMapping()
3: $q \leftarrow$ SampleStartIK($\xi(0)$)
4: $t \leftarrow 0$
5: while $t < T$ do
6:     $a \leftarrow \pi(t)$
7:     $\phi \leftarrow J^{\dagger}(q)a + h(q)$
8:     UpdateTrajectory($\xi'$, $q$, $t$)
9:     UpdateMapping($\pi'$, $\phi$, $t$)
10:    $q \leftarrow q + \Delta t \phi$
11:    $t \leftarrow t + \Delta t$
12:    if not Valid($q$) then return $(\emptyset,\emptyset)$
   return $(\xi', \pi')$
```

Our choice of physics simulator allows us to quickly model dynamic interactions, but only in 2D environments. However, we wish to generate trajectories for 7-DOF arm of the HERB robot. To do this, we first plan motions for the end-effector moving in the plane using our planner. Then, in a separate post-processing step we "lift" these trajectories to the robot's full configuration space by using the Jacobian pseudo-inverse to generate joint velocities that achieve these end-effector motions. Alg.6 shows the basic algorithm we use.

We initialize the conversion by sampling a full arm configuration from the set of inverse kinematics solutions that place the end-effector in the initial configuration specified in the trajectory (Alg.6- Line 3). During the conversion, we generate a new trajectory ξ' that describes the full arm configurations of the robot at each time step (Alg.6- Line 8). Additionally, we generate a new mapping π' that describes the joint motions the robot should execute at each time step (Alg.6- Line 9). During trajectory generation, intermediate configurations of the robot are checked to ensure there is no unmodeled collision, e.g. collision between the forearm and an obstacle or movable object (Alg.6- Line 12). If we encounter an invalid configuration, an empty path is returned and an outer process restarts the conversion.

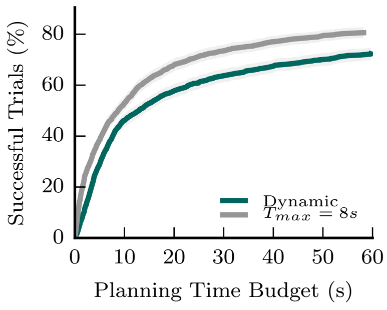


Figure 5.4: The success rate of the semi-dynamic planner with $T_{max} = 8$ s and the dynamic planner. Given a restricted time budget of 60 s, the semi-dynamic planner solves more scenes faster.

Quantitative analysis

Fig.5.4 shows the success rate of the semi-dynamic planner for $T_{max} = 8$ s and the dynamic planner on all scenes as a function of planning time budget. As the planners are allowed more time, more solutions are found and the success rate grows. For all time budgets, the semi-dynamic planner outperforms the dynamic planner. This support **H.2**: **Given a restricted time budget the semi-dynamic planner achieves higher success rate that the dynamic planner**.

Next, we compare the success rates for different choices of T_{max}. From Fig.5.5 we can observe two interesting trends. First, the planners with $T_{max} = 0\,\mathrm{s}$ and $T_{max} = 1\,\mathrm{s}$ perform worst. This can be explained by the fact that many scenes require manipulation of objects that behave dynamically. Consider the scenes in Fig.5.3b and Fig.5.3c. The robot's task is to move the ball into the goal. Finding actions that result in statically stable states after 0 s or 1 s is unlikely because the ball keeps rolling for some time after contact.

Second, the success rates for $T_{max} = 8\,\mathrm{s}, 20\,\mathrm{s}, 60\,\mathrm{s}$ do not differ largely from each other at a budget of 60 s. The planner achieves the highest success rate when $T_{max} = 8\,\mathrm{s}$, followed by $T_{max} = 20\,\mathrm{s}$ and then $T_{max} = 60\,\mathrm{s}$. This weakens **H.1**: **Increasing T_{max} does not necessarily lead to a higher success rate for a restricted planning time budget.**

While the results support our hypothesis for $T_{max} < 8\,\mathrm{s}$, the benefit of increasing T_{max} vanishes in the range of $[8\,\mathrm{s}, 20\,\mathrm{s}]$. At first glance, this is surprising since any solution found with some T'_{max} can also be found with $T''_{max} > T'_{max}$. An explanation can be found through closer examination of the effect of T_{max} on tree growth. In particular, we examine the average time to try a single extension under different values of T_{max} in Fig.5.6. As we increase T_{max}, we must spend more time running the physics simulator to see if objects come to rest, increasing propagation time. This time penalty means that given a fixed planning time budget, fewer extensions can be added to the tree as T_{max} increases, reducing the amount of exploration the planner can realize.

Finally, we examine the average path duration for the three scenes from Fig.5.3. Scene 1 contains only high friction objects. These objects are quasistatic in nature - they come to rest almost immediately after the robot stops imparting forces. For these scenes, path duration is similar for all values of T_{max} and the dynamic planner (Fig.5.7-(*left*)).

Scene 2 contains two low friction objects, depicted as balls. These objects continue moving after contact with the robot is lost. Here, the path duration increases with higher values of T_{max}. This indicates that these solutions make use of actions that allow the balls to roll for a longer duration after breaking contact with the robot. Fig.5.7-(*right*)

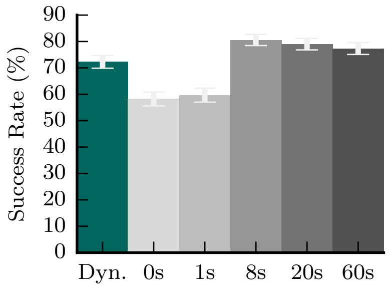


Figure 5.5: Success rate for different values of T_{max}.

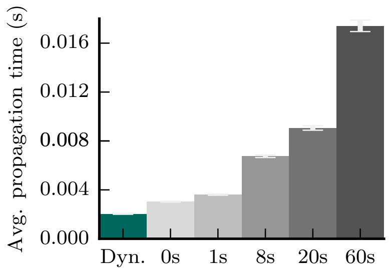


Figure 5.6: Average time to try a propagate a single extension of the tree. As T_{max} increases the propagation time increases.

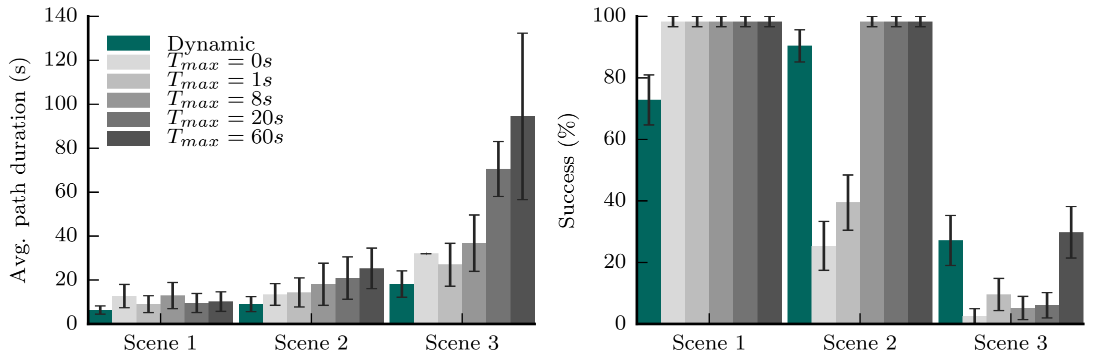


Figure 5.7: Results for three scenes in Fig.5.3. (*Left*) Average path duration. (Note: Only one path was found for $T_{max} = 0\,\text{s}$ in Scene 3) (*Right*) Success rate.

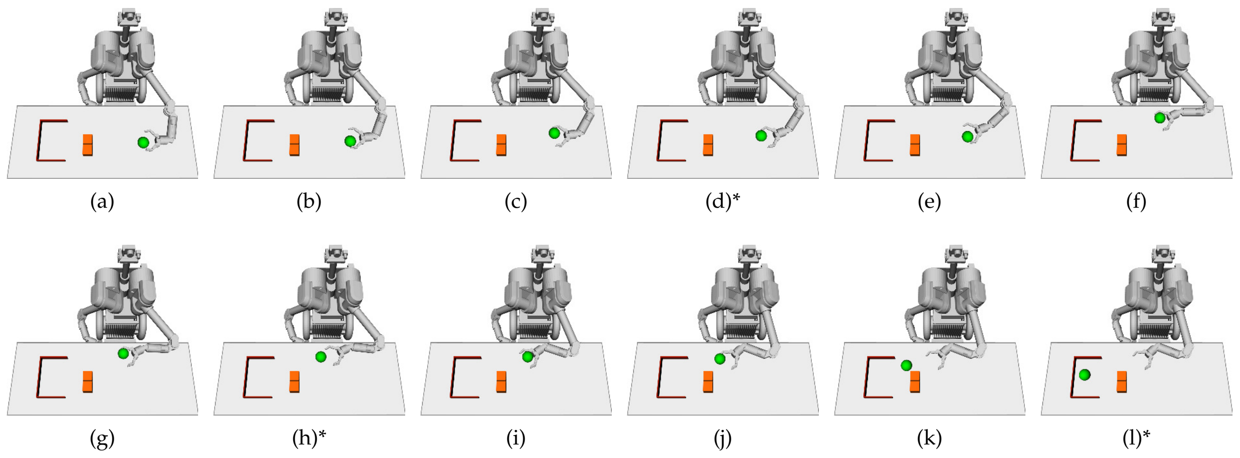
supports the need for these types of actions. The success rate on this scene when $T_{max} = 0\,\text{s}, 1\,\text{s}$ is significantly lower than the rate for higher values.

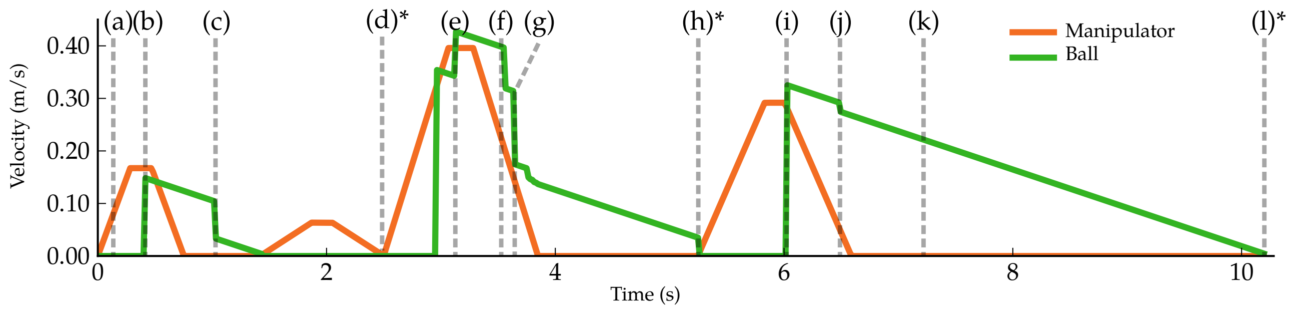
Scene 3 proves to be difficult for all planners. It contains a static wall blocking direct access to the goal. Furthermore, the goal region lies in open space, where no obstacle prevents the ball from rolling through the goal region. The semi-dynamic planner with $T_{max} = 0\,\text{s}, 1\,\text{s}, 8\,\text{s}, 20\,\text{s}$ found very few solutions. If a solution was found the path duration is on average shorter than for greater T_{max} as it involves less time waiting for objects to come to rest. For $T_{max} = 60\,\text{s}$, more solutions are found but the average duration of solutions is largest. The dynamic planner achieves similar success rate on this environment, but found paths with the shortest execution times on average. This highlights a fundamental trade-off in using the semi-dynamic planner: semi-dynamic planners find solutions quicker but typically exhibit longer path duration due to the need to wait for the environment to come to rest between actions.

In fact, on average the solutions found by the dynamic planner are shorter in execution time for all scenes with dynamically behaving objects.

Qualitative analysis

Fig.5.8 shows an example trajectory that was first planned in 2D then lifted to a full arm trajectory. The robot's task is to move the green ball, which is modeled as a low-friction disc, into the goal on the left side (red box). As can be seen in the velocity profile of the end-effector and the target object, the robot moves the target object from one statically stable state to another. A particularly interesting caging behavior occurs between Fig.5.8b, Fig.5.8c and Fig.5.8e, Fig.5.8f. In both cases, the robot first accelerates the ball using one side of the end-effector and then catches it at the end of the





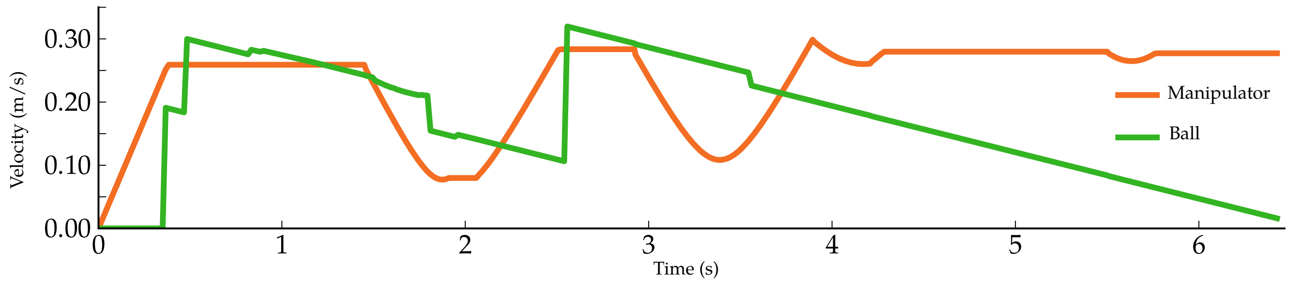
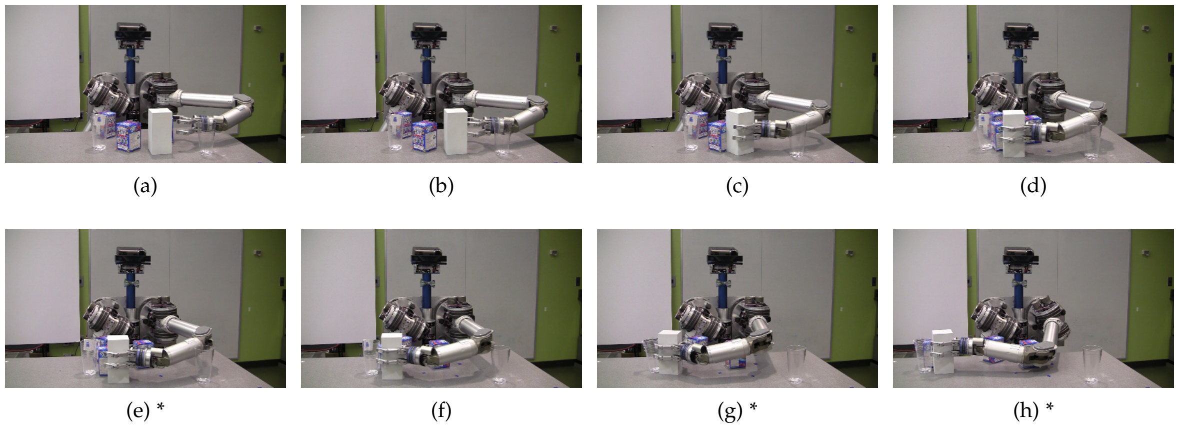
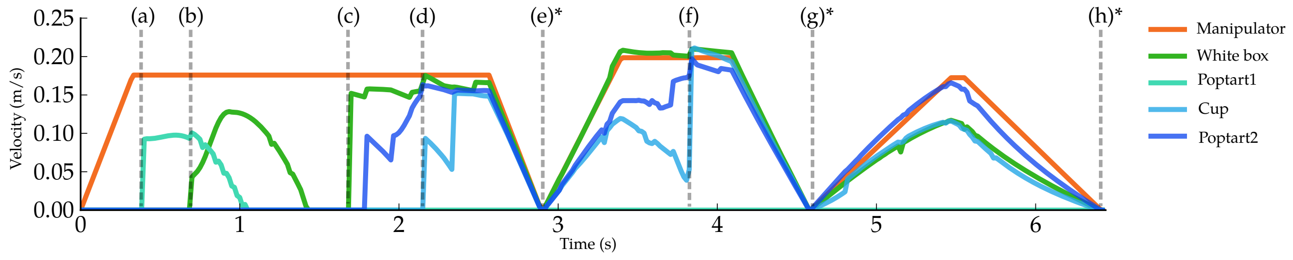


Figure 5.8: *Top:* Snapshots from a semi-dynamic trajectory lifted to a 7-DOF arm trajectory and executed on the robot in simulation ($T_{max} = 8s$). The task is to move the green ball into the red goal. The orange boxes are static obstacles. The states marked with $*$ are statically stable. *Middle:* Velocity profile of the end-effector and the target. Note how the robot waits until the ball comes to rest before it performs the next action. *Bottom:* Velocities for a dynamic trajectory for the same scene. In contrast to the semi-dynamic trajectory the environment does not come to rest after each action.

action using the other side of the end-effector. We observed this behavior frequently in many scenes. An action where the robot pushes a ball into open space is less likely to lead to a statically stable state within T_{max} than an action for which caging occurs.

Fig.5.8-*bottom* shows the velocity profile for a trajectory planned with the dynamic planner on the same scene. Note how both the manipulator and the target object keep in motion for the whole trajectory. As a result the total duration of the trajectory is shorter than the semi-dynamic one.





(i)

Figure 5.9: *Top:* Snapshots from a semi-dynamic trajectory lifted to a 7-DOF arm trajectory and executed on HERB ($T_{max} = 8s$). The task is to move the white box to the left. *Bottom:* Velocity profile for the trajectory as planned. Multiple objects are pushed simultaneously. Also the target object, the cup and one of the boxes are moved multiple times. The statically stable states are marked with $*$.

5.5 *Summary and Discussion*

In this chapter, we offered a method for moving beyond quasistatic interactions and planning for dynamic interactions. We proposed to limit the scope of the planner to consider only dynamic interactions that result in statically stable states – all objects come to rest between robot motions. This allowed us to keep the search in the joint configuration space of the robot and objects. We showed this speeds the search at the expense of limiting the types of solutions the planner can generate.

In our formulation, we require *all* objects come to rest between actions. We note that this constraint may be too restrictive, particularly when planning for highly dynamic objects that don't easily come to rest. We could instead employ a hybrid approach that incorporates the full state of some objects, i.e. the rolling ball. This would allow the planner to track velocities of these objects and remove the need for these objects to come to rest between actions. This increases the size of the state space but also increases the space of possible solutions.

The implementation in Sec.5.4 plans for the robot's end-effector in 2D and then lifts the resulting trajectory to the full configuration space of the robot. This method allows us to plan in 2D and use a fast physics model. However, it introduces two main limitations. First, by planning in 2D we are unable to take into account the kinematics of the arm at plan time. As a result, the planner can produce end-effector trajectories that are kinematically infeasible. When these trajectories are encountered, they are discarded and the planner is restarted.

Second, any contact between the arm and objects is not modeled. Again, this can lead the planner to produce infeasible trajectories, i.e. there exist no collision-free arm motions that realize the end-effector motion. In the previous chapters, we have shown that allowing contact between the full robot and objects proves beneficial to the planner by expanding the space of solutions that can be achieved. In order to allow these trajectories, we must substitute a full 3D physics model. This will result in overall slower tree extension, but at the added benefit of an expanded solution set.

Despite these limitations, our preliminary experiments show we are able to produce executable trajectories on a real robot (Fig.5.9). While we are able to achieve some success executing these trajectories open-loop, the dynamic interactions highlight the need to incorporate uncertainty into our planner. It is impossible to perfectly model the dynamic interactions, especially with low friction objects like the rolling ball. As a result, many of our executions of scenes with such objects resulted in failure. In Part II, we will present methods for incorporating the uncertainties prevalent in our real physical systems.

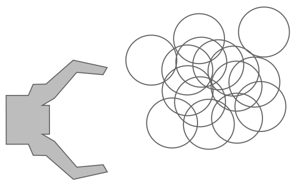
Part II

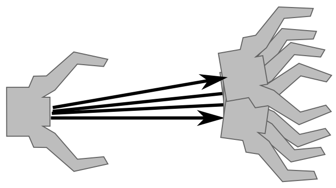
Rearrangement Planning under Uncertainty

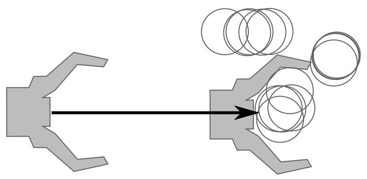
6
Planning Under Uncertainty

The algorithm presented in Ch.3- Ch.5 planned assuming perfect knowledge of the environment and perfect action execution. As a result, while the plans generated by Alg.1 are feasible, they are often prone to failure due to uncertainty. In general, open-loop plans are susceptible to failure due to uncertainty in the initial pose of objects in the scene (Fig.6.1a), and uncertainty in the motion of the manipulator (Fig.6.1b). Alg.1 introduces a third source of uncertainty: uncertainty in the evolution of the physical interaction (Fig.6.1c). This could be due to poor selection of parameters for the model, noisy contacts or errors in the geometric representation of the scene. In the next chapters, we formulate methods for characterizing and handling this uncertainty in order to produce robust open-loop trajectories.

Planning under these uncertainties has been studied extensively in the robotics community. In this chapter, we provide a brief description of the most widely adopted techniques and how they relate to the rearrangement planning problem.


(a) Uncertainty in initial object pose


(b) Uncertainty in action execution


(c) Uncertainty in physics parameters
Figure 6.1: Three sources of uncertainty in the system. (a) and (b) are prevalent in many planning domains. Our framework introduces a third source of uncertainty: uncertainty in the physics model (c)
.

6.1 Conformant Planning

The basic assumption of a *conformant planning* problem is that the system cannot be observed while the plan is being executed, and the state of the system is not known for certain even at the beginning of planning. The goal of conformant planning is to generate a sequence of actions that is guaranteed to achieve the goal despite any uncertainty in the system [32, 44, 114]. The problem is considerably more difficult that traditional planning, even under simplifying assumptions such as bounding the length of plans [50].

A less restrictive framework is the *probabilistic conformant planning* problem [57, 75]. Here, the goal is to generate a sequence of actions that achieves the goal with sufficient probability. Framing the problem under this less restrictive condition opens the space of solutions and eases the overall planning problem. We frame the rearrangement planning problem as an instance of probabilistic conformant planning in Ch.7.

6.2 Sampling Based Approaches

Some works have proposed methods that extend sampling based frameworks to solve the probabilistic planning problem. One of the most common approaches is to build on the Probabilistic Roadmap (PRM) algorithm [62] by accounting for stochasticity in robot pose [106], transition dynamics [10] and obstacle representation [47]. While the use of a PRM is infeasible for this problem, we borrow some key inspirations from these works. In particular, in Ch.8 we propose to use Monte Carlo simulations to estimate the probability a sequence of actions achieves success. This same method is employed to estimate success of individual actions in [10].

Alternate works have built uncertainty into RRTs [83, 93, 101] under assumptions such as bounded uncertainty. Indeed, we could modify our algorithms from Part I so that each node represents a set of possible states, or belief state, rather than a single known state. The difficulty arises in defining an informative distance metric between two belief states that allows us to effectively guide tree growth [80]. We provide two methods of incorporating uncertainty into RRTs in Ch.9.

6.3 Partially Observable Markov Decision Processes

An alternative and common method of handling planning under uncertainty is to formulate the problem as a Partially Observable Markov Decision Process (POMDP). POMDP solvers can reason about uncertainty and incorporate closed-loop feedback from local or global sensors. We refer the reader to one of several good surveys of the formulation, theory and algorithms [27, 82, 95, 124].

We are interested in a sub-domain of POMDP problems: Unobservable Markov Decision Process (UMDP). In these problems, the robot does not receive observations that help determine state. While the lack of observations reduces the complexity, the problem is still difficult to solve exactly, particularly in our continuous state and action space with non-Gaussian non-smooth distributions. In this work we consider online solvers that find approximate solutions [13, 74, 111, 121]. We use an algorithm inspired by Partially Observable Monte Carlo Planning (POMCP) [111] which relies on Monte Carlo simulations to estimate our difficult distributions in Ch.10.

6.4 *Rearrangement Planning under Uncertainty*

A few works have considered uncertainty in planning among movable objects. Levihn et al. [78] formulate the problem as a Markov Decision Problem and uses the Box2D physics simulator to estimate transition probabilities between states. The proposed planner uses a hierarchical approach, using the MDP to plan high-level actions that connect regions of free state space by moving objects, then using a state space planner to generate geometrically feasible instantiations of these actions. An alternative backtracking-based approach is used to solve rearrangement planning under uncertainty in [36].

Neither of these approaches can be applied directly to our problem without restricting the space of solutions our planner can consider. The complexity of tracking free space [78] is difficult for the high-dimensional robot state spaces we consider. The backtracking approach [36] limits object interaction to a single object at a time. However, we draw inspiration from both of these approaches in the solutions we present in the following sections.

7 Incorporating Uncertainty into Rearrangement Planning

Formally, we will augment the problem described in Ch.2. Rather than planning from a deterministic start state $x_0 \in X_{free}$ we instead begin in an initial *belief state* $b_0 = p(x_0)$ that is a probability distribution over the start state. Additionally we assume our state evolves as a stochastic non-holonomic system. Thus we have a distribution $p(\xi|\pi)$ of trajectories that results from starting in $x_0 \sim b_0$ and executing control inputs π under the stochastic transition dynamics. Our goal is to find a sequence of control inputs π that maximizes the probability $p_\pi = \Pr[\Lambda[\xi] = 1]$ of satisfying the *success functional* $\Lambda : \Xi \to \{0, 1\}$ where Ξ is the set of all trajectories. We define the success functional

$$\Lambda[\xi] = \begin{cases} 1 & : \xi(T_\xi) \in X_G \\ 0 & : \text{otherwise} \end{cases} \tag{7.1}$$

where T_ξ is the duration of trajectory ξ.

This problem is particularly hard for rearrangement planning. First, the problem is set in a high-dimensional space with continuous controls. Second, contact causes physics to evolve in complex, non-linear ways and quickly leads to multimodal and non-smooth distributions [68, 70, 104]. Third, finding good trajectories is inherently hard: most trajectories achieve success with zero probability.

In the following chapters, we outline methods for handling uncertainty under the challenges presented by rearrangement planning.

8
Trajectory Selection for Rearrangement Planning

This chapter is adapted from Koval et al. [69] and contains work completed in collaboration with Michael Koval.

In this chapter we formulate rearrangement planning under uncertainty as a *selection problem*. To do this we take advantage of three characteristics of the planning framework presented in Part I: (1) the method for generating trajectories is stochastic – several calls to the planner for the same query will result in different trajectories, (2) the use of a physics model allows us to easily forward-simulate the system's dynamics under different initial conditions, and (3) we can test whether a simulation is successful, i.e. achieves the goal.

Given these properties we first use the stochastic planner to generate a finite set of state space trajectory *candidates*. We then perform several noisy *rollouts*: forward-simulations under varying initial conditions. We use the success or failure of these rollouts to form an estimate of p_π, the probability of the trajectory being successfully executed on the real robot, and use this estimate to select the most robust trajectory.

Given a set of k trajectories, we could perform a fixed sufficient number n of rollouts on each trajectory, requiring a total of kn rollouts. However, rollouts are computationally expensive, requiring the evaluation of a full physics simulation. We seek an algorithm that can *efficiently* choose the best trajectory given a small rollout budget.

We can formalize this selection problem as an instance of the "best arm" variant [90] of the k-armed bandit problem [107]. In our formulation, each candidate trajectory is an "arm" and the goal is to identify the best arm given a fixed budget of rollouts. We use the *successive rejects* algorithm [11] to select the best candidate. In the following sections we detail important components of the method.

8.1 Trajectory Selection

Our goal is to find the most robust sequence of control inputs:

$$\pi_{best} = \operatorname*{argmax}_{\pi \in \Pi} p_\pi \qquad (8.1)$$

where Π is the set of all possible sequences of control inputs. Compute π_{best} exactly is not possible. Instead, we propose an approximate solution and provide intuition about the accuracy of the approximation.

8.1.1 *Approximating success probability*

We can write p_π as the expectation:

$$p_\pi = E\left[\Lambda[\xi]\right] = \int_\Xi \Lambda[\xi] p(\xi|\pi) d\xi$$

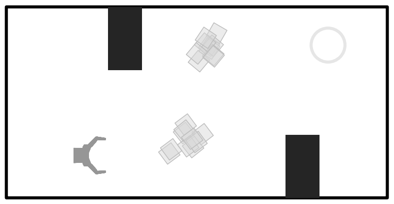
over the space of trajectories Ξ. Directly computing this integral requires the ability to evaluate $p(\xi|\pi)$. In the case of rearrangement planning, this is not available.

Instead, we approximate this expectation as the mean

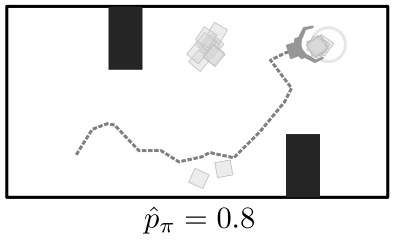
$$\hat{p}_\pi = \frac{1}{n} \sum_{j=1}^{n} \Lambda[\xi_j]$$

of Λ over n rollouts $\xi_1, \ldots, \xi_n \in \Xi$. The law of large numbers guarantees that $\lim_{n\to\infty} \hat{p}_\pi = p_\pi$: i.e. our approximation $\hat{p}_\pi$ approaches the true success probability p_π as the number of samples n increases.

Each rollout is an independent sample from the distribution $\xi_j \sim p(\xi|\pi)$. We generate rollout ξ_j by sampling an initial state $x_j \sim p(x_s)$, then forward-propagating x_j through stochastic dynamics while executing the control inputs dictated by π. Fig.8.1 shows an example of using rollouts to compute $\hat{p}_\pi$ when there is uncertainty in the initial pose of the objects and the physics models, but no uncertainty in action execution.



(a) Uncertainty in initial pose of objects in the scene



(b) Rollouts of each of these noisy initial states

Figure 8.1: Rollouts of the control sequence can be used to estimate probability of success under uncertainty.

8.1.2 *Trajectory selection*

Solving Eq.(8.1) also requires finding the global optimum of the infinite-dimensional set of sequences of control inputs Π. To gain tractability, we first *generate* a candidate set of control sequences $\Pi_{\text{can}} \subseteq \Pi$ and then *select* the most robust candidate

$$\pi^* = \underset{\pi \in \Pi_{\text{can}}}{\text{argmax}}\, p_\pi$$

from this finite set.

We populate Π_{can} by drawing samples from a distribution over Π. In our rearrangement planning problem, we generate candidates by repeatedly calling one of the planners from Part I. Our intuition is that the RRT will generate a diverse set of candidates that achieve the goal with varying success probabilities.

Given Π_{can}, the simplest selection algorithm is to choose

$$\hat{\pi}^* = \operatorname*{argmax}_{\pi \in \Pi_{\text{can}}} \hat{p}_\pi$$

using our approximation $\hat{p}_\pi$ of p_π. To do this, for each candidate π_i we perform n rollouts and count the number of success, s_i. Then $\hat{p}_{\pi_i} = s_i / n$.

8.1.3 Effect of approximation error

Approximating p_π with $\hat{p}_\pi$ comes at a cost: we may incorrectly select a sub-optimal candidate $\hat{\pi}^* \neq \pi^*$ due to error. An error occurs when a candidate π_i performs well on the n rollouts used to estimate p_{π_i}, but performs poorly on the underlying distribution. This phenomenon parallels the concept of *overfitting*.

The number of successful rollouts s_i is a Binomial random variable. The magnitude of the error $\|\hat{p}_\pi - p_\pi\|$ is unbounded for any finite number n of rollouts. However, we can use a Binomial confidence interval to bound the probability

$$Pr(\|\hat{p}_\pi - p_\pi\| > \delta) < \alpha \tag{8.2}$$

of an error with magnitude δ occurring. When the central limit theorem holds, the Wald interval states

$$\delta = z_{1-\alpha/2} \sqrt{\frac{1}{n} \hat{p}_\pi (1 - \hat{p}_\pi)} \tag{8.3}$$

where $z_{1-\alpha/2} = \Phi^{-1}(1 - \alpha/2)$ is the $(1 - \alpha/2)$-th percentile of the Gaussian distribution.

Given a desired δ and α, we can solve for the number of samples n required to satisfy Eq.(8.2). The value δ is related to the minimum difference between two trajectories that we can reliably detect. Ideally we would drive $\delta \to 0$, allowing us to differentiate trajectories with similar success rates. From Eq.(8.3) we see this requires a prohibitively large value of n; e.g. reducing δ by half requires increasing n by a factor of four.

8.2 Multi-Armed Bandit Formulation

The approach described in Sec.8.1 assumes that we need to perform the *same number of rollouts on all candidates*. Our analysis in Sec.8.1.3 suggests that this is wasteful; we can use fewer samples to differentiate between two candidates that have vastly different success probabilities.

We formalize the intuition by framing the trajectory selection problem as a variant of the *multi-armed bandit problem* [107]. This

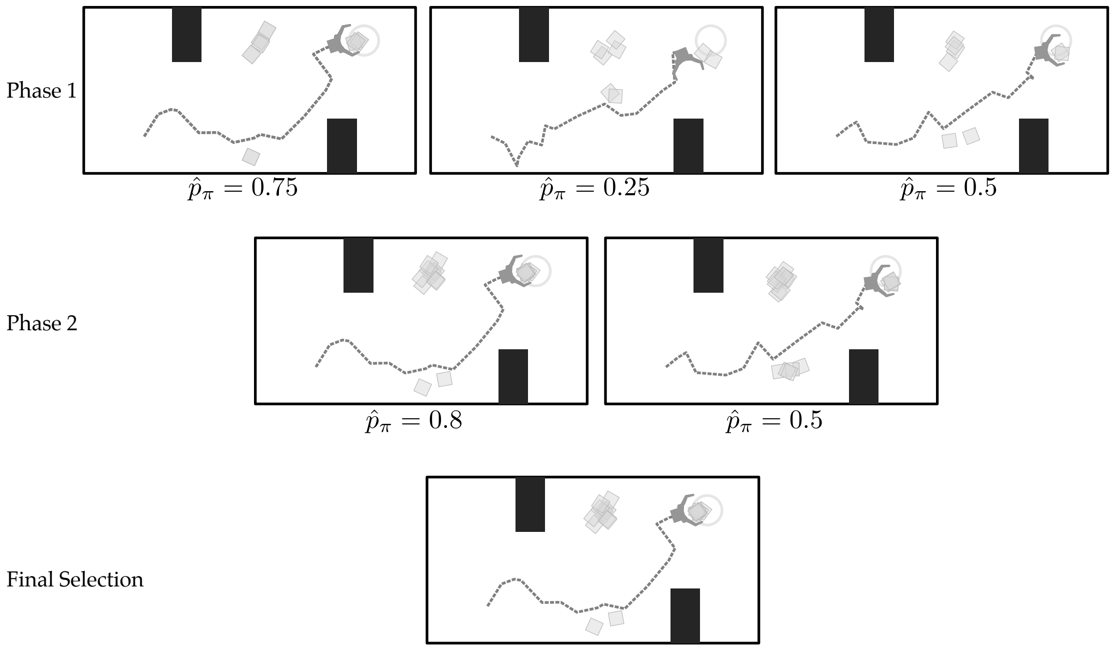


Figure 8.2: The successive rejects algorithm. During each phase, rollouts are performed on all remaining candidates to improve estimates of $\hat{p}$ for each candidate. After each phase, the candidate with lowest probability of success is eliminated. This continues until a single candidate remains in the set.

enables us to use the *successive rejects* algorithm [11] to efficiently identify the best candidate.

8.2.1 Multi-armed bandit formulation

A *multi-armed bandit problem* is a sequential process where an agent is presented with k arms and at each time step must choose only one arm to pull. After pulling an arm, the agent receives a reward. The goal of the agent is to maximize expected sum of reward. Since the agent does not know the distribution of rewards, it must trade off between *exploring* different arms and *exploiting* its estimate of the best arm.

Trajectory selection is an instance of the multi-armed bandit algorithm where each candidate $\pi_i \in \Pi_{\text{can}}$ is an arm. Pulling the i-th arm corresponds to performing one rollout, ξ_j of π_i. We receive a binary reward $\Lambda[\xi_j] \sim \text{Bernoulli}[p_{\pi_i}]$ depending upon whether the rollout, $\xi_j \sim p(\xi|\pi_i)$, achieves the goal.

Unlike the canonical bandit problem, the goal of trajectory selection is not to maximize the expected sum of reward across all rollouts. Instead, after exhausting a budget of B rollouts, we choose the single best candidate $\hat{\pi}^* = \text{argmax}_{\pi \in \Pi_{\text{can}}} \hat{p}_\pi$ and execute $\hat{\pi}^*$ on the real robot. Our goal is to optimally allocate the B rollouts among the k candidates to maximize the success probability $p_{\hat{\pi}^*}$ of our selection $\hat{\pi}^*$. This is known as the *best arm* or *pure exploration* variant of the bandit problem [11, 90].

Algorithm 7 Successive Rejects

```
Require: candidates $\Pi_{\text{can}} = \{\pi_1, \ldots, \pi_k\}$, initial belief $b_0$
 1: $A = \{1, \ldots, k\}$
 2: $s_i \leftarrow 0$ for all $i \in A$
 3: $n_0 \leftarrow 0$
 4: for $l = 1, \ldots, k-1$ do
 5:     $n \leftarrow n_l - n_{l-1}$                    ▷ See Eq.(8.4) for def'n
 6:     for all $i \in A$ do
 7:         for $j = 1, \ldots, n$ do                  ▷ Perform $n$ rollouts
 8:             $x_j \sim b_0$
 9:             $\xi_j \leftarrow \text{Rollout}(x_j, \pi_i)$
10:             $s_i \leftarrow s_i + \Lambda[\xi_j]$
11:     $i_{\text{worst}} \leftarrow \text{argmin}_{i \in A}(s_i / n_l)$
12:     $A \leftarrow A \setminus \{i_{\text{worst}}\}$           ▷ Reject the worst $\hat{p}_\pi$
13: $\{\hat{\pi}^*\} \leftarrow A$
```

8.2.2 *Successive rejects algorithm*

The *successive rejects* algorithm (Alg.7) is a principled method of solving the best arm problem. The intuition behind the algorithm is to partition the B rollouts between several phases (Line 4). A set $A \subseteq \Pi_{\text{can}}$ is repeatedly shrunk until it contains the single best candidate. In each phase, we perform an equal number of rollouts n (Line 5) on each remaining candidate $\pi \in A$ and remove the candidate $\pi_{worst} = \text{argmin}_{\pi \in A} \hat{p}_\pi$ with the lowest estimated success probability from A (Line 12). This repeats until we have completed $k-1$ phases. At this point, we return the remaining candidate $\hat{\pi}^*$ in A as our selection (Line 13). Fig.8.2 illustrates the algorithm.

The key component of the successive rejects algorithm is how to select the number of rollouts to perform in phase l. If we have k candidates and a total budget B of rollouts, then we choose

$$n_l = \left\lceil \frac{1}{\overline{\log}\ k} \cdot \frac{B-k}{k+1-l} \right\rceil \tag{8.4}$$

where $\overline{\log}\ k = 1/2 + \sum_{i=2}^{k} 1/i$ and $\lceil \cdot \rceil$ denotes the ceiling operator [11]. n_l is the *total number* of rollouts performed across all phases on each candidate remaining in phase l. Only $n = n_l - n_{l-1}$ of these rollouts are performed in phase l.

Given the choice of Eq.(8.4) for n_l, prior work [11] shows that the probability ϵ of Alg.7 making an error is bounded by:

$$\epsilon \leq \frac{k(k-1)}{2} exp \left[-\frac{b-k}{H_2 \overline{\log}\ k} \right]$$

where $H_2 = max_{2 \leq i \leq k}(i\Delta_{(i)}^{-2})$ and $\Delta_{(i)} = p_{\pi^*} - p_{\pi_{(i)}}$ is the gap between the best candidate and the i-th best candidate $\pi_{(i)}$. We can additionally bound $H_2 \leq H_1$ with $H_1 = \sum_{i=2}^{k} \Delta_{(i)}^{-2}$.

The quantities H_1 and H_2 formalize the difficulty of the problem. Since $\Delta_{(i)}$ is the denominator of H_1, a problem is more difficult if the gaps $\Delta_{(1)}, \ldots, \Delta_{(k)}$ are small. This confirms our analysis from Sec.8.1.3 that it is difficult to differentiate between two candidates with similar success probabilities.

8.3 *Experiments and Results*

First, we verify the following two properties hold in our test environment:

P.1 The state space planner can generate candidate trajectories with varying success probabilities.

P.2 Increasing the number of rollouts per trajectory improves the estimated success probability, allowing us to make a better selection.

Finally, we test two hypotheses:

H.1 The successive rejects algorithm requires fewer rollouts to find the best trajectory than a baseline that uses a fixed number of rollouts.

H.2 Selecting a good trajectory increases the likelihood of successful execution on a real robot.

We evaluate our algorithm on the trajectories generated for the HERB robot tests in Sec.4.3.3 using the hybrid planner with $p = 0.5$. In these tests, HERB was tasked with pushing an object on the table through clutter.

8.3.1 *Robustness to uncertainty*

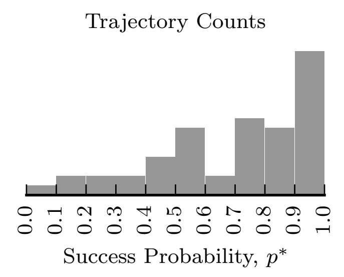


Figure 8.3: Histogram of the success probabilities achieved by 50 candidate control sequences that solve the scene in Fig.8.4.

We test **P.1** by evaluating the 50 candidate trajectories Π_{can} generated for one of the HERB scenes (Fig.8.4). We execute 400 noisy rollouts of each candidate $\pi_i \in \Pi_{\text{can}}$ and count the number of rollouts s_i that achieve the goal to compute $\hat{p}_{\pi_i} = s_i/400$. Using $n = 400$ rollouts gives us 95% confidence that our estimate $\hat{p}_{\pi_i}$ is withing 5% of the true success probability p_{π_i}.

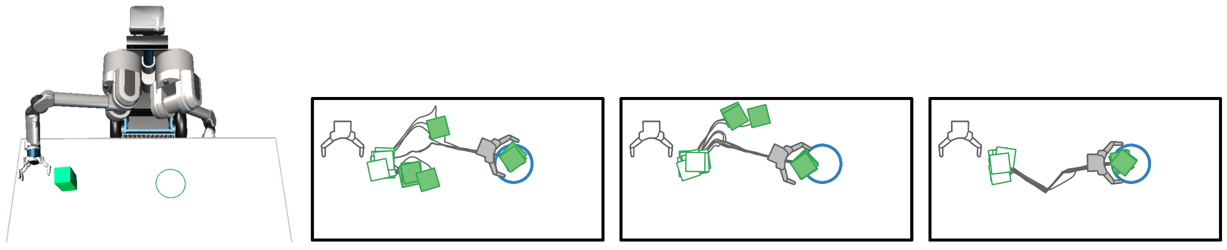

(a) Original scene (b) $\hat{p} = 0.28$ (c) $\hat{p} = 0.59$ (d) $\hat{p} = 1.0$

Figure 8.4: Start poses for the object in the scene are drawn from a Gaussian with distribution $\mu = \mathbf{0}, \Sigma^{1/2} = \mathtt{diag}\{2\mathtt{cm}, 2\mathtt{cm}, 0.1\mathtt{rad}\}$. The robot must push the box into the goal region. For simplicity, only the robot hand is show, though contact is allowed with the full robot arm.

Fig.8.3 shows the distribution of success probabilities for Π_{can}. Each of 400 noisy rollouts samples the initial pose of the object from a Gaussian distribution with zero mean and $\Sigma^{1/2} = \mathtt{diag}\{2\mathtt{cm}, 2\mathtt{cm}, 0.1\mathtt{rad}\}$.

These results show that we can easily generate different trajectories with our state space planner. More importantly , this confirms that the trajectories produced by our planner differ in robustness to uncertainty.

8.3.2 *Fixed rollout method*

Nest, we verify **P.2** with a baseline selection algorithm that uses a fixed number n of rollouts to evaluate each trajectory. We compare multiple values of $n \in \{15, 50, 150, 400\}$ using the same set of candidate trajectories as from Sec.8.3.1. We use the calculated $\hat{p}_{\pi_i}$ values as the ground truth success probabilities p^*_π for each trajectory. We then discard the previous rollouts and generate 400 new rollouts to test the selection algorithm.

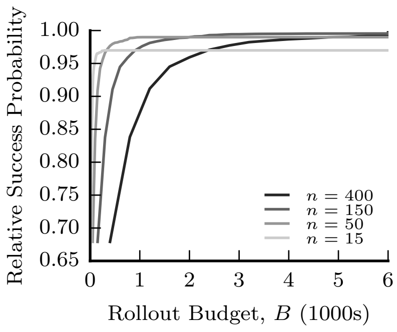


Figure 8.5: Achieved success probability as a function of budget for the scene in Fig.8.4.

Fig.8.5 shows the ground truth success probability p^* of the trajectory selected with a budget of B total rollouts. Results are averaged across 300 trials. With a small rollout budget, small values of n find trajectories with higher success probability. This is expected, as these planners are able to evaluate more trajectories using B rollouts. For example, with a budget of $B = 800$ rollouts, the $n = 400$ planner can only evaluate two trajectories while the $n = 15$ planner can evaluate all 50.

Large values of n find better solutions as B increases. This is also expected. Increasing n shrinks the confidence interval and allows the planner to accurately differentiate between trajectories with similar success probabilities.

8.3.3 *Success rejects algorithm*

We test **H.1** by comparing the successive rejects algorithm described in Sec.8.2.2 against a baseline algorithm that uses a fixed number n of rollouts. We begin by allocating a budget of $B = 800$ rollouts to each algorithm. We run multiple iterations, increasing B by 400 rollouts each time until we reach $B = 20000$. At this point, the $n = 400$ algorithm can evaluate all 50 candidates. We record the ground-truth success probability p^* of the trajectory selected in each iteration.

Fig.8.6 shows the relative success probability of each selection algorithm averaged over 300 trials. In each trial, we randomize the order of Π_{can} and the outcome of the noisy rollouts for each candidate $\pi_i \in \Pi_{\text{can}}$. The ordering of both trajectories and the outcomes is kept constant within a trial.

For a fixed budget of 20000, both planners find the same near-optimal trajectory. However, the successive rejects algorithm finds the trajectory with far fewer rollouts on average. This supports our hypothesis: **The successive rejects algorithm requires fewer rollouts to find the best trajectory than a baseline planner that uses a fixed number of rollouts.**

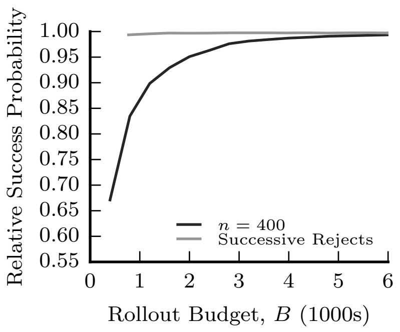


Figure 8.6: The successive rejects algorithm finds better solutions than an algorithm that applies a fixed number of rollouts to every trajectory given the same budget B.

8.3.4 *Real robot experiments*

We test **H.2** by executing trajectories selected by our algorithm on a real robot. We generate four new random scenes for HERB, construct each by measuring the nominal location of each object relative to HERB, and perturb the pose of each object by an offset drawn from a Gaussian distribution. We use the successive rejects selection algorithm to generate five trajectories for each of the four scenes and record the success rate estimated by the planner for each candidate trajectory. Finally, we select seven of the twenty generated trajectories with varying estimated success rates and execute each 10 times on HERB. We record success or failure of each execution.

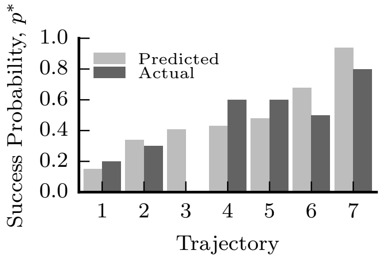


Figure 8.7: Comparison of the success probability estimated through Monte Carlo rollouts (predicted) and observed during ten executions on the real robot (actual). Trajectories with a high estimated probability of success tend to succeed more often when executed on the real robot.

Fig.8.7 shows the estimated and actual success rate of each trajectory. The estimated success rate does not perfectly predict the results that we see on the real robot. However, there is a clear correlation between the estimated success rate and the probability that executing the trajectory succeeds on the real robot. This supports our hypothesis: **Selecting a good trajectory increases the likelihood of successful execution on a real robot.**

The qualitative aspects of the real robot experiments are perhaps more interesting. Fig.8.8 shows *Trajectory 7*. In this scene, the robot is tasked with moving the white box into the circle. This trajectory exhibits the highest success rate. As can be seen, a sweeping motion is used to move the box to the goal. Prior work [36] shows that sweeping primitives are particularly effective at reconfiguring objects under uncertainty. It is encouraging that our selection algorithm can produce similar behavior.

Next, we examine *Trajectory 3*. Our planner estimated the success rate of this trajectory to be $\hat{p} = 0.41$. However, we were unable to

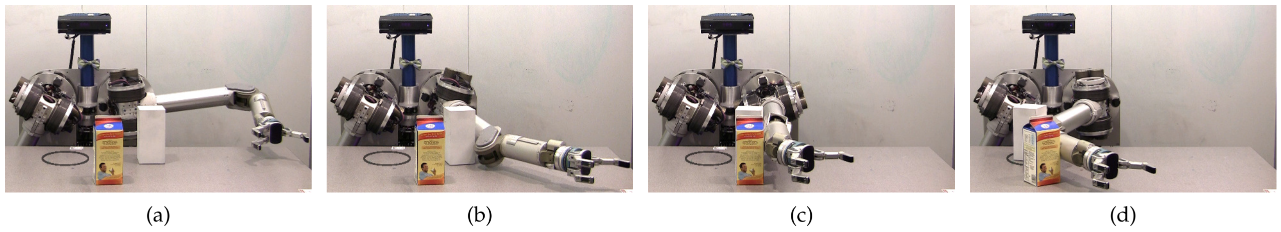


Figure 8.8: Trajectory 7, from the results presented in Fig.8.7, executed on HERB. The trajectory selection algorithm chose to use a sweeping motion to robustly push the white box into the circular goal region.

achieve any successful executions on the real robot. Examining the rollouts used to evaluate this trajectory reveals unmodeled errors in the physics model. This highlights a fundamental limitation of our formulation: our estimate of $\hat{p}_\pi$ can only be as good as our model of noise in the system dynamics.

8.3.5 Modeling control errors

Our HERB experiments accounted for errors in the initial pose of the objects in the scene. These errors are the most prevalent source of failure when executing trajectories on HERB (though our real robot experiments pointed out other physics modeling errors do exist). When executing our KRex experiments in Ch.4 we noted that the method of controlling KRex was much more prevalent to inaccuracies than the control method for HERB. The trajectory selection method is able to account for these uncertainties.

To test this, we generate 400 noisy rollouts of the trajectories from the KRex clearance experiments from Ch.4. Like the HERB experiments, we sample the initial pose of objects from a Gaussian distribution with zero mean and $\Sigma^{1/2} = \texttt{diag}\{\texttt{2cm}, \texttt{2cm}, \texttt{0.1rad}\}$. However, we also sample the initial pose of KRex from a Gaussian distribution with zero mean and $\Sigma^{1/2} = \texttt{diag}\{\texttt{2cm}, \texttt{2cm}, \texttt{0.0rad}\}$ to reflect the initial pose errors prevalent when executing on KRex. Finally, we sample noise into the duration each control is executed from a Gaussian with zero mean and $\sigma = 0.2\,\mathrm{s}$ to reflect the noisy control.

Fig.8.9 shows the distribution of Π_{can} for this task. This distribution contains far fewer candidates that succeed with high probability. This is due to the nature of the modeled uncertainty: the noise in the control execution grows unbounded (Fig.8.11).

Fig.8.10 uses the same method from Sec.8.3.3 to compare the performance of the successive rejects algorithm vs. using a fixed $n = 400$ rollouts per candidate. Here the performance improvements are more drastic than the HERB case. The successive rejects algorithm finds the best trajectory using less than 4000 rollouts on average while the fixed rollout algorithm requires an average of over 16000 rollouts to select the best.

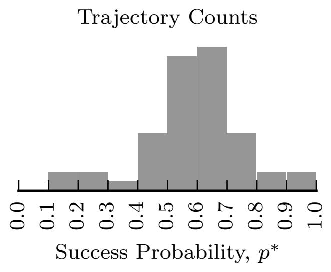


Figure 8.9: Histogram of the success probabilities achieved by 50 candidate control sequences that solve the scene in Fig.8.11.

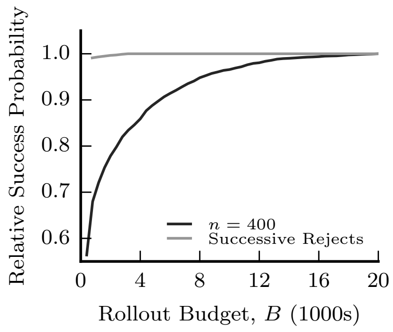


Figure 8.10: The successive rejects algorithm finds better solutions than an algorithm that applies a fixed number of rollouts to every trajectory given the same budget B.

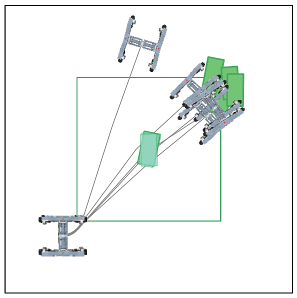
8.4 *Summary and Discussion*

Our results show that selecting the best trajectory from a set of candidates is a surprisingly effective method of improving the likelihood of the plan executing successfully. Additionally, we show that using the success rejects algorithm dramatically reduces the number of rollouts required to achieve a desired level of performance. This algorithm is simple to implement and performs strictly better than using a fixed number of rollouts.

The performance of this algorithm depends entirely on the quality of the trajectories included in Π_{can}. First, the successive rejects algorithm is most beneficial when few candidates achieve the goal with high probability, as is the case in the KRex experiments from Sec.8.3.5. Second, the output trajectory can only be as good as the best trajectory in Π_{can}. If all trajectories in the set are of poor quality, the selected trajectory will be brittle.

We may be able to help this issue by seamlessly trading off between evaluating existing candidates and expanding Π_{can} to possibly include better candidates. This is a *smooth bandit problem* [66] defined over the continuous set of control sequences Π. The key challenge is to define a meaningful metric over Π and insure that Λ is sufficiently smooth. This formulation may result in an *anytime* planner that continually outputs better trajectories over time.

Ideally, we would incorporate uncertainty directly into the planner. Prior work [34] has show that some pushing actions reduce uncertainty, while others increase it. Currently, we rely on such actions being randomly generated by a state space planner. In Ch.9 and Ch.10, we provide two methods for incorporating uncertainty at plan time, allowing our planner to explicitly include such actions.


(a) $\hat{p}_\pi = 0.8$

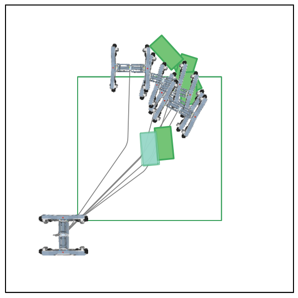

(b) $\hat{p}_\pi = 0.4$

Figure 8.11: Success rate for two trajectories for the same clearance task. The top trajectory is more robust to the control uncertainties prevalent for KRex.

9 Convergent Rearrangement Planning

This chapter is adapted from Johnson et al. [59] and contains work completed in collaboration with Dr. Aaron Johnson.

Framing rearrangement planning under uncertainty as a trajectory selection problem imposes one fundamental limitation: we cannot optimize for robustness at plan time.

Prior work [24, 35] has shown that nonprehensile interactions such as pushing can be inherently uncertainty reducing (Fig.9.1). In this chapter, we formulate a method for characterizing the robustness of an action or set of actions inspired by results from *contraction analysis* [81]. We propose three *divergence metrics*, partially based on this analysis, and use these metrics to identify and select uncertainty reducing actions at plan time. We show this allows us to actively *generate* robust trajectories.

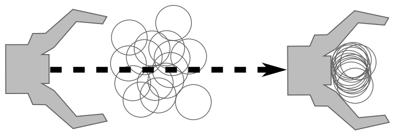


Figure 9.1: An example pushing action that reduces uncertainty in object pose

9.1 Contraction Analysis and Divergence Metrics

Contraction analysis [81] provides a proof of global exponential convergence for a controller over a contraction region (a subset of the configuration space where all states will converge to a single trajectory). In this section we review the main results from contraction analysis [81]. We then define *divergence metrics* partially based on this analysis, as well as corresponding numerical approximations and path metrics. These metrics provide a way to quantify the convergence of a path as well as guide the search for a convergent plan.

9.1.1 Contraction analysis

Consider our system with state $x \in X$, control input $u \in \mathcal{U}$ and (possibly time-varying) vector field, $f : X \times \mathcal{U} \times \mathbb{R}^{\geq 0}$. Define F as the symmetric part of the Jacobian of f, i.e.,

$$F(x,u,t) := \frac{1}{2}\left(\frac{\partial f(x,u,t)}{\partial x} + \frac{\partial f(x,u,t)}{\partial x}^T\right) \tag{9.1}$$

Let $\delta\xi(t)$ define the virtual displacement (an infinitesimal displacement at a fixed time t) in a trajectory formed in f. This virtual displacement is bounded by the magnitude of the initial displacement, $\delta\xi(t_0)$ and the integral of $\lambda_{max}(x, u, t)$, the maximum eigenvalue of F at ξ at time t, [81, Eqn. 3],

We use the notation $x = \xi(\tau)$ and $u = \pi(\tau)$ where ξ is a solution to a vector field $\dot{\xi}(t) = f(\xi(t), \pi(t), t)$ under control π.

$$\|\delta\xi(t)\| \leq \|\delta\xi(t_0)\| e^{\int_{t_0}^{t} \lambda_{max}(x,u,\tau)d\tau}. \tag{9.2}$$

Define the maximal divergence metric $D_m := \lambda_{max}$. In particular if D_m (and therefore also F) is uniformly negative definite everywhere in a region around a nominal trajectory, any differential length at the start of a trajectory will vanish exponentially along its length, [81, Thm. 1],

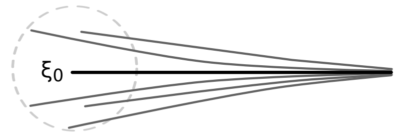


Figure 9.2: Theor.9.1.1: Any trajectory within a radius r of a nominal ξ_0 will converge to ξ_0 if $D_m < 0$.

Theorem 9.1.1. *Given a nominal trajectory, ξ_0, that is the solution to a vector field, $\dot{\xi}_0(t) = f(\xi_0(t), \pi_0(t), t)$, under control π_0, any other trajectory that begins within region defined by a ball of radius r around the nominal trajectory will converge exponentially to that trajectory so long as F, Eq.(9.1), is uniformly negative definite over that region, i.e. if,*

$$\exists\beta > 0, \forall t \geq t_0, x \in R(t), \qquad D_m(x, \pi_0(t), t) \leq -\beta < 0,$$

where $R(t) := \{x : \|x - \xi_0(t)\| < r\}$. By bounding D_m we conclude that all neighboring trajectories converge to a single trajectory (Fig.9.2).

Consider now the evolution of a differential volume, δV, around the trajectory,

$$\|\delta V(t)\| = \|\delta V(t_0)\| e^{\int_{t_0}^{t} \operatorname{div} f(x,u,\tau)d\tau}. \tag{9.3}$$

Define the average divergence metric, $D_a := \operatorname{div} f$. As a relaxation of Theor.9.1.1, consider [81, Sec. 3.9],

Theorem 9.1.2. *Given a nominal trajectory, $\xi_0(t)$, that is the solution to a vector field, $\dot{\xi}_0(t) = f(\xi_0(t), \pi_0(t), t)$, under control $\pi_0(t)$, any other trajectory that begins within a volume element δV around the nominal trajectory will converge exponentially to a set of measure zero around that trajectory so long as* div *f is uniformly negative definite at every point of the nominal trajectory, i.e. if,*

$$\exists\beta > 0, \forall t \geq t_0 \qquad D_a(\xi_0(t), \pi_0(t), t) \leq -\beta < 0.$$

This theorem says that if the average eigenvalue of F is negative (since $\operatorname{div} f = \operatorname{tr} F = \sum \lambda_F$) then a volume around a given trajectory will collapse on average. There may still be some differential directions which do not collapse down to the nominal trajectory (and, indeed, may diverge), however the differential volume will shrink to zero and the trajectories will lie on some set of measure zero.

Extending beyond the results of contraction analysis, consider the evolution of the expected value of a virtual displacement, $E\left[\|\delta\xi(t)\|\right]$, taken over some distribution,

$$E\left[\|\delta\xi(t)\|\right] = E\left[\|\delta\xi(t_0)\|\right] e^{\int_{t_0}^{t} D_e(x,u,\tau)d\tau}, \tag{9.4}$$

$$D_e(x,u,t) := \frac{d}{dt} \ln E\left[\|\delta\xi(t)\|\right]. \tag{9.5}$$

This form of the expected divergence metric, D_e, may not seem particularly useful. However we will show in the next section that it is easy to compute numerically.

D_m	Maximal divergence
D_a	Average divergence
D_e	Expected divergence

Table 9.1: Divergence metrics

Tab.9.1 lists the metrics presented in this section for easy reference during the remainder of this chapter.

9.1.2 *Numerical approximation*

The contraction analysis of [81] assumes a closed form differentiable vector field. In rearrangement planning, this vector field is defined by the non-holonomic constraint (Eq.(2.1)) that describes the motion of the manipulator and objects. This vector field is not smooth – contact is inherently discontinuous – and lacks an analytic representation (although for simple problems this is theoretically possible [60]). We have shown that f can be effectively approximated by a physics simulator, however analytic divergence measures cannot be derived. Instead, to approximate D_m, D_a, and D_e we introduce numerical divergence metrics that approximate the virtual displacement, $\delta\xi$, with finite samples.

Given a nominal trajectory, $\xi_0(t)$, generated by applying some action $\pi_0(t)$ to a system with dynamics f, a perturbed trajectory (or *noisy rollout*), $\xi_i(t)$, is the solution to the same system and action as the nominal trajectory, $\dot{\xi}_i(t) = f(\xi_i(t), \pi_0(t), t)$, but with a different initial condition, $\xi_i(t_0) = \xi_0(t_0) + \delta\xi_i$. Thus Eq.(9.2) may be modified as,

Here $x_0 = \xi_0(\tau)$ and $u_0 = \pi_0(\tau)$.

$$\|\xi_i(t) - \xi_0(t)\| \leq \|\xi_i(t_0) - \xi_0(t_0)\| e^{\int_{t_0}^{t} D_m(x_0,u_0,\tau)d\tau}, \tag{9.6}$$

which holds in the limit as $\delta\xi_i$ goes to zero. Thus if $D_m < 0$, the ratio, $\|\xi_i(t) - \xi_0(t)\| / \|\xi_i(t_0) - \xi_0(t_0)\|$, goes to zero exponentially. To get the closest approximation, consider the largest such ratio, each of which abides by the bound in Eq.(9.6),

$$\max_i \frac{\|\xi_i(t) - \xi_0(t)\|}{\|\xi_i(t_0) - \xi_0(t_0)\|} \leq e^{\int_{t_0}^{t} D_m(x_0,u_0,\tau)d\tau} \tag{9.7}$$

For a small time step δt, we have that,

$$\hat{D}_m(x_0,u_0,t) := \frac{1}{\delta t} \ln \max_i \frac{\|\xi_i(t+\delta t) - \xi_0(t+\delta t)\|}{\|\xi_i(t) - \xi_0(t)\|} \tag{9.8}$$

and we arrive at the numerical approximation, $\hat{D}_m \approx D_m$.

Similarly, for the average divergence D_a, we will approximate the differential volume by taking the volume spanned by a finite set of points. Let $V(\xi(t))$ define such a volume, then $\hat{D}_a \approx D_a$ is a numerical approximation where,

$$\hat{D}_a(x_0, u_0, t) := \frac{1}{\delta t} \ln \frac{V(\xi(t+\delta t))}{V(\xi(t))} \tag{9.9}$$

Finally, to estimate the divergence of expectation $\hat{D}_e \approx D_e$, consider the ratio of the average displacements,

$$\hat{D}_e(x_0, u_0, t) := \frac{1}{\delta t} \ln \frac{\frac{1}{N}\sum_{i=0}^{N} \|\xi_i(t+\delta t) - \xi_0(t+\delta t)\|}{\frac{1}{N}\sum_{i=0}^{N} \|\xi_i(t) - \xi_0(t)\|} \tag{9.10}$$

The approximation holds exactly in the limit as N goes to infinity and δt goes to zero, as can be derived from Eq.(9.10) or from standard results in Monte Carlo estimation, e.g. [37, Sec. 1.3.1]. Note that the condition in Eq.(9.4) will hold over the length of a path if the set of noisy samples, $\{\delta\xi_i\}$, is drawn once at time t_0 and not independently at each time step.

9.1.3 Path metrics

Define for each metric D_i the exponential of the integral of that metric along a trajectory,

$$E_i := e^{\int_0^T D_i(x,u,\tau) d\tau} \tag{9.11}$$

where note that some of the divergence metrics admit the following simplifications,

$$\begin{aligned}
E_a &= \exp\left(\int_{t_0}^{t} \operatorname{div} f(\xi(\tau), \pi(\tau), \tau) d\tau\right) = \frac{\|\delta V(\xi(t))\|}{\|\delta V(\xi(t_0))\|} \\
\hat{E}_a &= \exp\left(\lim_{\delta t \to 0} \sum_{\tau} \ln \frac{V(\xi(\tau+\delta t))}{V(\xi(\tau))}\right) = \frac{V(\xi(t))}{V(\xi(t_0))} \\
E_e &= \exp\left(\int_{t_0}^{t} \frac{d}{d\tau} \ln E\left[\|\delta\xi(\tau)\|\right] d\tau\right) = \frac{E\left[\|\delta\xi(t)\|\right]}{E\left[\|\delta\xi(t_0)\|\right]} \\
\hat{E}_e &= \exp \lim_{\delta t \to 0} \left(\sum_{\tau=t_0}^{t} \ln \frac{\frac{1}{N}\sum_{i=0}^{N} \|\delta\xi_i(\tau+\delta t)\|}{\frac{1}{N}\sum_{i=0}^{N} \|\delta\xi_i(\tau)\|}\right) \\
&= \frac{\frac{1}{N}\sum_{i=0}^{N} \|\delta\xi_i(t)\|}{\frac{1}{N}\sum_{i=0}^{N} \|\delta\xi_i(t_0)\|}.
\end{aligned}$$

9.2 Planning with Divergence Metrics

The contraction analysis described in the previous section provides conditions for convergence but it does not provide a method of finding such regions. In this section we define two methods for incorporating the divergence metrics presented in Sec.9.1 into the planners presented in Ch.3– Ch.5.

The *Contraction Region RRT* (CR-RRT) sets a threshold on divergence in order to find a monotonically converging trajectory (e.g. to meet the requirements of [81]). The *Biased RRT* (B-RRT) uses the divergence as a heuristic bias in order to find more robust trajectories even when a monotonically converging trajectory is not possible.

9.2.1 *Contraction Region RRT (CR-RRT)*

To allow our planner to meet the requirements of Theor.9.1.1 (or Theor.9.1.2) we modify the extension step to only consider algorithms such that,

$$D_m(\xi(t), \pi(t), t) < 0 \text{ for all } t_i \leq t \leq t_{i+1}, \tag{9.12}$$

(respectively, $D_a < 0$) where $d = t_{i+1} - t_i$ is the duration of the sampled action and t_i is the duration of the existing path to the start of this extension in the tree. If no such actions are sampled, the tree is not extended. If one or more such actions are sampled, the selection criteria for choosing the best of all valid actions remains unchanged from the original algorithm.

Not every problem will have a solution that meets the requirement of Theor.9.1.1, and in such cases this algorithm will never terminate with a solution. We can relax the requirement by altering the constraint,

$$D_m(\xi(t), \pi(t), t) < d_m \text{ for all } t_i \leq t \leq t_{i+1}, \tag{9.13}$$

where $d_m \in \mathbb{R}$ is a parameter that corresponds to the maximum admissible divergence value.

9.2.2 *Biased RRT (B-RRT)*

An alternative method of incorporating the divergence metrics is to include them in the distance computation, allowing them to bias the search by altering the values used in the selection criteria at each extension. Rather than selecting the best action based on a Euclidean distance metric, as proposed in previous chapters, the biased RRT (B-RRT) algorithm scales the original distance by a factor of $s = e^{bD_j}$. Here $b \in \mathbb{R}$ is a *bias* and D_j is the chosen divergence metric. The modification updates Alg.1-Line 8,

$$i^* = \operatorname*{argmin}_i e^{bD_j} \texttt{Dist}(x_i, x_{rand})$$

With this, actions that perform well with respect to the divergence metric are preferred even if they are not the most direct path. Thus, the B-RRT heuristically tries to reduce the divergence, without enforcing the strict conditions of the CR-RRT. Note that when $b = 0$, this algorithm is identical to our original algorithm.

9.3 *Experiments and Results*

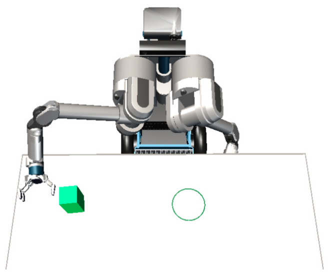


Figure 9.3: Test scenario.

We test the modifications to the planning framework presented in Sec.9.2.1 and Sec.9.2.2 on one of the scenes from the HERB robot experiments in Sec.3.5.1 (Fig.9.3). As mentioned in Sec.9.1, we cannot analytically compute divergence metric values for our problem. Instead, we use the derived approximations. We evaluate plans generated by our previous planners, as well as with the B-RRT and CR-RRT modifications using the exponential divergence $\hat{E}_e$ calculated using $N = 100$ and sampling the initial object poses from a Gaussian distribution with standard deviation $\sigma = \texttt{diag}\{2\texttt{cm}, 2\texttt{cm}, 0.1\texttt{rad}\}$. This distribution was selected to reflect the actual distribution of noise from our object detection system used to initialize real robot experiments in Sec.9.3.4.

9.3.1 *Baseline*

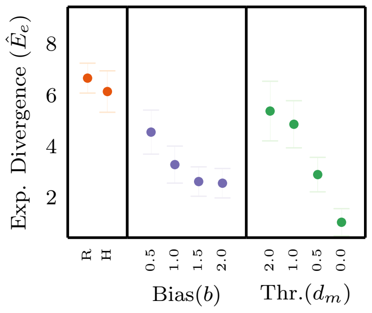


Figure 9.4: Comparison of $\hat{E}_e$ values for the robot-centric (R), hybrid (H), B-RRT and CR-RRT. Increasing the bias in the C-RRT and decreasing the threshold in the CR-RRT both lead to better performing paths.

We first compute the exponential divergence $\hat{E}_e$ for all paths created using the robot-centric planner from Ch.3 and the object-centric/robot-centric hybrid planner (p=0.5) from Ch.4. These results are denoted R and H respectively. Computing these values establishes a baseline for which to compare our algorithm modifications.

Fig.9.4-(left) shows the $\hat{E}_e$ values for all 50 paths for each planner. The incorporation of object-centric actions reduces $\hat{E}_e$, indicating an increased robustness to uncertainty. This is not surprising. The push primitive is inherently uncertainty reducing [34].

9.3.2 *B-RRT*

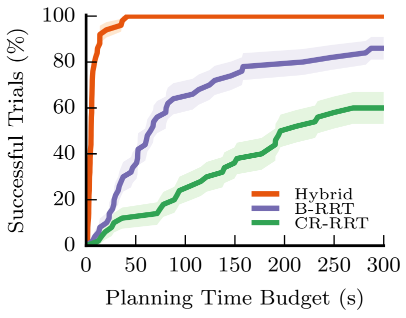


Figure 9.5: The success rate of the planners as a function of plan time. The increased computational burden of evaluating the divergence metric on each extension increases planning time and lowers success rate under a fixed time budget. This highlights the trade-off required to produce more robust paths.

We implement the B-RRT using $\hat{D}_e$ as the divergence metric, i.e. the numerical approximation to the expected value divergence. We use the planner to generate 50 trajectories that solve the scene. This is repeated for values of $b = \{0.0, 1.0, 1.5, 2.0\}$. Each trial used $N = 10$ samples at each extension step to calculate $\hat{D}_e$.

Fig.9.4-(middle) shows the $\hat{E}_e$ value as a function of b. As we increase the value of b, the mean expected divergence $\hat{E}_e$ decreases. Fig.9.5 compares the success rate vs. plan time for the B-RRT with $b = 2.0$ to the hybrid planner with $p = 0.5$ from Ch.4. The two plots highlight a trade-off in planning robust paths. The B-RRT exhibits slower planning times and lower overall success. This is due to two reasons. First, the additional computation time required to compute

$\hat{D}_e$ for each extension. Second, the tree spreads slowly because the bias leads to selecting shorter actions where error has less time to accumulate.

9.3.3 *CR-RRT*

Next, we test whether this problem admits a solution that falls strictly inside a contraction region. To test this, we use the CR-RRT with $\hat{D}_m$ computed with $N = 10$ samples. We test with decreasing values of $d_m = \{2.0, 1.0, 0.5, 0.0\}$.

Fig.9.4-(right) shows the $\hat{E}_e$ value of trajectories found by the CR-RRT as a function of d_m. As can be seen, lowering the threshold d_m leads to more robust trajectories. The best solutions are found with $d_m = 0.0$. Trajectories in this category are composed entirely of convergent actions - each individual action reduces the uncertainty in the system. Fig.9.6-(bottom) shows an example of such a trajectory.

Again, the increased robustness comes at a cost. Fig.9.5 shows the success rate as a function of plan time for the CR-RRT with $d_m = 0$ vs. the B-RRT ($b = 2.0$) and hybrid planners. Imposing a strict threshold on extension leads many potential candidates to be rejected and an lead to complete failure on some extensions due to lack of valid candidates. This imposes additional computational burden over the B-RRT, leading to longer planning times and lower overall success rate.

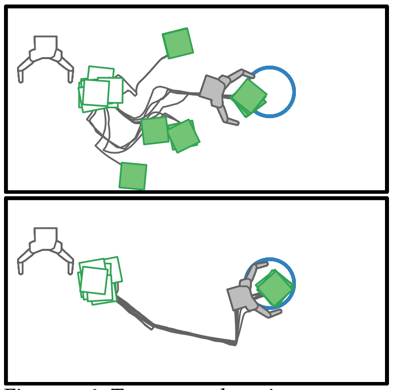


Figure 9.6: Two example trajectories.*Top*: Exponential divergence $\hat{E}_e = 10.50$. *Bottom*: Exponential divergence $\hat{E}_e = 0.15$.

9.3.4 *Real robot experiments*

In this section, we demonstrate that trajectories with lower $\hat{E}_e$ succeed more often in execution than trajectories with high $\hat{E}_e$ values. We generate several solutions to rearrangement problems for the HERB robot similar to the scene used in the planning experiments. We use AprilTags [98] to detect the pose of the box to be pushed. Similar to prior results, plans are evaluated using $\hat{E}_e$ calculated with $N = 100$ and sampling the object poses from a Gaussian distributed with $\sigma = \mathtt{diag}\{\mathtt{2cm}, \mathtt{2cm}, \mathtt{0.1rad}\}$ to reflect actual computed error in AprilTag measurements.

Fig.9.7 shows the results of 11 plans with varying $\hat{E}_e$ each executed 10 times on HERB. For each execution, the initial pose of the box is disturbed by an offset drawn from the same Gaussian used in the evaluation of $\hat{E}_e$. This allows us to simulate several noisy detections. An execution is a success if the final pose of the box is within 15 cm of the planned final pose. The trials shaded in dark blue demonstrate a negative correlation between success rate and $\hat{E}_e$, i.e. trajectories with lower $\hat{E}_e$ are more likely to succeed. Two trials (light blue) do not follow the expected trend. A closer look at these paths attributes

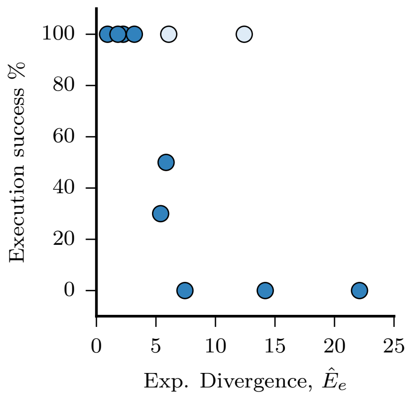


Figure 9.7: Measured success rate over 10 executions of trajectories on HERB as a function of exponential divergence, $\hat{E}_e$.

the discrepancy to unrealistic behavior in the physics simulator used to calculate $\hat{E}_e$.

9.3.5 *Relationship to trajectory selection*

We note that the path metrics proposed Sec.9.1.3 could be used within the trajectory selection framework proposed in Ch.8. Each of the numerical approximations to the exact path metrics use a finite set of samples to estimate the true value of the metric. As mentioned, this estimate becomes more accurate as the sample size increases.

Following the same ideas from Sec.9.1.3, we could use the Successive Rejects Algorithm to allocate noisy samples (rollouts) to each path. These samples can then be used to refine the estimate of $\hat{E}_e$. As time, or rollout budget, expires, we select the candidate with minimum $\hat{E}_e$ value. In fact, we see a strong correlation between $\hat{E}_e$ value and the success probability metric used in the original trajectory selection formulation (Fig.9.8).

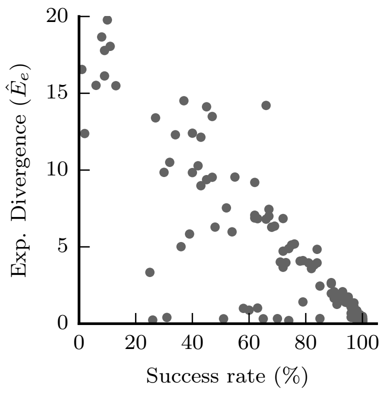


Figure 9.8: $\hat{E}_e$ vs. $\hat{p}_\pi$, the estimated probability of success used to guide trajectory selection in Ch.8, across all trajectories generated by the B-RRT. These two values are correlated: high $\hat{E}_e$ indicates low $\hat{p}_\pi$.

9.4 *Summary and Discussion*

We have proposed new convergent path planning methods that can search for open-loop trajectories that are inherently robust to state uncertainty prevalent in robotics. We introduced analytic and numerical divergence metrics that the convergent planners seek to minimize. Using the strongest of these planners and metrics, we showed the first planning based method to find contraction regions where all states converge to a single trajectory.

We note that the divergence metrics are fundamental properties of the underlying vector field, and motion planning will be most effective when it considers these properties. Convergent motion planners, like those presented here, provide a new way to generate behaviors that are robust to uncertainty that is always present when running robots in the real world.

10
Unobservable Monte Carlo Planning

The planning modifications proposed in Ch.9 allow us to improve the robustness of the trajectories generated by our planner while maintaining the original planning framework from Part I. The use of the proposed *divergence metrics* allow us to identify and select uncertainty reducing actions when possible. However, the decisions made during the search use only local information. At each extension of the tree, individual actions are evaluated for robustness, but the planner lacks the ability to track the evolution of uncertainty through the tree. This limits the effectiveness of the overall planner in two ways. First, the actions may not be evaluated under the true uncertainty conditions in which they will be applied. Second, the planner is overly conservative. It attempts to reduce all uncertainty. Often, some uncertainty is okay, especially in movables that are not relevant to the goal.

To overcome these limits we must track uncertainty as it evolves under sequences of actions. To do this, we modify our planning approach to plan in belief space, $\mathcal{B}$, where each point $b \in \mathcal{B}$ represents a probability distribution over possible states, $b = p(x)$. In this chapter, we frame the planning problem as an instance of an Unobservable Markov Decision Process (UMDP), a subclass of the widely used Partially Observable Markov Decision Process (POMDP). We show that we can extend Monte Carlo Tree Search methods [25] used to solve large MDPs and POMDPs to the UMDP domain. These methods rely on using Monte Carlo simulations to estimate the "goodness" or value of an action sequence under unknown initial state. The *contact* critical to successfully solving rearrangement planning problems requires careful selection of the algorithm components, particularly the actions and default policies, in order to guide the planner to portions of the belief space likely to lead to goal achievement. We outline methods that draw on our lessons and observations from Part I for creating an informative action set and default policy and show that these allow us to produce robust open-loop trajectories.

10.1 *Unobservable Markov Decision Processes*

In our original planning formulation in Ch.3 we used a deterministic transition function $\Gamma : X \times \mathcal{A} \rightarrow X$ to approximate the non-holonomic constraint imposed by the physical interactions of pushing. In reality, it is impossible to create a transition function that perfectly represents the real-world evolution of a state under an action, due to the uncertainties prevalent when executing in real environments. In order to capture the true evolution of uncertainty in our state, we must use a non-deterministic transition function that can account for the unknown or misrepresented elements in our model of the environment dynamics.

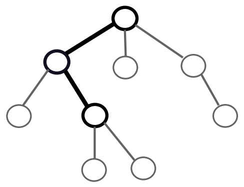
The most common formulation for problems with non-deterministic transition functions is a Markov Decision Process (MDP). MDPs are defined by four elements: $\langle X, \mathcal{A}, T, R \rangle$ where X is the state space, $\mathcal{A} : \mathcal{U} \times \mathbb{R}^{\geq 0}$ is the space of actions to be applied, $T = P(x'|a, x)$ describes the stochastic transition function and $R : X \times \mathcal{A} \rightarrow \mathbb{R}$ describes a reward received for a transition from state x under action a. A solution to an MDP provides an optimal policy, $\pi^* : X \rightarrow \mathcal{A}$ that determines the best action to take from any state in X.

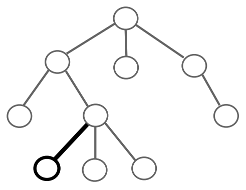
The MDP formulation assumes that the state is fully observable at all times, e.g. in our problem the robot knows x, the exact state of itself and all objects in the scene. The robot uses the policy to select an action from its current state, executes the action in the real world, observes the reached state and iterates. Our problem falls outside this domain for two reasons. First, we have a noisy estimate of the initial state, i.e. it is not possible to perfectly detect the initial pose of objects, or the robot. Second, we assume open-loop execution making it impossible to observe the state achieved after executing an action.

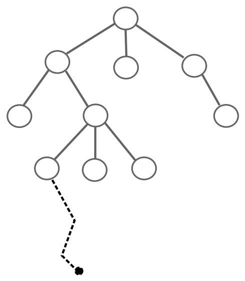
The Partially Observable Markov Decision Process (POMDP) extends the MDP formulation to partially observable domains where the robot has some uncertainty in the state. POMDPs are comprised of an MDP with two additional elements: O the set of observations the robot can obtain while executing, and $Z = Pr(o|x', a)$ the distribution of observations given a state x' reached by executing action a. A solution to a POMDP provides an optimal policy, $\pi^* : \mathcal{B} \rightarrow \mathcal{A}$ that determines the best action to take from any belief $b \in \mathcal{B}$.

Our lack of observations during execution mean our problem can be formulated as a sub-domain of POMDPs called Unobservable MDPs (UMDPs). These are a special case of POMDPs in which there is a single observation, the null observation, that is generated with probability 1 at every time step and gives no information about the current state.

Solving the UMDP that describes the rearrangement planning


(a) *Tree policy* used to traverse the tree


(b) Single node added to the tree


(c) *Default policy* used to perform a simulation

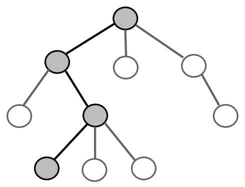

(d) Simulation result back-propagated through tree

Figure 10.2: The MCTS algorithm

problem exactly is difficult for three reasons: (1) we search across a continuous state and action space, (2) the dynamics of the system make closed form representation of the belief transition function difficult and (3) most actions in the continuous action space fail to make meaningful contact with objects.

The MCTS algorithm naturally account for difficulties (1) and (2). In the following sections we show how we can extend this algorithm to UMDPs and carefully select the important components of the algorithm to handle difficulty (3).

10.2 *Monte Carlo Tree Search*

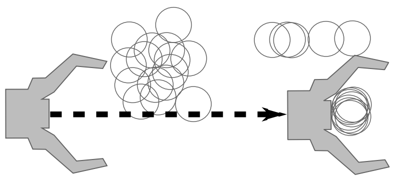


Figure 10.1: The evolution of the uncertainty when performing a simple pushing action. As can be seen, the uncertainty quickly becomes non-Gaussian and non-smooth.

In our domain, the evolution of the uncertainty when using nonprehensile interactions is non-smooth and non-Gaussian. Consider the toy example of a hand pushing a disc under uncertainty in the pose of the disc (Fig.10.1). After performing a simple straight line pushing action, the distribution of object poses becomes multimodal and has rigid edges. These dynamics make closed form representation of the belief transition function difficult.

Monte Carlo methods have been used in MDPs [22, 67, 122] and POMDPs [111, 121] when the true transition probabilities are unknown. These methods use a generative model, G, or black-box simulator, to sample successor states and rewards given a current state and action: $(x', r) \sim G(x, a)$. The generative model allows us to estimate the value of actions and generate near-optimal policies without closed form models of the environment dynamics.

Monte Carlo Tree Search [30, 111] (MCTS) is one such method for solving MDPs using this paradigm. The MCTS algorithm iteratively builds a tree using Monte Carlo simulations (Fig.10.2). Each node in the tree represents a state $x \in X_{free}$ and each edge in the tree represents an action. A node stores a count $N(x)$ of the number of times the state has been visited during the search, and a value, $\hat{Q}(x, a)$ for each outgoing action a, or edge, from the node. The value

$\hat{Q}(x, a)$ estimates the true underlying value function $Q(x, a)$ that describes the expected value, or reward, of taking action a when in state x. The estimate is formed by tracking the mean reward obtained from all Monte Carlo simulations that select action a when visiting the node representing x during planning.

The tree is built incrementally. At each iteration, a *tree policy* is used to search through the tree until a leaf node is reached for expansion (Fig.10.2a). The tree policy attempts to balance exploration of new regions of state/action space, with exploitation of visited and promising regions. Once the search leaves the tree (Fig.10.2b), a *default policy* is used to rollout the remainder of the simulation (Fig.10.2c). Then the reward resulting from the full simulation is back-propagated through the tree and used to update value estimates for each node (Fig.10.2d). The algorithm iterates until a termination criteria is achieved, i.e. timeout.

A policy is extracted from the tree according to the following:

$$\pi^*(x) = \underset{a \in \mathcal{A}}{\text{argmax}}\, \hat{Q}(x, a)$$

MCTS algorithms are a good fit for our problem. We can easily use our physics model to perform black-box simulations of control sequences that build up our estimates of the value function at each node in the tree. However, these simulations have some computational expense. The MCTS framework focuses computational resources to relevant regions of state space. In addition, the algorithm is anytime and highly parallelizable.

10.2.1 Unobservable Monte Carlo Planning (UMCP)

The POMCP [111] algorithm applies the MCTS framework to partially observable environments by augmenting the search to allow each node to represent a history: a sequence of actions and observations. The tree then tracks an estimate of the belief state for each history in the tree. This algorithm naturally deals with the *curse of dimensionality*[1]by considering only belief states reachable from a known initial belief state b_0.

[1] The dimensionality of the belief space is related to the number of states. A problem with n states renders an n-dimensional belief space. A continuous state space leads to an infinite dimensional belief space.

We use a similar approach to find approximate solutions for our unobservable MDP. We build a tree such that each node represents a unique history, $h = \{a_1, \ldots a_t\}$. Three values are stored for each node: $N(h)$ - the number of times the history, or action sequence, has been explored, $\hat{Q}(h, a)$ - an estimate of the value of taking action a after applying history h, and $\hat{\mathcal{B}}(h)$ - an estimate of the true belief achieved when applying the actions in h.

Alg.8 shows the MCTS algorithm applied to our UMDP. The tree is rooted with an initial belief state s_0 that contains a set of states drawn

Algorithm 8 Unobservable Monte Carlo Planning

```
 1: $s_0 \leftarrow$ GenerateInitialSamples()
 2: while not timeout do
 3:     $x \leftarrow$ SampleState($s_0$)
 4:     Simulate($x$,$\{\}$,$0$)
 5: function Simulate($x$, $h$, $d$)
 6:     if $\gamma^d < \epsilon$ then return 0
 7:     if NotVisited($h$) then
 8:         InitializeHistory($h$) return DefaultPolicy($x$)
 9:     $a \leftarrow$ TreePolicy($h$)
10:     $x' \leftarrow$ NoisyPhysicsPropagate($x$,$a.u$,$a.d$)
11:     $r \leftarrow$ Reward($x$, $a$) $+\gamma\cdot$ Simulate($x'$, $h \cup \{a\}$, $d+1$)
12:     $\hat{\mathcal{B}}(h) \leftarrow \hat{\mathcal{B}}(h) \cup \{x\}$
13:     $N(h) \leftarrow N(h)+1$
14:     $\hat{Q}(h,a) \leftarrow \hat{Q}(h,a)+r$ return $r$
```

from an initial distribution defined on the state space. Then, during the search an initial state $x \sim s_0$ is drawn from the belief state (Line 3). This state is propagated through the tree by using the *tree policy* to select actions (Line 9) and using a noisy physics model to forward propagate the state under the selected actions (Line 10). After each propagation, the new state is added to the belief state of the new history (Line 12). The search recurses through the tree, propagating a single state through the noisy transition dynamics. Over time, the belief states represented at the nodes of the tree grow to represent the true belief distribution.

Once the search reaches a previously unvisited history, a *default policy* is used to rollout the remainder of a simulation and accumulate reward (Line 8). This reward is propagated back through the tree to update the value function estimate stored for each history/action pair.

Reward model

As stated in Ch.7, our goal is to generate paths that maximize the success functional Λ (Eq.(7.1)). We encode this functional into our reward model:

$$R(x,a) = \Lambda(x) \tag{10.1}$$

In Ch.7 we defined Λ as a functional that determines if the endpoint of a trajectory is a goal state. In this chapter we overload Λ to describe whether a particular state represents a goal state:

$$\Lambda(x) = \begin{cases} 1 & : x \in X_G \\ 0 & : \text{otherwise} \end{cases}$$

Action set

MCTS-based planners search across a discrete action set. In our formulation from Ch.2, we search across a continuous action space.

The naive method for generating a discrete action set from a continuous space is to divide the space into partitions, and select a single representative action from each partition, e.g. the mean action. However, this method ignores a key aspect of our problem: *contact* is critical to success in rearrangement planning. Consider the simple example of a robot pushing an object in Fig.10.3. Discretization of the continuous action space leads to a discrete action set that moves the robot in the four cardinal directions. The object is "trapped", i.e. there is no motion that can create enough contact to move it out of the current cell.

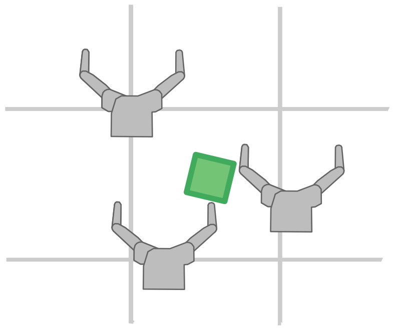


Figure 10.3: A simple example of a hand pushing objects. The *basic* primitives allow the hand to translate along the grid lines. In this simple example, the object is "trapped", i.e. there are no primitives that allow it to move out of the cell.

We could expand the action set by discretizing the control space more finely. However, the size of the primitive set is directly related to the branching factor of the search, so any large increase affects the computation time. We wish to focus the discretization to promising areas of action space. To do this we follow ideas from Sec.4.3. We first select a set of primitives that move the robot without the explicit intent of creating contact or interaction with objects. These *basic* primitives are small motions of the robot defined by a coarse discretization of the control space.

These *basic* primitives are similar in motivation to the *robot-centric* primitives used by our deterministic state space planners. The primitives are context agnostic; they are not specific to the rearrangement planning problem. *Basic* primitives may achieve some contact with objects, but it is not guaranteed. We know contact with objects in the environment is critical to goal achievement. With this in mind, we augment the primitive set applied at each set with *contact* primitives aimed at creating contact with objects in the scene. These are similar in motivation to the *object-centric* primitives from Sec.4.3.

We instantiate these primitives using the first state in the estimated belief $x_0 \in \hat{B}(h)$ for each history. A contact primitive is generated by solving the two-point BVP that moves the robot to a pose in contact with an object based on the object's pose in x_0. We create one *contact* primitive for each object in x_0 that must be moved to achieve the goal. For example, for the clearance task described in previous chapters, we create one primitive for each object in the region to be cleared.

Tree Policy

The tree policy is used to select actions, or edges, in the UMCP tree to traverse. For a given node in the tree corresponding to history h, the policy first uses the method from the previous section to generate a discrete set of actions $\mathcal{A}_{active}$. Once the discrete action set is obtained, the tree policy must select a single action to traverse. We follow the UCT algorithm [67] and use UCB1 [12] as the tree policy. Under this paradigm outgoing actions from each node are treated as arms in a multi-armed bandit problem. UCB1 then selects an action, or arm, as follows:

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}_{active}} \frac{\hat{Q}(h,a)}{N(h \cup \{a\})} + c\sqrt{\frac{\log N(h)}{N(h \cup \{a\})}} \tag{10.2}$$

where $c > 0$ is an exploration constant. Note that this selection method requires all actions are tried at least once.

The use of such a method is ideal because it provides a formal method for trading between exploration and exploitation.

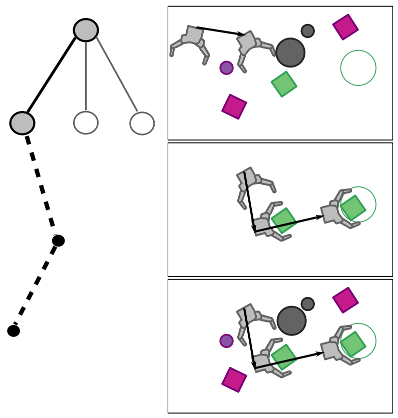


Figure 10.4: An example of MCTS applied to rearrangement planning. A *tree policy* is used to select actions that make contact (top-right). Then the *default policy* plans in the a lower dimensional space (middle-right). The result is propagated through the full space to generate a reward (bottom-right).

Default policy

Each time the search reaches a leaf in the tree, the default policy is used to estimate the reward that will be obtained if we follow a path that leads through this leaf. The most common default policy is to randomly select a sequence of actions to apply. For our rearrangement planning problem, this policy will rarely be informative: most action sequences fail to create and maintain the contact with objects that is critical to goal achievement.

Instead, we define an informed default policy that is capable of quickly searching for a path to the goal. We perform the search in the lower dimensional subspace containing only the elements of the full state space that are defined in the goal. We allow the search to simply ignore all other movable objects. Movable objects that are not defined in the goal are not checked for collision or included in any physics simulations during the search. This implicitly assumes that all movable objects can trivially be moved out of the way.

After generating a set of actions that solve the problem in the lower dimensional subspace, the actions are then forward simulated through the full state space to generate the reward value that is backpropagated through the tree. Fig.10.4 illustrates this method.

Our key insight is that by reducing the dimensionality of the state space in the default policy search, we allow for the possibility of using fast planners or exact solvers that provide much more information than random action sequences. For tasks such as the K-Rex traversal task that involve only the manipulator we can either solve the two-point BVP directly or use a local planner. For tasks that require explicitly moving objects such as the clearance tasks for HERB and KRex, we can use the planners from Part I. We provide a faster heuristic planner in Appendix A capable of solving tasks that involve moving a single object.

Path extraction

We use our tree to create an anytime algorithm for extracting paths. Upon request, a path π is extracted from the tree as follows. First, we extract π_{tree} by repeatedly picking the action a_t such that:

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}_{active}} \hat{Q}(h, a)$$

Once a leaf is encountered, we query the belief represented by the history $\hat{\mathcal{B}}(h)$ to obtain a probability of success $\hat{p}_{\pi_{tree}} = \sum_{x \in \hat{\mathcal{B}}(h)} \Lambda(x)$. If this probability is lower than a threshold p_{goal}, we randomly select a state from the belief $x \in \hat{\mathcal{B}}(h)$ and use the *default policy* to generate a path π_{def} from x to the goal. If successful, all remaining samples in $\hat{\mathcal{B}}(h)$ are forward propagated through this path to get an updated probability of success $\hat{p}_{\pi}$ of the combined path $\pi = \pi_{tree} + \pi_{def}$ formed from appending π_{def} to π_{tree}. If $\hat{p}_{\pi}$ is better than the success probability from previous requests, the path is returned. Otherwise, the previous best path is returned.

The use of a planner to generate the default policy means we can often find paths that achieve the goal with non-zero probability. Our insight is that these path segments can be particularly useful when there is not enough planning time to deeply grow the tree.

10.3 Experiments and Results

We test the capabilities of the UMCP algorithm using the task of HERB pushing an object to a goal region through clutter from Sec.3.5.1. We test three hypothesis:

H.1 The planner using the **planned** default policy allows us to generate paths with higher probability of success than the planner with a **random** default policy.

H.2 Using explicit **contact** primitives allows the planner to generate paths with high probability than a planner that uses a **basic** action set formed by discretization of each dimension of the action set.

H.3 Our UMCP planner that uses **contact** primitives and the **planned** default policy is able to produce paths that exhibit higher probability of success compared to anytime versions of baseline planners presented in previous chapters.

H.1 verifies our intuition that using a powerful default policy capable of explicitly attempting to achieve the goal will outperform the common default policy that selects random action sequences, despite the extra resources and slower tree growth that result from using a planner to compute the default policy. **H.2** tests the need for primitives that explicitly try to create contact with goal critical objects. Finally, **H.3** verifies the need to track the evolution of uncertainty through sequences of actions. We expect this to be especially important on problems where it is difficult to find sequences solely comprised of uncertainty reducing actions, as preferred by the C-RRT.

In the following sections we detail our planning setup and provide results for each hypothesis.

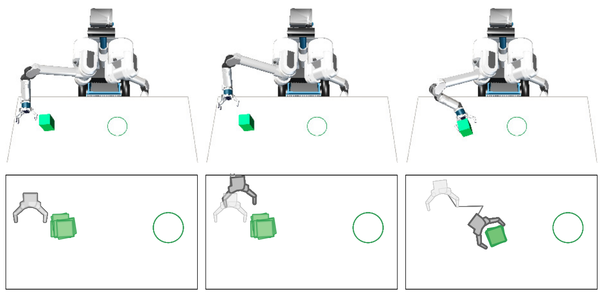
10.3.1 Test setup

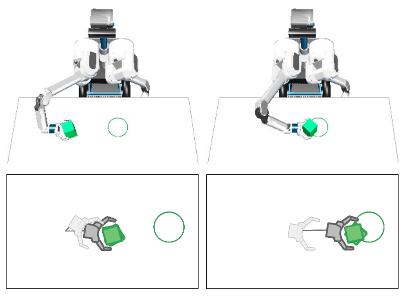
We use the heuristic structured search planner from Appendix A as the **planned** *default policy*. Specifically, we use the action set and heuristics described in the appendix in a weighted A* search with $w = 5$. This strongly biases our search toward the goal. We allow the planner 1 s to find a path.

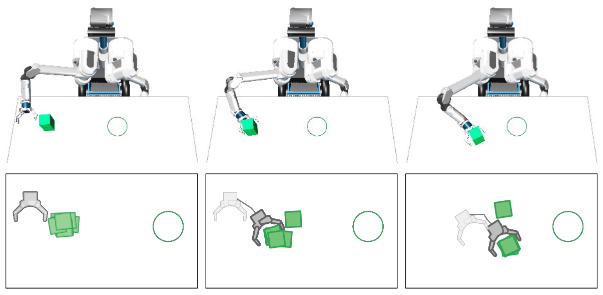
We run each version of the UMCP planner 50 times on a low clutter scene (Fig.10.6) and a high clutter scene (Fig.10.10). In each scene, we generate the initial belief s_0 by sampling noise into the initial pose of each object from a Gaussian with distribution $\mu = \mathbf{0}, \Sigma^{1/2} = \texttt{diag}\{\texttt{2cm}, \texttt{2cm}, \texttt{0.1rad}\}$. We allow the UMCP planner to run for 300 s. We request and record a path every 15 s.

10.3.2 Baseline planners

We compare the estimated success rate of the paths generated by the UMCP algorithm to anytime versions of the B-RRT planner (with bias $b = 2.0$) from Ch.9 and the original PCRRT from Ch.3. To create anytime versions of these planners, we make as many repeated calls to the planner as possible within 300 s. When a call completes we perform a set of 100 noisy rollouts on the resulting control sequence π to generate an estimate $\hat{p}_\pi$ of the probability of success. We generate these noisy rollouts using the same noise parameters used to create the initial belief s_0 in the UMCP planner. We keep π only if it has higher estimated success probability than all previous paths generated by the planner. This is a similar algorithm to the AMD-RRT described in [59] though we use probability of success rather than performance of the path under a divergence metric.


(a) *Tree policy*


(b) ***Planned** default policy*


(c) *Tree policy*

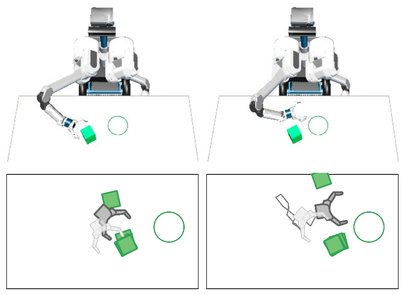

(d) ***Random** default policy*

Figure 10.6: An example path computed with the **planned** default policy (*top*) and **random** default policy (*bottom*) at $t = 45\,\text{s}$.

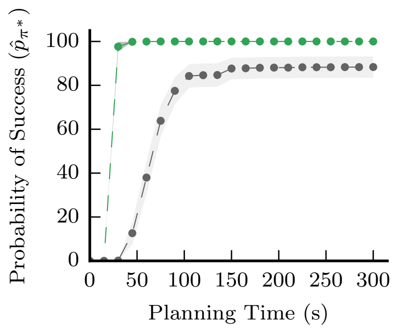
10.3.3 *Effect of default policy*

We first examine the effect of our choice of default policy. Fig.10.5 shows the estimated probability of success of the chosen path, $\hat{p}_{\pi*}$, as a function of planning time. As can be seen, the use of the **planned** default policy allows us to generate better paths faster, especially in high clutter scenes. This is despite the planned default policy taking almost 10x as long to compute as the random default policy (mean time 0.5 s and 0.06 s respectively). This supports **H.1**: **the use of an informed default policy leads to better overall success rate in generated paths.**

Fig.10.6 shows an example path at $t = 45\,\text{s}$ for each version of the UMCP planner. The left column shows the portion of the path extracted from the tree. The paths are similar, though the UMCP planner that uses the **planned** default policy finds a more robust sequence (Fig.10.6-*top*). This is because the default policy is more informative and allows better estimates of $\hat{Q}(h, a)$ early in the tree.

The main difference in the two results comes from the portion of the path extracted using the default policy. The **planned** default policy maintains contact with the goal object and eventually moves the full belief either into or near the goal region. The **random** default policy loses contact with the object quickly and fails to move any of the belief to the goal.

Fig.10.7 shows a path requested later in the planning call ($t =$



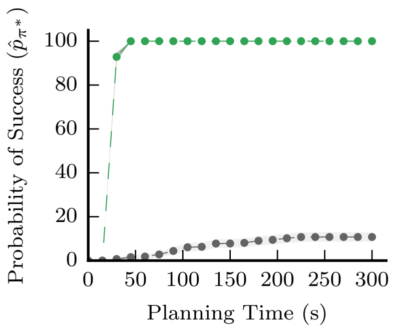
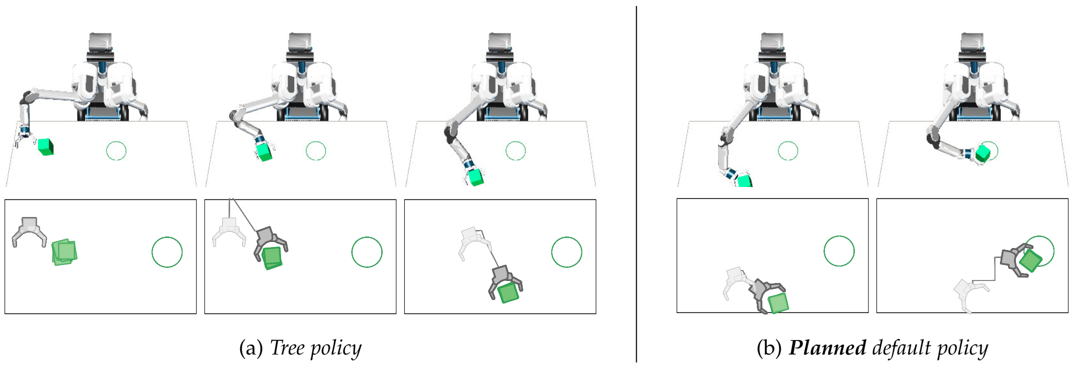


Figure 10.5: Use of a policy explicitly seeking to achieve the goal (—) results in overall better paths when compared to using a policy that randomly selects actions (—) for both low clutter (*top*) and high clutter scenes (*bottom*).


(a) *Tree policy* (b) ***Planned** default policy*

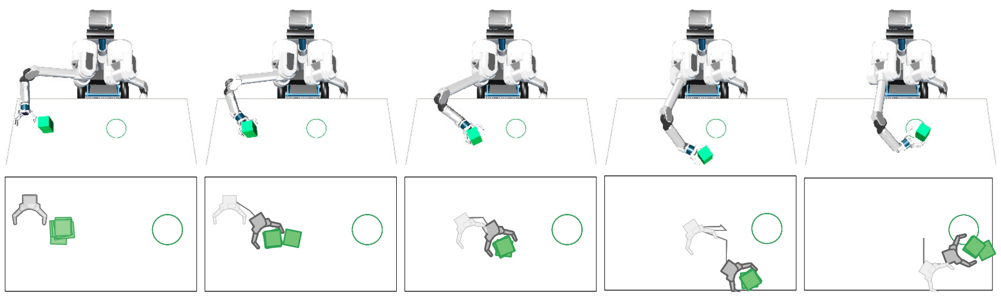

(c) *Tree policy*

Figure 10.7: An example path computed with the **planned** default policy (*top*) and **random** default policy (*bottom*) at $t = 250$ s.

250 s). At these later times we begin to see the advantage of using a faster default policy. Here the tree using the **random** rollout policy is able to grow deep enough that a full path can be extracted without needing the rollout policy to supplement the path. Still, the **planned** rollout policy is more informative, enabling better estimates of $\hat{Q}(h, a)$ leading to a more robust path.

10.3.4 Effect of contact primitives

Next we examine the effect of the **contact** primitives in the action set. We compare to a baseline planner that uses only the **basic** action set. Fig.10.8 compares the probability of success $\hat{p}_{\pi*}$ of the path returned by the planner at each time step.

In low clutter scenes, the usefulness of the **contact** primitive is limited (Fig.10.8-*top*). The UMCP planner with the **contact** primitive finds paths only slightly faster. This is due to the use of the **planned** default policy used to complete paths extracted from the tree. Examination of the generated paths shows that the default policy is heavily relied upon to generate the contact needed for success.

The benefit of these primitives is much more prevalent in high clutter scenes (Fig.10.8-*bottom*). Here, the **planned** default policy fails to be applied without first moving either the bowl or bottle (Fig.10.10-*left*). The **simple** primitive set is not rich enough to create

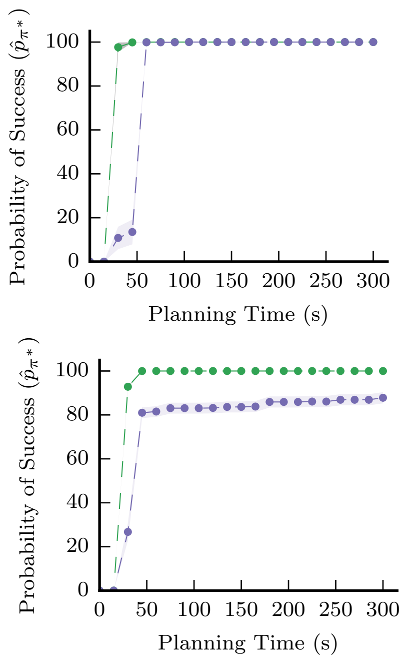
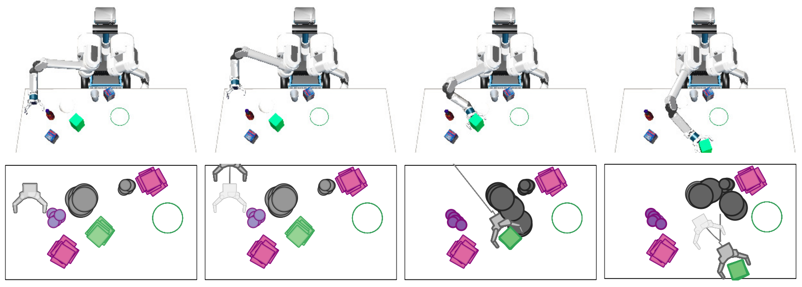
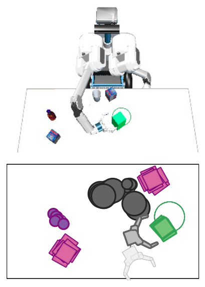


Figure 10.8: Using **contact** primitives (—) allows for finding better paths sooner than using **basic** primitives (—).



(a) *Tree policy*



(b) ***Planned** rollout policy*

Figure 10.10: In high clutter scenes, the UMCP algorithm with **contact** primitives performs well. The algorithm allows for increasing uncertainty in the pose of the bowl, as long as it does not inhibit goal achievement.

useful contact. In contrast, the **contact** primitive easily moves both objects out of the way in order to make contact with the box. Then the **planned** default policy can be applied to achieve the goal. Fig.10.10 depicts an example path found by the UMCP planner with **contact** primitives.

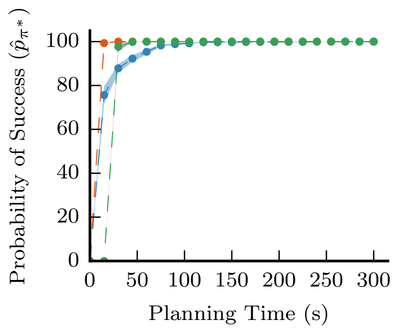
10.3.5 Comparison to baseline planners

Finally, we compare the UMCP planner using **contact** primitives in the action set and the **planned** rollout policy to the baseline planners described in Sec.10.3.2. Fig.10.8 shows the estimated probability of success $\hat{p}_{\pi^*}$ of each planner as a function of planning time.

For the low clutter scene the PCRRT and B-RRT are easily able to find solutions (Fig.10.9-*top*). The B-RRT performs exceptionally well here because there exists a solution comprised almost entirely of uncertainty reducing, or low divergence, actions.

The advantage of the UMCP algorithm can be seen for the high clutter scene (Fig.10.9-*bottom*). Here, the B-RRT performs poorly because the actions that reduce uncertainty in the goal object increase uncertainty of the pose of other objects, such as the bottle or bowl (Fig.10.10). Such actions perform poorly under the divergence metrics. As a result, the B-RRT is slow to explore and find solutions. The UMCP allows for increasing uncertainty along dimensions that are not important for goal achievement. This allows the planner to find solutions more easily.

Overall, for planning time budgets greater than 30 s the UMCP algorithm finds solutions as good as the solutions found by the B-RRT and PCRRT algorithms in the low clutter scene. The UMCP algorithm outperforms both baseline planners in the high clutter scene. This partially confirms **H.3**: **Our UMCP planner that uses contact primitives and the planned default policy is able to produce paths that exhibit higher probability of success compared to anytime versions of baseline planners presented in previous chapters.**



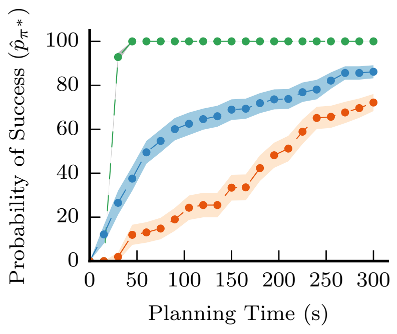


Figure 10.9: On simple scenes (*top*), the B-RRT from Ch.9 (—) is able to find better paths quickly. The UMCP algorithm (—) is able to find better paths more quickly than the PCRRT from Ch.3 (—). On difficult scenes (*bottom*), the UMCP outperforms both the C-RRT and the PCRRT.

10.4 Summary and Discussion

In this chapter we present Unobservable Monte Carlo Planning, an algorithm that extends MCTS to the Unobservable MDP domain. We show that by carefully selecting an informative default policy and an action set capable of generating contact with important objects in the scene, we are able to plan solutions that are robust to uncertainty. The result is an anytime algorithm that returns good solutions fast in our example scenarios.

This algorithm represents a step toward planning rearrangement planning by nonprehensile manipulation in belief space. We believe there are two promising directions that may improve the quality of the planner. First, we can expand the set of actions considered by the planner by using gradient free methods to make local adjustments to the action set. This idea was successfully demonstrated in [109] with the Adaptive Belief Tree algorithm. Second, we believe this algorithm could be extended to closed-loop planning by incorporating feedback as observations in a full POMDP formulation. This is not trivial. Careful thought must be applied to allow us to maintain tractability under the exponential increase in histories. However, recent work in using contact sensing [70] during pushing interactions shows it is possible. We discuss this and other areas for future extensions in our concluding chapter.

11
Conclusion

In this thesis we proposed a set of planners capable of solving rearrangement planning problems using nonprehensile manipulation. We showed that by incorporating uncertainty into our planners, we can create paths that are more likely to be executed successfully on the robot. Each of the planners described in this thesis trades off between implementation complexity, planning efficiency and robustness of the output to real world uncertainties. We do not offer a single solution that best solves rearrangement planning under all conditions and constraints, we doubt that such a planner exists. Throughout the document, we have highlighted limitations in each individual solution. In this final chapter, we discuss limitations that apply across the suite of planners and directions for future work that address these limitations and could expand the effectiveness of the ideas we have presented.

11.1 Lessons Learned

Development and test of the planners described in this thesis have led to number of lessons, some surprising and some expected. We identify a few important ones here:

Full arm interaction and simultaneous object contact: In this work, we have taken an initial step in developing plans that exhibit full arm interactions and simultaneous object contact. We hope we have made a convincing argument for the importance and usefulness of these interactions – they expand the space of problems a robot can solve and allow the robot to work more effectively in clutter. These interactions are made possible by our use of nonprehensile interaction – other modes of interaction such as grasping would benefit less from this expanded set of interaction modes. Still we believe that many manipulation problems can benefit from the ability to reason about nonprehensile manipulation as a means to achieving manipulation goals.

Planning with physics models: Prior to beginning this work, we identified two main concerns with integrating physics models at plan time: (1) speed and (2) validity. Typical physics models perform many intensive computations each time they are stepped. These computations include several small integrations and optimizations to resolve collisions. Indeed, our plan times even under the improvements in Ch.4 are much larger than desirable – taking significantly longer than most modern geometric planners. For our KRex experiments in Ch.4 we traded fidelity of the physics model for speed. Here we used a 2D physics engine rather than a full 3D model. Surprisingly, we were still able to achieve several successful executions of the planned trajectories. We believe this affirms the findings in prior work [19, 38] that using goal regions rather than specific goal configurations can drastically improve performance in a task. These goal regions allow the imperfect models to still be "good enough" for goal achievement.

Unmodeled uncertainties: In Part II we outlined three strategies for coping with uncertainties prevalent when executing planned trajectories on the robot. We showed these strategies to be effective at combating known uncertainties, i.e. uncertainties that we can model. However, as highlighted by our real robot experiments in Ch.8, the methods we described are powerless against unmodeled uncertainties. To account for such uncertainties, we believe we must incorporate feedback directly into the physics model either through a pre-processing step or during trajectory execution. We discuss this further in the next section.

11.2 *Future Work*

Our work on this thesis has illuminated a number of promising directions for future work:

Learning heuristics: Humans accomplish the rearrangement tasks described in this thesis effortlessly and often quite elegantly. Our intuition is that we may be able to use human demonstrations to guide our planning, allowing us to find more solutions faster. We have recently completed preliminary work that reinforces this intuition. In this work, we use Amazon Mechanical Turk to collect demonstrations from users guiding a robot hand to perform rearrangement tasks on a table. From this data, we extract features of the scene that can be provided as input to a multi-class learning algorithm. The learned model provides a mapping from a state

$x \in X_{free}$ to a single action in a discrete set. This mapping can be used to guide action selection within the randomized planners from Ch.3–Ch.5. Alternatively, the mapping can be used as a default policy in the UMCP planner from Ch.10.

Our current structure is limited in two ways. First, to simplify understanding and input from the user, we only allow the user to input discrete motions that move the hand in the plane. Our learned model then maps state to an action from this discrete set. Ideally, we would like to consider continuous action spaces. To do this we must make two modifications to our method. First, we must allow the users more flexibility in their inputs. Second, we must switch from a multi-class classifier to a regression algorithm capable of generating motions in the continuous action space.

The second limitation falls directly out of our test setup. We allow the user to guide the hand, rather than the full manipulator, in order to simplify the input mapping and cognitive load on the user. However, by having the user move only the hand we eliminate the ability of the user to use the full arm to solve the task. In addition, it allows the user to perform actions that are kinematically infeasible or introduce unresolvable collisions (e.g. between the robot arm and obstacles in the environment). Still we find this is a promising direction for future examination.

Integration with hierarchical task planning: Our experience indicates movable clutter can be categorized into two groups: (1) items that can easily be moved through incidental contact and (2) items that should be explicitly moved out of the way. The planners we present in this thesis deal easily and naturally with items in the category 1. Items in the category 2 lead to longer planning times or failure of our planners.

Prior work has structured rearrangement planning as an instance of hierarchical task planning [16, 53, 100]. Under this approach, a high-level task planner is used to identify a set of objects to move and an ordering for moving these objects. Then a low-level geometric planner is used to find feasible trajectories to move each object. This framework naturally deals with category 2 of clutter: items that should be explicitly moved out of the way. An unfortunate side effect is that items in the first category (objects that can easily be moved through incidental contact) are often treated as though they are in category 2. In these cases, the planner spends extra computation and execution time planning unnecessary object movement.

The difficulty comes in properly categorizing an item, i.e. it is difficult to know how much "trouble" an object will cause our planner and which objects require explicit interaction. We believe an interesting body of future work may be tightly coupling our state space planners presented in this thesis with higher level task planners. In particular, as we plan we can identify sources of failure. For example, when performing the randomized search we may invalidate an action because the robot motion leads to an object being pushed off the edge of the table. If several actions lead to the same failure, our planner can inform the task planner that the offending object must first be moved away from the edge or removed from the table before this plan can succeed. The task planner can then insert a sub-goal into the task that attempts to move the object. Our hypothesis is that the introspection of the planner in regards to failures during planning, coupled with reporting to a higher level task planner that can force changes in the environment, may lead to overall better success at rearrangement problems.

Incorporating feedback: This thesis focused on planning open-loop trajectories. In reality, our robots have a corpus of sensors constantly providing feedback during execution. We believe we can use this feedback to improve both planning and execution.

Given access to sensor feedback, we can augment our UMDP planner from Ch.10 to solve a full POMDP. Recent works have considered tactile feedback [70] for improved robustness in grasping tasks. Similar feedback could be used in our planning problems to allow the planner to incorporate information gathering actions that ensure objects are properly localized and to gain a more robust estimate of state when selecting actions to execute. The main challenge in constructing and solving such problems is managing the large state space and longer time horizons required to accomplish rearrangement tasks.

Alternatively, our quasistatic and semi-dynamic plans from Ch.3–Ch.5 ensure that objects will be at rest between actions. We can exploit this fact to incorporate feedback at execution time in lieu of plan time. In particular, after executing each action in the plan, the robot can observe the world and, if different from expectation, forward simulate the remainder of the plan to check goal achievement. If the goal is not achieved, a replan can be triggered. This paradigm is ideal because it allows for incorporating feedback while avoiding the need to solve a full POMDP. The main drawback to the approach comes from the use of non-holonomic nonprehensile interactions, i.e. the robot cannot easily reverse a push. This can lead the robot to begin executing a path, fail, and be unable to recover.

11.3 *The Last Word*

This thesis is a small step towards increasing the capabilities of autonomous robots. As we allow robots to interact more freely with objects in the environment, we open them to plan for and exhibit the types of interactions humans use naturally. In addition, we empower robots to perform a broader range of tasks more effectively and efficiently. We hope this work can serve as a stepping stone on the way to enabling robots to perform meaningful tasks that help humans in their every day lives.

A
Rearrangement Planning via Heuristic Search

In this section we describe a method for solving a specific rearrangement task where the goal can be expressed in terms of a single object $M^j \in \mathcal{M}$. Example tasks may be clearing a single item from a region or pushing an object into a goal.

Our goal is to harness the power of heuristically guided structured search algorithms [49, 105]. These algorithms are desirable because they produce near-optimal plans quickly, given a sufficiently expressive heuristic. To create such a heuristic we make three assumptions on the planning instance:

Assumption 1: Contact between the robot and goal object is restricted to the end-effector. (We do allow contact between the full robot and all other objects in $\mathcal{M}$.)

Assumption 2: The goal object can only be moved by contact with the robot. The robot cannot use other objects in the environment to push the goal object.

Assumption 3: All motions of the robot and objects are quasistatic.

We note that when using this planner as the *default policy* in our UMCP algorithm (Ch.10), we search across the lower dimensional state space $X' = X^R \times X^j$ where Assumption 2 is unnecessary. However, this algorithm can be applied to the full state space X containing movable objects other than the goal object M^j.

We construct an action set comprised of *basic* primitives that move the robot along a lattice with *contact* and *pushing* primitives that explicitly attempt to create and maintain contact with objects. We then define an informative and admissible heuristic that is used to quickly guide the search toward the goal. In the following subsections we briefly describe these two basic elements.

A.1 Action Selection

We select a discrete set of primitives to apply to each state that allows us to perform a feasible and focused search. We first select a set of *basic* primitives: small motions of the robot defined by a coarse discretization of the control space. *Basic* primitives may achieve some contact with objects, but it is not guaranteed. To improve performance of the search, we augment the primitive set applied at each state with a dynamically-generated *contact* or *pushing* primitive aimed at creating or maintaining contact with the goal object. During the search, we forward propagate all primitives through a quasistatic model of physics to ensure our state transitions properly model object motion under pushing contact. This allows us to include motions that exhibit full arm manipulation and simultaneous object contact. We use our insights from Ch.4 to create the *contact* and *pushing* primitives.

A.2 Heuristic

We define the cost of a control sequence, π, as the distance the end-effector of the robot moves in the configuration space of the goal object. Formally, assume we have a distance metric, $\mathrm{d} : X^R \times X^j \to \mathbb{R}^{\geq 0}$, and a function $\mathrm{FK} : C^R \to C^j$ that computes forward kinematics from the robot's configuration space to the goal object's configuration space.

We compute the cost of a single primitive, a, applied to a state $x \in X$ in two steps. First we compute the set $Q = \{q_1 \dots q_{p+1}\}$ of robot configurations achieved by the primitive. This can be obtained by forward propagating the controls in the primitive through the constraint f (Eq.(2.1)). Then the cost of a primitive is:

$$\mathrm{c}_a(a, x) = \sum_{i=1}^{p} \mathrm{d}(\mathrm{FK}(q_i), \mathrm{FK}(q_{i+1})) \tag{A.1}$$

And the cost of a path, $\pi = \{a_1, \dots, a_n\}$:

$$\mathrm{c}_\pi(\pi, x_0) = \sum_{i=1}^{n} \mathrm{c}_a(a_i, x_i) \tag{A.2}$$

where x_i is the state reached by sequentially applying primitives $a_0 \dots a_{i-1}$ to x_0.

Two observations of the problem can be used to generate a useful heuristic that underestimates the cost-to-go from a state $x \in X_{free}$. First, by definition of the problem, contact with the goal object is required for goal achievement. Due to Assumptions 1 and 2, this contact must be between the end-effector and the object. Second, the robot must stay in contact with the goal object for the object to move, due to Assumption 3. Using these observations we develop a two part heuristic to estimate the distance between state x and X_G:

$$h(x) = \hat{\mathrm{d}}_{con}(x) + \tag{A.3}$$
$$\hat{\mathrm{d}}_{move}(x) \tag{A.4}$$

Eq.(A.3) estimates the distance to make contact with the goal object. Eq.(A.4) estimates the distance the end-effector must move to push the goal object to the goal region.
Distance to contact: We compute $\hat{\mathrm{d}}_{con}$ by approximating the end-effector with the smallest enclosing sphere. If this sphere penetrates the object, $\hat{\mathrm{d}}_{con} = 0$. Otherwise, $\hat{\mathrm{d}}_{con}$ is the translational distance between the closest points on the sphere and object under the metric d.

Proposition $\hat{\mathrm{d}}_{con}$ is a lower bound on the true cost to make contact with the goal object.

Proof. Approximating the end-effector pose with the sphere means all rotations of the end-effector have $\hat{\mathrm{d}}_{con} = 0$. Thus our estimate of the rotation distance is a lower bound of the true distance. The shortest translational distance the end-effector can move to make contact is the distance between the two closest points on the end-effector and object. Selecting the closest point on the sphere to the object ensures we underestimate this distance. Since we underestimate translational and rotational distance, we must underestimate the true distance. □

Distance to goal: We compute $\hat{\mathrm{d}}_{move}$ as the straight line distance from the object location to the closest point in the goal region.

Proposition $\hat{\mathrm{d}}_{move}$ is a lower bound on the true cost to move the object to the goal.

Proof. $\hat{\mathrm{d}}_{move}$ is the shortest distance the object can move and still achieve the goal. By the quasistatic assumption, contact must be maintained between robot and object for the object to move. As a result, $\hat{\mathrm{d}}_{move}$ must also be the shortest distance the robot could move. □

A.3 Expanding applicability

We define this algorithm to be applied only to rearrangement problems where the goal can be expressed in terms of a single movable object, e.g. clear an object from a region or push an object to a goal location. As we add more movable objects to the goal, computing an informative heuristic requires solving the traveling salesman problem. Consider a clearance task where the robot must push n items out of a defined goal region. We can easily define and compute $\hat{d}_{move}$ as the sum of the distance of the n objects from the edge of the goal region. Computing $\hat{d}_{con}$ is more difficult. We must compute the shortest path for the robot to move between objects. An alternative is to define $\hat{d}_{con}$ as the distance to the furthest object. This is easy to compute and still provides a lower bound, though admittedly looser.

Additionally, we would like to loosen our assumptions in order to expand applicability of this algorithm. Assumption 1 is particularly unsatisfying because it prevents the whole-arm interaction that is useful in many rearrangement problems. We believe we can remove this assumption by computing $\hat{d}_{con}$ as the distance between the object and the *closest* point on the manipulator. Computing this distance is computationally more difficult. This will have meaningful impact on overall planning times because the heuristic is computed on every expansion.

To remove Assumption 2, we can adjust $\hat{d}_{con}$ to be the distance between the robot and the *closest* movable object rather than the distance between the robot and the goal object. To remove Assumption 3, we must eliminate $\hat{d}_{move}$ from the heuristic all together, as a single strike of an object may allow it to slide or roll all the way to the goal with no further movement of the robot.

It is clear that eliminating all three assumptions quickly weakens the power of the heuristic, reducing its ability to meaningfully guide the search. Still our experimental results with this planner inspire us to consider alternate meaningful heuristics that allow us to apply structured search to rearrangement planning.

List of Figures

List of Tables

B
Bibliography

[1] Open Dynamics Engine. `http://www.ode.org`, 2000 (accessed August 2014.

[2] NVIDIA PhysX: Physics simulation for developers. `http://www.nvidia.com`, 2009.

[3] Box2D. `http://box2d.org`, 2010 (accessed August 2014).

[4] Bullet physics library. `http://bulletphysics.org`, (accessed August 2014).

[5] P. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Non-holonomic path planning for pushing a disk among obstacles. In *IEEE International Conference on Robotics and Automation*, 1997.

[6] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1993.

[7] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Sensorless parts orienting with a one-joint manipulator. In *IEEE International Conference on Robotics and Automation*, 1997.

[8] S. Akella and M. T. Mason. Posing polygonal objects in the plane by pushing. In *IEEE International Conference on Robotics and Automation*, 1992.

[9] R. Alami, J. P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *Workshop on the Algorithmic Foundations of Robotics*, 1994.

[10] R. Alterovitz, T. Siméon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and Systems*, 2007.

[11] J-Y Audibert and S. Bubeck. Best arm identification in multi-armed bandits. In *COLT Conference on Learning Theory*, 2010.

[12] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[13] H. Bai, D. Hsu, W. Lee, and A. Ngo. Monte carlo value iteration for continuous-state POMDPs. In *Workshop on the Algorithmic Foundations of Robotics*, 2010.

[14] J. Barraquand and J. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 3:2328–2335, 1993.

[15] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Manipulation with multiple action types. In *International Symposium on Experimental Robotics*, 2012.

[16] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *IEEE International Conference on Robotics and Automation*, 2013.

[17] O. Ben-Shahar and E. Rivlin. Practical pushing planning for rearrangement tasks. In *IEEE International Conference on Robotics and Automation*, 1998.

[18] O. Ben-Shahar and E. Rivlin. To push or not to push: On the rearrangement of movable objects by a mobile robot. In *IEEE Transactions on Systems, Man and Cybernetics*, 1998.

[19] D. Berenson, S.S. Srinivasa, D. Ferguson, A. Collet, and J.J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, 2009.

[20] D. Berenson, S.S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation*, 2009.

[21] R-P Berretty, K. Goldberg, M.H. Overmars, and A. Frank van der Stappen. Orienting parts by inside-out pulling. In *IEEE International Conference on Robotics and Automation*, 2001.

[22] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling*, 2003.

[23] M. Brokowski, M. Peshkin, and K. Goldberg. Curved fences for part alignment. In *IEEE International Conference on Robotics and Automation*, 1993.

[24] R. C. Brost. Automatic grasp planning in the presence of uncertainty. *The International Journal of Robotics Research*, 7(1):3–17, 1988.

[25] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo Tree Search methods. In *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

[26] S. Caselli and M. Reggiani. ERPP: An experience-based randomized path planner. In *IEEE International Conference on Robotics and Automation*, 2000.

[27] A.R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Moarkov Decision Processes*. PhD thesis, Department of Computer Science, Brown University, 1998.

[28] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE International Conference on Robotics and Automation*, 1995.

[29] L. Chang, S. Srinivasa, and N. Pollard. Planning pre-grasp manipulation for transport tasks. In *IEEE International Conference on Robotics and Automation*, 2010.

[30] G.M.J.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo Tree Search: A new framework for game AI. In *Artificial Intelligence Interactive Digital Entertainment Conference*, 2008.

[31] P. C. Chen and Y. K. Hwang. Practical path planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, 1991.

[32] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. In *Journal of Artificial Intelligence Research*, 2000.

[33] M.R. Dogar, K. Hsiao, M. Ciocarlie, and S.S. Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and Systems*, 2012.

[34] M.R. Dogar and S.S. Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[35] M.R. Dogar and S.S. Srinivasa. A framework for push-grasping in clutter. In *Robotics: Science and Systems*, 2011.

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

[37] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Science & Business Media, 2001.

[38] A. Dragan, N. Ratliff, and S.S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *IEEE International Conference on Robotics and Automation*, 2011.

[39] L.E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[40] M. Erdmann. An exploration of nonprehensile two-palm manipulation: Planning and execution. *Seventh International Symposium on Robotics Research*, 1995.

[41] M. A. Erdmann. Using backprojections for fine motion planning with uncertainty. In *IEEE International Conference on Robotics and Automation*, 1985.

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

[43] C. Van Geem, T. Siméon, and J-P. Laumond. Mobility analysis for feasibility studies in cad models of industrial environments. In *IEEE International Conference on Robotics and Automation*, 1999.

[44] R.P. Goldman and M.S. Boddy. Expressive planning and explicit knowledge. In *International Conference on Artificial Intelligence Planning and Scheduling*, 1996.

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

[46] D.D. Grossman and M.W. Blasgen. Orienting mechanical parts by computer-controlled manipulator. In *IEEE Transactions on Systems, Man and Cybernetics*, 1975.

[47] L.J. Guibas, D. Hsu, H. Kurniawati, and E. Rehman. Bounded uncertainty roadmaps for path planning. In *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[48] M. Gupta and G.S. Sukhatme. Using manipulation primitives for brick sorting in clutter. In *IEEE International Conference on Robotics and Automation*, 2012.

[49] P.E. Hart, N.J. Nillson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[50] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *European Conference on Planning*, 1999.

[51] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *IEEE International Conference on Robotics and Automation*, 2010.

[52] J.A. Haustein, J.E. King, S.S. Srinivasa, and T. Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable configurations. In *IEEE International Conference on Robotics and Automation*, 2015.

[53] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *IEEE International Conference on Robotics and Automation*, 2014.

[54] H. Hitakawa. Advanced parts orientation system has wide application. In *Assembly Automation*, 1988.

[55] R.D. Howe and M.R. Cutkosky. Practical force-motion models for sliding manipulation. *The International Journal of Robotics Research*, 15(6):557–572, 1996.

[56] W. Huang and M.T. Mason. Mechanics, planning, and control for tapping. In *Workshop on the Algorithmic Foundations of Robotics*, 1998.

[57] N. Hyafil and F. Bacchus. Conformant probabilistic planning via CSPs. In *International Conference on Automated Planning and Scheduling*, 2003.

[58] J. Ichnowski and R. Alterovitz. Parallel sampling-based motion planning with superlinear speedup. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[59] A. Johnson, J.E. King, and S.S. Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051, 2016.

[60] Aaron M. Johnson, Samuel E. Burden, and D. E. Koditschek. A hybrid systems model for simple manipulation and self-manipulation systems. *arXiv preprint arXiv:1502.01538 [cs.RO]*, 2015.

[61] L. Kavraki and J-C Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation*, 1994.

[62] L. Kavraki, P. Svestka, J-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[63] J.E. King, M. Cognetti, and S.S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE International Conference on Robotics and Automation*, 2016.

[64] J.E. King, J.A. Haustein, S.S. Srinivasa, and T. Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *IEEE International Conference on Robotics and Automation*, 2015.

[65] J.E. King, M. Klingensmith, C. Dellin, M. Dogar, P. Velagapudi, N. Pollard, and S.S. Srinivasa. Pregrasp manipulation as trajectory optimization. In *Robotics: Science and Systems*, 2013.

[66] R.D. Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *Conference on Neural Information Processing Systems*, 2004.

[67] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, 2006.

[68] M.C. Koval, M.R. Dogar, N.S. Pollard, and S.S. Srinivasa. Pose estimation for contact manipulation with manifold particle filters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[69] M.C. Koval, J.E. King, N. Pollard, and S.S. Srinivasa. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[70] M.C. Koval, N.S. Pollard, and S.S. Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *Robotics: Science and Systems*, 2014.

[71] A. Krontiris and K. Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems*, 2015.

[72] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. Bekris. Rearranging similar objects with a manipulator using pebble graphs. In *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[73] J.J. Kuffner and S.M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 2000.

[74] H. Kurniawati and V. Yadav. An online POMDP solver for uncertainty planning in dynamic environment. In *International Symposium on Robotics Research*, 2013.

[75] N. Kushmerick, S. Hanks, and D.S. Weld. An algorithm for probabilistic planning. 76:239–286, 1995.

[76] S.M. LaValle. Rapidly-exploring Random Trees: A new tool for path planning. 1998.

[77] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[78] M. Levihn, J. Scholz, and M. Stilman. Planning with movable obstacles in continuous environments with uncertain dynamics. In *IEEE International Conference on Robotics and Automation*, 2013.

[79] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Robotics: Science and Systems*, 2008.

[80] Z. Littlefield, D. Klimenko, H. Kurniawati, and K. Bekris. The importance of a suitable distance function in belief-space planning. In *International Symposium on Robotics Research*, 2015.

[81] Winfried Lohmiller and Jean-Jacques E Slotine. On contraction analysis for non-linear systems. *Automatica*, 34(6):683–696, 1998.

[82] W. Lovejoy. A survey of algorithmic methods for partially observed Markov decision process. *Annals of Operations Research*, 28(1):47–65, 1991.

[83] B. Luders, M. Kothari, and J.P. How. Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *AIAA Guidance, Navigation, and Control Conference*, 2010.

[84] K. Lynch. Locally controllable polygons by stable pushing. In *IEEE International Conference on Robotics and Automation*, 1997.

[85] K. Lynch. Toppling manipulation. In *IEEE International Conference on Robotics and Automation*, 1999.

[86] K. Lynch and M.T. Mason. Controllability of pushing. In *IEEE International Conference on Robotics and Automation*, 1995.

[87] K. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. In *Workshop on the Algorithmic Foundations of Robotics*, 1995.

[88] K. Lynch and M.T. Mason. Dynamic nonprehensile manipulation: Controllability, planning and experiments. *The International Journal of Robotics Research*, 18(1):64–92, 1999.

[89] Y. Maeda, H. Kijimoto, Y. Aiyama, and T. Arai. Planning of graspless manipulation by multiple robot fingers. In *IEEE International Conference on Robotics and Automation*, 2001.

[90] O. Maron and A.W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Conference on Neural Information Processing Systems*, 1993.

[91] M.T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[92] M.T. Mason. Progress in nonprehensile manipulation. *The International Journal of Robotics Research*, 18(11):1129–1141, 1999.

[93] M.A. Melchior and R. Simmons. Particle RRT for path planning with uncertainty. In *IEEE International Conference on Robotics and Automation*, 2007.

[94] T. Mericli, M. Veloso, and H.L. Akin. Achievable push-manipulation for complex passive mobile objects using past experience. In *International Conference on Autonomous Agents and Multi-agent Systems*, 2013.

[95] G. Monahan. A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[96] D. Nieuwenhuisen, A. Stappen., and M. Overmars. An effective framework for path planning amidst movable obstacles. In *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[97] D. Nieuwenhuisen, A. Frank van der Stappen, and M.H. Overmars. Path planning for pushing a disk using compliance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[98] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *IEEE International Conference on Robotics and Automation*, pages 3400–3407, May 2011.

[99] J. Ota. Rearrangement planning of multiple movable objects by realtime search methodology. In *IEEE International Conference on Robotics and Automation*, 2002.

[100] J. Ota. Rearrangement planning of multiple movable objects - integration of global and local planning methodology. In *IEEE International Conference on Robotics and Automation*, 2004.

[101] R. Pepy, M. Kieffer, and E. Walter. Reliable robust path planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[102] M.A. Peshkin and A.C. Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6):569–598, 1988.

[103] M.A. Peshkin and A.C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5):524–531, 1988.

[104] R. Platt, L.P. Kaelbling, T. Lozano-Pérez, and R. Tedrake. Efficient planning in non-Gaussian belief spaces and its application to robot grasping. In *International Symposium on Robotics Research*, 2011.

[105] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1970.

[106] S. Prentice and N. Roy. The belief roadmap, efficient planning in linear POMDPs by factoring the covariance. In *International Symposium on Robotics Research*, volume 66, pages 293–305, 2008.

[107] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[108] G. Sánchez and J-C. Latombe. On delaying collision checking in PRM planning - application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

[109] K.M. Seiler, H. Kurniawati, and S.P.N Singh. An online and approximate solver for POMDPs with continuous action space. In *IEEE International Conference on Robotics and Automation*, 2015.

[110] S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *The International Journal of Robotics Research*, 17(8):840–857, 1998.

[111] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *Conference on Neural Information Processing Systems*, 2010.

[112] T. Siméon, J-P. Laumond, J. Cortés, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7–8):729–746, 2004.

[113] S.Jentzsch, A.Gaschler, O.Khatib, and A.Knoll. MOPL: A multi-modal path planner for generic manipulation tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[114] D.E. Smith and D.S. Weld. Conformant graphplan. In *AAAI Conference on Artificial Intelligence*, 1998.

[115] S.S. Srinivasa, D. Ferguson, C.J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M.V. Weghe. HERB: A Home Exploring Robotic Butler. *Autonomous Robots*, 28(1):5–20, 2010.

[116] M. Stilman. and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *IEEE-RAS International Conference on Humanoid Robots*, 2004.

[117] M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. In *Workshop on the Algorithmic Foundations of Robotics*, 2006.

[118] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner. Planning and executing navigation among movable obstacles. *Springer Journal of Advanced Robotics*, 21(14):1617–1634, 2007.

[119] M. Stilman, J. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, 2007.

[120] I. Sucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*, 19(4):72–82, 2012.

[121] S. Thrun. Monte Carlo POMDPs. In *Conference on Neural Information Processing Systems*, 2000.

[122] J.N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72, 2002.

[123] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path planning among movable obstacles: a probabilistically complete approach. In *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[124] C.C. White. A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.

[125] G. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 41(4):764–790, 1991.

[126] S. Zickler and M. Velosa. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *International Conference on Autonomous Agents and Multi-agent Systems*, 2009.

[127] C. Zito, R. Stolkin, M. Kopicki, and J.L. Wyatt. Two-level RRT planning for robotic push manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[128] N. Zumel and M. Erdmann. Nonprehensile two palm manipulation with non-equilibrium transitions between stable states. In *IEEE International Conference on Robotics and Automation*, 1996.

[129] N. Zumel and M. Erdmann. Nonprehensile manipulation for orienting parts in the plane. In *IEEE International Conference on Robotics and Automation*, 1997.