Reliably arranging objects: a conformant planning approach to robot manipulation

by

Ariel S. Anders

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctorate of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author . . . . . .
Department of Electrical Engineering and Computer Science
January 15, 2019

Certified by . . . . . .
Tomas Lozano-Perez
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . . . .
Leslie Pack Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Reliably arranging objects: a conformant planning approach to robot manipulation

by

Ariel S. Anders

Submitted to the Department of Electrical Engineering and Computer Science
on January 15, 2019, in partial fulfillment of the
requirements for the degree of
Doctorate of Philosophy in Electrical Engineering and Computer Science

Abstract

A crucial challenge in robotics is achieving reliable results in spite of sensing and control uncertainty. In this work, we explore the conformant planning approach to reliable robot manipulation. In particular, we tackle the problem of pushing multiple planar objects simultaneously to achieve a specified arrangement without using external sensing. A conformant plan is a sequence of manipulation actions that reliably achieve a goal arrangement in spite of uncertainty in object pose and nondeterministic action outcomes, and without assuming the availability of additional observations. To find conformant plans, we explored two different approaches:

Conformant planning by construction. This approach formalizes conformant planning as a belief-state planning problem. A belief state is the set of all possible states of the world, and the objective is to find a sequence of actions that will bring an initial belief state to a goal belief state. To do forward belief-state planning, we created a deterministic belief-state transition model from on-line physics-based simulations and supervised learning based on off-line physics simulations.

Conformant planning through plan improvement. This approach takes a deterministic manipulation plan and augments it by adding fixtures (movable obstacles) to push parts up against. This method uses an optimization-based approach to determine the ideal fixture placement location.

This thesis provides insight and develops approaches toward scalable methods for solving challenging planar manipulation problems with multiple objects or concave shape geometry. We show the success of these approaches based on planning times and robustness in real and simulated experiments.

Thesis Supervisor: Tomas Lozano-Perez
Title: Professor of Computer Science and Engineering

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Acknowledgments

I started my graduate studies at MIT in July 2012, when I was only 21 years old. To say I was somewhat lost and unsure of my research topic would be an understatement. I greatly appreciate my research advisors, Leslie Pack Kaelbling and Tomas Lozano-Perez, for their patience and support throughout all of these years.

Leslie and Tomas have been tremendous mentors and rolemodels. Together, they have shown a successful example of work-life balance. They are extremely dedicated to their research and passionately work on their research problems. Looking back at my project, Tomas helped by guiding me in my ideas and steering the research direction in robot manipulation. Leslie helped me complete the narrative of the project and asked questions which required answers that eventually bolstered and improved the quality of my work. They gave their time (about an hour every week!) and effort to help me complete my degree, from start to finish. To them, I express my hearty appreciation knowing that I will always use the skills they have taught me.

I am also very fortunate to have my final committee member be Sertac Karaman. Sertac offers wonderful insight on research, and I am fortunate to have also interacted with him through teaching undergraduate and highschool classes on autonomous vehicles.

At MIT, I worked in the Learning and Intelligent Systems (LIS) group. I have been extremely lucky with this research group because every member has made my degree more memorable. I was also fortunate to be a member of three communities on campus: Ashdown House, CSAIL Student Committee, and the University Center for Exemplary Mentoring (UCEM). My research group and these communities were my main social life while completing my studies.

Prior to MIT, I worked in the Bionics Lab under Jacob Rosen as an undergraduate at UC Santa Cruz. Professor Rosen was my first research mentor, and there is no doubt in my mind that I would never have embarked on my graduate studies without his encouragement and support during my undergraduate years. At UC Santa Cruz, I was involved with the National Society of Black Engineers (NSBE), Society of Women Engineers (SWE), and the STEM Diversity programs. Each of these groups contributed to my professional success.

Before I went to college, my time was mostly spent horseback riding. My first lessons in responsibility and work ethic were gathered raising my horse “Sam”. From middle school onwards, my relationship with Sam kept me on track in pursing my education. Around this time, I had two math teachers, Mr. Little and Mrs. Long, who realized my potential and helped me excel.

I am fortunate to have made incredible friendships along this journey through graduate school. My friends have made graduate school so much more than just an education. I want to mention a few colleagues and peers who have truly made this experience special: Sarah Ross, Lawson Wong, Patrick Barragan, Jennifer Barry, Gustavo Goretkin, Elena Glassman, Rohit Singh, David Rosen, Gautaum Kamauth, George Konidaris, Owen MacIndoe, Ilia Llebedev, Garrett Wollman, Teresa Cataldo, Chris Amato, Brian Axelrod, Candace Ross, Zi Wang, Caris Moses, Caelan Garrett, Rohan Chitnis, Abhishek Agarwal, Corey Walsh, Winter Guerra, Jane Conner, Jocie Kluger, Alin Tomescu, Michal Grzadkowski, Frank Permenter, Marie Giron, Ifueko Igbinedion, Danielle Pace, Kyle Wilke, Elise Strobach, Raghav Argarwal, Zelda Mariet, and Andie Lehn. This list is mostly in order of appearance.

Finally, I would like to acknowledge the people closest to me: my significant other, Daniel Preston, and my family. Dan has made graduate school much more enjoyable with his partnership, enthusiasm, and passion. My siblings and parents have always provided unconditional love and support while I was working on my graduate studies. My parents worked extremely long hours to ensure their children had a worry-free childhood, encouraged me to shoot for the stars, and cheered me on along the way; my parents are the main reason I have completed my doctorate education.

Biography

Ariel Anders is a roboticist with almost a decade of experience blending the fields of robotics and computer science. As a PhD candidate at MIT, she researched robust robotic manipulation methods with the vision of enabling household helpers. As a teaching assistant at MIT she has taught three undergraduate hands-on robotics laboratory courses and instructed two highschool autonomous vehicle classes. In fall 2017, she lectured a mechanical engineering

course on Dynamics at Olin College of Engineering. Her passions extend outside research and education: at MIT, she helped found a clean energy assessment center, developed an iOS app for people with vision impairment, and won a grant to install therapy lamps across campus.

Ariel attained a Bachelor of Science in Computer Engineering from the University of California: Santa Cruz in 2012. AT UC Santa Cruz she worked in the Bionics Lab on dental robotics under Professor Jacob Rosen. She earned a Master of Science in Electrical Engineering and Computer Science from MIT in 2014 on reinforcement learning for grasping. Ariel is originally from Bakersfield, California, where she discovered her love of horseback riding.

Contents

List of Figures

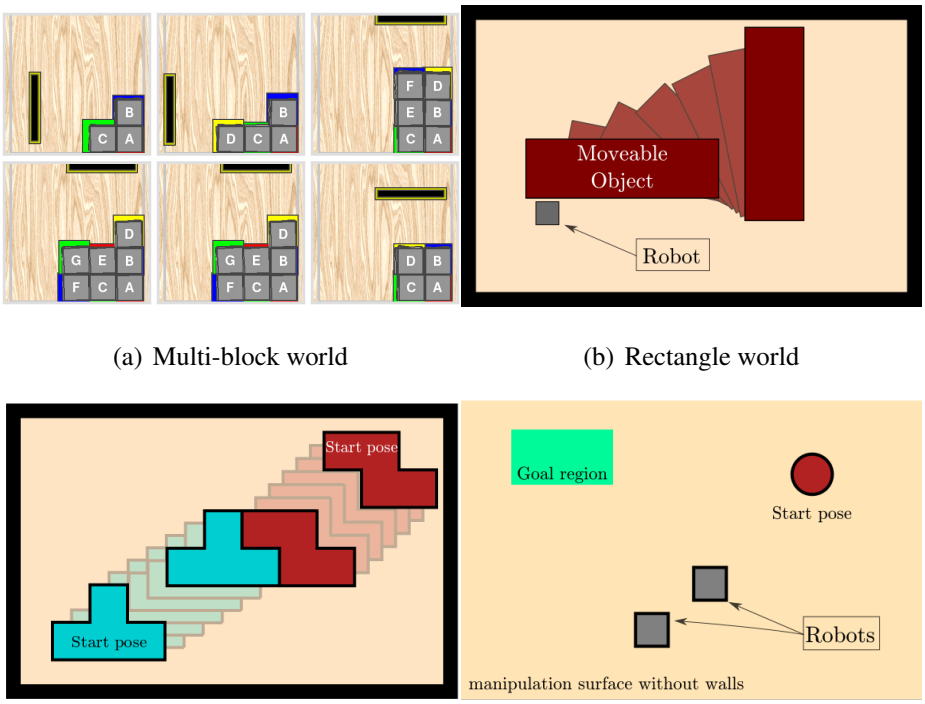
List of Tables

Chapter 1

Introduction

Constructing multi-object arrangements is an important component of a variety of robot tasks: arranging boxes in a warehouse, placing groceries on a shelf, packing objects in boxes, etc. The fundamental goal in these tasks is to achieve a set of relative position constraints among a set of parts; in general, these constraints require contacts among multiple objects. These contacts are difficult or impossible to perceive visually and difficult or impossible to achieve via unconstrained open-loop positioning actions.

Figure 1-1: Robot arrangement task. *Real robot reliably assembling an arrangement. First the robot places a block, then pushes it using a paddle.*



(a) Multi-block world (b) Rectangle world

(c) T and Z shapes (d) Circle circle square square

Figure 1-2: Experimental manipulation domains. *Four example tabletop manipulation domains.*

A key strategy for robustly constructing object arrangements is to use actions, such as pushing and other compliant motions, that achieve desired contact relationships between objects by *exploiting the task mechanics*. For example, the robot can achieve contact between two object faces by pushing one up against the other as in Figure 1-1, or achieve an insertion by using a remote-center compliance strategy. These actions have a tendency to act as "funnels," [34, 31] which can map a large set of possible initial configurations into a more compact set.

In general, the result of these actions is not predictable, depending on the actual (partially observed) initial state of the world and (unobserved) properties of the robot and objects. Given this setting, we are faced with the question of how to plan sequences of actions that can achieve a desired goal state without the ability to sense the precise outcome of each action. We frame this problem as one of *conformant planning*: given a set of possible initial object configurations, a set of actions with non-deterministic outcomes, and a set of goal configurations, the objective is to find a sequence of actions that is guaranteed to drive the

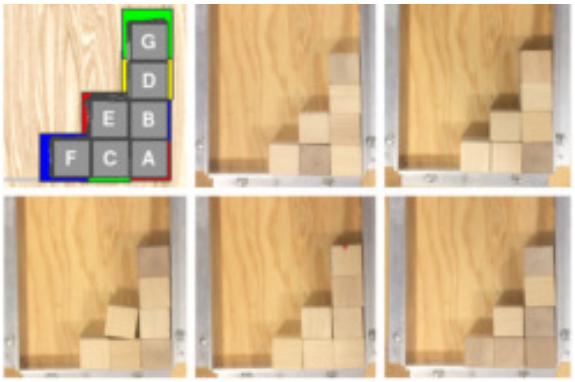


Figure 1-3: Real robot execution. *Robust execution of 7 block arrangement on a real robot; results shown for 1000 noisy simulated executions and 5 executions on a real robot.*

objects into a configuration satisfying the goal.

The problem of conformant planning can be seen as a search through a space of *belief states*, which are sets of possible configurations of the objects in the arrangement. Each primitive action is modeled using a transition function that maps an initial belief state into a resulting belief state; the resulting belief state is the union of the possible configurations resulting from applying that action primitive to every configuration in the initial belief state.

For all but the simplest primitive actions in the most ideal circumstances, it is difficult to analytically derive an accurate and exact belief-state transition function, because it depends on interactions among multiple objects which may be affected by detailed physical properties of the objects and robot.

Instead, we present two different approaches for approximating belief-state transitions. Our first approach relies upon multiple physics-based simulations applied to a set of states; these states are analogous to "particles" in a particle filter. In the presence of uncertainty, we can use multiple simulations with added noise to model belief-state transitions. The alternative approach uses machine-learning methods to acquire belief-state transition models from physics simulations prior to planning. We learn transitions for the belief state of individual objects given local context, called *compositional belief-state transition models* (CBSTs), and then compose these to construct a belief-state transition model for the overall system.

Although we demonstrate the learned model in the context of conformant planning, with

no external sensing, it could be used to implement a belief-space replanning strategy [38] that would incorporate intermediate sensing. The conformant setting offers a simpler, cleaner framework for evaluating the effectiveness of these learned models.

This thesis is organized as follows. In Section 1.1 we note the related work and compare our technical problem with prior work. In Section 1.2 we provide the background material on the *Arrangement Problem* (AP). Then, in Chapter 2 we introduce a specific instance of the AP that we used to analyze our approach and develop experiments. Chapters 3 and 4 presents different methods for solving the AP and corresponding experiment results. In Chapter 5, we summarize and reflect on this work and indicate some possible directions for future work.

1.1 Related work

Our approach to conformant planning for robot manipulation combines a variety of different fields: physics modeling of manipulation, planning under uncertainty, and machine learning. The academic community has studied each of these independent fields and acquired a tremendous amount of knowledge, which has been written in comprehensive textbooks. For modeling manipulation, we recommend Mason [33] for a helpful overview and Eberly [9] for a review of fast dynamic computations used in physics game engines. In the field of planning under uncertainty, we recommend LaValle's *Planning Algorithms* textbook [30] for a review of traditional planning techniques and Thrun and Fox's well-known *Probabilistic Robotics* textbook [44] for handling uncertainty within robotic systems. Finally, for an overview of traditional supervised machine learning work that is used in this work, we recommend Friedman *et al* [13] and Bishop [4] in addition to the many readily available online resources.

Although the fields of modeling dynamics, planning, and learning have been studied extensively, the study of the intersection of these areas is a space for active research. One element of this intersection is conformant planning for manipulation; a key challenge in this area is to understand the task mechanics well enough to be able to choose actions to achieve the desired outcomes. A number of operations, most notably pushing, have been the subject of extensive study, and analytic models have been derived for the task mechanics. In many cases, however, the outcomes of actions depend on physical properties that we do not have access to, such as pressure distributions or frictional coefficients. Nevertheless, in some cases the understanding of task mechanics can be exploited to plan robust manipulation. For example, Lynch and Mason [32] showed how to exploit multiple contacts to produce stable pushing trajectories and Dogar and Srinivasa [8] showed how to reliably funnel a range of initial locations of an object with known shape into a target location in the hand.

Recently, there have been a number of examples of *physics-based manipulation*, which exploit the availability of physics simulation engines, originally developed for computer games, to predict the effect of actions in situations where analytic methods would be cumbersome at best. The work on rearrangement planning [20] addresses pushing an object

to a target location in the presence of "clutter", objects that are allowed to be pushed out of the way. Planning these operations relies on being able to predict, using physics simulations, the motion of multiple interacting objects being pushed by the robot. Related work on "grasping through clutter" also relies on such simulations [7, 21]. This work is focused on placing or grasping a single object, even though several "clutter" objects will end up being moved in the process; it does not address the goal of achieving a final arrangement of multiple objects.

An alternative to using analytical models or on-line simulation for physics-based planning is to learn compact models from experience (real or simulated). In the observable case, where the state of the world after an action can be determined, this is a simple regression problem: given many observations of *(state, action, next state)*, learn a function to predict the *next state* given *(state, action)*. Given such a learned function, planning proceeds as before. A number of instances of this approach exist [39, 10, 35].

There are alternative approaches to this idea of learning a model and using it for planning. For example, it is also possible to learn a policy directly and bypass planning. Laskey *et al* [29] learned a policy for grasping in clutter that had to handle multiple object interaction during the manipulation task. However, such learned policies tend not to generalize as well as learning a model and then planning. Additionally, this work assumed the initial state is known and that actions are reliable.

Outside the context of multi-step manipulation planning, there has been work in learning to predict the effect of object interactions without full state information [22, 12, 26]. For example, Kroemer *et al* [26] worked on classifying object contact distributions to predict whether a set of inputs would yield a stable grasp or place. This contact classifier in conjuction with sampling object positions enabled a 1-step planner that could be used sequentially to stack objects resting on a table.

Even without considering uncertainty, there is the problem of planning the sequence of manipulation actions to arrange objects into a desired configuration. Stillman and Kuffner reasoned about a single robot navigating from a start location to a goal location where the robot had the ability to move a set of movable obstacles that were in the way [42]. Barry looked at manipulation problems that specified the initial and final configurations for the

Problem characteristics	This thesis	King, Koval *et al* [19, 23]	Dogar *et al* [8, 7]	Elliot *et al* [10]	Koval *et al* [23, 25]	Krontiris and Bekris [27]
Multiple objects in goal arrangement	X					X
Multiple object interaction during manipulation	X	X				
Learned/estimated object-contact models	X		X	X	X	
No external sensing	X	X	X		tactile	

Table 1.1: **Closely related work problem comparison.** *This table compares the most closely related work in conformant manipulation. This thesis tackles a technical problem that has a variety of challenging characteristics, whereas prior work only has a subset of these characteristics.*

robot and objects; created a sampling-based algorithm that found a plan using multiple types of manipulation action like grasping, tilting, and pulling [3]. Krontiris and Bekris defined the *prehensile rearrangement problem*, where the robot could pick and place objects to achieve a desired arrangement; however, this work did not allow for any collisions between objects [27]. Additionally, many of these rearrangement problems can be solved using a task and motion planner such as the one described in the work by Garret *et al* [14].

The work described typically assumes that the initial state is known and the actions are reliable. In the presence of uncertainty in the initial placements of objects or in the outcome of the actions, we are interested in finding actions, such as pushing and compliant motions, that reliably achieve the goal state in spite of this uncertainty and without assuming the availability of additional observations. This class of problems is known as *conformant planning*; it has been explored in robotics for planning compliant motions [31] and sequences of tray tilting operations [11] and, more recently, for rearrangement planning [24, 17].

Conformant planning is only one paradigm in the regime of planning under uncertainty. In robot manipulation "planning under uncertainty" is the problem of choosing actions in a partially observable stochastic domain, which is formalized as a Partially Observable

Markov Decision Process (POMDP). The solution to a POMDP is a policy, which is a mapping from histories of observations to actions. This idea of selecting actions based on immediate or historical sensory information is *contingent planning*. There are known algorithms for optimally solving a POMDP, however, these problems are PSPACE complete for problems with discrete state and action spaces and undecidable for continuous state and action spaces [5]. Since manipulation occurs in a continuous state and action space alternative approaches are considered.

Instead of solving a POMDP, there are various strategies that solve a relaxed problem, like assuming the most likely state and most likely observations will occur and handle errors by replanning. In replanning, a robot executes a step of the plan, then observes the effect of the execution and replans if the effect was not the predicted one, until the goal is achieved or the robot reaches an irreversible state (like dropping or breaking an object). This approach has the benefit that the planning problem is easier, because planning in a deterministic domain is much easier than in a non-deterministic one, however, replanning relies upon the ability to observe the results of intermediate steps in the plan.

Both contingent planning and replanning require observations. In manipulation problems, one common source of observations are intensity or depth cameras. In manipulation it may not be possible to use vision sensors due to occlusions: the robot may block the viewpoint of the camera with its arms during the manipulation task. Additionally, extracting state information from images remains a difficult problem and inaccuracies might contribute to a lack of robustness in the overall manipulation strategy.

This leads to the question: is it possible to do manipulation without frequent sensing? Conformant planning, often called sensorless planning, is a potential solution. In conformant plans, a robot takes a sequence of actions that are guaranteed to achieve the goal. An example of this is pushing an object into a corner to reduce the possible locations of the object. Conformant plans, by definition, are robust to uncertainty; however, there are cases when a conformant plan may not exist or be prohibitively expensive to find.

More broadly, the notion of conformant planning has been explored in the AI community starting with Kushmerick *et al* [28], who addressed it within the framework of probabilistic planning, and Goldman and Boddy [15] who used expressions in a logic of knowledge to

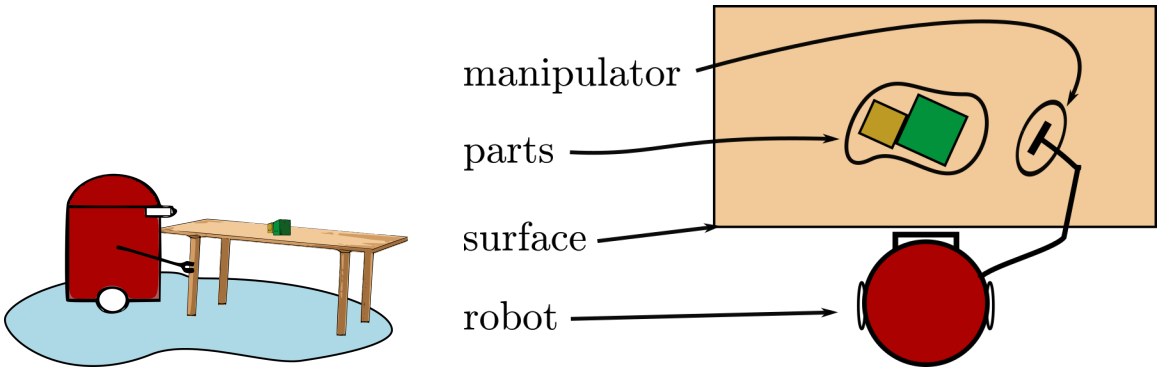
characterize belief states. Early work from theoretical computer science [37] shows that finding a finite-horizon optimal policy for a completely unobservable MDP is NP-complete, making this class of problems more efficient to solve than POMDPs in general. Yu *et al* [45] found that conformant planning is an effective strategy for a multi-robot "tag" domain.

Decision making when there is uncertainty falls into these three categories of contingent planning, replanning, and conformant planning. Most manipulation applications requires some combination of the three when applied for a real robotic system. In this thesis, we are focusing on the specific case of conformant planning; however, it is possible to take advantage of some of the methods and representations developed in this thesis within other approaches. For example, conformant plans could be used in a replanning framework to improve the likelihood of success. Additionally, the prediction methods we use in conformant planning are a common component in most contingent algorithms.

The conformant arrangement problem we study in this work has some characteristics that distinguish it from prior work listed above. Most significantly, we allow problems that have multiple objects in the goal arrangement. Additionally, we allow multiple objects to come into contact and interact with each other during manipulation, and we model the motion of this object interaction for use in planning. Finally, our work assumes no external sensing during the manipulation actions. Table 1.1 provides an overview of the most similar related work and compares the technical problems that are tackled in each work. As you can see, our work is unique in that our technical problem encompasses each of these challenging characteristics.

1.2 Background on the arrangement problem

The *Arrangement Problem* (AP) is a planning problem in which a robot constructs an *arrangement* from a set of *parts* on a flat *manipulation surface*. The robot interacts with these parts using one or more *manipulators*. Figure 1-4 shows an overview of the components in an arrangement problem. There are related "rearrangement" problems that have been studied in prior work[27, 19]; however, we address a different class of problems, which we define formally below.



(a) Arrangement problem side view

(b) Arrangement problem top view

Figure 1-4: Arrangement problem components. *Figure (a) shows the side view of an arrangement problem and Figure (b) shows the top view of an arrangement problem.The robot (in red) uses a* manipulator *to interact with* parts *resting on the* manipulation surface.

The overall objective in the AP is to construct an arrangement based on initial and final state specifications. Within this objective, there are multiple criteria for constructing a good plan. As a *planning problem*, one criterion is to find plans with a minimal amount of computation; an additional criterion is to find optimal plans that have the fewest actions or lowest overall cost if actions have an associated cost. In the AP we will assume the domain is deterministic so reliability will not be an issue. In Section 1.2.2, we will extend AP to define problems that have non-deterministic actions and state uncertainty.

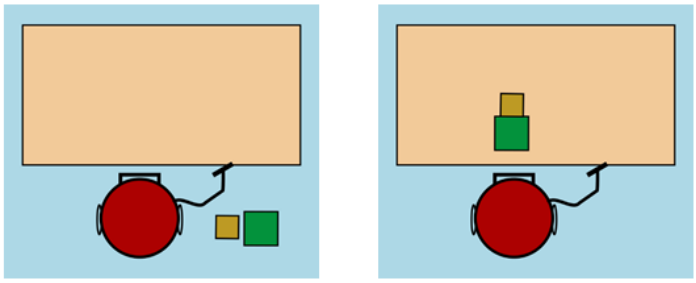


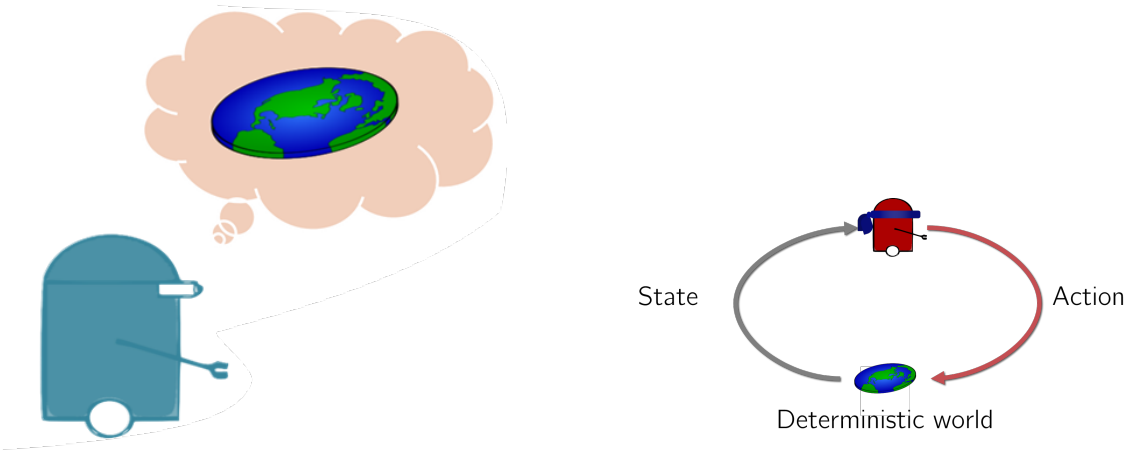
Figure 1-5: Arrangement problem goal. *An arrangement problem specifies an initial state of the world and a desired arrangement.*

In this work, we use predictive models to represent object motion given a robot's manipulation actions. The planner uses these models in a search algorithm to find plans.

In Section 1.2.1 we formally define the AP which assumes the world is deterministic and uses a deterministic state transition model of the world for planning.

In Section 1.2.2, we will address the *Conformant Arrangement Problem* (CAP) where the world is nondeterministic, but we use a deterministic belief-state transition model that describes this stochastic behavior of the world.

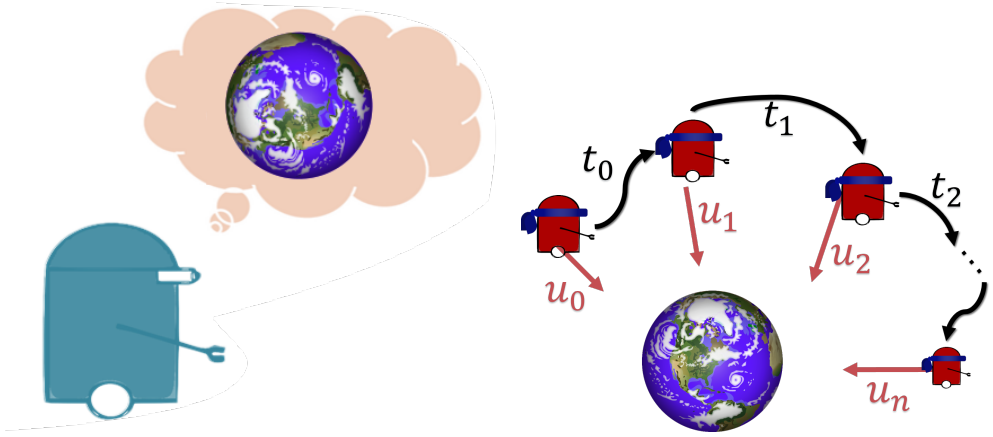
The distinction between the CAP and AP is elucidated in Figure 1-7. In both cases a solution to the AP or CAP is a sequence of manipulation actions that constructs a desired arrangement. No additional sensing actions are considered in the planning framework.



(a) Deterministic world model for planning

(b) Deterministic world model plan execution

Figure 1-6: Deterministic world model. *Figure (a) shows a robot planning with a deterministic world model that does not have any uncertainty. This deterministic model of the world is simplistic and incorrect. Figure (b) represents executing a deterministic plan: the robot assumes the world transitions based on the incorrect deterministic model. Note that the robot has a blindfold to represent the lack of sensing actions.*



(a) Stochastic world model for planning

(b) Stochastic world model plan execution

Figure 1-7: Stochastic world model. *Figure (a) represents planning with a nondeterministic world model. This model may also be simplified (since we can never represent the true world dynamics), but the robot will have to consider nondeterministic action effects and lack of state information. In Figure (b) the robot executes a* conformant *plan on the nondeterministic world, but never assumes state information. Similarly, the robot wears a blindfold to represent the lack of sensing actions taken during plan execution.*

1.2.1 Deterministic arrangement problem

The AP is a hybrid motion planning-problem [30], with actions that are defined by motion primitives and a discrete-time model. The AP formulation also assumes that some of the manipulation primitives will handle the introduction and removal of objects onto the manipulation surface. The objective is to find an optimal, or near optimal, sequence of manipulation primitives that achieve a goal configuration.

Arrangement Problem **formulation**

- $\mathcal{W}$ is the manipulation surface, where $\mathcal{W} \subseteq \mathcal{R}^2$; this surface is a rigid body object with some defined geometry.
- Ω is a set of known rigid objects; Ω contains the set of $\mathcal{O}$, $\mathcal{P}$, and $\mathcal{A}$.
  1. $\mathcal{O}$ is a set of fixed obstacles on the manipulation surface.
  2. $\mathcal{A}$ is a set of r manipulators used by the robot. Each manipulator is represented as $\mathcal{A}^i$ and $\mathcal{A} = \{ \mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^r \}$
  3. $\mathcal{P}$ is a set of p parts that make up the desired arrangement. Each part is represented as $\mathcal{P}^i$ and $\mathcal{P} = \{ \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^p \}$.
  4. Each manipulator, $\mathcal{A}^i$, and part, $\mathcal{P}^i$, located on the manipulation surface has a pose in $\mathcal{SO}_2$.
  5. $\mathcal{C}_F$ is a set of collision-free complete configurations of all parts and manipulators, a subset of the whole configuration space $\mathcal{C}$; we additionally define $\mathcal{C}_O$, where $O \subseteq \Omega$, to be the space of collision-free configurations of the subset of objects O.
- x_{obj} is an *object* state, where $x_{obj} = \langle id, q_{id} \rangle$ when the object is on the manipulation surface or $x_{obj} = \langle id, None \rangle$ when the object is not on the manipulation surface. The first element in the state tuple, id, represents a unique string object identifier like "BlockA". The latter element q_{id} represents the configuration of object id.

World state $\mathbf{x}$ is a tuple of movable object states: $\mathbf{x} = \langle x_{obj_0}, x_{obj_1}, \ldots, x_{obj_n} \rangle$. Note that the movable objects consist of both the manipulators and parts.

Actions $\mathcal{U}$ is the set of all possible actions defined $\mathcal{U} = \mathcal{U}_{\Omega_{place}} \cup \mathcal{U}_{\Omega_{pick}} \cup \mathcal{U}_{push}$. Each action $u_k \in \mathcal{U}$ is a *motion primitive*.

- $\mathcal{U}_{\Omega_{place}} = \cup_{o \in \Omega} \mathcal{U}_{o_{place}}$, where $\mathcal{U}_{o_{place}}$ is a set of actions that introduce object o onto the manipulation surface and is defined for each $o \in \Omega$.
- $\mathcal{U}_{\Omega_{pick}} = \cup_{o \in \Omega} \mathcal{U}_{o_{pick}}$, where $\mathcal{U}_{o_{pick}}$ is a set of actions that remove object o from the manipulation surface and is defined for each $o \in \Omega$.
- $\mathcal{U}_{push}$ is a set of actions that change the poses of some or all objects currently in the arrangement.

Transition function $\mathcal{F}$ is the set of transition functions, where $\mathcal{F}$ is the union of the transition functions $f_{place,u,o,O}$, $f_{pick,u,o,O}$, and $f_{push,u,O}$. We assume that an action, u_k, deterministically modifies the current state $\mathbf{x}_k$, the discrete-time model is of the form:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, u_k).$$

Initial state $\mathbf{x}_{init}$ is the initial world state (for the deterministic AP only), defines an object state for all of the parts and manipulators relevant to the problem. Note that the initial state does not have to have any parts or manipulators enabled on the manipulation surface, it could simply be $\mathbf{x}_{init} = (\langle id_1, None \rangle, \ldots, \langle id_n, None \rangle)$.

Goal condition $G \subset \mathcal{C}_F$ is the set of states that are successful assemblies. A state $\mathbf{x}$ is a goal state if it satisfies an absolute or relative goal criterion specified by the problem.

Given an *Arrangement Problem*, a solution is a sequence of actions:

$$\pi = u_1, u_2, \ldots u_k.$$

The solution must have the property that forward simulating the actions from the initial state results in a goal satisfying state.

1.2.2 Conformant arrangement problem

In this section we extend the deterministic *Arrangement Problem* into a *Conformant Arrangement Problem* (CAP), where the goal is to find a conformant plan, which is an open loop sequence of actions capable of constructing a desired arrangement when there is uncertainty in the domain.

The problem of conformant planning can be seen as a forward search in *belief space*; a belief space (or information space [30]) is a space of elements that characterize the robot's information or uncertainty about its domain. Elements in a belief space are typically subsets of, or probability distributions over, the underlying world state space. In this work, we will let the belief space $\mathcal{B} = \mathcal{P}(\mathcal{C}_F)$, that is, the set of all subsets of $\mathcal{C}_F$, and $\mathcal{B}_O = \mathcal{P}(C_O)$ be the restriction to subset of objects O.

A node in this forward search is a *belief state*, which represents the set of possible configurations for objects in the arrangement. Each primitive action is modeled using a transition function that maps an initial belief state into a resulting belief state; the resulting belief state is the union of the possible configurations resulting from applying that action primitive to every configuration in the initial belief state. A conformant plan is a sequence of actions that brings an initial belief state into a goal-satisfying belief state; a belief state is goal-satisfying if all possible states it represents are in a goal-satisfying configuration.

Conformant Arrangement Problem formulation

In the CAP, the manipulation surface, objects, and actions $(\mathcal{W}, \Omega, \mathcal{U})$ are the same as in the AP. To transform the AP into a conformant problem, the planning state, transition function, initial planning state and goal condition transform as follows:

Planning state b is a belief state, which represents all possible object configurations in the world. In this work, we assume there is no uncertainty in determining which movable objects (parts and manipulators) are located on the manipulation surface. We will let the belief space $\mathcal{B} = \mathbf{P}(\mathcal{C}_F)$, where $\mathbf{P}$ denotes the *powerset* operator; in this case, it defines the set of all subsets of $\mathcal{C}_F$. We additionally define $\mathcal{B}_O = \mathbf{P}(C_O)$ to be the restriction to subset of objects O.

Transition function $f' : \mathcal{B} \rightarrow \mathcal{B}$, is a mapping from a set of configurations into the set of possible resulting configurations. This model is derived from $f_{ND} : \mathcal{C}_F \rightarrow \mathbf{P}(\mathcal{C}_F)$, where f_{ND} is a nondeterministic mapping from a configuration into a set of configurations. Thus, the transition function f' that maps a belief state into a resultant belief state is directly constructed as:

$$f'(b) = \bigcup_{c \in b} f(c) \ .$$

The nondeterministic functions, f_{ND}, are the following:

- $f_{place,u,o,O} : \mathcal{C}_O \rightarrow \mathbf{P}(\mathcal{C}_{O \cup \{o\}})$ is a transition function characterizing the addition of part o to the arrangement currently consisting of parts in O by mapping a configuration in $\mathcal{C}_O$ to a set of configurations that is a subset of $\mathcal{C}_{O \cup \{o\}}$ since there are a set of possible placed object states due to non-determinism in the place action; note that we don't expect the configuration of objects currently on the manipulation surface to change during the place action.
- $f_{pick,u,o,O} : \mathcal{C}_O \rightarrow \mathbf{P}(\mathcal{C}_{O \setminus \{o\}})$ is a transition function characterizing the removal of part o from the arrangement currently consisting of parts in O. Note that there is no uncertainty in the ability to pick the object from the table. We assume that this object will be removed and that no other objects on the manipulation surface will be modified.
- $f_{push,u,O} : \mathcal{C}_O \rightarrow \mathbf{P}(\mathcal{C}_O)$ is a transition function characterizing the effects of manipulation action u on objects in the current arrangement.

Initial state b_{init} is the initial planning state, which specifies all possible configurations for objects currently on the manipulation surface.

Goal condition $G \subset \mathcal{C}_F$ is the set of states that are successful arrangements. A belief state, b, is a goal belief state if all possible configurations represented by the belief state are goal satisfying states.

1.2.3 Approximate belief-state representations

In a continuous configuration space, it will be impossible to finitely represent all possible subsets of $\mathcal{C}_F$, and so given any concrete domain we will have to make a choice about how to represent elements of $\mathcal{B}$. In this work we explore two approximate belief-state representations. First, we present a particle-based belief-state representation and then we explore a factored belief-state representation. We finish this section with a discussion on the trade-offs between different belief-state representations.

Particle Belief-State Representation

One approximation to a belief-state is to use a sample based approach that is akin to "particles" in a particle filter. A world state ($\mathbf{x}$) as defined in the AP specifies the configuration of every movable object in the workspace. To approximate a belief state, b, we will sample world states or "particles". We define the parameter for this representation as P, the number of particles used to represent a belief state. Thus, $\hat{b} = \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P \}$, where each state $\mathbf{x}_i$ is a sampled world state from the true belief state b. Sub-figure 1-9(b) illustrates a particle-based representation of the belief state for a single object.

This sample-based representation may require a large number of particles to appropriately approximate the belief state. This leads to the potential problem that the particle representation may under-represent the true belief state when using an insufficient number of particles.

Factored Belief-State Representation

An alternative representation for approximating a belief state is to factor it into independent components. We represent every object on the manipulation surface with an individual object belief. Then, the composed world belief-state is a tuple of object beliefs. We can use a variety of parameterizations to representation each individual object belief, such as a set of particles; in this work, we considered an interval representation to represent the belief of an object as shown in Figure 1-8.

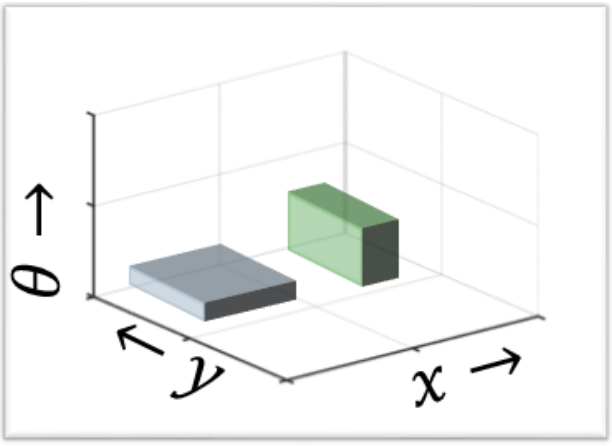


Figure 1-8: Factored belief-state in $\mathcal{SO}_2$ represented with intervals. *This picture shows the belief-state representation for two objects.*

Formally, if we think of a configuration $q = (\ q_{\mathcal{A}}^1, q_{\mathcal{A}}^2, \ldots, q_{\mathcal{A}}^r\ ,\ q_{\mathcal{P}}^1, q_{\mathcal{P}}^2, \ldots, q_{\mathcal{P}}^p\ )$, where $q_{\mathcal{A}}^i$ is the configuration of a manipulator and $q_{\mathcal{P}}^i$ is the configuration of a part, then $\mathcal{C} = {\mathcal{C}_{\mathcal{A}}}^1 \times \ldots {\mathcal{C}_{\mathcal{A}}}^r \times {\mathcal{C}_{\mathcal{P}}}^1 \times \ldots {\mathcal{C}_{\mathcal{P}}}^p$ is the Cartesian product of the configuration spaces of the individual objects. We cannot generally represent $\mathcal{C}_F$ as a Cartesian product due to collisions between the objects.

This decomposition leads us to the idea of representing the belief space as the product of independent beliefs about the configuration of each individual object, so that $\widehat{\mathcal{B}} = \widehat{\mathcal{B}}_1 \times \ldots \times \widehat{\mathcal{B}}_n$, where $\widehat{\mathcal{B}}_i \subset \mathcal{P}(\mathcal{C}_i)$. Representing sets of possible configurations of each object rather than sets of possible complete configurations is much more compact, although not as expressive. For example, we must augment such a factored representation with a constraint that the entire configuration must be collision free, so we will in general define

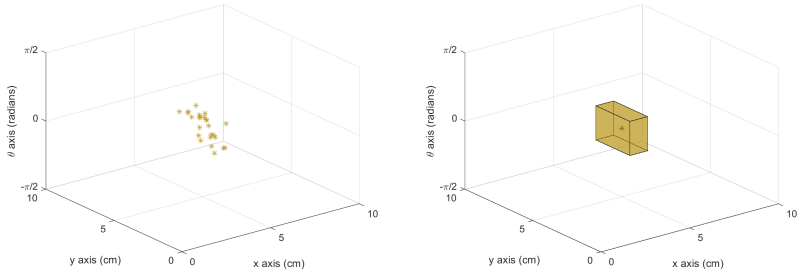
$$\widehat{\mathcal{B}} = (\widehat{\mathcal{B}}_1 \times \ldots \times \widehat{\mathcal{B}}_n) \cap \mathcal{C}_F \quad .$$

Using this factored belief space, we denote an approximate belief state as

$$\hat{b} = \langle \hat{b}_1, \ldots, \hat{b}_n \rangle.$$

Belief-state representations summary

We can use different parameterizations to represent a robot's uncertainty about the world. Some of the most popular methods use parameterized probability distributions to represent a robot's uncertainty. In this work we focused on set-based belief representations rather than probability distributions. A plan is conformant, if it reaches a goal-satisfying configuration regardless of initial state uncertainty and nondeterministic actions. In conformant planning we are not interested in the most likely outcome— instead, we are interested in forcing all outcomes into goal configurations; this is why our methods focus on sets rather than distributions.



(a) particle object belief (b) interval object belief

Figure 1-9: Approximate object belief state comparison. *Sub-figure (a) shows a particle-based representation of an object's belief. Sub-figure (b) shows the interval-based object belief representation of the same object.*

We present a factored interval representation and a non-factored particle representation for world belief states. The factored interval representation compactly represents a large variety of potential object states; whereas the particle-based representation may require a prohibitive number of particles to accurately represent a belief state, and intuitively, this number increases as the number of objects in the world increases. Although the factored belief state represents a large range of potential world states, some of these world states may be invalid due to collisions or may not be present in the true belief state. We must augment the factored representation with the collision free constraint.

Approximate belief-state transition functions

We model each primitive action using a transition function that maps an initial belief state into a resulting belief state; the resulting belief state is the union of the possible configurations resulting from applying that action primitive to every configuration in the initial belief state.

For all but the simplest primitive actions in the most ideal circumstances, it is difficult to analytically derive a belief-state transition function because it depends on interactions among multiple objects which may be affected by detailed physical properties of the objects and robot.

Instead, we present two different approaches for approximating belief state transitions. Our first approach relies upon multiple physics-based simulations, that is similar to "particles" in a particle filter. In the presence of uncertainty, we can use multiple simulations with added noise to model belief-state transitions, which we call *particle belief-state transition models* (PBSTs). The alternative approach uses machine learning methods to create belief-state transition models from physics simulations prior to planning. We learn transitions for the belief state of individual objects given local context, called *compositional belief-state transition models* (CBSTs), and then compose these to construct a belief-state transition model for the overall system. We show concrete examples of the PBST and CBST transition methods in Section 3.1 and Section 3.2, respectively.

Chapter 2

Planar tabletop conformant arrangement problem

In this section we provide concrete examples of the *Conformant Arrangement Problem* in a planar tabletop manipulation setting. In section 2.2.2 we describe useful distance metrics for manipulation problems to use with a heuristic search algorithm. Then, in section 2.4, we describe the implementation of the real world and simulated domain.

2.1 Arrangement problem components

As stated in Section 1.2.1, we define an arrangement problem with the tuple $(\mathcal{W}, \Omega, \mathcal{U}, \mathcal{F}, b_{init}, \mathcal{G})$. This section shows how we represented each of these components for tabletop manipulation.

2.1.1 Objects and state representation

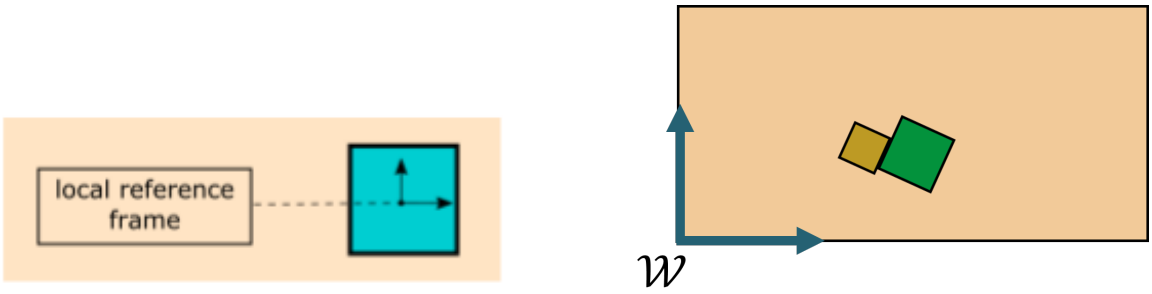
In this section, we describe the objects and state representation used in the planar tabletop conformant arrangement problems. In Section 2.1.2, we describe the extension of the state space representation to the belief space representation.

Manipulation surface $\mathcal{W}$ Every manipulation surface, $\mathcal{W}$, is a rectangular table defined by its dimensions. We assume the lower left hand corner is the origin for the world reference frame.

Rigid body objects Ω Every object, $o \in \Omega$, is a polygonal rigid body, which we define as a set of convex polygons (potentially a single polygon). We represent each polygon as a sequence of vertices. Note that this representation allows for non-convex polygons if properly decomposed into a set of convex polygons.

State representation $\mathcal{X}$ A world state, $\mathbf{x} \in \mathcal{X}$, defines the configuration of the movable objects located on the manipulation surface and which movable objects are not located on the surface. The set of movable objects consists of the parts $\mathcal{P}$ and manipulators $\mathcal{A}$.

We define each movable object configuration relative to the manipulation surface reference frame in $\mathcal{SO}_2$. Figure 2-1 provides an overview of defining a world state.



(a) Local reference frame (b) World state

Figure 2-1: Object and world state. *Figure (a) shows a local reference frame attached to a rigid body to represent the location of the object relative to the world reference frame. Figure (b) represents a world state, which has a global reference frame to specify the locations of different object states, such as the yellow block located at (7.3, 6.8) with a rotation of* $\frac{\pi}{4}$.

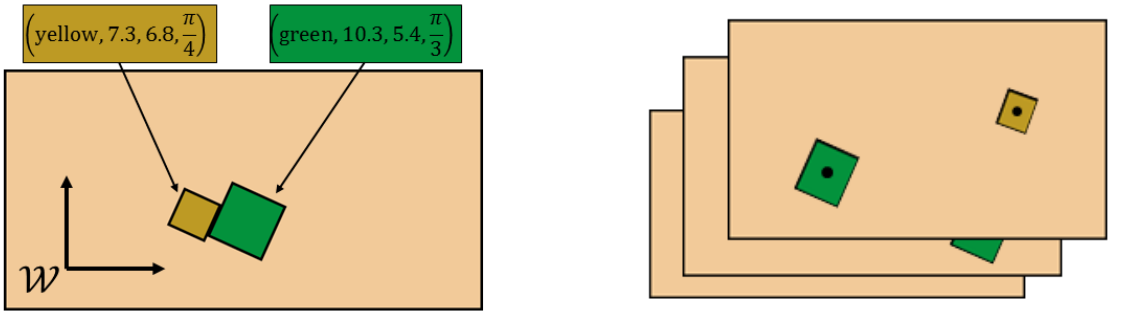
2.1.2 Belief state representations

In this section we will describe the two types of belief-state representation used in this thesis for the planar arrangement problem.

Particle belief-state representation

To approximate a belief state, b, we will sample a set of world states or "particles". We parameterize this representation by P, the number of particles used to represent a belief

state. Thus, $\hat{b} = \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P \}$, where each state $\mathbf{x}_i$ is a sample from the true belief state b. Figure 2-2 shows an example of representing a belief state using particles.



(a) Single world state $\mathbf{x}_i$

(b) Particle representation belief state, $\hat{b}$, from P samples

Figure 2-2: Particle belief-state representation. *Figure (a) shows a world state $\mathbf{x}_i$, which defines the x_{obj} for every object in the problem domain. Figure (b) shows a particle belief state that uses P samples of world states $\mathbf{x}_i$ to approximate the true belief state.*

Factored interval belief-state representation

The factored interval belief state defines an individual object belief for every movable object located on the manipulation surface. Thus, an approximate belief state would then be

$$\hat{b} = \langle \hat{b}_1, \ldots, \hat{b}_n \rangle.$$

The representable belief space for each object i, $\widehat{\mathcal{B}}_i$, is a "box" in $\mathcal{SO}_2$, represented with parameters $\langle x, y, \theta, \Delta x, \Delta y, \Delta\theta \rangle$ specifying the center and dimensions of the box; typically with $\theta = 0$. We will often denote the center of a belief box by $q = \langle x, y, \theta \rangle$ and its dimensions (the "uncertainty") as $\Delta q = \langle \Delta x, \Delta y, \Delta\theta \rangle$.

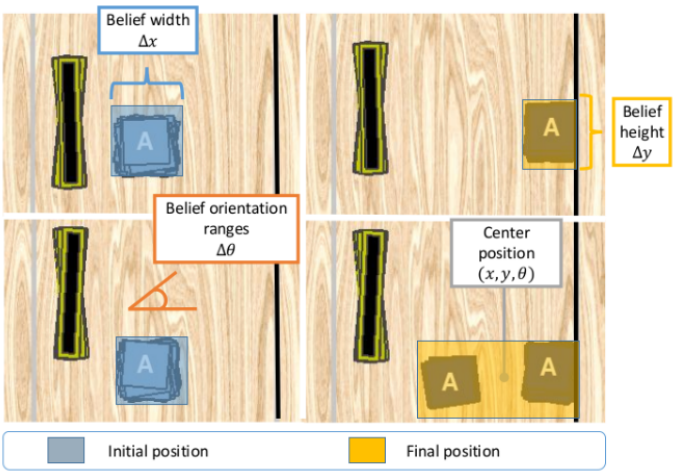


Figure 2-3: Object belief-state in $\mathcal{SO}_2$ represented with intervals. *This picture shows the belief-state representation for an individual object.*

We chose the representation of beliefs as boxes in the configuration space for compactness and computational efficiency benefits, but this representation also includes invalid world states. Thus, when using this representation, we must keep in mind that the complete belief space is the product of $\widehat{\mathcal{B}}_i$ for each object, together with the non-collision constraint.

Belief shadow

The belief-state representation defines all possible states of the world as a set of samples or ranges over object configurations. A useful representation for characterizing our uncertainty for a particular object is to construct a "belief shadow". A belief shadow is a belief uncertainty area that combines the occupied area of workspace into a single geometric structure based on an object's geometry and belief state. We used two compact geometric structures to represent a belief shadow. Figure 2-4 shows these belief shadow representations.

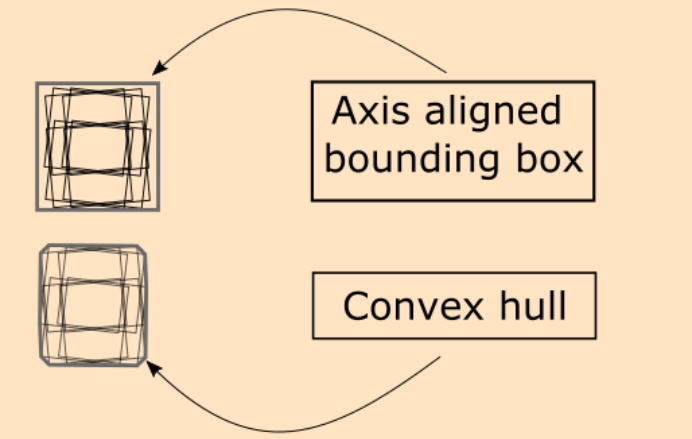


Figure 2-4: Object belief shadows. *This figure shows examples of object belief shadows. These object belief shadows approximate the occupied workspace given an object's uncertainty parameters. The first approximation is a simple axis-aligned bounding-box over all possible object locations; the second approximation is a convex hull over potential object placements.*

We define the axis aligned belief shadow in workspace as $\langle x, y, \Delta_w x, \Delta_w y \rangle$. To compute this belief shadow we use the configuration-space belief of an object. For example, a square block with an interval belief representation and dimension $2r$ has a belief bounding-box defined as

$$\mathrm{BB}(\langle x, y, \theta, \Delta x, \Delta y, \Delta\theta \rangle) = \\ \langle x, y, r + \Delta x + r\cos(\theta + \Delta\theta), r + \Delta y + r\sin(\theta + \Delta\theta) \rangle.$$

To construct the belief shadow bounding-box from a particle belief state we first use the method to convert a particle belief state to an interval belief state described in Section 2.1.2. Then, we use this estimated interval belief state to construct the belief shadow bounding-box. Notice that this belief shadow bounding-box estimation has no conservative guarantee to cover the true belief shadow.

Object belief domination

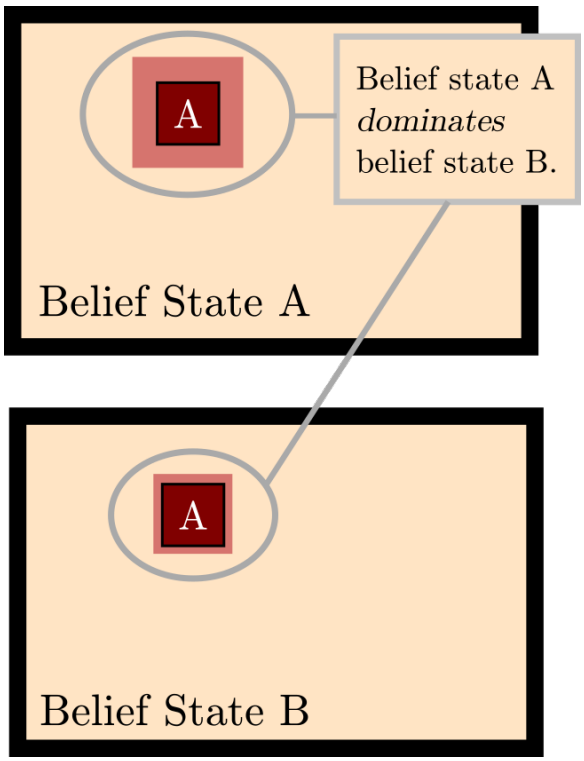
A useful property in the interval belief representation is the ability to quickly determine if a belief state *dominates* another belief state. We begin by observing that some belief states are contained by others: given two belief states, b and b', we say that b *dominates* b' if and only if:

- b and b' are both defined on the same set of objects O;
- for all $o \in O$: the configuration space box $b[o]$ is a subset of the configuration space box $b'[o]$, that is $b[o] \subseteq b'[o]$.

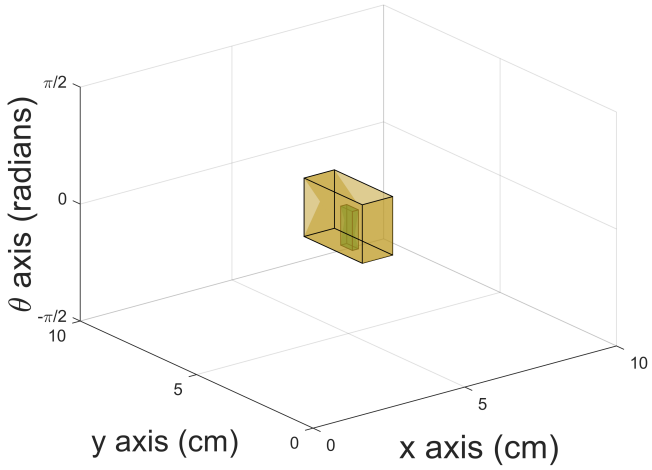
The ability to quickly determine if a belief state dominates another belief state is useful during a search. Intuitively, if b dominates b', then b may be a more useful state in the search: the objects have overlapping possible states and less uncertainty in b than in b'. Therefore, we consider a pruning strategy in which, during the process of any forward search, whenever the search reaches a state b' that is dominated by some state b that has already been visited, then b' is not explored.

This strategy can result in computational improvements and decreased search time since fewer paths may be explored; however, some of these removed paths may have a lower cost and can prevent a search algorithm's optimality conditions from being guaranteed.

Particle belief-state to factored interval belief-state



(a) Workspace object belief domination



(b) Configuration space object belief domination

Figure 2-5: Interval object belief state domination. *Figure (a) demonstrates object belief state domination using the workspace belief bounding-box representation of uncertainty. Figure (b) demonstrates the belief-state domination looking at the object configuration space. Notice how the smaller green box is completely contained within the larger yellow box; this means the green belief state dominates the yellow belief state.*

Figure 2-5 shows two examples of an object belief domination. Figure (a) demonstrates object belief state domination using the workspace belief bounding-box representation of uncertainty. Figure (b) demonstrates the belief-state domination looking at the object configuration space.

We can convert the particle belief representation to the interval representation by factoring the particle world state into a list of particle object states. Then, we convert each particle object belief state into an interval object belief state. To do so, we find the minimum and maximum ranges: $(x_{min}, y_{min}, \theta_{min})$ and $(x_{max}, y_{max}, \theta_{max})$ over all the particles. Using these boundary values, the interval belief state is then constructed as:

$$\begin{bmatrix} x \\ y \\ \theta \\ \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} x_{min} + \frac{1}{2}(x_{max} - x_{min}) \\ y_{min} + \frac{1}{2}(y_{max} - y_{min}) \\ \theta_{min} + \frac{1}{2}(\theta_{max} - \theta_{min}) \\ \frac{1}{2}(x_{max} - x_{min}) \\ \frac{1}{2}(y_{max} - y_{min}) \\ \frac{1}{2}(\theta_{max} - \theta_{min}) \end{bmatrix}.$$

2.1.3 Action space

Placing actions

The placing action is parametrized by an object id and desired place pose, which is specified as (x, y, θ). There are two preconditions for the place action. First, the expected placement of the object can not intersect any objects on the manipulation surface. Second, the object being placed must not already be located on the manipulation surface.

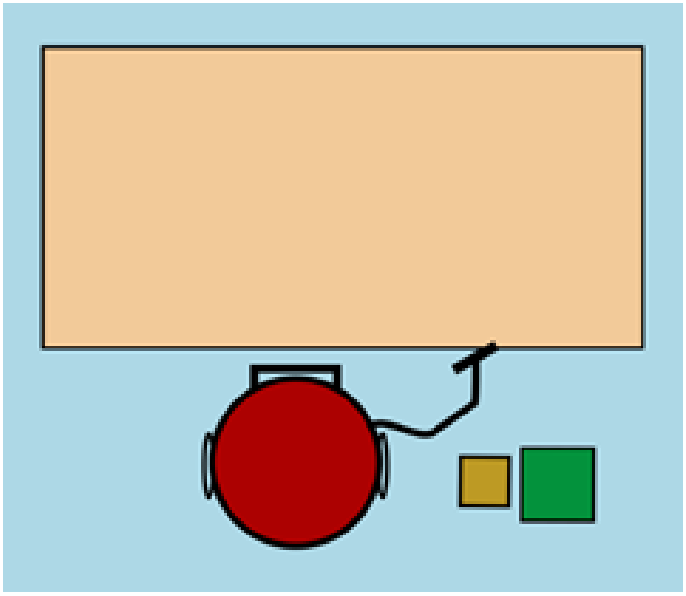


Figure 2-6: Place action parameters. *A place action is specified by* $(object\ id, x, y, \theta)$. *The uncertainty parameters are specified by translational offset* $(\pm x_{err}, \pm y_{err})$ *and rotational offset* $\pm\theta_{err}$.

To specify the place action uncertainty, we assume there is a maximum deviation from the intended placement specified by the translational offset $(\pm x_{err}, \pm y_{err})$ and rotational offset $\pm\theta_{err}$ These offset ranges are specified by the manipulation problem description. Figure 2-4 shows an example of the boundary pose locations given place uncertainty parameters; a simple approximation of the range of potential place poses (object belief shadow) is an axis-aligned bounding-box or a convex hull over potential object placements. The convex hull approximation is used in Section 4, where the orientation of the placed objects was not centered at zero.

To determine if a place action is viable when there is uncertainty in the poses of objects already on the manipulation surface and noise in the placing action, we use approximate belief shadows to detect potential object collisions. To do this, an approximate belief shadow is constructed for every object currently on the manipulation surface and the potential placed object. Then, a pairwise intersection test is conducted with the potential placed object and objects currently on the surface. If there are no interesections, then the place action is allowed to occur.

For the factored belief-state representation, the object belief shadow is constructed

directly from each object's belief parameters. For the particle representation, an interval object belief is fit over the set of samples, then the test is completed as previously described.

Pushing actions

A pushing action is specified by a push approach pose and distance. The manipulator is assumed to be a rectangular shape and the push direction is always perpendicular to the front rectangular face as shown in Figure 2-7.

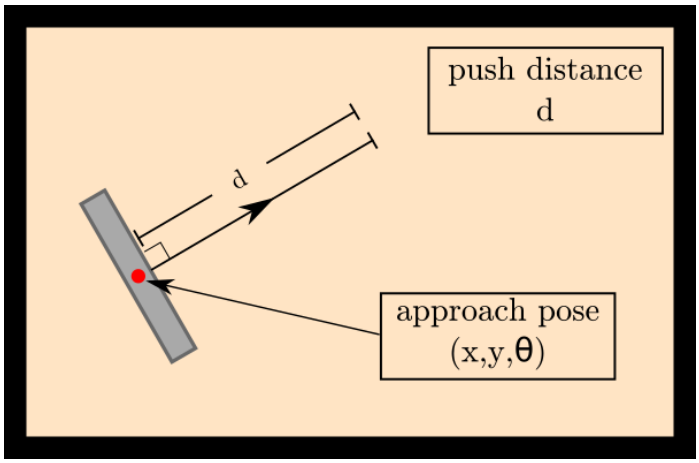


Figure 2-7: Push action parameters. *A pushing action is specified by a push approach pose and distance.*

The preconditions for a push action are similar to those for the place action. First, the manipulator must not be located on the surface and the expected approach pose location must not cause potential collisions with objects on the manipulation surface. Clearly it follows that this initial place of the manipulator has the same preconditions as the placing action: the projected belief shadow for placing the manipulator must not intersect any walls or object belief shadows of objects resting on the surface. Before, during, and after a pushing action it is possible for parts to have intersecting belief shadows with other object belief shadows on the surface. The intersection of these belief shadows is allowed and expected since parts may come into contact when constructing the arrangement. This may not appear obvious, but consider a case when the poses of two objects have uncertainty but the objects

are close enough together that we expect contact between the objects; in this case, the object belief shadows must certainly overlap.

If the preconditions are valid, then a pushing action is executed until the manipulator stops moving. Once all objects stop moving, the action is completed and the manipulator is removed from the manipulation surface. The manipulator may stop moving prior to achieving the desired pushing distance due to collisions with other objects; this outcome is expected in most pushing actions. In physical spaces with uncertainty, it is not possible to complete a push action to a specific location; for example, suppose we want the manipulator to be 1 inch away from the left side of the surface when there are no potential collision objects and the controller is at 1.0001 inches away from the left side of the surface. A properly tuned proportional-derivative controller will eventually move the manipulator a negligible amount. Thus, a the pushing action terminates when the motion is negligible or reaches below a pre-specified velocity threshold.

The pushing action has uncertainty parameters for translational offset and orientation uncertainty in the approach pose. The pushing uncertainty parameters are expected to be different from the placing parameters because the robot always maintains rigid contact with the manipulator; whereas placed parts may slip and roll during the placing action. Since the robot maintains rigid contact with the manipulator, the robot can pick the manipulator up after a pushing action. This ability to pick manipulators without any precondition (other than it is a manipulator and is located on the surface) is an important property used in Chapter 4.

The actual pushing controller is implemented by a proportional-derivative controller with low proportional gain. Since the proportional gain is low, the manipulator doesn't apply enough force to move the desired distance when there are collisions. Although this sounds like a poor controller since the manipulator does not reach the desired distance, it is useful for manipulation since excessive force could break the robot, objects, or the manipulation surface. The implementation for the pushing controllers are described in Section 2.4.

2.1.4 Manipulation goals

There are two types of specifications for a desired arrangement: 1) the *absolute* specification defines the desired position for each part in the arrangement, and 2) the *relative* specification specifies the desired relative positions between parts in the arrangement.

Absolute goals

An absolute goal specifies a target object location for every object in the arrangement. Each location has an allowable goal tolerance. In this work, we assume all goal tolerances to be the same for different objects within the arrangements, however it is easy to extend to different allowable goal tolerances for different movable parts. Figure 2-8 displays a sample goal specification.

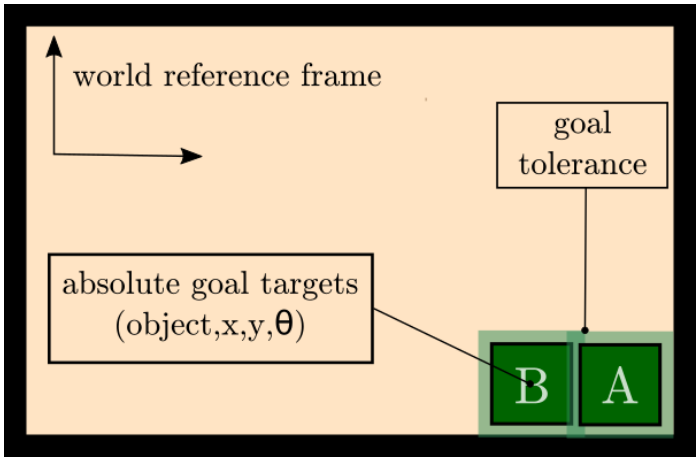


Figure 2-8: Absolute goal specification. *This figure shows an absolute goal where parts locations are specified with respect to the world reference.*

Relative goals

The relative goal specification defines the desired offset between all of the movable parts in an arrangement. Figure 2-9 shows an example of the relative goal configuration. The transparent regions in the figure represent allowable goal tolerance in the relative goal condition.

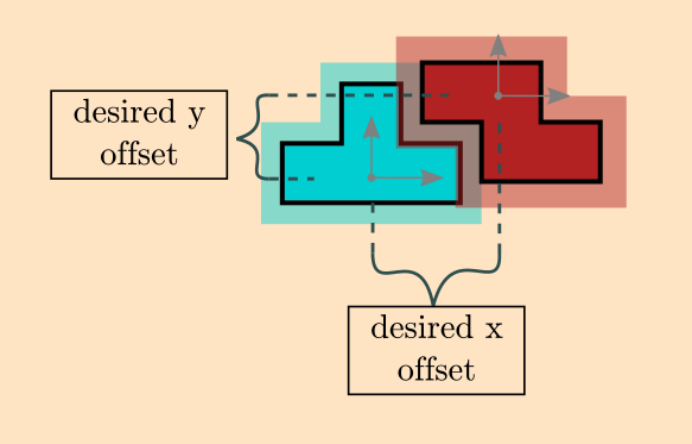


Figure 2-9: Relative goal specification. *This figure shows a relative goal where the translational relationship between the parts is specified.*

2.2 Graph search components

In this section we will discuss necessary components for implementing a graph search given a manipulation problem. First, we will look at how to generate candidate actions and then we will look at different heuristics for improving the speed of a forward search algorithm.

2.2.1 Action generation

Our action space is continuous, so there are infinite possibilities for candidate place and push actions. In this section we will discuss simple action generators for constructing a discrete set of potential actions given a state or belief state.

Place action generation

We have two methods for generating place actions. The first method looks at a part's absolute goal target and places a part near the goal target. We consider a discrete set of translational offsets from the goal target as shown in Figure 2-10. One way to look at this is that the place actions are a subset of the discretization of all possible place actions. This can be problematic since a poor choice of discretization can remove completeness guarantees

since no sequence of the provided actions will yield a successful arrangement.

Thus, the place offset must be considered carefully and is specified for each manipulation domain and may require some manual tuning. These offsets must be larger than the uncertainty regions so that the placing preconditions can be reached, but not too large since long pushing distances could accrue too much uncertainty and lead to no solution being found.

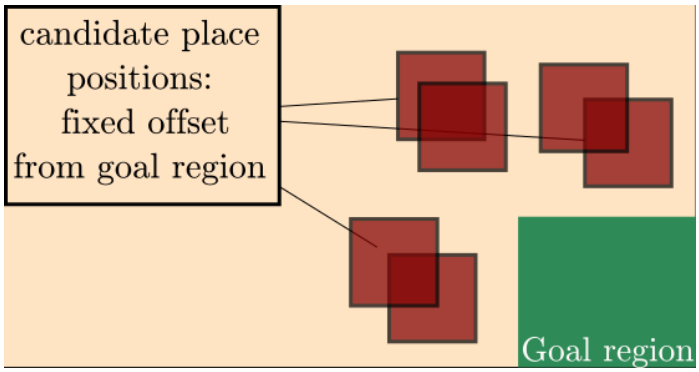


Figure 2-10: Place action generation. *This figure shows the place action generation. Given a part's target goal location, we generate several place candidates based on a fixed offset.*

The second place action generation is used in the plan improvement algorithm and will be described in Section 4-12.

Push action generation

To generate pushing actions, we only consider "object centric" pushes. For every movable part in the arrangement, we generate a set of pushes relative to the movable part. Figure 2-11 shows an example of generating a push action given a part's belief state.

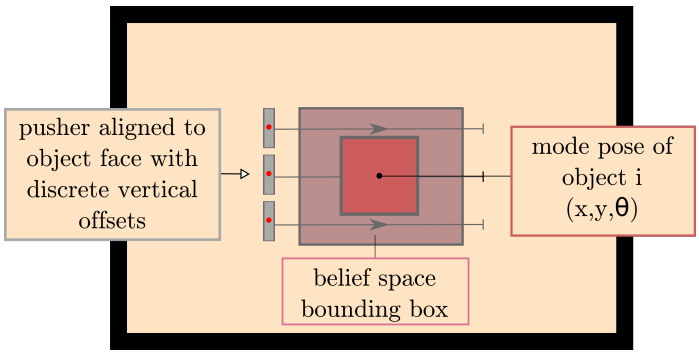


Figure 2-11: Push action generation. *This figure represents generating multiple candidate push actions given a bounding-box representation of a part's belief state.*

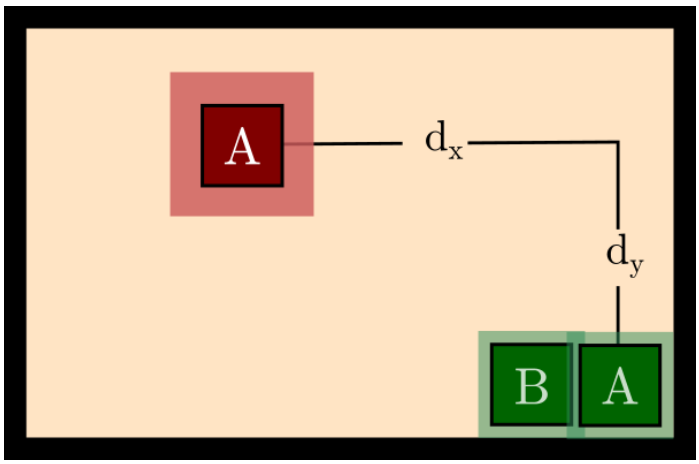
The mode pose of the object is the center of the belief interval, which we used for generating the push action. If the belief state is represented as a set of particles, then the factored interval belief-state representation is used to fit to the samples and then the push action is generated.

2.2.2 Heuristics

A heuristic estimates the distance to the goal. We created two types of heuristics that are useful for planar tabletop manipulation.

Manhattan distance

The first heuristic is simply the Manhattan distance of all parts to their goal poses in the arrangement. The Manhattan distance computes the center of the object's uncertainty to the target goal location and sums the x and y translational offset components separately as seen in Figure 2-12.

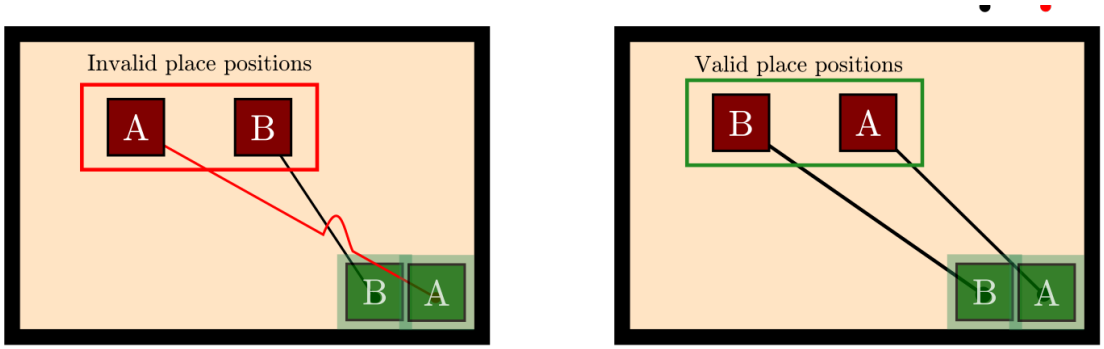


(a) Manhattan distance

Figure 2-12: Manhattan distance. *This figure shows the Manhattan distance, which is used as a heuristic to estimate a parts distance to its goal location.*

Part ordering

The second heuristic we generated looks at the parts currently located in the arrangement. Given a goal configuration, it is important to place parts in an order that will eventually yield the manipulation goal. Figure 2-13 provides the example of valid and invalid part placements. In this example parts labeled "A" and "B" end with part "B" located to the left of part "A". The invalid part (Figure 2-13 (a)) has part "B" located to the right of part "A". Although this configuration is recoverable: we could push "A" to its goal configuration followed by "B", it is clearly a slower solution that correctly placing "A" to the right of part "B".



(a) Invalid part locations

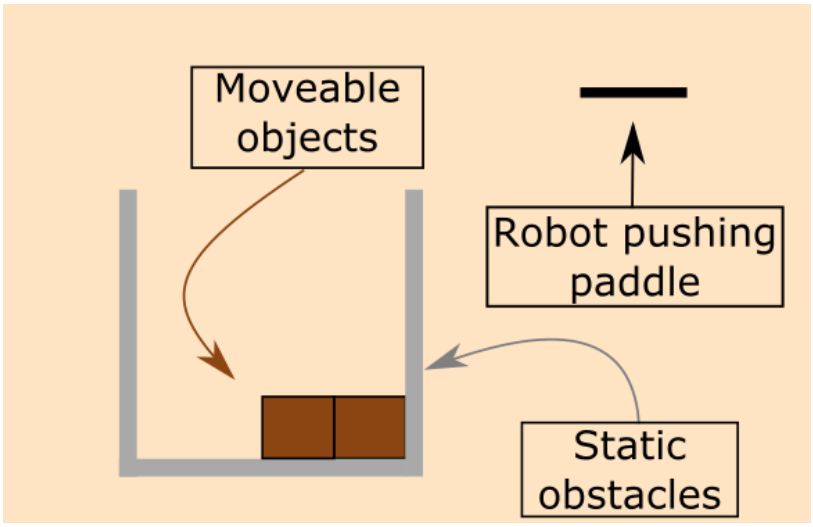
(b) Valid part locations

Figure 2-13: Part placement heuristic.

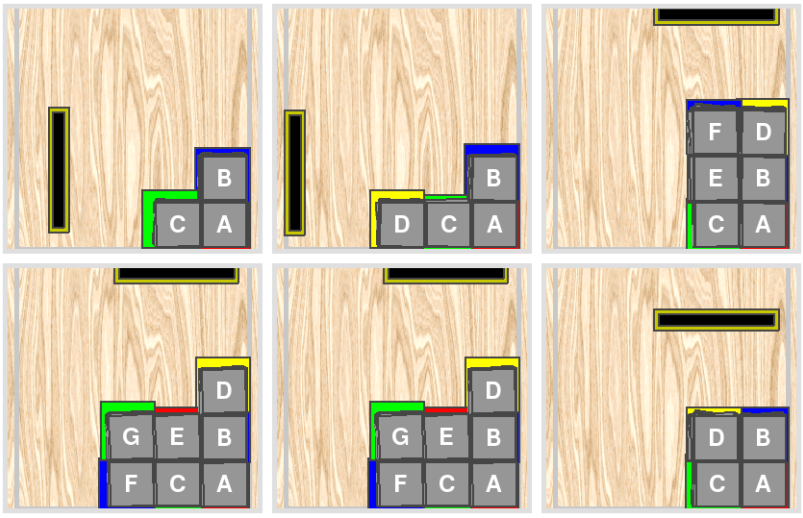
This part ordering condition is used as a pruning method to not explore parts of the search space that have inferior place positions. The heuristic assigns infinite cost to any invalid place positions and a cost of 0 for valid place positions. These places are not "invalid" in the sense that the goal cannot be achieved with this place configuration, but that they are inferior to the search since there is a place configuration that will take less steps. This leads to an effective pruning method because the search does not expand search nodes with an infinite cost.

2.3 Manipulation domains

In this section we provide concrete examples of the manipulation problems we solved in the planar tabletop manipulation setting. Figure 2-14 and Figure 2-16 show 2-D tabletop manipulation domains where a robot does pushing actions of rigid body objects and are instances of the arrangement problem.



(a) Block world

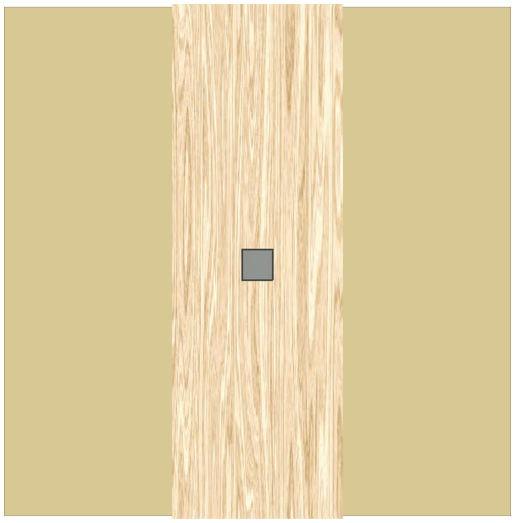


(b) Block world goal configurations

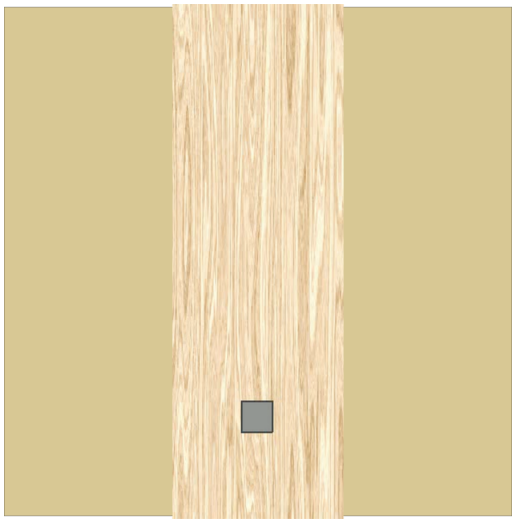
Figure 2-14: Arrangement construction problems. *In these problems, the robot places parts onto the manipulation surface. The number of parts in a desired arrangement is considerably higher with a maximum of 7 objects.*

In the "Arrangement construction problems", the robot introduces parts onto the manipulation surface using a place action. In the "Rearrangement problems", all of the parts in the desired arrangement start located on the surface. In both cases, the robot can take a mixture of manipulation actions to yield a desired arrangement. In all of these manipulation examples, the robot manipulator is a rectangular object that pushes perpendicular to its longest side and shown in Figure 2-6. Each of the individual domains provide a new element

of difficulty for reliable manipulation.



(a) Downwards push start



(b) Downwards push goal

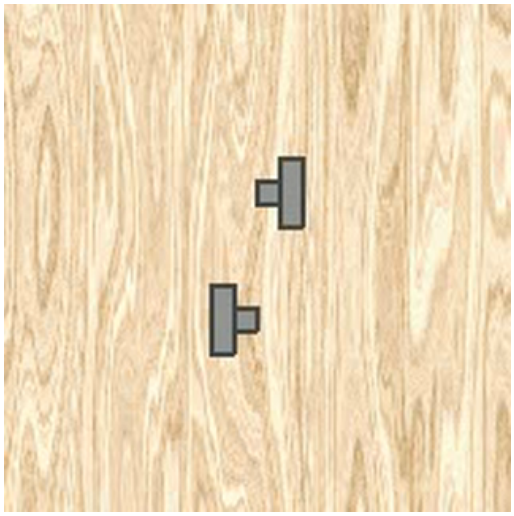
(c) Create a T start

(d) Create a T goal

Figure 2-15: Rearrangement manipulation problems (1/2). *In these manipulation problems, the parts are already located on the manipulation surface. The robot can use pushing and placing of additional manipulators to rearrange objects into a desired arrangement.*

(a) TZ mating start

(b) TZ mating goal



(c) TT mating start

(d) TT mating goal

Figure 2-16: Rearrangement manipulation problems (2/2). *In these manipulation problems, the parts are already located on the manipulation surface. The robot can use pushing and placing of fixtures to rearrange objects into a desired arrangement.*

2.4 Planar manipulation implementation

In this section we discuss the implementation methods for the planar tabletop manipulation problems. First, we implemented the planar tabletop manipulation domain in a physics simulator, and then implemented it using a real robot.

2.4.1 Simulated manipulation domain

We used Box2D, a 2D physics game engine, to simulate the planar tabletop manipulation problems. The physics simulator allows us to define movable and fixed bodies. Each body can be made up of multiple "Box2D fixtures"; a Box2D fixture defines the geometry, density, and friction properties of rigid body. Note that this Box2D fixture is a specified data structure in Box2D and is different from the *fixture* terminology introduced in Chapter 4.

The physics simulator works in discrete time steps. Each time-step is approximately $1/60$ seconds, a common framerate used for game engines. Before a time-step occurs, we can apply a force or impulse to a movable body. During the time-step, the simulator integrates the equations of motion to determine how objects move. After forward propagating the motion, the physics engine detects and corrects for collisions.

Ideally, the physics simulator is initialized with objects not in contact and then only forces and impulses are applied to simulate motion. In this work, however, we need the ability to write an arbitrary world state configuration and then apply a motion primitive. To reduce the error and allow the simulator to construct internal representation of collisions, we write the world state to the physics engine by directly modifying the body's position properties. Then, we run the physics engine for a couple of time steps before applying the motion primitive. In practice, this method reduces the nondeterminism that is inherent in the simulator.

Low level pushing controller

With Box2D, we can apply forces and torques to any location of a movable body, which we used to create our pushing controllers. For simplicity, we always applied forces and torques to the manipulator's center of mass.

We implemented a simple proportional derivative controller to apply pushing motions. Since the manipulator is a rigid body and we apply forces to the center of mass, we can define the equations of motion as:

$$F_{ext} = m\ddot{x}.$$

Given a desired set point x_{ref} with zero desired velocity, the external force that is applied is set to:

$$K_p(x_{ref} - x) + K_d(0 - \dot{x}) = m\ddot{x},$$

where $K_p, K_d > 0$.

This equation reduces to:

$$K_p \cdot x_{ref} = m\ddot{x} + K_d \cdot \dot{x} + K_p \cdot x.$$

This equation is a well known second order differential equation with constant coefficients. The solution for the equation of motion is then:

$$x(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t} + c_3.$$

Where r_1 and r_2 are found by the roots of the quadratic equation: $mr^2 + K_d r + K_p$:

$$r = -\frac{K_d}{m} \pm \frac{\sqrt{{K_d}^2 - 4mK_p}}{2m}.$$

This solution is stable if the real portion of r is less than or equal to 0. The solution will oscillate if there is a nonzero imaginary term, *e.g.* $K_d < \sqrt{4mK_p}$.

To create a stable controller without oscillations, we allow K_p to be a parameter greater than 0 and set $K_d = \sqrt{4mK_p}$ to reduce parameter tuning.

The push primitive is parametrized by (id, x, y, θ, d), where d is a pushing distance. A precondition for this action is that placing the manipulator at (x, y, θ) will not cause any collisions. The low-level controller does not check for this, it simply places the manipulator at (x, y, θ). Then, it computes a reference point that is perpendicular to the object face. In

this work, the manipulator was always a rectangle so the projected point was simply:

$$\begin{bmatrix} x_{ref} \\ y_{ref} \\ \theta_{ref} \end{bmatrix} = \begin{bmatrix} x_{ref} + d\cos(\theta + \frac{\pi}{2}) \\ y_{ref} + d\sin(\theta + \frac{\pi}{2}) \\ \theta \end{bmatrix}.$$

Then, we computed the control input $F_{ext} = (u_x, u_y, \tau)$ separately for x, y and θ:

$$F_{ext} = \begin{bmatrix} u_x \\ u_y \\ \tau \end{bmatrix} = \begin{bmatrix} clip(K_p(x_{ref} - x_{curr}) - K_d \cdot \dot{x}_{curr}) \\ clip(K_p(y_{ref} - y_{curr}) - K_d \cdot \dot{y}_{curr}) \\ clip(K_{p_\tau}(\theta_{ref} - \theta_{curr}) - K_{d_\tau} \cdot \dot{\theta}_{curr}) \end{bmatrix}.$$

Notice that the control inputs are clipped to prevent extremely large (and unrealistic) forces and torques from being applied. Finally, we ran the controller until the motion of the manipulator was approximately zero. As mentioned earlier, this can occur because the manipulator collided with other objects or because it reaches near the desired pushing distance and the control input goes to zero. At that time, we set the applied force and torque to 0 and then waited until all other movable objects on the manipulation surface's motion also went to approximately zero. Waiting for all objects to come to rest ensures the state representation, which only considers the movable object poses, is correct as opposed to keeping position and velocity information.

2.4.2 Real robot manipulation domain

We implemented the planar tabletop manipulation domain with the Willow Garage PR2 as shown in Figure 2-17. To program the robot, we took advantage of many open-source software packages provided by the Robot Operating System (ROS) [1].

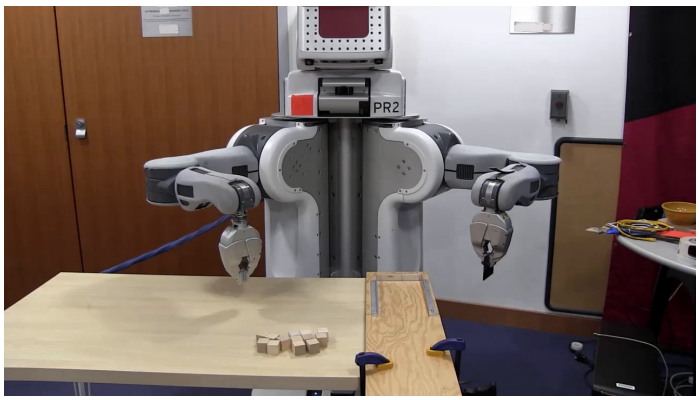


Figure 2-17: PR2 Robot. *This picture shows the real world planar tabletop manipulation setup using the Willow Garage PR2 robot.*

To create the rectangle manipulator, we built a custom paddle for the PR2 grippers to reduce wiggling motion. The paddle is 2.5 inches wide by 0.15 inches thick, laser cut from acrylic material. The laser cut patterns for the pushing paddle are shown in Figure 2-18. We also constructed a playing board that is 5 by 10 inches long with metal walls about 0.5 inches high. The surface of the board and the blocks are wood. The blocks are 1 inch cubes.

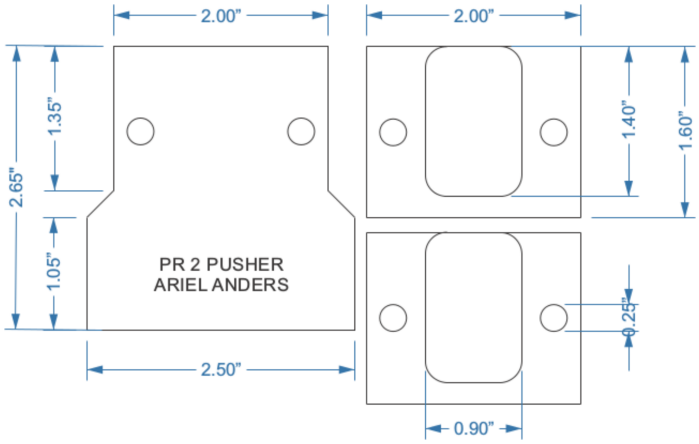
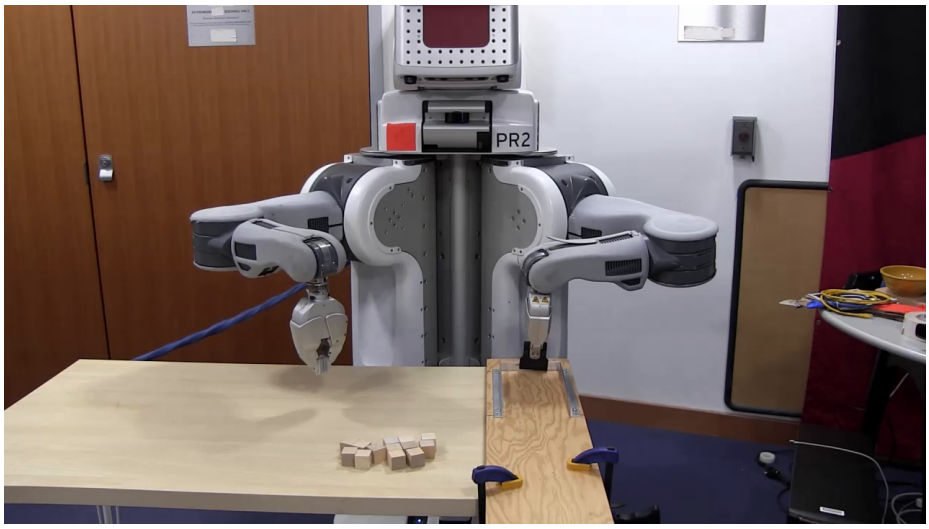
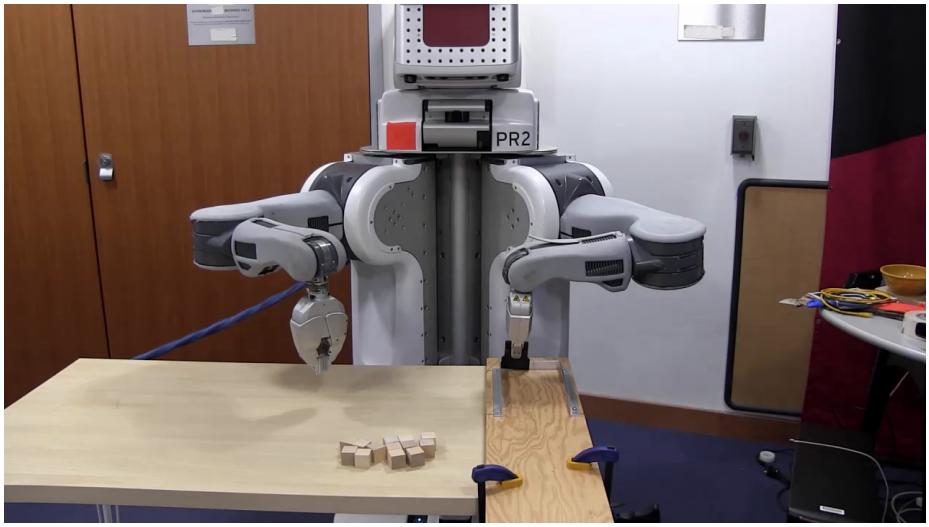


Figure 2-18: Custom manipulator. *The robot has a custom paddle fit to hold securely in the robot's grippers. When used in a top down fashion, this paddle replicates the rectangular manipulators in the planar arrangement manipulation problem.*

Before doing any experiments, we calibrate the location of the table by doing a sequence of two sliding moves along a horizontal and vertical axis of the board as shown in Figure 2-19. Then, we run an optimization program to determine the best fixed transform from the robot's base to the left hand corner of the board.



(a) Start horizontal slide



(b) End horizontal slide

Figure 2-19: Table calibration. *The robot slides the manipulator paddle in the horizontal and vertical directions. Figure (a) and Figure (b) shows the robot sliding the paddle in the horizontal direction.*

Although we don't use any external sensing in the algorithm, we set up a visual overlay to analyze the belief state predictions. Given the fixed transform from the robot to the table, we were able to project the belief shadows into RVIZ using the ROS visualization_msgs package. Additionally, we used ROS's image_geometry package to do a 3D projection onto the video from the head mounted Microsoft Kinect sensor as shown in Figure 2-20.

Figure 2-20: Belief shadow overlay. *The robot's belief of the objects and location of the manipulation surface are overlaid on the video feed from the head mounted Microsoft Kinect sensor camera.*

Push action primitive

The push action primitive is executed using two controllers. First, a joint trajectory controller is used to quickly move the robot's arm above the approach position. Then, a low level compliant Cartesian controller is used to move the arm down to the manipulation surface and execute the pushing action. Once the pushing action is complete the robot lifts its arm a few inches off the manipulation surface, then switches back to joint control to swiftly return its arm to its side location.

To determine the approach location the robot should move to, we used the KDL software package to do inverse kinematics. Then, we used the default joint trajectory controller from ROS for the PR2 to command joint actions.

The low level Cartesian controller only takes as input a single position to move the end effector to. We created a wrapper function that computed intermediate points along a pushing line to update as the set point based on time.

Low level compliant Cartesian controller

The pushing actions are implemented using a Jacobian-Transpose Cartesian controller [36]. We use a modified version of the JTCartesian controller distributed by ROS. This controller takes as input a posture and a set point. The posture is specified by seven joint angles that are seeded to the controller at runtime. We used a joint configuration that keeps the arm in an elbow up posture. The set-point is used to specify a desired pose and orientation of the gripper tool. This set-point is represented as a pose in 3 dimensions, (x, y, z), and a quaternion with 4 dimensions. The controller's input is the sum of two quantities: first, a PD control input to drive the end effector to the desired set-point and second, a null space control to try to keep a desired posture.

The low-level controller computes the error between the current end effector tool tip and the desired set point. Then, it multiplies this error and the velocity of the error by proportional and derivative parameters. Finally, it multiplies the Jacobian transpose and this scaled error vector to compute the actual joint torques for the arm.

Then we compute the desired torques for achieving a desired posture, by first computing the null space as $N = I - J^T J$ and multiplying this with the error in the posture scaled by some gain parameters.

Place primitive

The place primitive is implemented using the joint action trajectory package from ROS and an inverse kinematics service using the KDL library. The robot assumes a block has been placed into its right gripper. Then, it will move through a sequence of via points to safely move above the manipulation surface. The robot will then lower its arm to hover over the place position and drop the block.

This placing action tends to incur a great deal of uncertainty since the block can stick to the robot's gripper. We completed a simple placing experiment to determine the uncertainty of the place action of a 1 inch wooden cube. We specified four different place poses and requested the robot to complete a place action fifteen times for each place pose. Each time the block was placed, we traced the boundary of the cube on a paper resting on the

manipulation surface. We found the maximum displacement form the place pose was ± 0.2inches and ± 15 degrees. The traced objects from the experiment are shown in Figure 2-21.

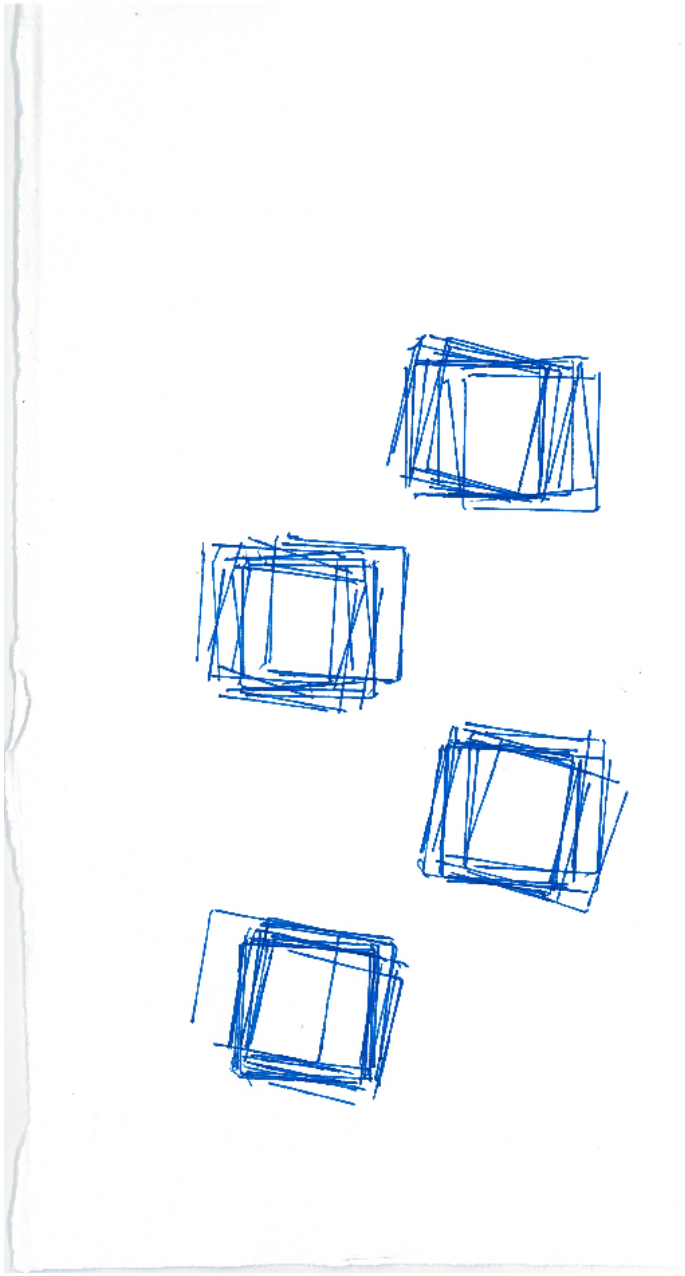


Figure 2-21: Place uncertainty. *Experiment to test the amount the blocks deviate when placed by the robot.*

Chapter 3

Conformant planning by construction

We described the problem of conformant planning as a search through a space of belief states and provided the planning problem formulation in Section 1.2.2. We followed this in Section 1.2.3 with two types of approximate belief-state representations. In this Chapter we will describe how to use these approximate representations in a forward search to find conformant plans. The central component is defining a belief-state transition function which maps an initial belief state into a resulting belief state.

Deriving a belief-state transition function is challenging because it depends on interactions among multiple objects which may be affected by detailed physical properties of the objects and robot. In this chapter, we present two different approaches for approximating belief state transitions.

In section 3.1, we provide the Particle Belief-State Transition (PBST) method and in Section 3.2, we explain the Compositional Belief-State Transition method (CBST). We also implement a *deterministic planner* by using a single particle to represent the nominal configuration of the objects and not adding noise during simulation. Then, in section 3.3 we compare the two belief-state transition functions and deterministic planner on our planar block world manipulation problem.

3.1 Particle Belief-State Transitions

The Particle Belief-State Transition (PBST) method relies on physics-based simulations and is similar to "particles" in a particle filter. In the presence of uncertainty, we can use multiple simulations with added noise to model belief-state transitions.

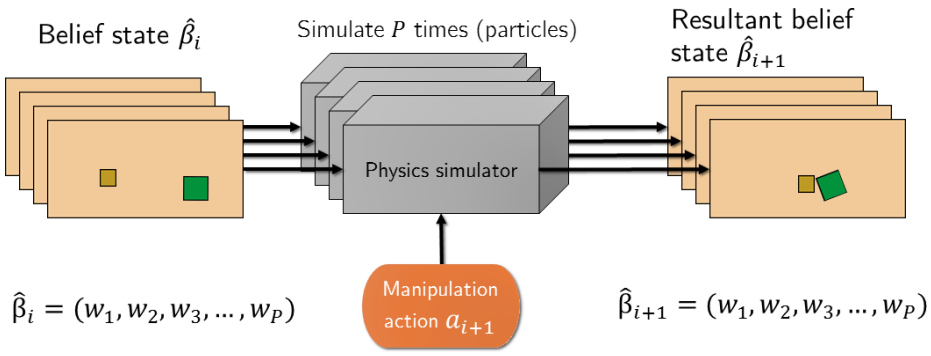


Figure 3-1: Particle belief-state transition. *This figure shows the particle belief-state transition d approximation method. Every world state is forward simulated with added noise to construct the resultant particle belief state.*

We approximate a belief state using P sampled states as described in Section 2-2. As input to the conformant planning problem, we give a description of the maximum uncertainty position and orientation ranges for each object. To provide the initial set of world states for the initial belief state, we uniformly sampled poses for each object and used rejection sampling for any invalid world states. We used rejection sampling because arbitrarily sampled world states may have collisions between the objects.

To compute a belief-state transition, we used a physics simulator to forward propagate every world state as shown in Figure 3-1. A physics simulators applies a deterministic update given properties of the objects in the world and the input action. To replicate the inherent uncertainty in the true world, we added noise to the simulation by modifying the domain parameters before every action. Every object defined in the simulator has its friction and density values randomly updated. In our experiments, we provide as input the ranges for the parameter updates and can increase the stochastic behavior of the simulator by the increasing the width of the parameter ranges.

We implemented a PBST function for three different types of actions: place, pick, and push. To complete the place action, we first checked the preconditions as defined in

Chapter 2. We constructed a belief shadow for the object intended to be placed given place uncertainty values defined by the problem definition and verified there were no collisions with objects on the manipulation surface. Then, for every world state defined in the particle belief state, we placed the object by updating is pose to the desired place position with noise added based on the place uncertainty values. There is no reason to double check that the placed object has collisions with other objects in the environment due to the action precondition. Once all word states have been updated, we construct the resultant belief state using the sampled poses for each object added.

The pick action we implemented is only valid for manipulators. We assume the robot never removes the manipulator from its end effector so it can always pick the manipulator up and remove it from the manipulation surface. The only precondition is that the manipulator is on the manipulation surface. The experiments described in this chapter do not make use of the pick action, but the pick action is used in Chapter 4.

Finally, the pushing action is executed as discussed in Section Chapter 2. A precondition test is completed to verify the approach pose of the manipulator does not have an overlapping belief shadow with any objects on the manipulation surface. Then, the pushing action is completed for every world state defined in the particle belief state. First, the manipulator is initially placed at the desired approach position with noise added based on the pushing uncertainty parameters defined by the problem definition. Then, the proportional-derivative controller is run for potentially thousands of simulation steps until no motion is detected. The full details for the pushing controller are described in Section 2.4. Once the motion of all objects has completed, the manipulator is removed from the manipulation surface. Finally, the world state is constructed from the current state of the physics simulation; all of the world states are combined to create the resultant particle belief state.

In the planner, we defined the heuristic value of a particle-based belief state to be the maximum over each individual particles's heuristic value. A further area of research could look at creating a heuristic based on the differences between the particles. For example, it is possible that having particles, where most objects are located near one another, is a desirable feature. At the same time, one feature of the particle method is the ability to represent multi-modal distributions. We wanted to compare the ability of the planner without

incorporating too much domain knowledge so we left this idea as an area for future research.

We also created a "deterministic planner" that uses a single particle. The deterministic planner assumes the domain is deterministic: objects are placed at their desired location and no noise is added to the domain. We call this the *nominal plan* in Chapter 4. For clarity: all plans in this thesis are a list of actions, to be executed sequentially. However the plans constructed using multiple particles and CBST functions (Section 3.2) consider uncertainty in the planning process.

3.2 Composable Belief-State Transitions

In Section 1.2.3, we defined a factored interval representation for a belief-state, where we represent the belief space as the product of independent beliefs about the configuration of each individual object, so that $\widehat{\mathcal{B}} = \widehat{\mathcal{B}}_1 \times \ldots \times \widehat{\mathcal{B}}_n$, where $\widehat{\mathcal{B}}_i \subset \mathcal{P}(\mathcal{C}_i)$.

This structuring also offers opportunities for compact representation of the transition functions; in most cases, there is a strong principle of locality, so that only some dimensions of the belief are changed by taking an action. Given a belief state $\langle \hat{b}_1, \ldots, \hat{b}_n \rangle$ and an action $f_{u,O}$, there will only be a set $O^* \subseteq O$ of objects that could possibly be affected by the action, so

$$f_{u,O}(\hat{b}) = \langle f(\hat{b}_1), \ldots, f(\hat{b}_n) \rangle$$

where $f(\hat{b}_i) = f_{u,i}(\hat{b}_i)$ if $i \in O^*$ and $f(\hat{b}_i) = \hat{b}_i$ otherwise.

The fewer objects that are potentially affected by any given action, the simpler the transition model is to apply. A transition model may be highly local when many of the objects are not yet placed near the objects being manipulated or when objects are already largely in a rigid formation (against one another or fixed objects in the world) so they do not move when they are contacted by other objects. It will generally be difficult to hand-specify the transition models sufficiently accurately; in our example implementation we learned the $f_{u,i}$ models from simulated data.

Structured transition model for pushing

The transition model for the push action has a structured decomposition and a local quantitative model that is learned from simulated data. Intuitively, the strategy is to find one or more sequences of blocks that will be pushed up against one another, constrained on one side by the robot paddle and on the other side by a wall (see Figure 3-11). Given such a sequence, we apply a learned quantitative local uncertainty model to compute the final center and delta values for the resulting object beliefs, starting from the object closest to the wall and working back toward the paddle.

Because we are ultimately going to use this model to search for a plan with reliable effects, it is only necessary to make predictions for actions for which we can confidently

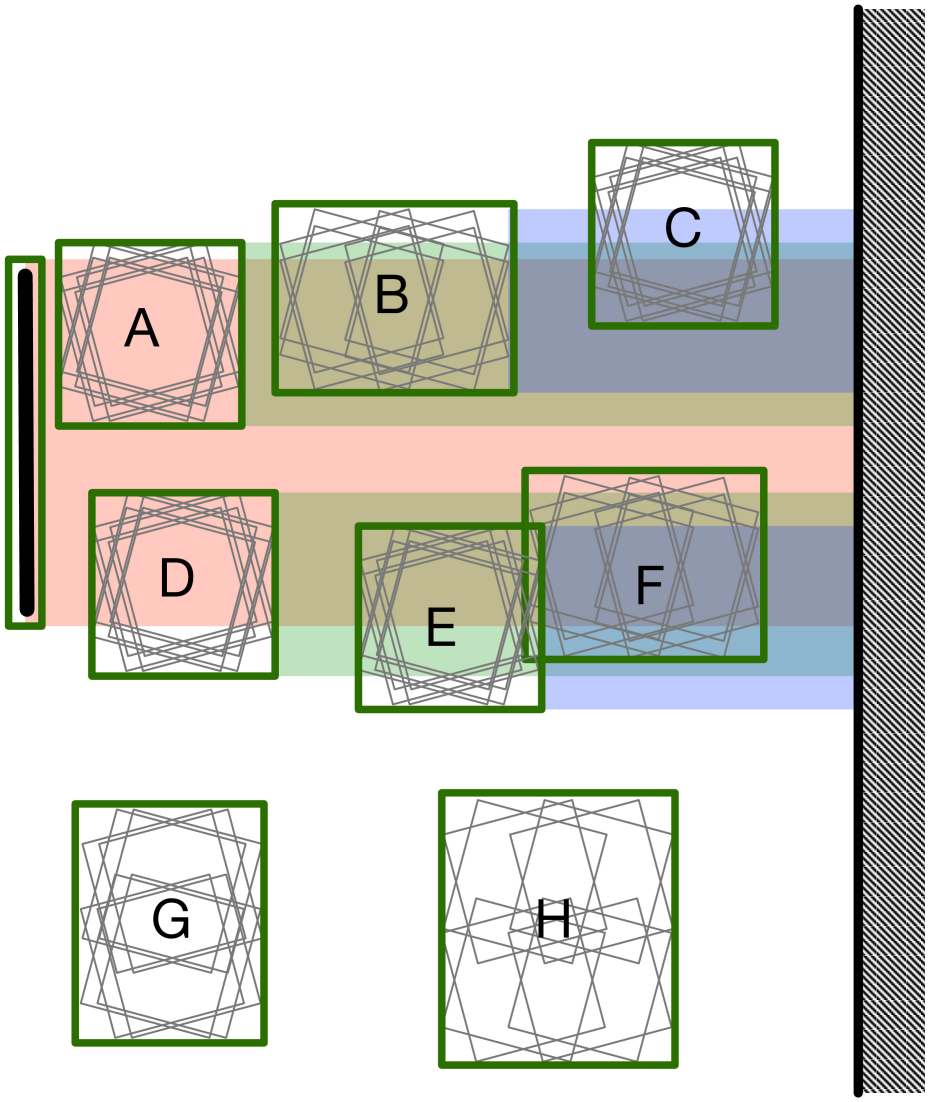


Figure 3-2: Swept volumes and bounding boxes. *This graphic shows the swept volumes and bounding boxes that are used for computing contact graph.*

predict the posterior belief state. Thus, the transition model will be partial, in some situations declining to make a prediction. This partiality will allow us to maintain the correctness of the planner, but may cause it to be incomplete (in the sense that there may be problem instances for which a legal plan exists, but our system is unable to find it.)

The first step is to determine which objects are affected by the pushing operation. To do this, we construct a *contact graph* and extract *contact paths* of objects that are mutually constraining. The contact graph has a node R for the robot, a node W for the wall, and nodes for objects that are potentially moved by this operation. Figure 3-2 illustrates a belief

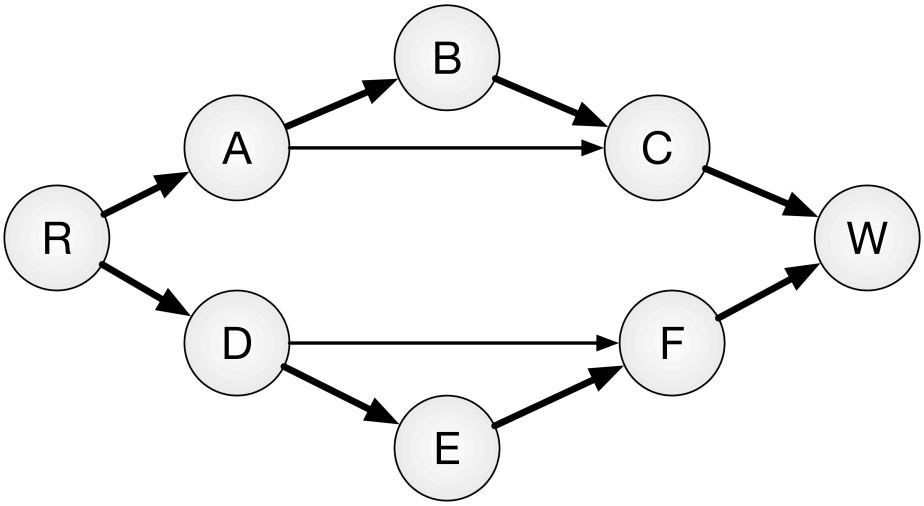


Figure 3-3: Example contact graph. *This figure shows the corresponding contact graph from Figure 3-2. For clarity some arcs are omitted. Every node (except W) has a directed edge pointing to W since moving the belief shadows of the manipulator and parts to the right will eventually have contact with the wall. Additionally, the manipulator belief shadow overlaps with all of the parts, so there should a directed age from R to A, B, C, D, E, and F.*

state and the process of determining possible contacts. It contains 6 objects, the robot paddle on the left and a wall on the right. For each object, we draw 8 copies of the object's shape, each of which represents a vertex of the belief box in configuration space; the green box is the workspace bounding box, BB, associated with each object. Note that the bounding boxes for objects E and F overlap; this is allowed and makes sense if we recall the implicit non-collision constraint. Uncertainty in the robot's position and orientation are represented by a green workspace bounding box.

For each object i, we construct a swept volume of the belief bounding box moving in the direction of the push action. Then we test the bounding box for every object j currently in the arrangement. Every intersection of object j's bounding box with the swept volume creates a potential contact from object i to object j. These potential contacts form directed edges in the contact graph.

The procedure MAKECG shown in Figure 3-4 outlines the process formally. For simplicity, we just describe the case for a push toward the right; for other axis-aligned pushes it is essentially the same, up to changing signs and/or swapping x and y coordinates.

In the example of figure 3-2, execution would go as follows:

```
MAKECG((x, y, θ), b):
    agenda = {robot}; V = {robot}; E = { }; E' = { }
    while not EMPTY(agenda):
        o = agenda.pop()
        vol = sweptVol(BB(b[o]), (x, y, θ))
        for o' ∈ Ω ∪ {wall}:
            if BB(b[o']) ∩ vol ≠ ∅:
                if o' ∉ V: agenda.ADD(o')
                V.procadd(o')
                E.ADD((o, o'))
    return V, E, E'

PUSHTRANS(b, (x, y, θ)):
    V, E, E' = MAKECG((x, y, θ), b)
    P = MAXIMALPATHS(V, E ∪ E')
    b' = copy(b)
    maxPaddleX = None; maxBlocks = None
    for p ∈ SORTEDLONGESTFIRST(P):
        if maxPaddleX == None:
            maxPaddleX = wall.x − (len)(p) · d
            maxBlocks = len(p)
        if len(p) < maxBlocks:
            firstObjB = b[p[1]]
            if firstObjB.x − firstObjB.Δx > maxPaddleX:
                pass // Objects sure not to be moved
            else
                return None // Objects moved unpredictably
        if p ∩ E' ≠ ∅:
            return None
        for i = LEN(p) − 1 downto 1:
            b'[p[i]] = PUSHTRANSOBJ(p[i − 1], p[i], p[i + 1], b', θ)
    return b'

PUSHTRANSOBJ(b, prev, cur, next, θ):
    // Just handling case of push to right
    Δx', Δy', Δθ' = PREDICT(b[prev], b[cur], b[next])
    x' = b[next].x − d − Δx'/2
    return (x', b[cur].y, 0, Δx', Δy', Δθ')
```

Figure 3-4: Belief-state transition. *Pseudo code for computing the transition function on belief states for the push (x, y, θ) action.*

- We begin with the robot and compute the volume of the workspace it would sweep through if it was able to move all the way to the wall; this is shown in light red in the figure.
- We add each object whose bounding box overlaps this swept volume to the graph, and add an edge from the robot node to the object to the set E. In this case, we add edges to objects A, through F.
- Now, we take each of these objects in turn, computing their swept volume and adding arcs to objects they will contact.
- The swept volume for A is shown in green. It overlaps with objects B and C.
- The swept volume for B is shown in blue. It overlaps with object C.
- Similar processing is done on D, E, and F.

Figure 3-3 shows the resulting contact graph.

The next step is to find maximal contact paths between nodes R and W. A contact path p is maximal if there is no other path p' between R and W such that $p \subset p'$. These paths represent "trains" of objects that will end up nearly in contact with one another, pushed between the hand and the wall. In this example, there are two maximal paths: $RABCW$ and $RDEFW$. If the paths are not of the same length, then the prediction process cannot be applied. Consider the case in which object D is not present in this example; the robot hand will compress objects A, B, and C against the wall but leave E and F unaffected. However, if D were present, but E and/or F were missing, then D would be pushed, but without a constraint on its right, we are unable to reliably predict its resulting position and uncertainty; in this case, the prediction model returns **None**, indicating that this operation cannot be used to construct a plan.

In some cases, we can still apply our prediction model when paths have uneven lengths. This is acceptable when we know the path of shorter length will not be affected by pushing action on the longer path. Consider the previous example, except with block D removed. If the robot pushes blocks A, B, and C against the wall but doesn't reach block E, then

the path of E and F will not be affected and does not need to be updated. To determine this case, we consider the closest distance to the wall the manipulator can get to: this is the sum of the widths of the objects in the longer contact path. We compare this distance to the location of the leading edge of the first block in the shorter contact path to determine if the manipulator will possibly hit the shorter path. If this test confirms the shorter path is not affected by a pushing action, we can safely update the longer path and ignore the shorter path; alternatively; if both paths are affected then we cannot apply the action to this belief state.

For each valid contact path, we use a local predictive model compositionally to compute the posterior belief for each object. The predictive model takes as input the belief states of three objects that occur sequentially in the contact path.

Learning a quantitative model for pushing

We will work with a factored version of f', which maps the belief state of an object and its neighbors on either side to its resulting belief state. In the transition model for a push to the right, we begin by predicting the uncertainties in the resulting object position, using a function PREDICT, which is learned from data. It has inputs and outputs in the form:

$$\Delta q' = \langle \Delta x', \Delta y', \Delta \theta' \rangle = \text{PREDICT}(b[prev], b[cur], b[next]) \ ,$$

where the inputs to the procedure are the current beliefs about three sequential objects in a pushing sequence. We assume that the center of the resulting uncertainty box has the same y coordinate as before and that the median rotation is 0. The median x coordinate is computed by finding the median x coordinate of the object to its right, subtracting the dimension of an object, d, and then subtracting the resulting x uncertainty, $\Delta x'/2$.

The problem of learning the PREDICT function can be treated as a supervised regression problem with three output dimensions. Again, for simplicity of exposition, we limit our attention to the push-right action. The inputs to the PREDICT procedure are

$(q_p, \Delta q_p, q_c, \Delta q_c, q_n, \Delta q_n)$. We compute from these inputs a feature vector ϕ:

$$\langle \Delta q_p, \Delta q_c, \Delta q_n, q_p.y - q_c.y, q_n.y - q_c.y, ct \rangle.$$

The fourth element of the feature vector $(q_p.y - q_c.y)$ represents the vertical offset between the previous object and current object uncertainty centers. Correspondingly, the fifth element represents the vertical offset between the current object and the next object.

Because generating training data on the real robot would be prohibitively costly in time, we generate training data using a Box2D simulation [6]. We construct a data set of 1800 examples, using the following process to construct each example:

- An initial belief state b is randomly constructed, with Δx and Δy values drawn uniformly in the interval $[0.0, 0.4]$ inches and $\Delta\theta$ values drawn uniformly in the interval $[0, 15]^\circ$; b may contain 1, 2, or 3 blocks.

- For 1000 iterations, an initial state is drawn uniformly from b and constructed in Box2D, with the robot paddle offset to the left and a fixed wall, parallel to the paddle, to the right of all the objects. The paddle's y coordinate is randomly varied to generate a variety of offset values. The static and kinetic friction coefficients for the simulation of robot, table, and objects are drawn uniformly in the range $[0.25, 0.55]$. The robot's paddle is moved in the desired direction using a position controller to a distant set-point with gains of 1.0 for position and angular error; the controller is run for 50 simulation steps with $\Delta_t = 0.01\ s$. The final pose of the each object (x_i, y_i, θ_i), together with its contact type, is recorded.

- The resulting belief state dimensions are computed as:

$$\begin{aligned}
\Delta x &= xmax - \min_i x_i \\
\Delta y &= \max_i \|y_i - y\| \\
\Delta\theta &= \max_i \|\theta_i\|
\end{aligned}$$

where $xmax$ is the maximum possible x coordinate for this object, if all the objects

to its right were perfectly aligned and as far to the right as possible. Notice that Δy and $\Delta\theta$ are computed based on the assumed structure of the transition function: the resulting uncertainty box has the same y coordinate as before and the median rotation is 0. Even though the median values may vary from this assumption, the computed uncertainty widths increase to account for this error making the structured belief-state transition a conservative over approximation.

This data is then used to train a multi-output random-forest regressor using the Scikit-Learn toolkit [41] using hyper-parameters that were found using a grid search with 8-fold cross-validation. We used 90% of the data for training and hyper-parameter optimization and held out 10% for final evaluation, in which we found a root-mean-squared test error of 0.509 in Δx, 0.079 in Δy and 2.799 in $\Delta\theta$.

Search

In Section 2.2 we described a generic forward-search problem that could be solved using A^* and in the previous section we specified the necessary state space and successor function for our example planar block world arrangement problem. We assign a cost of 1 to every action.

In order to make the A^* search process tractable, we specify two additional components: a dominance-based pruning method and a search heuristic. This pruning strategy was described in Chapter 2.

In addition, we experimented with two heuristics. The first, which is admissible, effectively acts as a binary filter on states, assigning infinite cost to any state in which the objects in a horizontal or vertical contact path are not in the order specified by the goal. It is not possible, given the operations in this space, to reach the goal from such a state. This heuristic provides some useful search guidance but is not very strong. We define an additional heuristic that is inadmissible in general, but highly effective at improving the speed of the search without much reduction in the solution quality. We define:

$$H(b, g) = \sum_{o \in \Omega} H(\mathrm{BB}(b[o]), g[o])$$

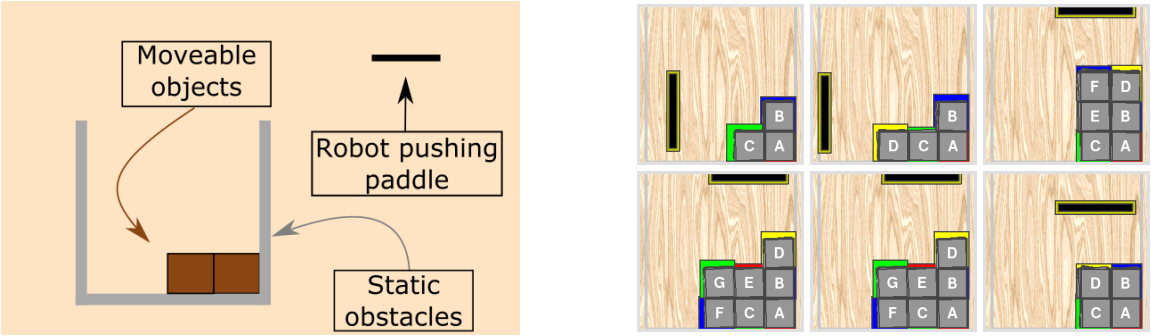
where

$$H(bo, go) = \begin{cases} 2 & \text{if } bo == \textbf{None} \\ 2 & \text{if } bo.x \not\subseteq go.x \text{ and } bo.y \not\subseteq go.y \\ 1 & \text{if } bo.x \subseteq go.x \text{ and } bo.y \not\subseteq go.y \\ 1 & \text{if } bo.x \not\subseteq go.x \text{ and } bo.y \subseteq go.y \\ 0 & \text{otherwise} \end{cases}$$

This heuristic estimates that it will take one place action and one push action to add a new object to the arrangement and one push per object dimension that is not currently contained within its goal interval. It is inadmissible because it is possible to push multiple blocks at once.

3.3 Results

In this section we present quantitative results of our approach in the block world domain, an instance of the "arrangement construction problem" described in Chapter 2 and re-shown here in figure 3-5.



(a) Block world

(b) Block world goal configurations

Figure 3-5: Arrangement construction problems. *In an arrangement construction problem, the robot places parts onto the manipulation surface and pushes the parts with a rectangular paddle to construct a desired arrangement. Figure (a) shows the components in the arrangement problem and Figure (b) shows example arrangements that were constructed in a simulated environment.*

Belief-state transition comparison

We compared the performance, both in terms of accuracy and speed, of our CBST planner against four different planners based on on-line simulation: a deterministic planner as well as particle-based planners with 5, 25, and 50 particles. We ran each planner on a 4-block arrangement problem and a 7-block arrangement problem, each a total of 10 times.

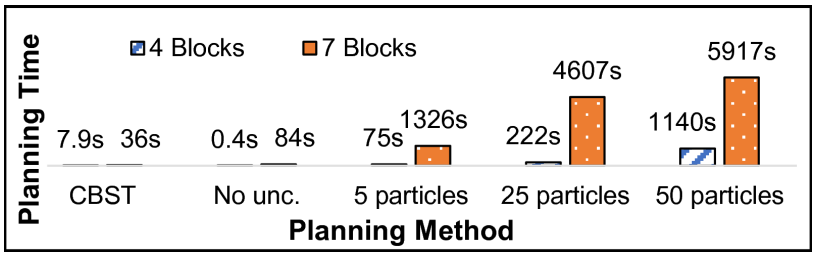


Figure 3-6: Planning time with different belief-state transition models. *This graph shows a comparison of planning time and different belief-state transition models for a 4-block and 7-block arrangement.*

Figure 3-6 compares the average planning time for different belief-state transition models. As expected, the deterministic model and CBST are fastest. The planning time with the particle transition models might be expected to scale linearly with the number of particles, however, as the number of particles increases, they will tend to be more diffused and might actually require a longer plan. We would expect the number of particles required to achieve a reliable planner to grow exponentially with the number of objects. For the 7-block problem, we capped the running time for an expansion at 5000 search nodes; none of the particle-based planners found a solution under this constraint. The reported times for the failed searches show the amount of time it took to reach the 5000 search node constraint.

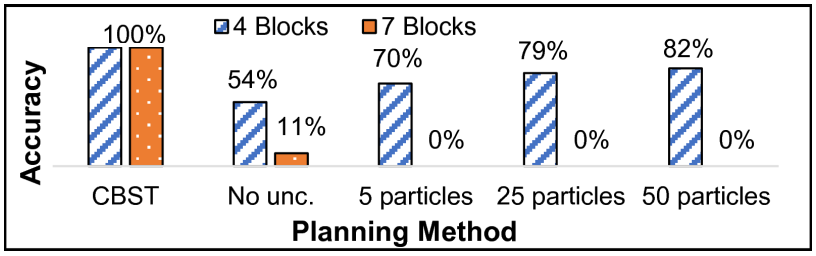


Figure 3-7: Execution accuracy with different belief-state transition models. *This graph shows the execution accuracy for different planning strategies.*

We evaluated the robustness of the solutions found by the different planners using the physics simulator. We tested each of the 10 plans 1000 times and added simulated noise as used in learning. We reported the percentage of tests in which all objects were in their goal region at termination in Figure 3-7. The learned CBST planner produced 100 percent successful results, whereas the on-line simulation approaches produced many non-conformant plans. The deterministic planner is successful a little more than half the time in the 4-block

case and a little more than 10% of the time in the 7-block case, showing the need to consider the effect of stochastic actions. In the 4-block case, the particle-based planners improve slowly as function of the number of particles and approach respectable levels of accuracy for 50 particles. In the 7-block case the running time to construct a plan with more than one particle was prohibitive.

Planner performance

We tested the CBST planner for four arrangements with 2, 3, 4, and 7 blocks. We varied the goal tolerances (the dimension of the goal regions) between ± 0.1 inches and ± 0.5 inches. The initial placement uncertainty was, unless stated otherwise, ± 0.2 inches in x and y and ± 15 degrees rotation. When a plan was obtained, we simulated it 1000 times. We found that whenever a plan was found, all the simulations satisfied the goal. However, for tight goal conditions, especially for larger assemblies, the search can exceed our limit of 5000 search nodes.

The search performance, as measured by number of search nodes expanded, is affected primarily by the quality of the heuristic used. In this experiment, the search node limit was 5000. In Table 3.1, we see that using the "inadmissible" heuristic cuts the number of expanded nodes substantially, at the cost of longer plans. The domination test has a small beneficial effect given this heuristic, but a very large effect when a weaker or no heuristic is used.

Search Type	Domination	Solutions Found	1 Block Nodes	1 Block Cost	2 Blocks Nodes	2 Blocks Cost	3 Blocks Nodes	3 Blocks Cost	4 Blocks Nodes	4 Blocks Cost	7 Blocks Nodes	7 Blocks Cost
BFS	No	50.00%	25	3	160	4	4483	6	--	--	--	--
BFS	Yes	50.00%	27	3	191	4	2982	6	--	--	--	--
Admiss.	No	50.00%	21	3	136	4	1149	6	--	--	--	--
Admiss.	Yes	50.00%	27	3	179	4	1123	6	--	--	--	--
Inadmiss.	No	100.00%	9	3	29	6	60	9	98	12	535	24
Inadmiss.	Yes	83.30%	9	3	28	6	54	9	95	12	284	21

Table 3.1: **Search results table.** *This table compares the search performance when using different heuristics, increasing the number of parts in an arrangement, and using belief-state domination as a search pruning technique.*

The effect of uncertainty

Figures 3-8 and 3-9 show the effects of goal tolerance and initial placement uncertainty on plan length. Tighter goals and higher initial uncertainty both increase the length of the required plans. Once the required plans exceed a length of about 16 actions, the search process exceeds the allowed number of search nodes (5000); in such cases, there is no corresponding result plotted in the graphs.

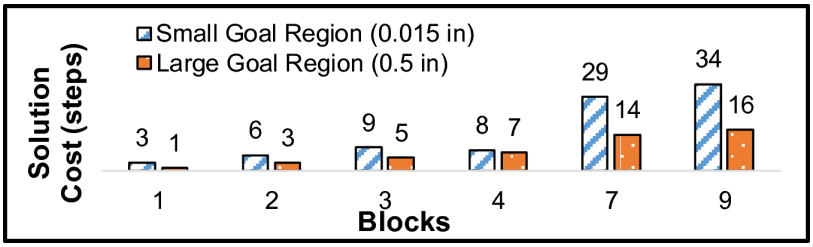


Figure 3-8: Goal tolerance and solution length. *This figure shows the effect of tigher goal regions on the solution length. The tight goal is* ± 0.15 *in and the loose goal is* ± 0.5 *in.*

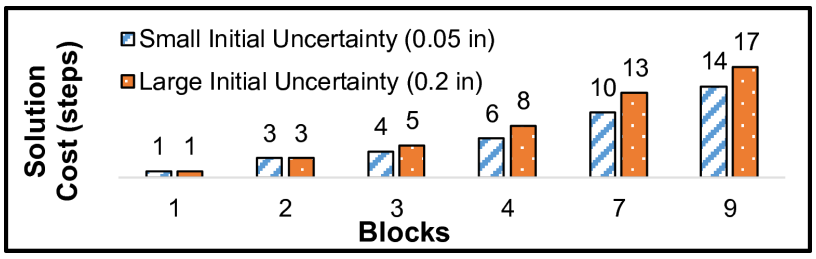


Figure 3-9: Initial place uncertainty and solution length. *This figure shows the effect of decreasing the initial uncertainty and the solution length. The small initial position uncertainty is* ± 0.05 *in and the large initial position uncertainty is* ± 0.2 *in; angle uncertainty is* ± 15 *deg in both cases.*

Real robot experiments

An example arrangement sequence found by the planner, as well as its execution using a real PR2 robot, are shown in Figure 3-10 and Figure 3-11. We obtained plans for 5 different assemblies (with 2, 3, 4, 7 and 9 blocks) and executed each one on the robot 5 times. The placement uncertainty was ± 0.2 inches and ± 15 degrees rotation; the goal tolerance was ± 0.2 inches. We saw one execution failure (in a 2-block arrangement) during the 25 assemblies, for a 96% success rate.

Figure 3-10: Real robot reliably assembling an arrangement. *These are a sequence of photographs of a robot constructing an arrangement. First the robot places a block, then pushes it using a paddle.*

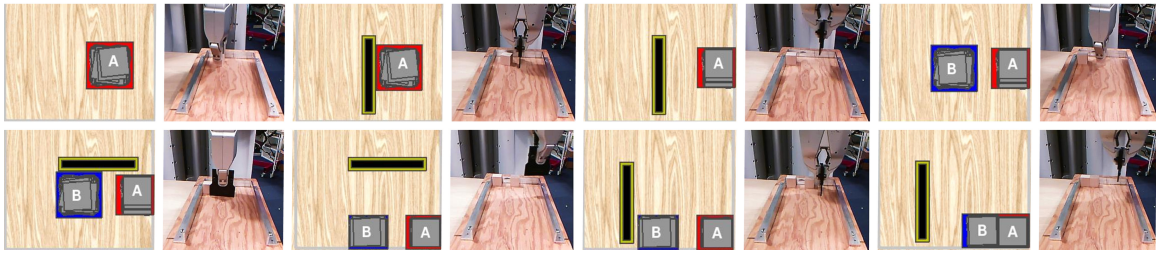


Figure 3-11: Execution of a sample 8-step plan. *In each pair of images, the left figure shows the results of 1000 simulations of the plan (gray blocks) as well as the predicted workspace bounding boxes from the transition model (in red and blue) and the right figure shows execution on the real robot.*

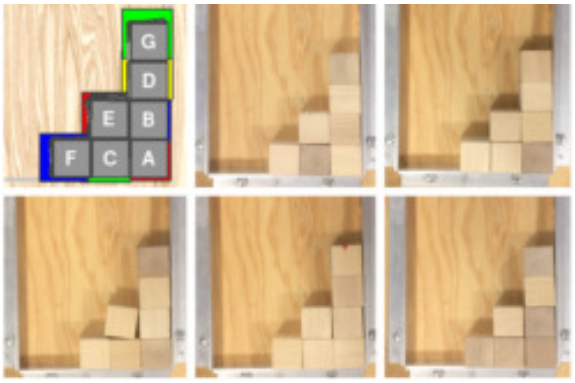


Figure 3-12: Real robot construction of a 7 block arrangement. *results shown for 1000 noisy simulated executions and 5 executions on a real robot.*

The execution failure appears to be due to an initial placement outside of the modeled placement uncertainty bounds. Increasing the modeled placement uncertainty could decrease this type of failure, at the expense of increasing the planning and execution times for the typical case. This is an unavoidable trade-off in conformant planning that could be ameliorated by moving to a belief-space replanning paradigm that adds some sensing, such as the final position of the paddle after a push.

3.3.1 Chapter discussion

One key question is: what fidelity of the transition model is needed for planning? Figure 3-13 shows a push action that usually pushes the block to the wall, but in some cases misses the block with the paddle due to uncertainty in the vertical offset. In this example, it took

over 50 simulations just to see the unlikely result. How should a robot deal with this? Doing on-line simulations to detect unlikely outcomes would be computationally difficult. Learning from on-line simulations is more promising, but representing and planning with a complex posterior distribution is problematic. Following the thread of this work, we could learn a relatively simple model and use sensing and replanning to detect prediction failures. We believe that the replanning approach is most practical, but probing the trade-off in fidelity of prediction vs sensing is important.

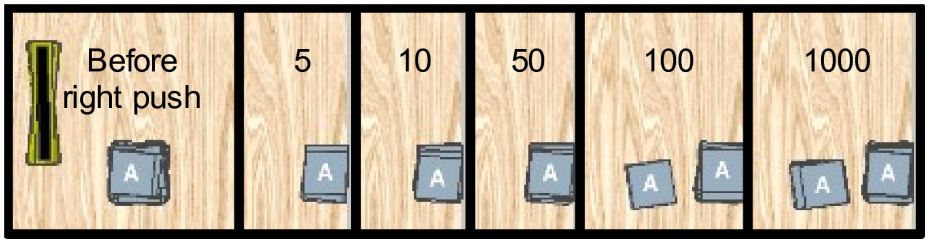


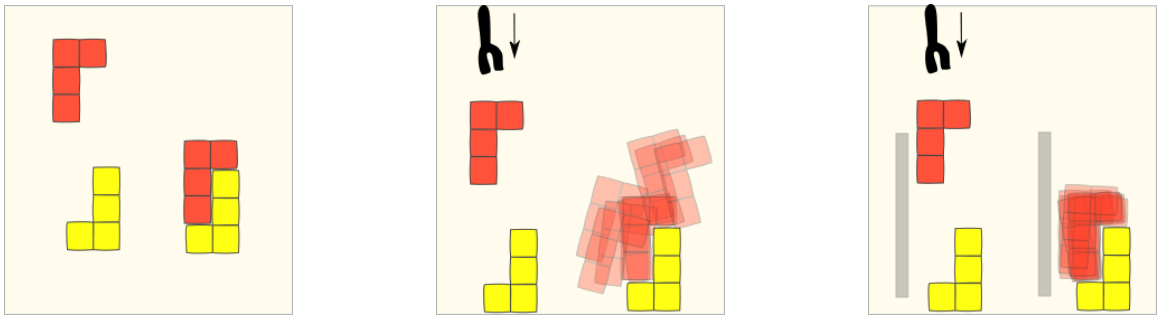
Figure 3-13: Increasing particles for belief-state approximation. *This figure shows the approximation of the belief state with a different number of particles.*

We have shown how a belief-space transition model can be acquired from off-line physics-based simulations and used to plan reliable planar push-arrangements in the presence of substantial uncertainty. We have compared this work with on-line physics-based simulation methods and found results showing much higher accuracy with significantly decreased search time in the box arrangement domain. Clearly, there are cases where the detailed approach described here will not work, such as a cylindrical robot pushing a cylindrical object. Future work should explore whether some of the basic ideas, especially learning factored models, can be generalized more broadly.

Chapter 4

Conformant planning by improvement

In this chapter, we explore the *fixture-placement problem*, which is the problem of identifying placements of *fixtures* to improve original pushing plans. A fixture is a movable obstacle that is placed onto the manipulation surface for assisting with a pushing action. Figure 4-1 provides a simplified example of using a fixture for aiding a pushing action.



(a) Problem set-up

(b) Nominal plan executed with uncertainty

(c) fixture-placement solution

Figure 4-1: fixture-placement example. *In this fixture-placement problem, the goal is to push two L-shaped pieces into a rectangle as shown in Figure (a). Figure (b) shows the effect of a downward push action that is substantially affected by the uncertainty in the domain. In Figure (c) a fixture is introduced and improves the original manipulation plan by blocking undesirable motions of the red object.*

In this simplified example, the goal is to push an 'L' shaped object down to create a rectangle of 'L' shapes as shown in sub figure (a). The subsequent sub figure (b) shows the execution of this push when there is uncertainty in the domain, which causes many pushing

executions to fail. Finally, sub figure (c) resolves this uncertainty by placing a rectangular fixture to block undesired motion during the downward push. In this chapter, we present an approach for determining fixture-placement poses to improve reliability.

Since we have introduced a new manipulation action, we must consider how to find conformant plans with fixture-placements. One approach is to directly plan in belief space with these additional manipulation actions. However, as we saw in Chapter 3, finding a conformant plan is expensive, since we must consider the effect of all possible actions at every search node. The new actions substantially increase planning time since they increase the branching factor and the length of plans may be increased (since we must place and pick fixtures).

Instead of directly planning in belief space, we propose to find a manipulation plan that solves a relaxed problem in state space and then add actions so that it is robust to uncertainty. In this case, our improved plans are approximately conformant, in the sense that we will try to improve a plan as much as possible, but note that this may not result in a truly conformant plan.

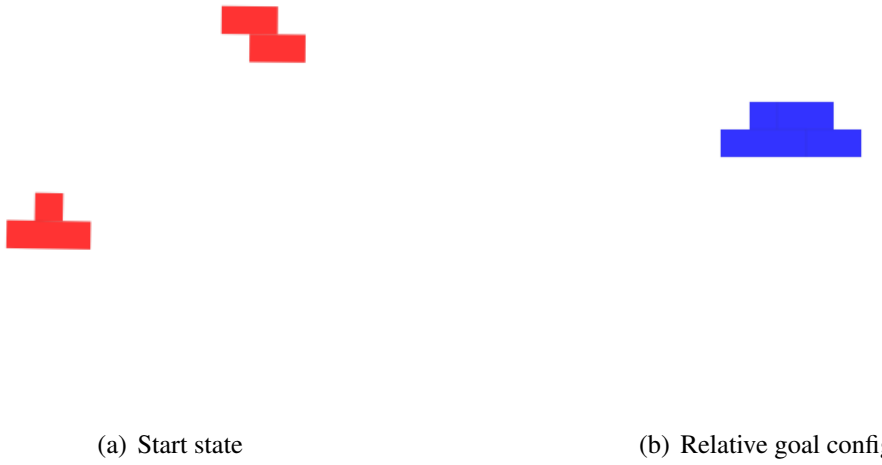
The relaxed manipulation problem is a deterministic version of the conformant arrangement problem. There is no uncertainty in the world state and the effect of every manipulation action is known. This simplified problem can be formulated as a planning problem, which can then be solved using graph search. In Chapter 3, we introduced a conformant planning method that searched over belief states. In comparison, this relaxed planner is a search over world states and uses the same implementation as the deterministic 1-particle planner used in Chapter 3 experiments.

Given a conformant arrangement problem, we can quickly find a plan for the deterministic version of the problem using a heuristic graph search. Many of these problems have multiple solutions; consider a simple example of pushing a block into a right-hand corner—we can push the block down and then to the right or to the right and then down. We can find these different plans for the same relaxed problem by rerunning the search. This is due to non-unique sorting behavior of the priority queue implemented in Python's standard libraries. Once we have the initial relaxed plan, we can proceed to the fixture-placement problem for improving the plan's reliability. Some of the relaxed plans are inherently more

robust than others; having the nondeterminism in the priority queue is useful because we acquire plans of similar length but different actions. Because of this randomness in relaxed plans, our experiments average over large numbers of initial relaxed plans.

4.1 Plan improvement with fixture-placement

We look at finding the fixture-placement for a multi-step manipulation plan as a sequence of single-step fixture-placement problems; that is, every push action is converted into a one-step fixture-placement problem and solved independently. To describe this approach, we provide a complete example from a manipulation problem used in our experiments shown in Figure 4-2. This manipulation problem consists of two concave parts: a 'T' shape and 'Z' shape. The goal is to mate the two parts together as shown in Figure 4-2(b) using a short rectangular manipulator and fixtures.

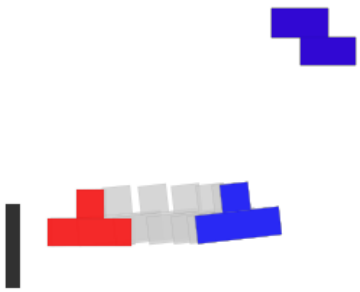


(a) Start state (b) Relative goal configuration

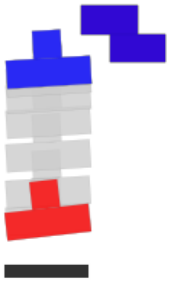
Figure 4-2: TZ mating Problem. *Figure (a) shows the example start state in the TZ mating problem. Figure (b) shows the relative goal configuration of the T and Z shapes. The rotation and position of the mated objects are not specified by a relative goal.*

First, we use the *Arrangement Problem* planner to find an initial deterministic manipulation plan that may not have consistent results when uncertainty is considered. Figure 4-3

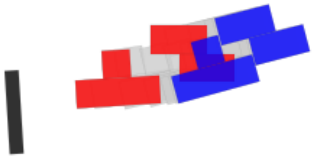
shows a four step manipulation plan found by our planner. It consists of two long pushes to move the 'T' shape into the general vicinity of the 'Z' shape, followed by two short, more precise positioning pushes to mate the two parts together.



(a) Action 1: right push



(b) Action 2: upward push



(c) Action 3: right push

(d) Action 4: downwards push

Figure 4-3: TZ mating nominal solution. *These figures show the solution steps found by the* Arrangement Problem *planner. Initial poses are shown in red and final poses after the pushing action are shown in blue. Only the initial position of the manipulator is shown.*

This deterministic manipulation plan is not robust to uncertainty: the average reliability score over 1000 noisy simulations was only 3.3%! Figure 4-4 shows an example execution of

the nominal plan with uncertainty added that fails to achieve a goal-satisfying configuration. The two long positioning actions add substantial uncertainty making the final two precise positioning actions fail to satisfy the relative goal conditions.

(a) Action 1

(b) Action 2

(c) Action 3

(d) Action 4

Figure 4-4: TZ mating nominal solution with uncertainty. *This figure shows the execution of the nominal solution when there is uncertainty. The parts deviate substantially from the nominal plan by the end of the third action and fails to reach a goal-satisfying state. The average reliability score over 1000 simulations was only 3.3%!*

To improve the reliability of the original manipulation plan, we propose an approach that introduces fixtures during each pushing action. Our detailed approach is detailed in Section 4.1.1, however, Figure 4-5 shows an example solution. Each of the four pushing actions is extended into a place-push-pick action sequence. The most dramatic results are shown in the two long pushing actions. In the first two pushes the augmented plan clearly exploits the fixtures as an uncertainty-reducing device. The final, more precise pushes have

fixture-placements that are not even used.

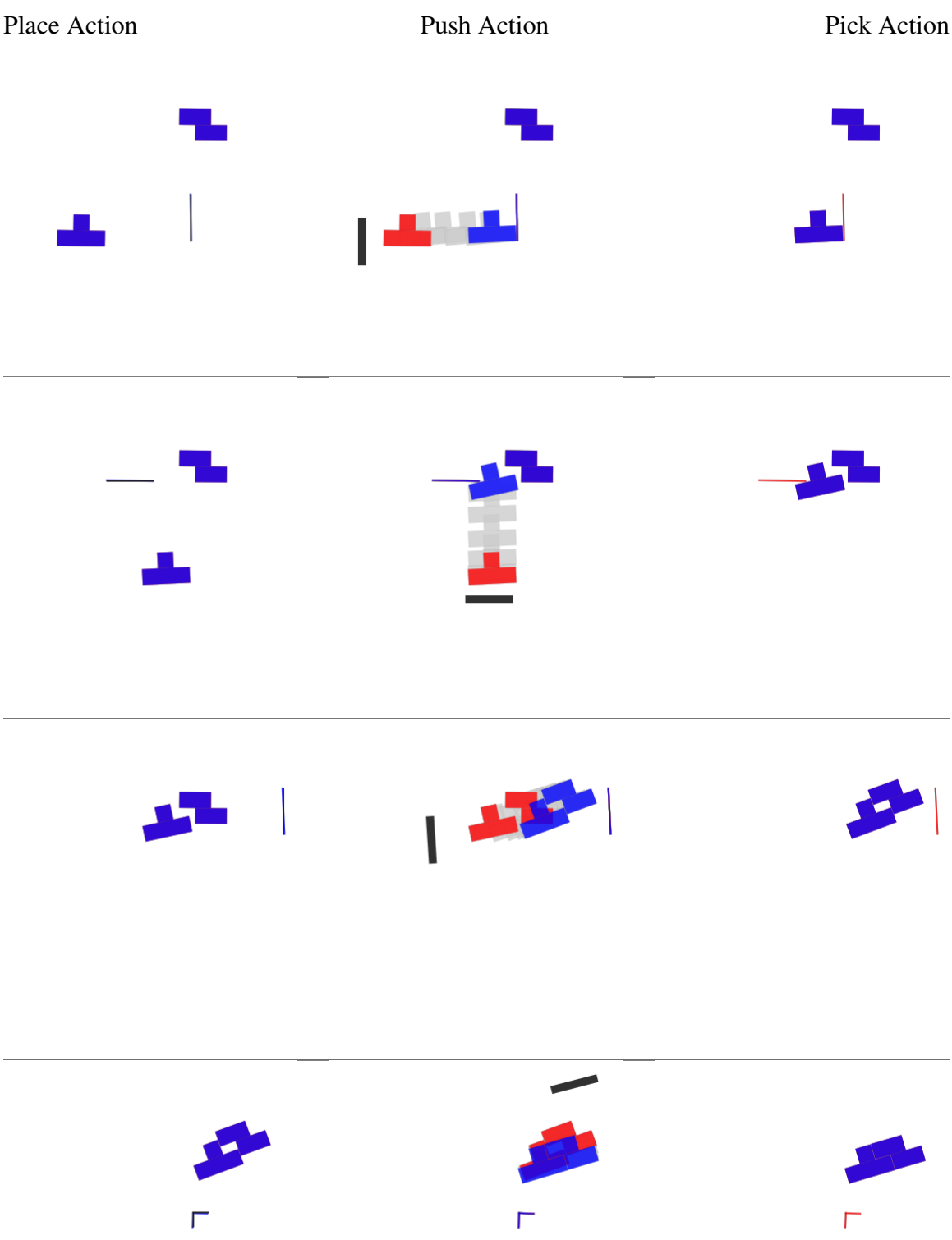


Figure 4-5: TZ mating with fixtures. *Every manipulation action is augmented into a 3 step place, push, pick action. Each row of this figure shows the modified plan using fixtures.*

Although the two latter fixtures are unused, the reason they are placed is elucidated in the approach section below. How useful is this approach? In this particular example, the average reliability of the fixture plan was 80.4% over 1000 simulations– a substantial improvement over the nominal manipulation plan of 3.3%. In the results section we show more rigorous analysis with additional experiments on other manipulation domains.

4.1.1 Approach

The *fixture-placement Problem* problem is formally defined as follows:

> Given a relaxed manipulation plan consisting of a sequence of actions, which we define as $(u_1, u_2, \ldots u_n)$, modify each push action into a sequence $u_{i_{new}} = [u_{place_i}, u_{push_i}, u_{pick_i}]$, that maximizes the average reliability of the augmented plan. The place action uses a *fixture*; the set of possible fixtures is defined by the problem domain definition.

We start with the deterministic *Arrangement Problem* planner to solve a manipulation problem. This planner provides a sequence of actions $(u_1, u_2, \ldots u_n)$, which can be used to find a corresponding sequence of states $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n)$. Typically, we will use the term *nominal plan* to refer to these sequences of states and actions. Our overall approach is to "control along the trajectory", that is, choose fixture-placement actions, $u_{i_{new}}$, that makes each sequential belief state stay as close as possible to the state from the nominal trajectory:

$$u_{i_{new}} = \arg\min_{u} \sum_{\mathbf{x}_{i-1} \in b_{i-1}} ||\mathbf{x}_{i_{nominal}} - f(\mathbf{x}_{i-1}, u)||,$$

where $u = (u_{place}, u_{push_i}, u_{pick})$ and b_{i-1} is the belief state at step $i-1$. The one-step fixture-placement procedure is shown in Figure 4-6.

ONESTEPFIXTUREPROBLEM$(\mathbf{x}_i, u_i, \mathbf{x}_{i+1}, b_i)$:

1 **if** ISPLACEACTION(u_i):

2 **return** u_i

3 $\mathcal{U}_{places}$ = GENERATEPLACEACTIONS$(\mathbf{x}_i, u_i, b_i)$

4 $\mathcal{U}_{pushes}$ = GENERATEPUSHMULTIPLIERS(u_i)

5 u_{place_i} = OPTIMIZEPLACEACTION$(b_i, \mathcal{U}_{places}, \mathbf{x}_{i+1})$

6 u_{push_i} = OPTIMIZEPUSHDISTANCE$(b_i, \mathcal{U}_{pushes}, \ldots$
$\mathbf{x}_{i+1}, u_{place_i}, u_{pick_i})$

7 u_{pick_i} = REVERSEPLACEACTION(u_{place_i})

8 $u_{i_{new}} = [u_{place_i}, u_{push_i}, u_{pick_i}]$

9 **return** $u_{i_{new}}$

Figure 4-6: one-step fixture-placement.

Since we are always trying to stay close to the nominal trajectory, we can consider every pushing action from the nominal plan as an independent fixture-placement problem. (We note that this is a strong assumption, that simplifies the problem, but may cause this approach to fail). For even the simple one-step fixture-placement problem, it is impossible to consider all possible fixture-placements since there are infinitely many possibilities. Instead, we will pose the problem as an optimization over a discrete set of possible placement actions (lines 3 and 5). In Section 4.1.2, we will describe the method for creating the set of possible placements. To reverse the fixture-placement, we will simply append a picking action to remove the fixture (line 7). Typically, a single fixture-placement action alone, will not be sufficient to improve the existing plan. The length of the pushing action itself, will need to be increased in order to 'push up' against a fixture. Therefore, simple push distance multipliers are considered for changing the original push action distance (line 6).

FIXTURESEARCH$(states, actions, b_0)$:

1 $optimizedActions = [\,]$;
2 $b_i = b_0$
3 **for** $i = 0$ **to** LENGTH$(actions)$:
4 $\quad \mathbf{x}_{curr} = states[i]$
5 $\quad u_i = actions[i]$
6 $\quad \mathbf{x}_{next} = states[i+1]$
7 $\quad u_{i_{new}} =$ ONESTEPFIXTUREPROBLEM$(\mathbf{x}_i, u_i, \mathbf{x}_{i+1}, b_i)$
8 $\quad b_i =$ PROPAGATEBELIEF$(b_i, u_{i_{new}})$
9 $\quad$ **if** b_i == **None**:
10 $\quad\quad$ **return None**
11 $\quad$ **else**
12 $\quad\quad optimizedActions$.APPEND$(u_{i_{new}})$
13 **return** $optimizedActions$

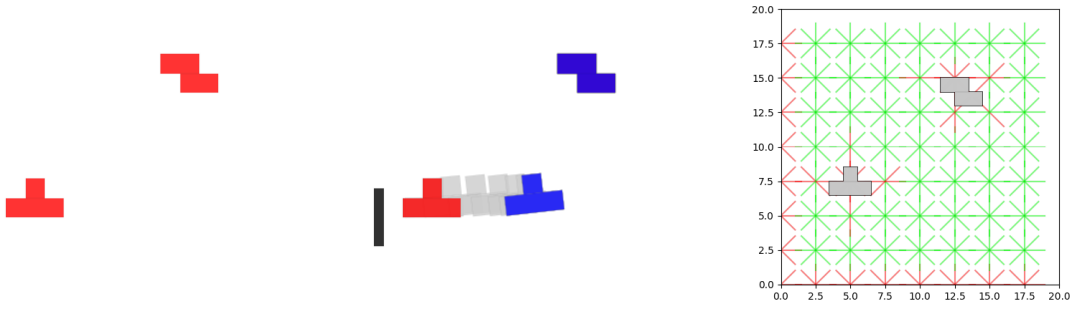
Figure 4-7: Search wrapper for multi-step fixture-placement problems.

The process of selecting an optimal place action followed by a push multiplier is then repeated for each of the n action steps; if an action is not a manipulation action, then the fixture-placement problem is skipped. The multi-step algorithm for fixture-placement is shown in Figure 4-7; as seen in line 8, a "propagateBelief" function is required after optimizing each action using the one-step fixture problem procedure. This belief propagation function is dependent on the type of belief state approximation used; in our case, we used the particle belief state transition, where the action is applied to every particle represented by the belief state to compute the approximate next belief state (see Chapter 3.1 for more details).

The remainder of this chapter proceeds as follows: in Section 4.1.2, we describe the methods for generating candidate fixture-placements and pushing multipliers; then, in Section 4.1.3, we describe the optimization procedure used to rank candidate actions. Finally, in Section 4.2 we construct fixture-placement experiments and analyze the results.

4.1.2 Generating improvement actions

In this section we describe an efficient method for generating candidate fixture-placements given the geometry of a fixture. It is impossible to consider all possible fixture-placements since there are infinitely many; therefore, we have posed the one-step fixture-placement problem as an optimization over a discrete set of possible placement actions. First we will describe a naive approach to fixture-placement followed by a method that looks at object motion during a manipulation action.



(a) initial state $\mathbf{x}_i$ (b) $\mathbf{x}_{i+1}$ (in blue) after right push (c) brute force discretization for fixture-placements

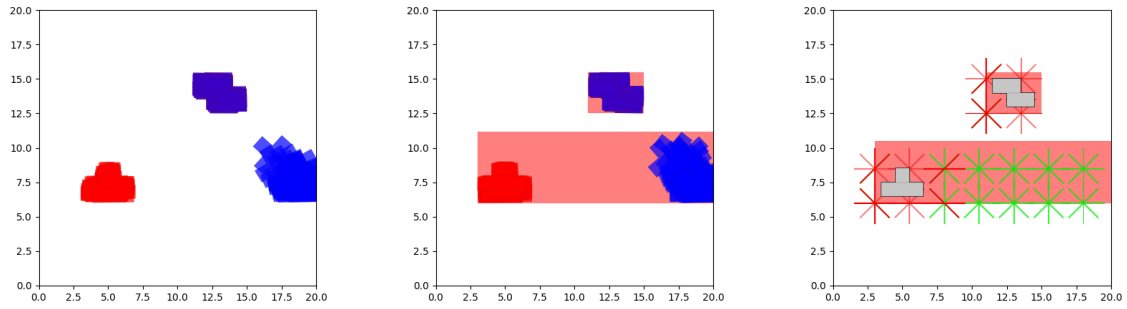
Figure 4-8: Naive fixture-placement. *Figure (a) shows the start state, $\mathbf{x}_i$ of the TZ mating problem. Figure (b) shows the resulting state $\mathbf{x}_{i+1}$ in blue. The push action, u_{push_i}, is also depicted in sub figure (b) by the black manipulator and timelapse part motion. Note that sub figures (a) and (b) show the nominal plan without uncertainty. Figure (c) shows candidate fixture-placements of a short rectangular bar using a workspace discretization method.*

To explain the naive approach for fixture-placement, we will consider the first pushing action of the TZ mating problem. Figure 4-8 shows the first right push action in the TZ mating problem. Sub figure (a) represents the initial state $\mathbf{x}_i$ and sub figure (b) shows the resulting state $\mathbf{x}_{i+1}$ in blue. The push action, u_{push_i}, is also depicted in sub figure (b) by the black manipulator and timelapse part motion. Note that sub figures (a) and (b) show the nominal plan without uncertainty. A simplistic approach for fixture-placement could discretize the workspace to generate candidate fixture-placements. Each candidate fixture-placement of a short rectangular bar is shown in sub figure (c); fixture-placements in red are

invalid due to the obstructions with the initial state or are not located within the workspace.

Using the naive method of fixture-placement generation creates many placement actions that are unlikely to be useful. To reduce the size of the optimization problem and only consider useful placements, we considered an approach that looks at part motion during the manipulation problem. Figure 4-9 presents an overview of this approach. Sub figure (a) shows the start state, $\mathbf{x}_i$ of the TZ mating problem in red and final state, $\mathbf{x}_{i+1}$, in blue. Although this problem has low initial position uncertainty, the final positions of the 'T' shaped object have a much larger spread.

Sub figure (b) shows the *part motion swath*, an axis aligned bounding box over the convex hull of initial and final object belief states. We use the belief propagation transition function with the original pushing action and a 2X distance multiplier to compute the resultant belief state. Then, an axis aligned bounding box is computed to cover the initial and final object belief states. Each part motion swath represents a volume of space that may have contained a part during the manipulation action. The enhanced method will only consider placements that intersect the part motion swath, as shown in sub figure (c).



(a) Noisy simulated pushes (b) part motion swath (c) Enhanced fixture-placements

Figure 4-9: Enhanced fixture-placement. *Figure (a) shows the current belief state, b_i, in red and shows the next belief state b_{i+1} in blue for a sample of noisy pushing actions. Figure (b) shows the* part motion swath, *an axis aligned bounding box over the convex hull of initial and final object poses. Figure (c) shows the enhanced fixture-placement method with a subset of fixture-placements that intersect the part motion swath.*

Pushing multipliers

In the fixture-placement problem, we make the assumption that it might be useful to push up against a fixture. To augment the pushing action, we assume the push starting location will stay the same, but the distance might increase by a factor in $\{1, 1.25, 1.5, 1.75, 2\}$. Additionally, when constructing the object motion swath and selecting the best placement action, we assume the initial pushing multiplier is $2\times$. This means the algorithm will have a tendency to select a fixture, however, the selection of a small multiplier near unity may not make use of the fixture, as seen in the previous TZ mating example for the last two precise positioning pushes.

4.1.3 Selecting improvement actions with optimization

To complete the one-step fixture-placement problem, it is necessary to rank the possible actions. The ranking method for choosing the best placement action and pushing multiplier are equivalent: take the current belief b_i and forward propagate the belief state with every candidate action. Then, compare every resultant belief state with the nominal next state $\mathbf{x}_{i+1}$. The comparison of a belief state with a nominal state is dependent on the representation used for the belief state. In our case, we looked at every simulated next state, $\hat{\mathbf{x}}_{i+1}$, and computed an average Manhattan distance for every part on the manipulation surface.

SCOREACTIONS$(\mathbf{x}_i, \mathbf{x}_{i+1}, b_i, \mathcal{U})$:

```
1   actionDists = [ ]
2   actionScores = [ ]
3   for u_k in U:
4       dists = [ ]
5       for x̂_i in b_i:
6           x̂_{i+1} = SIMULATE(x̂_i, u_k)
7           dist = DISTANCE(x̂_{i+1}, x_{i+1})
8           dists.APPEND(dist)
9       actionDists[k] = AVERAGE(dists)
    // scale all actions from 0 to 1
10  maxDist = MAX(actionDists)
11  minDist = MIN(actionDists)
12  for k = 1 to K:
13      distScore = (actionDists[k] − minDist) / (maxDist − minDist)
14      actionScores[k] = distScore
15  return actionScores
```

Figure 4-10: Scoring candidate actions. *This function shows how to score a set of candidate actions when using the particle belief state representation and transition function.*

The optimization algorithm for ranking candidate actions is shown in Figure 15. Lines 1-9 propagate the approximate belief state and compute the average Manhattan distance for every candidate action. Then, lines 10-14, normalize each candidate action's distance using a linear scaling from the minimum and maximum distances. In some cases the simulation of a candidate action will lead to an invalid state. Any action that leads to an invalid state automatically receive a score of infinity; note that the infinite values are not used when scaling action scores.

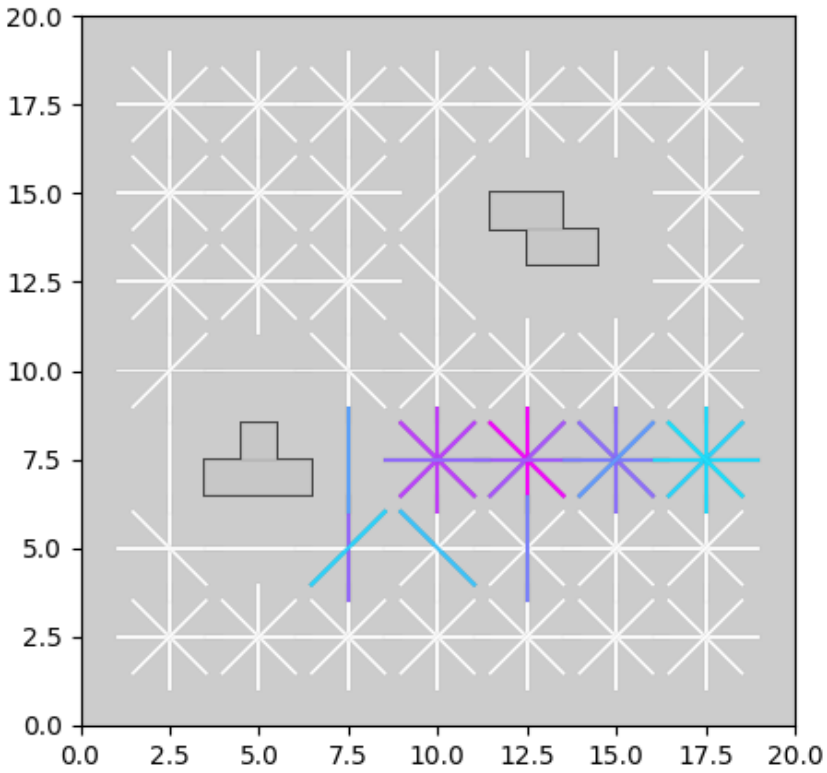


Figure 4-11: TZ mating fixture-placement heatmap. *This figure shows a heatmap of all the possible placement positions. Fixtures shown in white are invalid since they lead to action violations where the part moves off the allowable workspace. The bright cyan colors above the nominal part pose interfere with the pushing action yielding an undesirable, but legal final state. This cyan blue is the lowest possible score in the heatmap. The bright purple shown in the middle are the best locations for fixture-placement.*

Following our initial example of the TZ mating problem, we show a heatmap of all possible fixture-placements in Figure 4-11. Recall that the pushing distance is doubled when initially selecting the best fixture-placement location. Since the distance is increased the T shape may get pushed outside of the manipulation workspace, which leads to an invalid state or score of infinity. All of the colored placement fixtures are considered during the optimization, with magenta yielding the highest value and cyan having the lowest possible score.

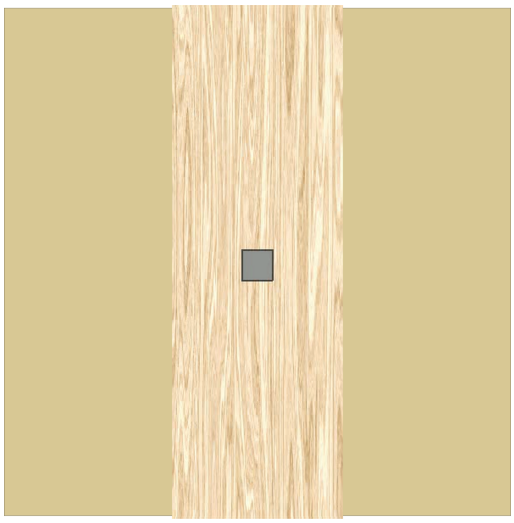
4.2 Experiments and results

4.2.1 Experiments

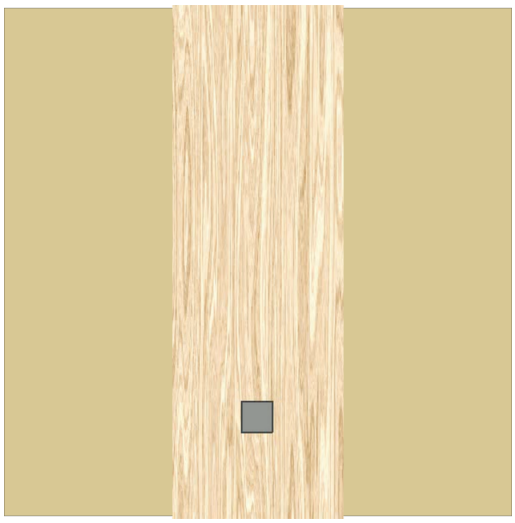
To evaluate the performance of the plan improvement algorithm, we used the percent reliability measure discussed in Section 3.2. Similarly to our previous experiments, we used a deterministic physics simulator and stochastically sampled domain parameters at every primitive action step to simulate noisy transitions. Table 4.1 shows the experiment set up. For every manipulation problem, we found 20 nominal plans assuming deterministic transitions. For each nominal plan we ran the plan improvement algorithm as specified in this chapter with a maximum search time of 24 hours. Each relaxed plan and improved plan was simulated with 1000 noisy roll-outs. Table 4.2 shows the uniform sample distribution for the domain parameters. Finally, Figure 4-12 and Figure 4-13 show the start and goal state for the manipulation problems we are evaluating the plan improvement on.

Parameter name	Value
Deterministic plans per manipulation problem	20
Maximum search time	24 hours
Simulations per reliability experiment	1000
Belief-state particles	10, 20, 40, 80, 150, 200, 500

Table 4.1: **Parameters for running experiments.** *This table shows the different experiment parameters that were used when running the experiments.*



(a) Downwards push start



(b) Downwards push goal

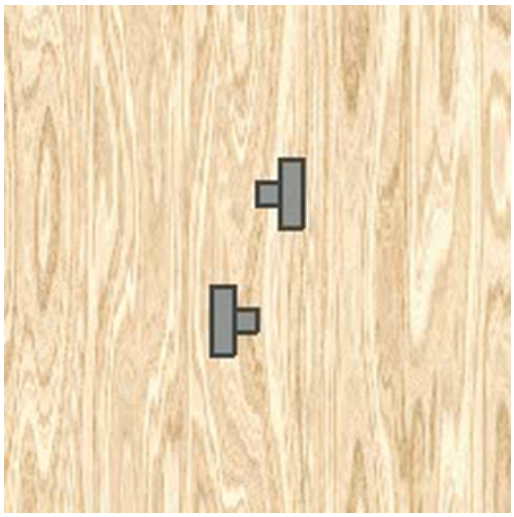
(c) Create a T start

(d) Create a T goal

Figure 4-12: Rearrangement manipulation problems (1/2). *In these manipulation problems, the parts are already located on the manipulation surface. The robot can use pushing and placing of fixtures to rearrange objects into a desired arrangement.*

(a) TZ mating start

(b) TZ mating goal



(c) TT mating start

(d) TT mating goal

Figure 4-13: Rearrangement manipulation problems (2/2). *In these manipulation problems, the parts are already located on the manipulation surface. The robot can use pushing and placing of fixtures to rearrange objects into a desired arrangement.*

Problem name	Initial pose unc (inches, degrees)	Action unc (inches, degrees)	Friction Coeff range ($\mu_k \in [0, 1]$)	Part density range	Manipulator density range
Create a T	$(\pm 0.35'', \pm 5^\circ)$	$(\pm 0.05'', \pm 5^\circ)$	$(0.5, 0.6)$	$(0.1, 0.15)$	$(0.35, 0.45)$
TZ mating	$(\pm 0.05'', \pm 1^\circ)$	$(\pm 0.05'', \pm 3^\circ)$	$(0.5, 0.6)$	$(0.1, 0.15)$	$(0.35, 0.45)$
TT mating	$(\pm 0.5'', \pm 5^\circ)$	$(\pm 0.05'', \pm 3^\circ)$	$(0.5, 0.6)$	$(0.1, 0.15)$	$(0.35, 0.45)$
TT mating (noisy domain parameters)	$(\pm 0.5'', \pm 5^\circ)$	$(\pm 0.05'', \pm 3^\circ)$	$(0.5, 0.6)$	$(0.01, 1.0)$	$(0.1, 1.0)$
Down Push	$(\pm 1.0'', \pm 15^\circ)$	$(\pm 0.2'', \pm 1^\circ)$	$(0.5, 0.6)$	$(0.1, 0.15)$	$(0.55, 0.6)$

Table 4.2: **Domain parameters for experiments.** *This table shows the different parameters used for sampling domain setting values.*

4.2.2 Results

Figure 4-14 provides an overview of the reliability analysis of the deterministic and improved plans. Every optimized plan used a belief-state particle transition with 150 particles. For every manipulation problem, the optimized plan has a higher reliability average. For some problems, the deterministic plan was very good, resulting in little improvement in the optimized plan. We analyze each optimized plan's performance as a function of particles below.

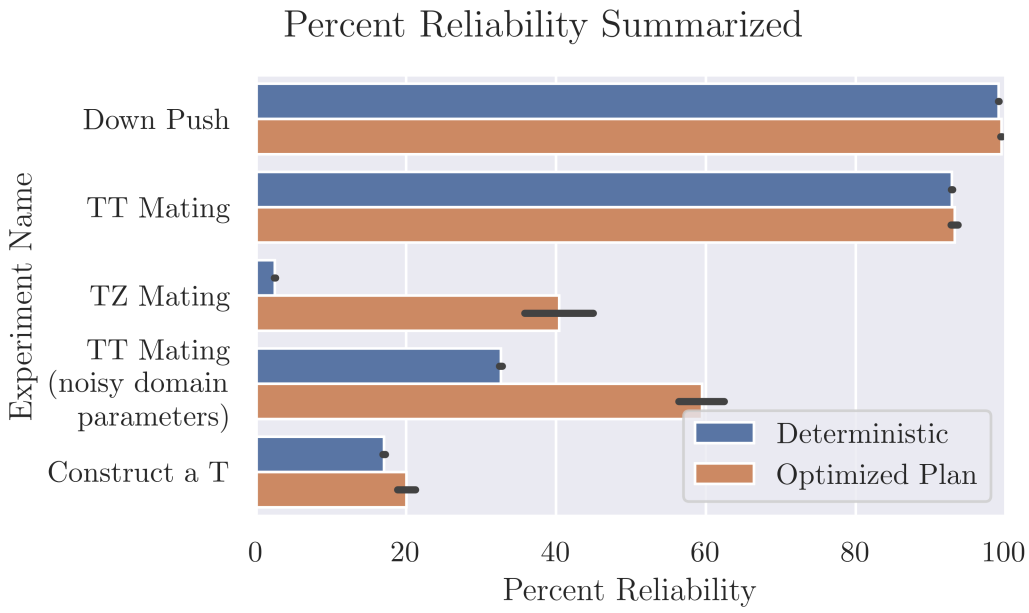
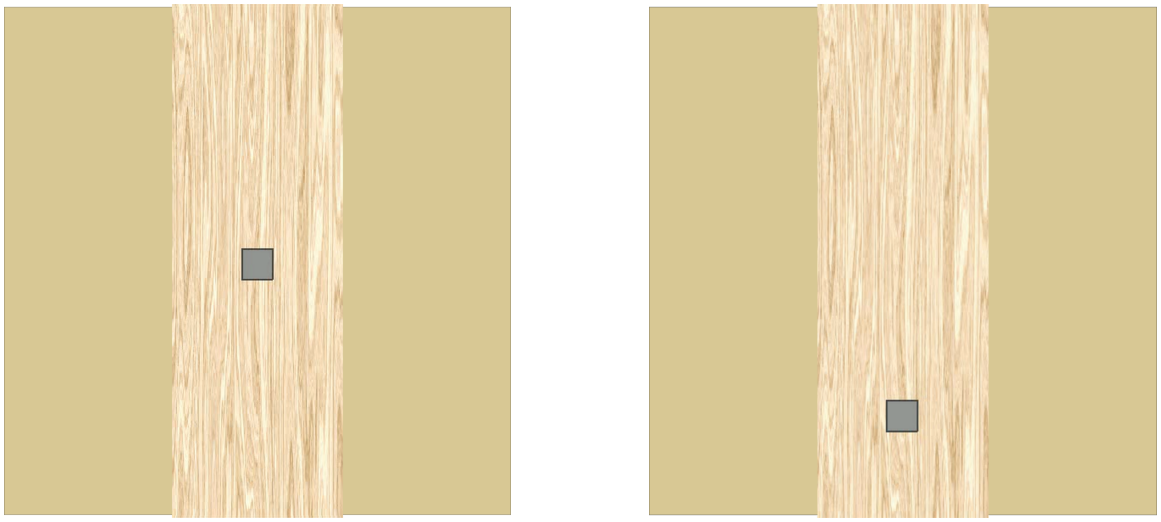


Figure 4-14: Average percent reliability. *This figure shows performance reliability across multiple manipulation problems. The number of particles used in the optimized planner was 150.*

Down Push results

In the Down Push problem, the goal is to push a block from the middle of the workspace downwards a set amount. This problem has only a single part and an absolute (rather than relative) goal target. Figure 4-15 shows the start and end location for the part in this manipulation problem.



(a) Downwards push start

(b) Downwards push goal

Figure 4-15: Down Push experiment. *Figure (a) shows the start location of the part and Figure (b) shows the goal location of the part. The goal is to push the part down to its desired goal target. In initial pose of the part has noise added and is not shown here.*

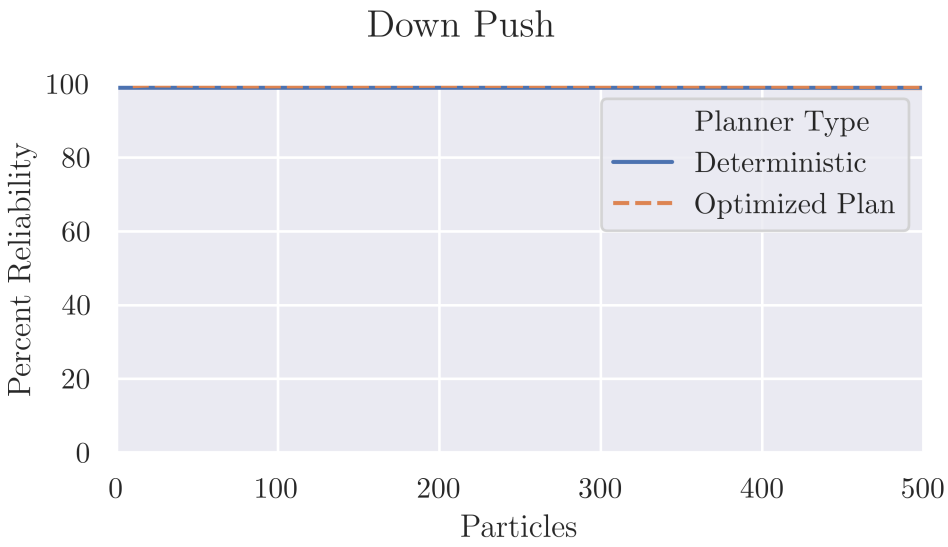


Figure 4-16: Down Push percent reliability plot. *This plot shows the average reliability of the fixture-improved plan as we increase the number of particles used to approximate a belief-state transition. The line for the deterministic reliability is super-imposed onto the graph but always used a single particle for finding the relaxed plan.*

This is a a relatively easy problem for planning and execution. As shown in the percent reliability plot in Figure 4-16, the deterministic plan has nearly perfect reliability. The optimized plan shows similarly strong reliability scores. There is no noticeable effect from increasing particles on the reliability score. The planning time, however, increases linearly as a function of number of particles as shwon in Figure 4-17. The optimized plan length is about 4 action steps higher than the deterministic plan. The increased action steps settles to 12 total actions as the number of particles increases. These 4 additional action steps correlate to two fixture-placements. Overall, the optimized plan was unnecessary for this manipulation problem since reliability performance was already high.

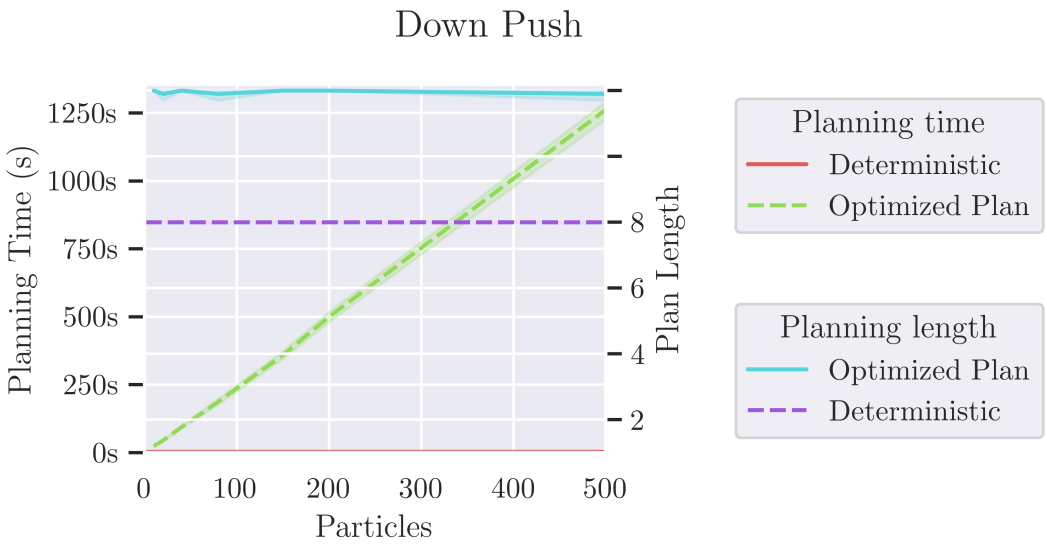
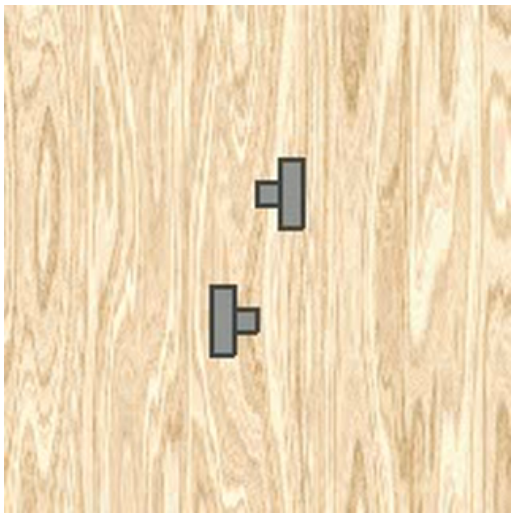


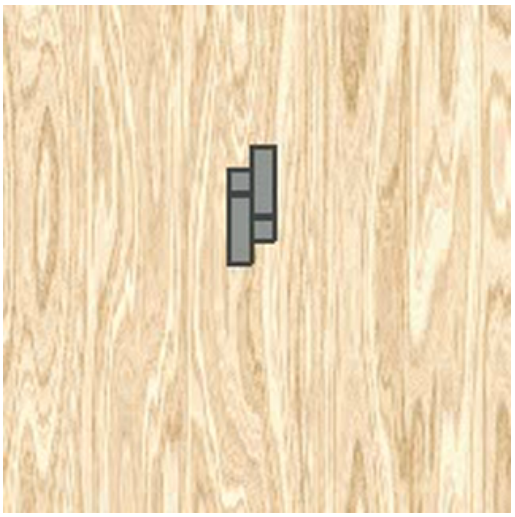
Figure 4-17: Down Push Planning time and plan length plot. *The independent axis shows the increase of the number of particles used to approximate a belief-state transition. This plot shows the timing performance with the vertical axis on the left; as expected, increasing the number of particles increases the planning time. The vertical axis on the right show the average number of steps in the manipulation plan. The augmented number of steps is typically flat: pushing actions from the deterministic plan get increased due to the added pick and place actions for the fixtures.*

TT Mating results

The TT Mating problem has two non-convex parts that are pushed together. The manipulation paddle is fairly long and the domain parameter ranges are relatively low. These features make the problem more reliable as evident in the percent reliability plot. The optimized plan has a lower reliability than the deterministic plan when implemented with 10 samples for the particle belief-state. Increasing the number of particles makes the optimized plan slightly better than the deterministic plan. However, the amount of improvement for the optimized plan is very small. As expected, the planning time increases with the number of particles and the plan lengths for both the deterministic and optimized case are static based on number of particles. In summary, the optimized planner was again not necessary in this problem because the deterministic plan was already very reliable.



(a) TT mating start



(b) TT mating goal

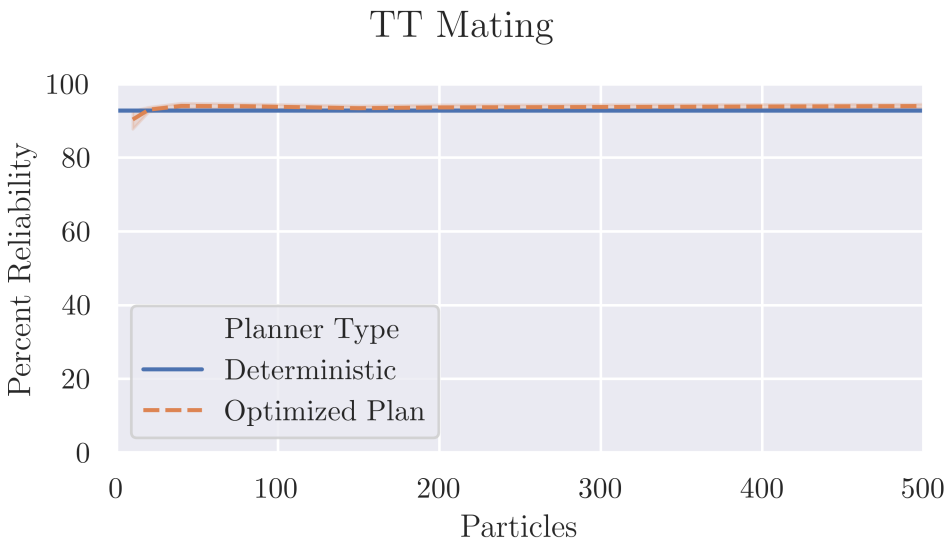


Figure 4-18: TT Mating percent reliability plot. *This plot shows the average reliability of the fixture-improved plan as we increase the number of particles used to approximate a belief-state transition. The line for the deterministic reliability is super-imposed onto the graph but always used a single particle for finding the relaxed plan.*

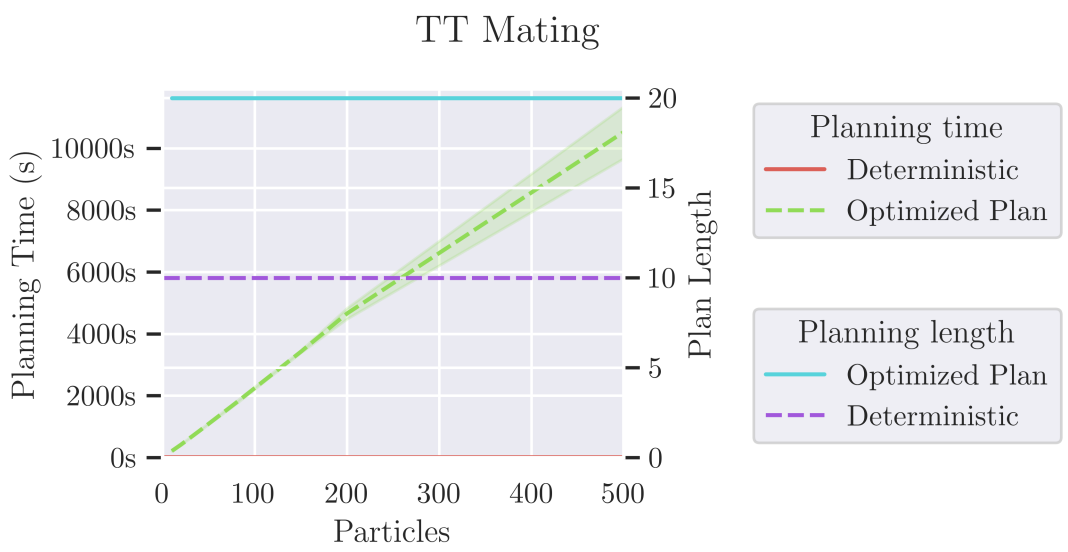


Figure 4-19: TT Mating Planning time and plan length plot. *The independent axis shows the increase of the number of particles used to approximate a belief-state transition. This plot shows the timing performance with the vertical axis on the left; as expected, increasing the number of particles increases the planning time. The vertical axis on the right show the average number of steps in the manipulation plan. The augmented number of steps is typically flat: pushing actions from the deterministic plan get increased due to the added pick and place actions for the fixtures.*

TZ mating results

The TZ mating problem has two different non-convex parts that are pushed together. The manipulation paddle is of medium length and the domain parameter ranges have a moderate size. These features make the problem very unreliable for a deterministic planner as evident in the percent reliability plot.

The optimized plan always has a higher reliability score than the deterministic plan. Increasing the number of particles makes the optimized plan improve its reliability score, however there is always a large range of noise in the reliability of the optimized plan. As expected, the planning time increases with the number of particles and the plan lengths for both the deterministic and optimized case are static based on number of particles. Overall, the optimized planner was beneficial for this manipulation problem and greatly improved the reliability of the deterministic plan.

(a) TZ mating start

(b) TZ mating goal

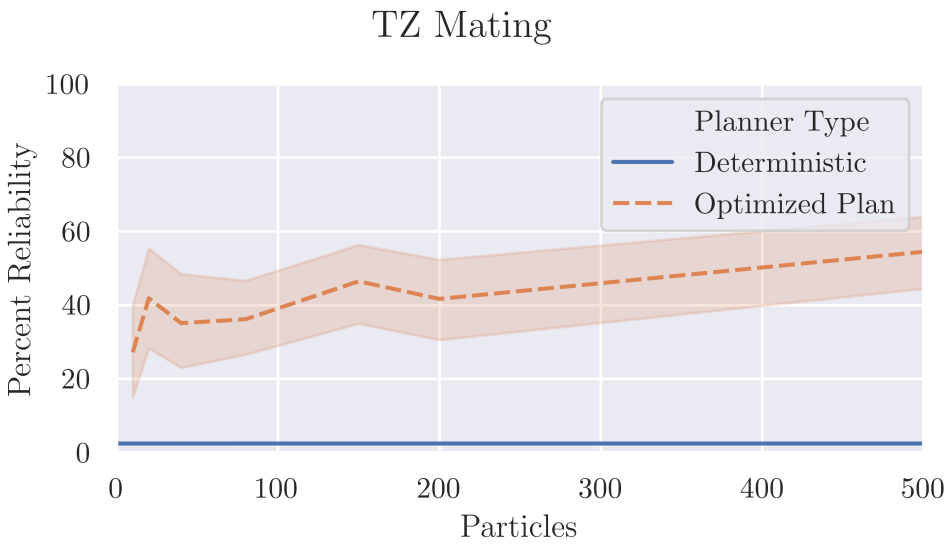


Figure 4-20: TZ mating percent reliability plot. *This plot shows the average reliability of the fixture-improved plan as we increase the number of particles used to approximate a belief-state transition. The line for the deterministic reliability is super-imposed onto the graph but always used a single particle for finding the relaxed plan.*

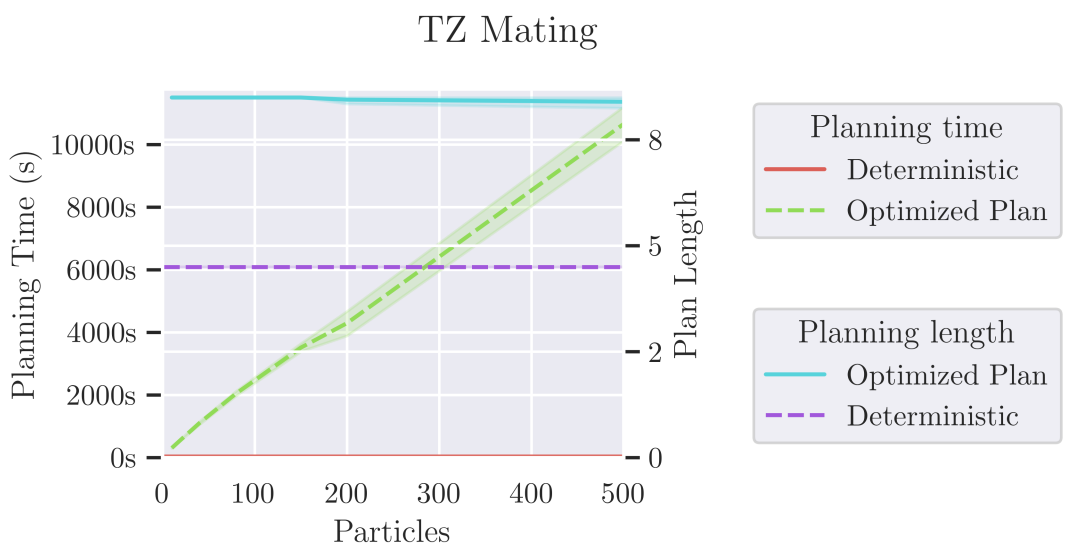
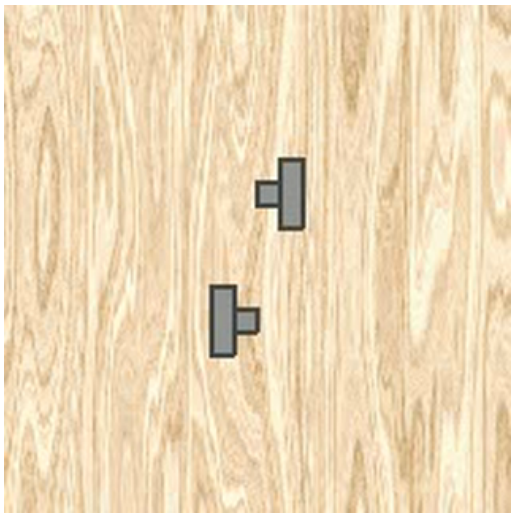


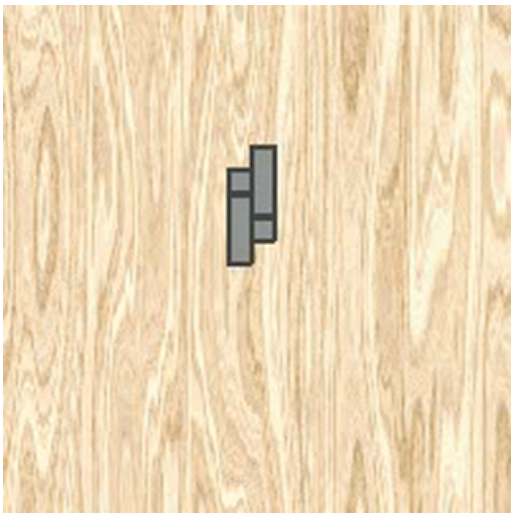
Figure 4-21: TZ mating Planning time and plan length plot. *The independent axis shows the increase of the number of particles used to approximate a belief-state transition. This plot shows the timing performance with the vertical axis on the left; as expected, increasing the number of particles increases the planning time. The vertical axis on the right show the average number of steps in the manipulation plan. The augmented number of steps is typically flat: pushing actions from the deterministic plan get increased due to the added pick and place actions for the fixtures.*

TT Mating results (noisy domain parameters)

This TT Mating problem has very noisy domain parameters, which makes the deterministic plan very unreliable. The optimized plan improves the reliability of the plan from a little over 30 percent to around 60 percent reliability. One unique feature of this problem is that increasing the particles tends to have very little change on the overall reliability. This implies that the initial plan has very little room for improvement with fixture-placement and a different planning method would be better suited. As expected, the planning time increases with the number of particles and the plan lengths for both the deterministic and optimized case are mostly static based on number of particles.



(a) TT mating start



(b) TT mating goal

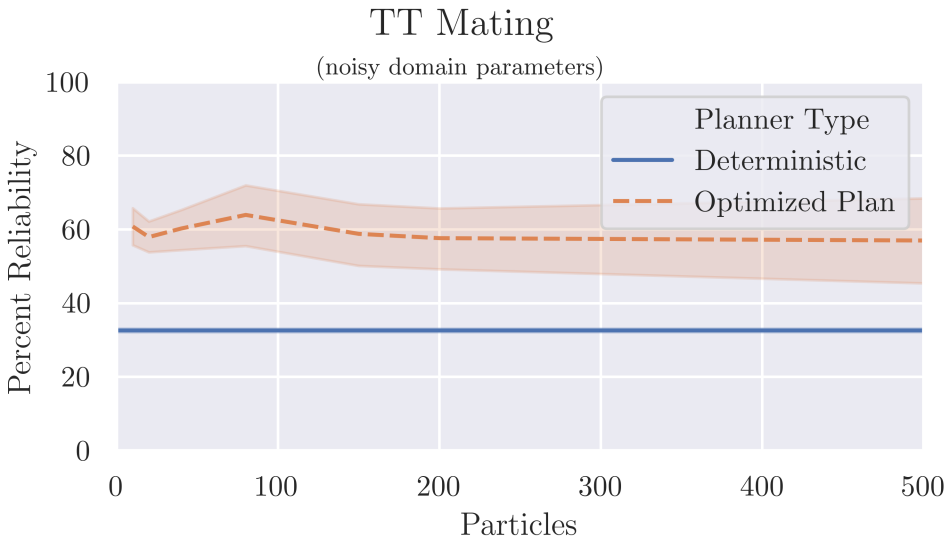


Figure 4-22: TT Mating (noisy domain parameters) percent reliability plot. *This plot shows the average reliability of the fixture-improved plan as we increase the number of particles used to approximate a belief-state transition. The line for the deterministic reliability is super-imposed onto the graph but always used a single particle for finding the relaxed plan.*

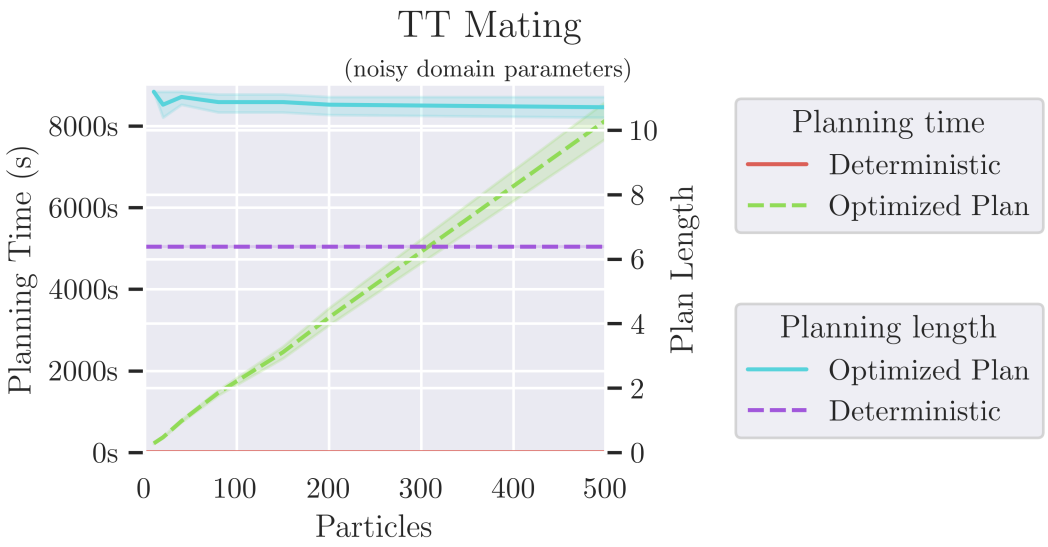


Figure 4-23: TT Mating (noisy domain parameters) Planning time and plan length plot. *The independent axis shows the increase of the number of particles used to approximate a belief-state transition. This plot shows the timing performance with the vertical axis on the left; as expected, increasing the number of particles increases the planning time. The vertical axis on the right show the average number of steps in the manipulation plan. The augmented number of steps is typically flat: pushing actions from the deterministic plan get increased due to the added pick and place actions for the fixtures.*

Construct a T results

The construct a T problem is by far the most challenging manipulation problem in this dataset. Both the deterministic and optimized plan fail to achieve very reliable performance. This problem was unable to be improved by adding fixtures: the average reliability is virtually flat while increasing the number of particles. As expected, the planning time increases with the number of particles and the plan lengths for both the deterministic and optimized case are mostly static based on number of particles. Overall, the optimized planner was unable to improve the reliability performance.

(a) Construct a T start

(b) Construct a T goal

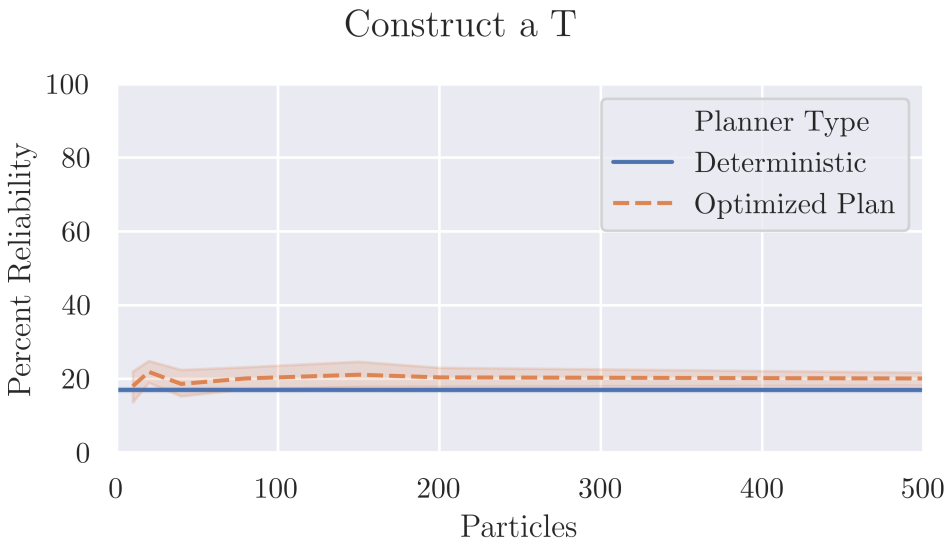


Figure 4-24: Construct a T percent reliability plot. *This plot shows the average reliability of the fixture-improved plan as we increase the number of particles used to approximate a belief-state transition. The line for the deterministic reliability is super-imposed onto the graph but always used a single particle for finding the relaxed plan.*

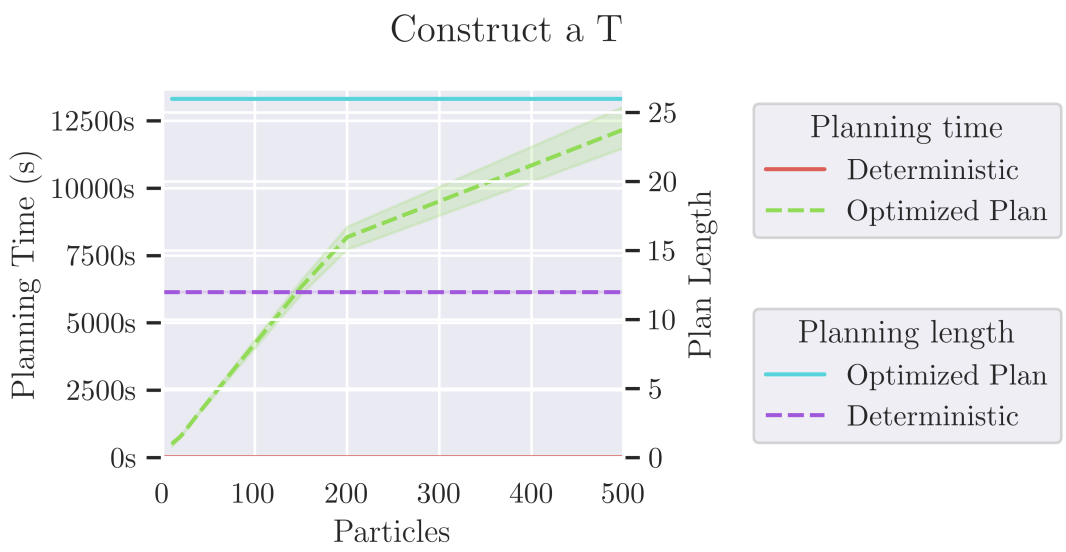


Figure 4-25: Construct a T Planning time and plan length plot. *The independent axis shows the increase of the number of particles used to approximate a belief-state transition. This plot shows the timing performance with the vertical axis on the left; as expected, increasing the number of particles increases the planning time. The vertical axis on the right show the average number of steps in the manipulation plan. The augmented number of steps is typically flat: pushing actions from the deterministic plan get increased due to the added pick and place actions for the fixtures.*

4.2.3 Chapter discussion

Placing fixtures as a method of planning improvement has mixed results. In some manipulation problems, the optimized planner was significantly better than the deterministic method; however, in other problems the optimized planner failed to dramatically improve the reliability of the plans. In all cases, conformant performance, or 100 percent accuracy, was never achieved. Not achieving fully conformant plans could be caused by a multitude of reasons: perhaps the candidate fixture-placement actions was insufficient and we should have generated more candidates. Another reason for poor performance could be the deterministic plan skeleton is unable to be improved.

Although plans were unable to reach full conformant performance, every manipulation problem had some percent increase in terms of reliability. This improvement confirms that even noisily placed fixtures are useful tool for reducing uncertainty. A future direction for

this work would combine the fixture-placement methods in this chapter with the "plan by construction" approach described in Chapter 3.

Chapter 5

Conclusion

In this final chapter, we discuss the contributions of this thesis, three areas for future work, and conclude with two unaddressed challenges for future work in robot manipulation.

5.1 Contributions

In this thesis we looked at the problem of conformant robot manipulation. Our main problem was to find a sequence of actions that construct a desired arrangement and are robust to uncertainty. Our approach considered uncertainty in initial part position, actuation noise, and unknown domain characteristics, such as friction and density. The detailed models of manipulation actions were constrained to 2D and modeled as a tabletop pushing problem. The main contributions of this work are the following:

1. A unifying approach for conformant manipulation that allows for adaptable belief-state transition models and primitive actions.

2. An extension of prior work to more challenging construction problems with multiple objects in goal arrangements.

3. A novel approach for plan improvement with fixture placements.

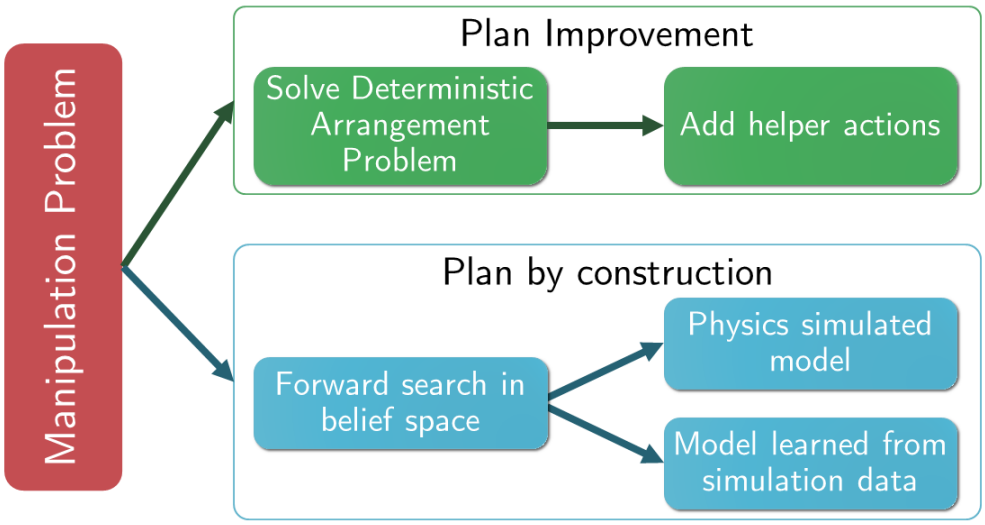


Figure 5-1: Approaches summary. *In this thesis we looked at two main approaches for finding a conformant plan.*

5.2 Future work

We looked at two main approaches for finding a conformant plan that are shown in Figure 5-1. In Chapter 3, we considered finding a conformant plan by construction and in Chapter 4 we found a relaxed plan and improved it using fixture placement. Both of these approaches have their merits: the "by construction" approach yields very robust plans, but takes a significant amount of planning time (or prior belief-state transition learning time). Alternatively, the "improvement" approach improves the reliability of a plan, but might not improve the accuracy very much because the initial plan was too unreliable. An interesting area for future work would combine the "by construction" and "improvement" approaches into a singular and more efficient planning method. Agboh and Dogar's recent work moves in this direction— they created a planning and control approach that determines whether to use a "fast" or "slow" push based on task accuracy and uncertainty level [2].

In this work, we did not require additional external sensing actions; however, this restricted the space of manipulation problems we can solve. In some situations, a conformant

plan might not be available and some other method, such as visual servoing, would be required to do a manipulation action. This leads to a different direction of future work, that would combine this manipulation framework with a contingency planner. The simplest extension, would be to use this in a replanning framework, but we can imagine a more advanced system that can reason between using a conformant manipulation planner or a contingency planner when completing a long-horizon manipulation problem.

A third and very important direction for this work is to increase the manipulation problem space. We considered a very simple pushing action with a rectangular shaped manipulator and perpendicular pushing distance; however, it would be interesting to consider a variable speed pushing action that does not have perpendicular constraints. Although a more complicated pushing action significantly increases the potential action space yielding a more challenging planning problem, it could potentially provide a very compliant pushing action for doing complex manipulation tasks.

Another dimension for increasing the problem space is to expand to general 3D dynamics. The sample based approaches presented here can be directly applied with 3D physics simulators, but would likely require a prohibitive amount of simulation to accurately represent a belief-state transition. The factored belief-state transition models explored in Section 3.2 would be an interesting place for further exploration to expand to more general models and more viable with the recent advances in machine learning.

5.3 Unaddressed challenges

Lastly, we must spend time considering the original problem of a robot completing useful manipulation tasks in a real world environment such as the ones in Figure 5-2.

(a) A messy kitchen. (b) Closet of boxes

Figure 5-2: Motivating manipulation problems. *These are photos of real-world environments that a household helper could potentially work in. Figure (a) is a photo of a kitchen in the wild that is a challenging environment for a robotic household helper and Figure (b) is a storage facility with haphazardly stacked boxes. Both of these settings are challenging since there are many different objects that are stacked upon one another, poor lighting conditions, and tightly constrained spaces.*

In a real world environment, we won't have complete control over all of the physical objects. For example, in a household task, a robot might have to pick up a plate, open a cabinet, and stack the plate upon a set of other plates. Immediately, there are hundreds of unknown variables about the *domain properties* of this household setting: what are the friction properties of the hinges on the cabinet? what is the mass of the plate? Suppose we know all of the characteristics about the domain *a priori*, we may not know the current *state* of the kitchen: What position is the plate at? Where is the robot located within the kitchen? In this work, we considered only a subset of unknown object dynamics. Before robots can exist in the wild, we must expand to more general models, create more accurate estimation techniques, and find planning methods that are capable of handling unexpected and unknown objects in the environment.

Planning in open domains is an active area of research and many interesting ideas for this area are being studied: for example, Talamadupula *et al* use replanning whenever a robot discovers a new object in the enviornment [43]. Recent work from Kaelbling *et al* that explores robot manipulation in open and uncertain domains concludes that a tight integration of state estimation and belief-state planning mechanisms are capable of communicating

goals to robots even when relevant oject properties are unknown [18]. Another solution area uses human dialog to help with clarifying object references and world uncertainty [40, 16]. In this thesis, we did not address open domains, however, robots will need this capability when interacting in real world household domains.

Figure 5-3: Confused robot. *A robot in a real world environment will not have prior knowledge about all of the objects in the world. This robot is currently confused about the properties of objects it found in the kitchen.*

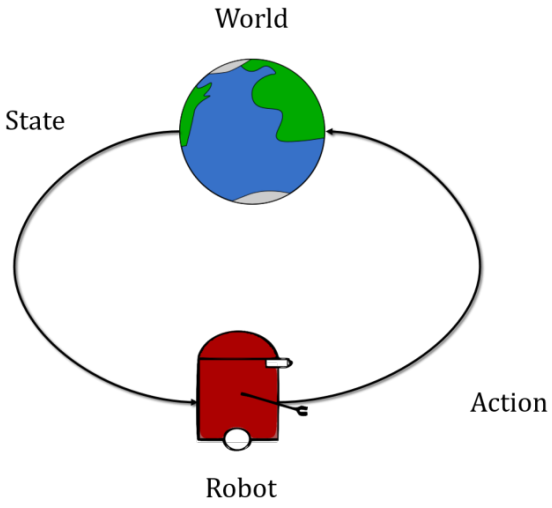


Figure 5-4: Closed world assumption. *In a sequential robotic task, a robot takes an action given the state of the world. This action may involve physically interacting with items in the environment, thus changing the current state of the world. Note that model of the world does not consider other agents behaving and changing the environment.*

We will conclude with pointing out that most of the modeling and planning of robot

manipulation uses a closed world assumption. Figure 5-4 depicts a robot completing a sequential robotic task. A robot takes an action given the *state* of the world. This action may involve physically interacting with items in the environment, thus changing the current state of the world. Then, the robot executes the next action, once again changing the state of the world. Note that is a strong assumption for modeling the world because it does not allow for other agents to exist and change environment. A major hurdle for robot manipulation, is to consider safe and robust manipulation when there are adversaries in the environment, such as a human trying to make the robot fail at its manipulation task.

Bibliography

[1] ROS: an open-source robot operating system. `http://www.ros.org`.

[2] Wisdom C. Agboh and Mehmet Remzi Dogar. Pushing fast and slow: Task-adaptive MPC for pushing manipulation under uncertainty. *Computing Research Repository (CoRR)*, abs/1805.03005, 2018.

[3] Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation with multiple action types. In *Experimental Robotics*, pages 531–545. Springer, 2013.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence*, pages 1023–1028, 1994.

[6] Erin Catto. Box2D: A 2D physics engine for games, 2008.

[7] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa. Physics-based grasp planning through clutter. *Robotics: Science and Systems (RSS)*, 2012.

[8] M. Dogar and S. Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and Systems (RSS)*, 2011.

[9] David H. Eberly. *Game physics*. CRC Press, 2010.

[10] S. Elliott, M. Valente, and M. Cakmak. Making objects graspable in confined environments through push and pull manipulation with a tool. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[11] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *Robotics and Automation*, 1988.

[12] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. *Computing Research Repository (CoRR)*, 2016.

[13] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, NY, USA, 2001.

[14] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.

[15] R. Goldman and M. Boddy. Expressive planning and explicit knowledge. *Artificial Intelligence Planning and Scheduling (AIPS)*, 1996.

[16] Julian Hough and David Schlangen. It's not what you do, it's how you do it: Grounding uncertainty for a simple robot. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 274–282. ACM, 2017.

[17] Aaron Johnson, Jennifer King, and Siddhartha Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters (RA-L)*, 2016.

[18] Leslie Pack Kaelbling, Alex LaGrassa, and Tomas Lozano-Perez. Specifying and achieving goals in open uncertain robot-manipulation domains. In *Proceedings of the Robotics: Science and Systems 2018 Workshop on Models and Representations for Natural Human-Robot Communication*, July 2018.

[19] Jennifer King. *Robust Rearrangement Planning using Nonprehensile Interaction*. PhD thesis, Carnegie Mellon University, 2016.

[20] Jennifer King, Joshua Haustein, Siddhartha Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[21] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[22] M. Kopicki, S. Zurek, R. Stolkin, T. Moerwald, and Jeremy L. Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, 2017.

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

[24] Michael Koval, Jennifer King, Nancy Pollard, and Siddhartha Srinivasa. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[25] Michael C. Koval, Nancy S. Pollard, and Siddhartha S. Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264, 2016.

[26] O. Kroemer, S. Leischnig, S. Luettgen, and J. Peters. A kernel-based approach to learning contact distributions for robot manipulation tasks. *Autonomous Robots*, 2017.

[27] Athanasios Krontiris and Kostas E. Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems (RSS)*, 2015.

[28] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 1995.

[29] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. Pokorny, A. Dragan, and K. Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. *CASE*, 2016.

[30] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[31] T. Lozano-Perez, M. Mason, and R. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research (IJRR)*, 1984.

[32] K. Lynch and M. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research (IJRR)*, 1996.

[33] M. Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

[34] Matthew T. Mason. The mechanics of manipulation. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 544–548. IEEE, 1985.

[35] T. Meriçli, M. Veloso, and H Akın. Push-manipulation of complex passive mobile objects using experimentally acquired motion models. *Autonomous Robots*, 2015.

[36] Jun Nakanishi, Michael Mistry, and Stefan Schaal. Comparative experiments on task space control with redundancy resolution. In *in IEEE International Conference on Intelligent Robots and Systems*, pages 1575–1582. IEEE, 2005.

[37] C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 1987.

[38] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. *Robotics: Science and Systems (RSS)*, 2010.

[39] J. Scholz and M. Stilman. Combining motion planning and optimization for flexible robot manipulation. *International Conference on Humanoid Robots*, 2010.

[40] Niels Schütte, Brian Mac Namee, and John Kelleher. Robot perception errors and human resolution strategies in situated human–robot dialogue. *Advanced Robotics*, 31(5):243–257, 2017.

[41] Scikit-learn. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR)*, 2011.

[42] Mike Stilman and James J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[43] Kartik Talamadupula, J. Benton, Subbarao Kambhampati, Paul Schermerhorn, and Matthias Scheutz. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):14, 2010.

[44] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[45] C. Yu, J. Chuang, B. Gerkey, G. Gordon, and A. Ng. Open loop plans in POMDPs. Technical report, Stanford University CS Dept, 2005.