Manipulation with Diverse Actions

by

Jennifer L. Barry

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author ....
Department of Electrical Engineering and Computer Science
May 16, 2013

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

Certified by..
Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ..
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Manipulation with Diverse Actions

by

Jennifer L. Barry

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We define the Diverse Action Manipulation (DAMA) problem in which we are given a mobile robot, a set of movable objects, and a set of diverse, possibly non-prehensile manipulation actions, and the objective is to find a sequence of actions that moves each of the objects to a goal configuration. We argue that classic sampling-based techniques cannot solve DAMA problems because of the need to move through lower-dimensional subspaces, and we give two sampling-based algorithms for this problem, DARRT and DARRTConnect, based on the RRT and RRTConnect algorithms respectively.

We also show that the DAMA problem can be framed as a multi-modal planning problem [14] and describe a hierarchical algorithm, DARRTH(Connect), that takes advantage of this multi-modal nature. This algorithm finds a high-level sequence of transfer manipulations by planning a path only for objects in the domain. It then attempts to achieve each transfer manipulation individually.

We present experimental results for all four algorithms for a set of nine problems in two complicated mobile manipulation domains. We show that the bi-directional algorithms are faster than their forward search counterparts and that the hierarchical algorithms perform better than the monolithic searches. We also formally define the conditions under which DARRT is exponentially convergent and prove that these conditions hold for two example manipulation domains, one of which includes non-prehensile manipulation.

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

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering

Acknowledgments

I must first thank my thesis advisors, Leslie Kaelbling and Tomás Lozano-Pérez, for their support and guidance. I am in awe of their ability to listen to half-baked, incoherently explained ideas (in my case, usually accompanied by a diagram of some circles connected with lines) and ask precisely the right question. I cannot count the times I spent hours wrestling with a problem only to have Tomás or Leslie solve it in under five minutes. They were also unfailingly kind and good humored. In six years, I think Tomás made at least one joke and Leslie said something cheerful and encouraging in every single meeting I had with them. They were the ones to realize that I was unhappy with my previous research direction and suggest I switch to robotics. I cannot imagine better advisors.

My thesis committee members, David Hsu and Nicholas Roy, gave me some great comments on my work. Nick met with me several times over the last two years to discuss ongoing research and interesting new directions. David read the drafts of this thesis in detail and helped me correct the math.

I am also grateful to everyone at Willow Garage, both for the use of and help with the PR2 robot and for my internship there. At one point, we had a problem with our robot that we could not fix on-site but were up against a deadline. They shipped the robot across the country, fixed it, and shipped it back in a total of ten days. Kaijen Hsiao mentored me during my internship and then continued watching out for me even after I left. She has been both a role model and a good friend.

CSAIL and particularly the LIS group were incredibly supportive. Teresa Cataldo made sure that every presentation or trip I took with the group went smoothly and included food. The discussions with my lab- and office-mates, both past and present, ranged from technical to ridiculous, but always provided new insight. I would especially like to thank Ariel Anders, Patrick Barragan, Sam Davies, Jared Glover, Martin Levihn, Owen MacIndoe, Alejandro Perez, and Lawson Wong who shared office space with me and commented only rarely on my propensity to talk to myself. Finale Doshi, George Konidaris, and Stefanie Tellex answered my endless questions about the process of obtaining a Ph.D. Huan Liu and Garratt Gallagher braved my impressive record of breaking hardware by introducing me to the PR2. Dylan Hadfield-Menell went above and beyond the call of a labmate's duty in reading and correcting the proof in this thesis. Annie Holladay worked with me as a UROP for over two years and doubled my productivity. She even put up with having MIT news record her experiments.

My family and friends always knew when to push me to leave the office and when to leave me alone and let me work. Many of them, including my awesome in-laws Doug, Karen, and Em German, even got up at the crack of dawn to watch the broadcast of my thesis defense. Michelle Tomasik dragged me to the gym at least twice a week for five years. Lily Cohen and Marci Foster-Molina were always willing to listen and never more than a phone call or a plane, train, or automobile ride away.

My brother Andy has been a graduate student in CSAIL with me for the past three years. He was always willing to take a break to go running or grab some candy and listen to whatever was on my mind. The foundation for Chapter 3 of this thesis was laid during a three-and-a-half mile run with him by the Charles River. He also entirely changed my thesis and very likely my career by yelling at me that if I thought robots were so cool I should just take the PR2 class already. The thing I will miss most about my time here is working only a floor away from my little brother.

My mother Sue and father Dan have always not just encouraged me to follow my dreams, but provided living examples of how to do that. From math puzzles in the car and books read aloud to dishes of swamp water and broken Mettler balances (sorry about that, Mom), they spent my childhood showing me that learning is fun. If I began to forget that while working on my thesis, my dad would find some crazy new robot for me to program. My mom has been a source of inspiration, constantly reminding me that we must examine every dimension of a problem.

My husband David German was my port in a storm. He soothed me through bad times and laughed with me through good ones. He complained remarkably little about how infrequently I was home during the last two years, but just brought me dinner at work and gently reminded me that sleep and food are important to productivity. Many times I have come back to my desk after a meeting and found a vase full of flowers, and I have rarely wanted for hot chocolate. I cannot imagine the last six years without him.

Lastly, I would like to thank Juan and Cindy for just being there.

Contents

1 Introduction **13**
- 1.1 Problem Overview 13
- 1.2 Approach 14
- 1.3 Thesis Organization 16

2 Background **17**
- 2.1 Configuration Space and Sampling-Based Search 17
- 2.2 Rapidly-exploring Random Tree Algorithm 20
  - 2.2.1 General Conditions for Exponential Convergence 22
  - 2.2.2 Exponential Convergence in Holonomic Spaces 25
  - 2.2.3 The RRT Algorithm in Non-Holonomic Spaces 36
  - 2.2.4 RRTConnect Algorithm 37
- 2.3 Related Work 40
  - 2.3.1 Non-Prehensile Manipulation 40
  - 2.3.2 Re-Grasping 41
  - 2.3.3 Navigation Among Movable Obstacles 43
  - 2.3.4 Sampling and Constrained Motion Planning 44
  - 2.3.5 Multi-Modal Planning 46

3 Sampling-Based Algorithms for Diverse Action Manipulation **49**
- 3.1 Diverse Action Manipulation Problem 49
- 3.2 Diverse Action Rapidly-exploring Random Tree Algorithm 54
  - 3.2.1 Motivating Example 54
  - 3.2.2 Overview 59
  - 3.2.3 Distance Function 59
  - 3.2.4 Empty Space Planner 61
  - 3.2.5 Projection Functions 64
- 3.3 DARRTConnect Algorithm 66
  - 3.3.1 Motivation 66
  - 3.3.2 Algorithm 67

4 A Hierarchical Approach to Diverse Action Manipulation **71**
- 4.1 Manipulation as Multi-Modal Planning 71
  - 4.1.1 MM-DAMA Problem 72
  - 4.1.2 Explicit Multi-Modal Planning 73

4.2 DARRTH Algorithm 75
4.2.1 Finding an Object Path 77
4.2.2 Manipulation Primitive Subgoals 78
4.2.3 DARRTH(CONNECT) Algorithm 79
4.3 DARRT as a Multi-Modal Planner 80

5 Diverse Action Manipulation Experiments 81
5.1 Plate Domain 81
5.1.1 Implementation Details 81
5.1.2 Results 87
5.2 Tool Use Domain 89
5.2.1 Implementation Details 90
5.2.2 Results 98
5.3 Discussion 100
5.3.1 Problem Difficulty 100
5.3.2 Forward vs Bi-Directional Planners 100
5.3.3 Flat vs Hierarchical Planners 101
5.3.4 Reset Times 103

6 Exponential Convergence of the Search Algorithms 105
6.1 Exponential Convergence of the DARRT Algorithm 105
6.1.1 DARRT Input 106
6.1.2 DARRT Analysis 109
6.2 Examples 117
6.2.1 Preliminaries: Notation and Cross Product Spaces 117
6.2.2 Point Rigid Transfer 119
6.2.3 Disc Pushing 132
6.2.4 In Defense of Projection Functions 157
6.3 Exponential Convergence of DARRTH(CONNECT) 157

7 Conclusion 161
7.1 Summary of Contributions 161
7.2 Future Work 162

A Proofs 167

B Tables 173
B.1 Plate Domain 174
B.2 Tool Use Domain 186

List of Figures

2.1 A round robot translating in the plane has a two dimensional configuration space . . . 18
2.2 A robot arm with seven degrees of freedom . . . 19
2.3 The steps of the RRT algorithm. . . . 21
2.4 An example of a situation in which the RRT algorithm is not complete 23
2.5 An illustration of the argument for exponential convergence . . . 24
2.6 A holonomic point robot navigating in a two dimensional plane . . . 26
2.7 The free space must be open for completeness of the RRT algorithm . 27
2.8 Radius of locality . . . 29
2.9 The necessity of a local path . . . 30
2.10 A local path . . . 31
2.11 An empty space planner for a car . . . 36
2.12 RRTConnect is more efficient than RRT . . . 39
2.13 A one-dimensional pushing world . . . 43
2.14 Round robot pushing a round object . . . 45
2.15 An example of a problem with continuous modes . . . 47

3.1 An example world in which a robot manipulates a plate. . . . 51
3.2 Holonomic extension fails in manipulation domains . . . 55
3.3 Empty space planner example . . . 56
3.4 Failure of the RRT for manipulation . . . 57
3.5 Projection functions . . . 58
3.6 DARRT distance function . . . 61
3.7 Primitives each define a set of constraints . . . 64
3.8 Narrow passages in manipulation . . . 67

4.1 We assume the plate can only be grasped at a single position on the table. . . . 73
4.2 The mode adjacency graph in the Plate World. . . . 74

5.1 Worlds 0-2 in the Plate Domain . . . 82
5.2 Worlds 3-4 in the Plate Domain . . . 83
5.3 An example execution in the Plate Domain . . . 88
5.4 Using the spatula to lift the CD in the Tool Use Domain . . . 90
5.5 Worlds 0-1 in the Tool Use Domain . . . 91
5.6 Worlds 2-3 in the Tool Use Domain . . . 92

5.7 The `use-spatula` primitive 96
5.8 An example execution in the Tool Use Domain 99

6.1 Tubes should be cross products of subspace tubes, not a union of open balls 111
6.2 The empty space planner for a point robot and object 120
6.3 The local path for an object when only the robot is moving 123
6.4 The empty space planner for a robot pushing a disc 133
6.5 The local path for a point robot pushing a disc 139
6.6 Projection functions are necessary 158

7.1 A poor choice for an empty space planner that avoids contact with the object 164

List of Tables

5.1 Results in the Plate Domain 87
5.2 Subgoal times in the Plate Domain 89
5.3 Results in the Tool Use Domain 98
5.4 Subgoal times in the Tool Use Domain 98

B.1 Full results for DARRT and DARRTConnect in Plate Domain Worlds 0-2. 174
B.2 Full results for DARRT and DARRTConnect in Plate Domain Worlds 3-4. 175
B.3 Full results for DARRTH in Plate Domain World 0 176
B.4 Full results for DARRTHConnect in Plate Domain World 0 177
B.5 Full results for DARRTH in Plate Domain World 1 178
B.6 Full results for DARRTHConnect in Plate Domain World 1 179
B.7 Full results for DARRTH in Plate Domain World 2 180
B.8 Full results for DARRTHConnect in Plate Domain World 2 181
B.9 Full results for DARRTH in Plate Domain World 3 182
B.10 Full results for DARRTHConnect in Plate Domain World 3 183
B.11 Full results for DARRTH in Plate Domain World 4 184
B.12 Full results for DARRTHConnect in Plate Domain World 4 185
B.13 Full results for DARRT and DARRTConnect in Tool Use Domain Worlds 0-1. 186
B.14 Full results for DARRT and DARRTConnect in Tool Use Domain Worlds 2-3. 187
B.15 Full results for DARRTH in Tool Use Domain World 0 188
B.16 Full results for DARRTHConnect in Tool Use Domain World 0 . . 189
B.17 Full results for DARRTH in Tool Use Domain World 1 190
B.18 Full results for DARRTHConnect in Tool Use Domain World 1 . . 191
B.19 Full results for DARRTH in Tool Use Domain World 2 192
B.20 Full results for DARRTHConnect in Tool Use Domain World 2 . . 193
B.21 Full results for DARRTH in Tool Use Domain World 3 194
B.22 Full results for DARRTHConnect in Tool Use Domain World 3 . . 195

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Consider a personal robot performing chores in a home. This robot will manipulate many different types of objects in many different ways. To put a book away in a bookcase, it must consider the placement of the book when planning to pick it up. A bad choice of grasp could prevent the robot from being able to slide the book onto the shelf. To clear a table, the robot needs to manipulate plates and platters that are too flat for grippers to slide under. If the robot cannot reach a toy under a couch, it could find a yardstick to use as a tool to sweep the toy to a position where it can be grasped.

These are all examples of manipulation with diverse actions. The robot has many actions available to it (grasp, push, place, sweep, etc) and it must sequence them to accomplish a complicated task. The different actions all constrain the robot differently, but later actions in the sequence may depend upon earlier actions. For instance, in order to grasp a plate, the robot may first need to slide it to the edge of the table.

As humans, we use many different types of manipulation to accomplish daily tasks. We routinely slide flat books on tables, sweep papers into piles, and use common objects as tools. Robots, with much less dexterous hands than humans, should use these strategies even more often. However, these actions are also fraught with challenges in control, perception, and planning. Here, we focus on planning; we outline methods for understanding which actions are appropriate in which circumstances and how to sequence them.

1.1 Problem Overview

We study manipulation problems in which we have a robot manipulating some number of objects. In this context, "manipulation" means any interaction with the object: the robot might be pushing, pulling, shoving, or throwing the object. We usually assume that the task is to use the robot to re-position some of the objects.

We are interested in problems with three characteristics:

- Diverse actions: There are multiple actions available to the robot for manipulating objects.

- Multiple actions per object: In order to accomplish the task, the robot must manipulate the same object more than once.

- Non-prehensile actions: There are some manipulation actions in which the robot is not rigidly attached to the object.

The problem of pushing a plate to the edge of a table to grasp it fulfills these characteristics. There are diverse manipulation actions (push and grasp), the robot must push and grasp the same object, and push is a non-prehensile action. We use variations of this problem as examples throughout the thesis.

Another problem of interest is tool use. A robot should be able to use objects in the environment as tools to manipulate other objects. For instance, in the introduction we talked about a robot sweeping a toy out from under a couch using a yardstick as a tool. A robot could use tweezers to pick up a small object or push two objects by using one object to push the other. Tool use is almost always non-prehensile because there are at least two objects, and the robot is rigidly attached to at most one of them.

There is a large body of work on the dynamics and control of non-prehensile actions like pushing, pulling, throwing, and striking [7, 9, 18, 26, 32, 42]. Previous work in non-prehensile manipulation tends to focus, however, on a single manipulation action rather than on finding sequences of manipulation actions. Additionally, many of these actions introduce uncertainty into the execution because the robot does not control all of the degrees of freedom of the object it is manipulating. Here, we assume that we can use the techniques discussed in prior work to treat the outcomes of actions as deterministic. Incorporating planning for uncertain actions is beyond the scope of this thesis.

The navigation among movable obstacles problem [36, 46, 47, 48, 52] and the regrasping task [30, 43] both require using multiple manipulation actions, and the focus is on properly sequencing these actions. However, these approaches both assume the manipulated object is rigidly attached to the robot and leverage that in planning. Our algorithm can plan for both prehensile and non-prehensile actions.

Other frameworks for robotic motion planning have not focused explicitly on the type of manipulation, but have taken a more general approach of finding a framework for motion planning. Berenson and Srinivasa [4, 5] developed a planner for a broad set of end-effector constraints while Hauser and Ng-Throw-Hing [14, 15] discuss planning in non-expansive spaces. Our work builds on these ideas, but can solve problems out of the scope of both of these algorithms.

We expand upon this discussion of related work in Chapter 2.

1.2 Approach

The types of problems described in Section 1.1 are challenging because they require making long plans in which early choices in the plan affect later ones. For instance, when placing a book on a shelf, the book must be grasped in such a way that the gripper does not keep the book from sitting stably. If we break the problem into sub-problems, we have to be careful to carry these constraints through all sub-problems.

Additionally, the space in which manipulation occurs is continuous and high-dimensional. For instance, in our experiments, we plan for a robot with a seven degree of freedom arm mounted on a three degree of freedom base. Objects are usually rigid with six degrees of freedom. This gives us a continuous search space with at least sixteen dimensions.

Moreover, manipulation usually requires specific relative configurations of the robot and object. For instance, the robot may need to be grasping or pushing an object. These configurations occur in low-dimensional subspaces of the full sixteen dimensional search space. Thus we must search a very large space for a plan while also ensuring that we find configurations in these small, constrained subspaces.

We approach the problem using sampling-based search. This type of search starts from a given initial position or "configuration" of the robot and objects and samples a new configuration for the robot and objects randomly from the space of all possible configurations. It then tries to connect the initial configuration to the new configuration, truncating the path at the first collision with an obstacle. The algorithm repeats these steps until a path to the goal is found.

When connecting one configuration to another, we must be careful to use a path that the robot can actually execute. In some mobile manipulation domains, this path can be a straight line. For instance, a disc-shaped robot translating in the two-dimensional plane can move along any line in the plane. In our case, however, we have objects in the domain. A "straight line" path in these domains would require that the objects move on their own, which is not possible. To plan paths in these domains, we use an "empty space planner". This planner plans a path from one configuration of the robot and objects to another that is executable by the robot. For instance, to move the robot and a graspable object, it would plan a path to move the robot to the object, pick the object up, move the object to its new position, place it there, and then finally move the robot to the robot's new position. This planner operates in "empty space" because it ignores any collisions with obstacles.

In Chapter 2, we give an overview of sampling-based planning and in Chapter 3 we discuss two sampling-based algorithms for manipulation. The first, based on the Rapidly-exploring Random Tree (RRT) algorithm [28], starts from an initial configuration and extends in random directions using the empty space planner until it reaches a goal configuration. The second, based on the RRTCONNECT algorithm [23], runs two searches, one forwards from the initial configuration and one backwards from the goal configurations and attempts to meet them in the middle.

These sampling-based searches are "flat" in that they search in the entire space without attempting to break it into smaller spaces. However, spaces for manipulation have a structure that can be used to approach the problem hierarchically. For instance, in most manipulation problems the first step is to move the robot somewhere near the object it is manipulating. Ideally, we would define this as a sub-problem and solve it separately from planning, for instance, how to push the object. We show that the specific structure of the manipulation space is *multi-modal* [14]. A multi-modal problem has a set of (usually) low-dimensional subspaces or "modes" amongst which the system must transition. In our case, the modes correspond to different ways in which the robot can manipulate the object. For instance, the robot pushing an object is one mode while the robot grasping the object is a different mode. We use this structure to define a good set of sub-problems for a manipulation problem and then solve each sub-problem separately using the flat planners.

1.3 Thesis Organization

In the next chapter, we give a more thorough overview of the work related to this thesis. We also present sampling-based planning in detail and discuss the algorithms most related to our work.

In Chapter 3, we formally define a manipulation problem and present the Diverse Action Rapidly-exploring Random Tree (DARRT) and DARRTConnect sampling-based algorithms for solving these problems.

In Chapter 4, we show how to take advantage of the structure of manipulation problems to create hierarchical algorithms, DARRTH and DARRTHConnect, that use the sampling-based planners as subroutines. In this chapter, we also present a characterization of general manipulation problems as multi-modal problems.

In Chapter 5, we give results for DARRT, DARRTConnect, DARRTH, and DARRTHConnect on a set of problems in two complicated mobile manipulation domains. We show that DARRTConnect is usually much more efficient than DARRT and that the hierarchical planners usually perform better than their flat counterparts.

In Chapter 6, we present the theoretical results for the sampling-based and hierarchical planners. We show that under certain assumptions, both planners should converge quickly to a solution.

This thesis makes three contributions:

1. We formally define the diverse action manipulation problem (Chapter 3) and show that all diverse action manipulation problems have a multi-modal structure (Chapter 4).

2. We present two sampling-based algorithms (Chapter 3) and two hierarchical algorithms (Chapter 4) for the diverse action manipulation problem, and show that they are able to plan in a variety of problems in two mobile manipulation domains (Chapter 5).

3. We prove the exponential convergence of DARRT and DARRTH(Connect) under a set of carefully stated assumptions and show there are at least two manipulation domains that fulfill the assumptions for DARRT (Chapter 6).

Chapter 2

Background

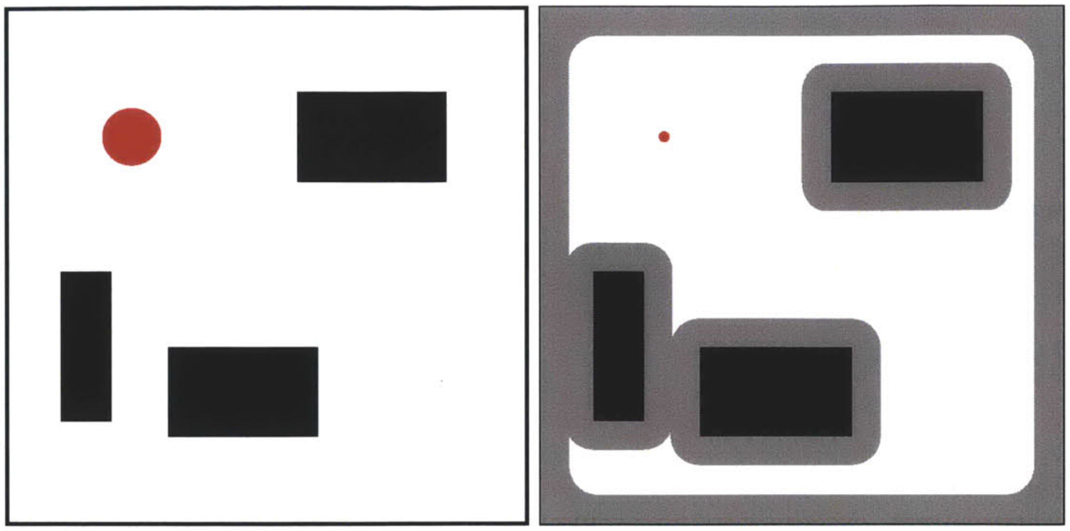
In this chapter, we discuss the previous work upon which this thesis builds. We begin by describing the space we search and then discussing common methods used to search this space. We close with a review of the literature.

2.1 Configuration Space and Sampling-Based Search

In this thesis, we plan for robots manipulating objects. Planning a path for a robot is usually done in the robot's *configuration space* [29]. The configuration space is the coordinate space in which assigning a value to every coordinate determines the location of every point on the robot. This space has dimensionality equal to the number of degrees of freedom of the robot and represents the robot as a point. The *work space* is the space in which the robot and obstacles are defined. This is usually the six dimensional space of translation along three axes, roll, pitch, and yaw. Obstacles in the configuration space are configurations for the robot in which some point on the robot would be in collision with some obstacle.

Consider, for instance, the round robot translating in the two dimensional plane shown in Figure 2.1. The work space for this robot, shown in Figure 2.1a, is two dimensional because we restrict the robot to the plane. In this work space there are a number of obstacles with which the robot should not collide. Specifically, the robot's center must be at least a distance of the robot's radius from any obstacle. This gives us a set of points for the robot's center in which the robot is not in collision with an obstacle, the *free space*, and a set of points in which the robot is in collision with an obstacle, the *configuration space obstacle* as shown in Figure 2.1b. We can plan a path for the robot by planning a path for just its center point such that the center point is never inside the configuration space obstacle. In other words, specifying only the coordinates of the robot's center point allows us to determine whether any point on the robot is in collision. Therefore, the configuration space for the round robot is also two dimensional as shown in Figure 2.1b.

Planning for the robot's center is an example of planning in configuration space. In the case of the round robot translating in the plane, the configuration space is two dimensional because the robot has two degrees of freedom: it can translate in



(a) (b)

Figure 2.1: A round robot translating in the plane has a two dimensional configuration space. (a) The work space of the robot. The robot is the round red disc while the obstacles are shown as filled black rectangles. (b) The configuration space of the robot. In this space, the robot is represented as a point (red) and the obstacles have all been "grown" by the radius of the robot. The configuration space obstacle is shown in gray superimposed on the work space obstacles.

the x direction or it can translate in the y direction. Therefore, in order to specify the robot's configuration, we have to specify two numbers: the coordinate of its center point in x and the coordinate of its center point in y. In general, however, the configuration space is of higher dimensionality than the work space. Assume we make the robot triangular. Now it can translate in the plane or rotate. To specify its configuration we need both its x and y coordinates and its angle. Therefore, the configuration space is three dimensional. The position of every point on the robot depends on its orientation as well as its position.

Now consider a robot arm with seven joints like the one shown in Figure 2.2. To specify the position for all points on this arm, we have to specify an angle for every joint. This configuration space is seven dimensional and transforming an obstacle from the work space into the configuration space is hard. If we mount the arm on a mobile base, the configuration space becomes ten dimensional: we must specify an angle for every arm joint and an x and y coordinate and an angle for the base. If there are obstacles, we must be able to decide for every ten dimensional point whether the robot is in collision with an obstacle or not. In our case, we plan not only for a robot arm on a mobile base, but also for some objects in the environment. These objects are rigid bodies so they each have six degrees of freedom (x, y, z, roll, pitch, yaw). This gives us configuration spaces with upwards of sixteen dimensions. Converting a work space description of a set of obstacles into the configuration space obstacle is not, in general, an easy problem and becomes harder as the dimensionality of the configuration space increases. We omit the discussion here and refer the reader to

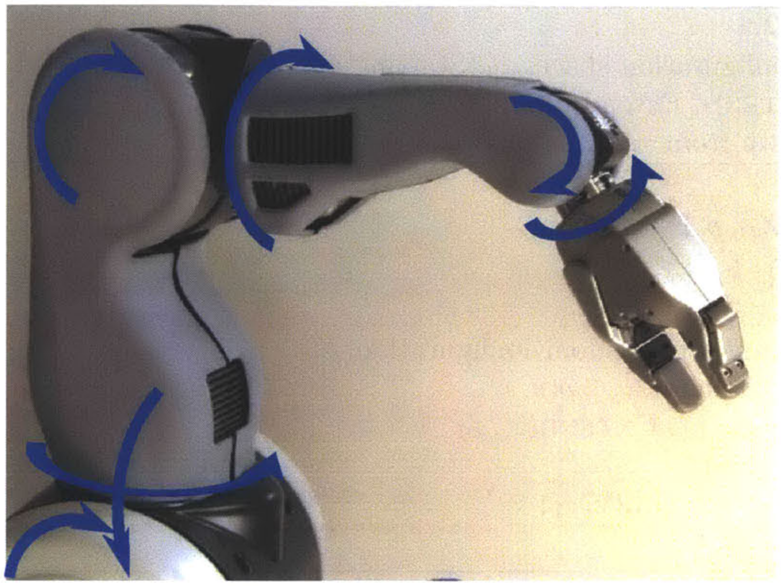


Figure 2.2: A robot arm with seven joints has seven degrees of freedom (wrist rotate, wrist lift, elbow rotate, elbow lift, shoulder rotate, shoulder lift, shoulder pan) and a seven dimensional configuration space.

LaValle [27].

The goal is to find collision free paths for the robot and objects. We do this by searching in their configuration space, which is both continuous and high-dimensional. It is difficult to plan in such spaces because it is hard to discretize them. The high dimensionality of the space means the discretization is either extremely coarse or extremely large. Therefore, we cannot use classic deterministic search techniques like A* that require an a priori discretization of the space.

A common strategy is to adaptively discretize the space as we search. Sampling-based search algorithms are one method for this. These are non-deterministic search algorithms that try to explore the connectivity properties of the space. There are many such algorithms [17, 20, 22, 23, 27, 28] and we discuss one in more detail in Section 2.2, but in general they build a structure of configuration space points connected by lines. At each iteration, they sample a new point from the space and attempt to connect it to the current structure with a collision free path. The method used for sampling and for connecting a point to the structure is specific to each algorithm.

These methods have the advantage that it is not necessary to create a full description of a configuration space obstacle, but just a method for testing whether paths are collision free. Additionally, they require no a priori discretization of the space. However, these algorithms must also carefully balance exploration of the space with exploitation of their current structure. For instance, given a point in free space, the algorithm could try to grow paths from that point as far as possible until they hit an obstacle. This is an "exploitation" of the structure because the algorithm is using its current knowledge about the space to find a path. Once a path does hit an obstacle, however, the algorithm must choose how to move around this obstacle. This cannot

Algorithm 2.1
Input: X, Configuration space; x_0, Starting configuration; X_G, Goal set; ρ, Distance function; EXTEND, Extend function
Output: A path from x_0 into X_G.

```
RRT($X, x_0, X_G, \rho$, EXTEND)
1  $V_0 \leftarrow \{x_0\}$, $k \leftarrow 1$
2  while $V_{k-1} \cap X_G = \emptyset$
3      $x \leftarrow$ uniformRandomConfiguration($X$)
4      $x' \leftarrow \arg\min_{v \in V_{k-1}} \rho(v, x)$
5      $V_k \leftarrow V_{k-1} \cup$ EXTEND($x', x$)
6      $k \leftarrow k + 1$
7  return ExtractPath($V_{k-1}$)
```

be done without a method for exploring the space. If the algorithm spends too much time on exploitation, it may be unable to move around complicated obstacles. If it spends too much time on exploration, it may never find paths to points that are far from its initial configuration.

The algorithms we discuss in this thesis are based on a particular sampling-based algorithm, the Rapidly-exploring Random Tree (RRT) algorithm. In the next section, we discuss this algorithm in detail.

2.2 Rapidly-exploring Random Tree Algorithm

The Rapidly-exploring Random Tree (RRT) algorithm [28] is one choice for the balance of exploration and exploitation in sampling-based search. The RRT algorithm explores the space by moving in a random direction at each iteration. It exploits its current structure by choosing the piece of the structure to expand greedily.

Pseudo-code for the RRT algorithm is shown in Algorithm 2.1 and an illustration of the algorithm is shown in Figure 2.3. The input is a configuration space X, a starting configuration x_0, a non-zero measure goal set X_G, a distance metric ρ, and an extend function EXTEND$(x', x) : X \times X \to 2^X$. EXTEND$(x', x)$ returns a collision free path from x' towards x. For simplicity, we treat the EXTEND function as returning a set of configurations although in practice there needs to be information about the connectivity of these configurations to later recover the path. Additionally, in most applications, the EXTEND function returns a finite set of configurations that represent a discretization of some path. However, it is also possible for the EXTEND function to return a representation of an infinite set of configurations. For instance, in Section 2.2.2, the EXTEND function returns a line segment in configuration space.

The RRT algorithm works by building a tree that explores the space. It begins with only the initial configuration in the tree (Figure 2.3a). It then samples a configuration uniformly at random from the space (Figure 2.3b) and locates the con-

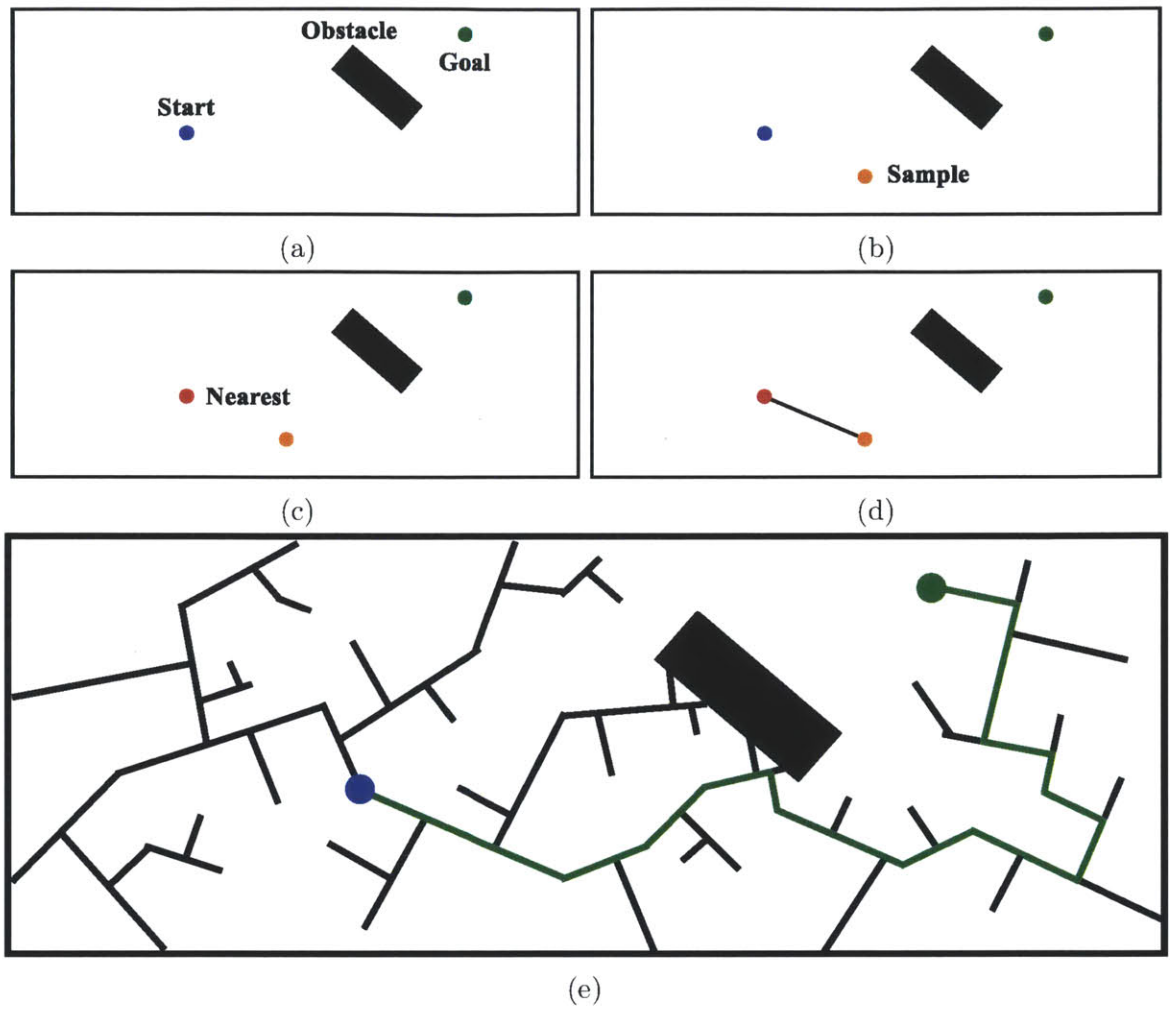


Figure 2.3: The steps of the RRT algorithm. The initial configuration is shown in blue, the goal set is shown in green, and obstacles are black rectangles. (a) The search begins with only the initial configuration (blue dot) in the tree. (b) A random configuration (orange) is sampled from the space. (c) The nearest configuration in the tree (red) to the sampled configuration (orange) is found. (d) This configuration (red) is extended to the sampled configuration (orange). The extension (solid black line) is truncated to the first obstacle added to the tree. (e) If these steps are repeated an infinite number of times, we can guarantee a path (green solid line) to the goal will be found.

figuration in the tree nearest this sample (Figure 2.3c). Note that if the sets returned from EXTEND are infinite in size, there must be an analytical method for finding the nearest configuration. For instance, in Section 2.2.2, EXTEND returns a line segment in configuration space and we use Euclidean distance as the distance metric. We can find the nearest configuration on a line segment to a given configuration analytically in Euclidean space. Once the nearest configuration in the current tree is determined, the RRT algorithm extends this configuration towards the sampled configuration using the Extend function (Figure 2.3d). If the set returned from Extend includes a configuration in the goal set, the algorithm terminates and returns the path to the goal set (Figure 2.3e).

With some restrictions on the extend and distance functions, we can guarantee that after k iterations the probability that the algorithm has not added a configuration in X_G to the tree (provided X_G has non-zero measure in X) is $O(2^{-ak})$ for positive constant a. We first provide a sketch of this argument in the general case and then give a formal proof for a holonomic configuration space.

2.2.1 General Conditions for Exponential Convergence

In this section, we give an overview of the types of extend and distance functions and configuration spaces for which the RRT algorithm is exponentially convergent. Because our algorithm is based on the RRT algorithm, understanding these properties at a high level is important.

We begin by formally defining two main properties of interest to us.

Definition 2.1 (Probabilistic Completeness): A sampling algorithm is *probabilistically complete* if the probability that the algorithm returns a solution when one exists approaches 1 as the number of samples approaches infinity.

Definition 2.2 (Exponential Convergence): A sampling algorithm *converges exponentially* if the probability that the algorithm returns a solution when one exists is $1 - O(2^{-ak})$ after k samples for some positive constant a. Exponential convergence implies probabilistic completeness.

The RRT algorithm is not necessarily either complete or exponentially convergent, because these properties depend on the configuration space and the choice of the extend and distance functions. For example, the extend function $\text{Extend}(x', x) = \{x'\}$ clearly leads to an incomplete algorithm. However, for any given configuration space, extend function, and distance function, we can decide whether the RRT algorithm is complete and exponentially convergent.

The main requirement on the configuration space is that there is no "important" measure-zero set. Because the RRT algorithm samples the space uniformly at random, there is zero probability that any configuration in a measure-zero set is sampled from the space. Therefore, the RRT algorithm is incomplete if the path from start to goal requires moving through some measure-zero set. For example, consider the situation shown in Figure 2.4, in which the space is divided by an obstacle into two halves connected by two "wormhole" configurations. If the robot is in either of these configurations, it enters the wormhole and comes out in the other half of the space. Because an obstacle divides the two halves of the space, the robot cannot reach the other half in any other way. Assume the robot starts in the bottom half of the space and that the goal set is in the top half of the space. A path from the initial configuration into the goal set exists so a solution exists. However, there is zero

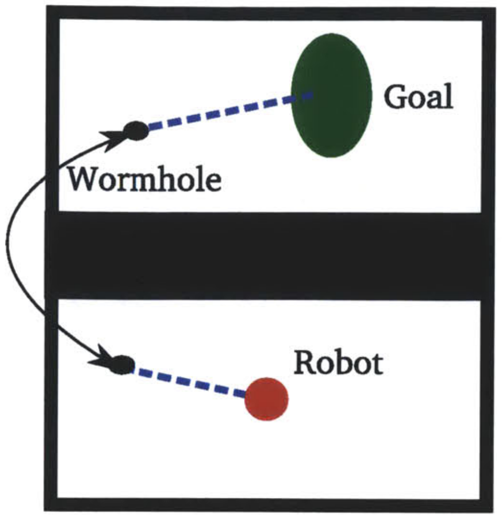


Figure 2.4: In this world, the round robot (red disc) moves in the plane. Obstacles are filled black rectangles. The plane is divided in half by an obstacle; the only way for the robot to move between the two halves of the plane is to enter the "wormhole". This wormhole can be entered at exactly two configurations in space. For instance, the robot can move from its initial configuration in the bottom half of the space to the goal region in the top half (filled green oval) using the path shown (blue dashed line). However, an algorithm that samples uniformly at random from the space never samples the wormhole (or a line that crosses the wormhole) and has zero probability of finding a path like this. The problem is that the wormhole configurations have very different reachability properties than any of the configurations near them.

probability that the RRT algorithm samples the wormhole configuration or even a line that crosses the wormhole configuration because it is a zero-measure subset of the two dimensional configuration space. Therefore, the RRT algorithm never returns a solution to the problem shown in Figure 2.4, and is not complete is this space.

To avoid situations like the one shown in Figure 2.4, we put a condition on the configuration space. The exact statement of this condition depends on the EXTEND and distance functions, but it is usually the assumption that the space is open. Here we give an intuition for why that condition is important. In Section 2.2.2, we state it formally for the case in which the configuration space is Euclidean and EXTEND returns a line segment.

We require that all configurations "act" like the configurations near them, where near is defined by the distance function ρ. Let $x \in X$ be a configuration in the space and let $B(x)$ be the set of configurations close to x. The exact definition of "act like" depends on the configuration space and the EXTEND function, but, in essence, if the robot is not in contact with any obstacles in configuration x then the robot should also not be in contact with any obstacles in any configuration in $B(x)$. Additionally, if x can be extended to x', all configurations in $B(x)$ should be able to extend to all configurations in $B(x')$.

Consider a collision free path in the space from some configuration $x_s \in X$ to

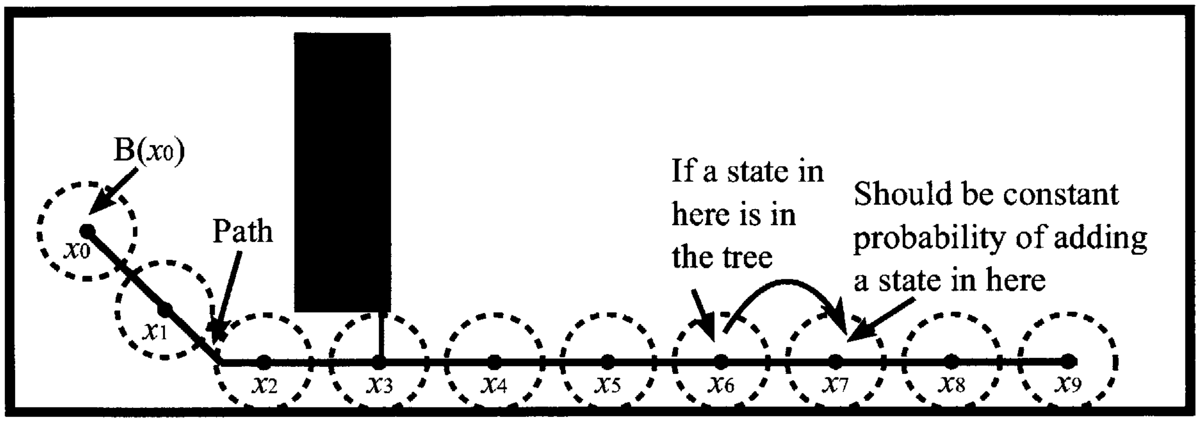

B(xo)
xo
Path
x1
x2
x3
x4
x5
x6
x7
x8
x9
If a state in here is in the tree
Should be constant probability of adding a state in here

Figure 2.5: This figure illustrates the main argument behind exponential convergence. The path shown (thick black line) is a path that could be taken by a point robot navigating in a two dimensional plane. Obstacles are shown as black rectangles. The RRT algorithm converges exponentially if we can discretize the path as shown into a sequence of configurations $\{x_0, ..., x_9\}$ such that, if we have a configuration in the tree in the open ball around x_j, we have a constant probability (with respect to iteration number) of adding a configuration in the open ball around x_{j+1}.

some configuration $x_g \in X$. Since all configurations close to any configuration are interchangeable with their near neighbors, we can expand this path into a "tube" by considering the configurations close to the path. As shown in Figure 2.8, every path through this tube is a collision free path from $B(x_s)$ into $B(x_g)$. While the RRT algorithm could not have found the single path from x_s to x_g (since this path is a zero-measure set), it *can* find some path in the tube. Note that if there are infinitely narrow passages in the space that prevent the expansion of every path in the space into a tube, as shown in Figures 2.4 and 2.7, then the RRT algorithm is not complete in this space.

The condition that all configurations act like their near neighbors is sufficient to guarantee that there is some non-zero probability of sampling a set of configurations that could be connected to form a path from a starting configuration to a configuration in the goal set. More specifically, let there be a collision free path from $x_s \in X$ to $x_g \in X$. Then for some sequence of sets $\{B_0, ..., B_m\}$ with non-zero measures in X, and for all $i \in \{0, ..., m\}$, all configurations in B_{i-1} can be directly extended to all configurations in B_i. Now if the algorithm samples a configuration in each B_i in order and connects it to the configuration sampled in B_{i-1}, it finds a path into B_m, which is the set of configurations near x_g. This is shown in Figure 2.5. The probability of sampling from each B_i in order is greater than zero.

However, while we can guarantee the algorithm can sample from each B_i in order, there is no guarantee that the algorithm tries to extend the configuration sampled in B_{i-1} to the configuration sampled in B_i. When a configuration x is sampled from B_i, the algorithm chooses to extend the configuration x' in the tree nearest to x. This configuration may not be in B_{i-1} as shown in Figure 2.9. Thus we need another condition that ensures that the greediness of the RRT algorithm does not preclude exponential convergence: If a path exists from a configuration x_s to a configuration x_g, then there must be a way to discretize this path $\{x_0, ..., x_n\}$ such that if any configuration near x_j is in the tree and we sample a configuration near x_{j+1}, then a configuration near x_{j+1} is added to the tree. This is shown in Figure 2.5.

For exponential convergence, we require that the definition of "near" does not depend on the size of the tree. In other words, if there is a configuration near x_j in the tree then there is a non-zero probability that we add a configuration near x_{j+1} to the tree on the next iteration and this probability is *independent of iteration.* The result is that at every iteration there is some constant chance that the tree progresses along the path. The laws of probability give us that, as the number of iterations increases, the probability that the tree has not progressed along the path must decrease exponentially. We make this argument formally in Theorem 2.7.

In the next section we show how to prove exponential convergence for a specific choice of configuration space, EXTEND function, and distance function.

2.2.2 Exponential Convergence in Holonomic Spaces

To ground the arguments we made in Section 2.2.1, we formally prove exponential convergence of the RRT algorithm for a holonomic robot. For our purposes, a robot is holonomic if it can execute any straight line path in its configuration space. For instance, a round robot translating in the plane is holonomic because it can move along any straight line path. A car, however, is non-holonomic because it cannot move sideways.

Previously, Kuffner and LaValle [23] proved probabilistic completeness for the RRT algorithm in holonomic spaces and Kavraki et al. [21] showed exponential convergence for the related probabilistic roadmap algorithm. LaValle and Kuffner [28] gave general conditions on exponential for the RRT algorithm. Here, we show that the RRT algorithm in holonomic spaces fulfills these conditions and then use LaValle and Kuffner's proof to complete the proof of exponential convergence. We structure the proof given here to be analogous to the proof for exponential convergence for the DARRT algorithm that we give in Chapter 6. Throughout this section, we use a point robot navigating in the plane as an example of a holonomic robot. This is shown in Figure 2.6.

The proof we give in this section may also be useful in that it identifies the constants that are important in the convergence rate of the algorithm.

We assume we are given the configuration space X and some subset $X_{free} \subseteq X$ that represents the free space. X_{free} is defined implicitly by a function, COLLISION : $X \rightarrow \{\textbf{True}, \textbf{False}\}$, that tests if a configuration is in X_{free}. A path is *collision free* if and only if it is a continuous piecewise linear path contained entirely in X_{free}. We use the RRT algorithm to find collision free paths.

In holonomic spaces, two configurations are close to each other if the Euclidean distance between them is small so we use Euclidean distance as our metric. (In this case, this metric is also the distance the robot must travel between two configurations in the absence of obstacles, but that is not the reason it is a reasonable choice for a distance metric in the space.)

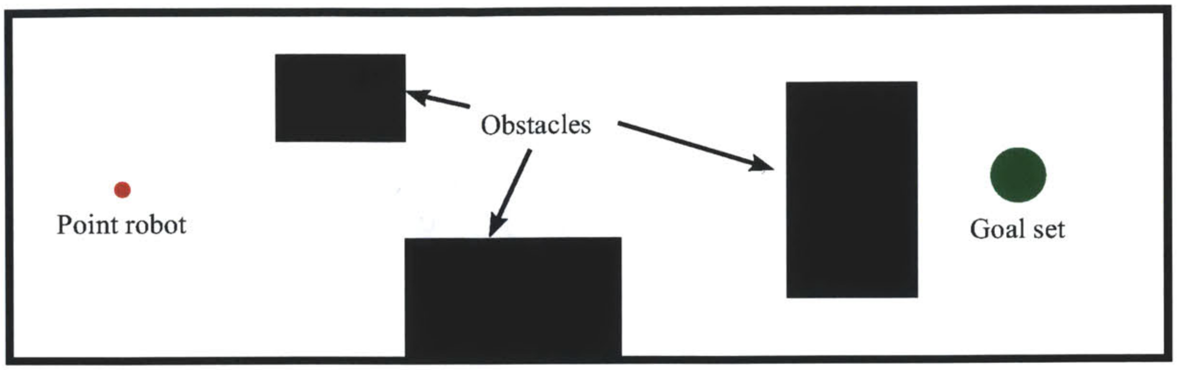


Figure 2.6: A point robot navigating in a two dimensional plane. The robot point is shown in red, its goal set in green, and obstacles as filled black rectangles. X_{free} is the set of points not contained in an obstacle or on the boundary of an obstacle (X_{free} is an open set). The robot can move instantaneously in any direction.

Algorithm 2.2
Input: x', Configuration to extend from; x, Configuration to extend towards, COLLISION: Function to check a configuration for collision
Output: The collision free part of the line segment from x' to x.

```
EXTENDHOLONOMIC($x', x$)
1  for $\alpha \in [0, 1]$ // Often discretized in practice
2       if COLLISION($x' + \alpha(x - x')$)
             // Line segment from $x'$ to $\alpha(x - x')$ including $x'$ but not $\alpha(x - x')$
3            return Line$[x', \alpha(x - x'))$
4  return Line$[x', x]$ // Line segment from $x'$ to $x$ including both endpoints
```

Given two configurations, $x', x \in X$, we extend x' towards x using a straight line path from x' to x. For notational simplicity, we treat x' and x as vectors, writing the point at fraction α along the line from x' to x as $x' + \alpha(x - x')$. This straight line path may not be collision free. The return from the EXTEND function is the straight line path truncated to the first collision. This is shown in Algorithm 2.2.

For $x', x \in X$, we let

$$\pi(x', x) = \bigcup_{\alpha=0}^{1} (x' + \alpha(x - x')) \tag{2.1}$$

be the set of configurations on the line segment from x' to x. The set of configurations a given configuration can reach with a straight line is the *locally reachable* set:

Definition 2.3 (Locally Reachable): For $x' \in X$, the *locally reachable* set of

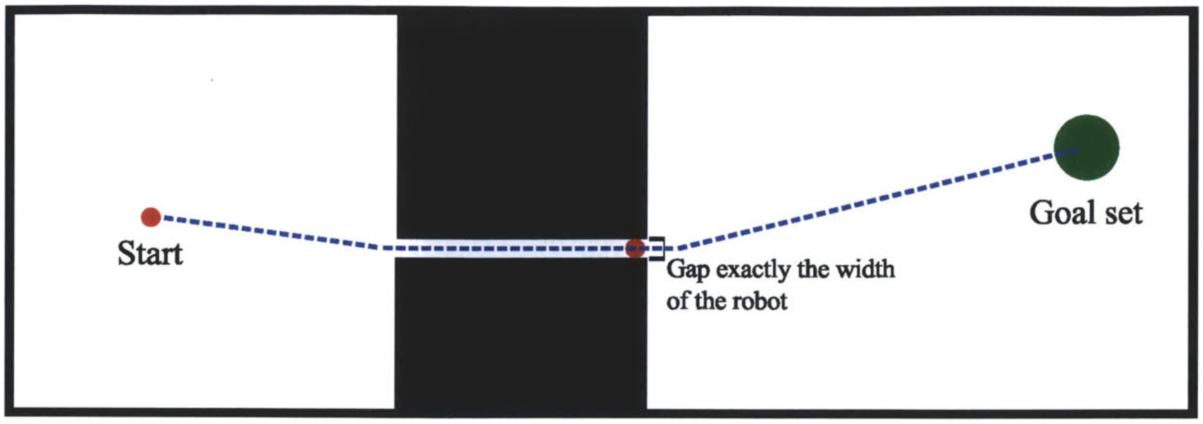


Figure 2.7: The gap between the two obstacles is exactly the width of the robot. If we allow the robot to contact, but not overlap, the obstacles, there is a path (blue dashed line) through free space from the starting configuration (red) to the goal configuration (green). However, this path must include the line segment exactly in the middle of the two obstacles. We have zero probability of sampling along this line. Therefore the RRT algorithm cannot find a path and is incomplete in this space. The problem is that the free space is not open because it includes points on the boundaries of the obstacles. For a configuration x in free space in the gap, there is no $\delta > 0$ with $B_\delta(x)$ in the free space.

configurations from x',

$$U(x') = \{x \in X \big| \pi(x', x) \subseteq X_{free}\}, \tag{2.2}$$

is the set of configurations for which the straight line path from x' is collision free. Note that $x \in \text{EXTENDHOLONOMIC}(x', x)$ if and only if $x \in U(x')$.

Recall from Section 2.2.1 that we consider nearby configurations interchangeable. The configurations "near" a configuration x are the ones within some small radius or "ball" of x. The open ball of radius δ around x is

$$B_\delta(x) = \{x' \in X \big| \rho(x, x') < \delta\}. \tag{2.3}$$

Smaller balls are contained in larger balls:

Lemma 2.1: For all $x \in X$, for all $\zeta > 0$, for all $\delta \leq \zeta$, $B_\delta(x) \subseteq B_\zeta(x)$.
Proof: We have that

$$B_\delta(x) = \{x' \in X \big| \rho(x, x') < \delta\} \subseteq \{x' \in X \big| \rho(x, x') < \zeta\} = B_\zeta(x). \tag{2.4}$$

■

Now we need a way of ensuring that all configurations in $B_\delta(x)$ are interchangeable with x for some δ. Certainly we must at least have that if $x \in X_{free}$ then for some $\delta > 0$, $B_\delta(x) \subseteq X_{free}$. This is not always the case. Figure 2.7 shows an example of a free space where such a δ does not exist for every configuration. In order to prove exponential convergence of the RRT algorithm in a holonomic space, we must assume that such a δ exists:

Assumption 2.1: For all $x \in X_{free}$, for some $\delta > 0$, $B_\delta(x) \subseteq X_{free}$.

This is actually the assumption that X_{free} is open. We state it in this way because we make an analogous assumption about our configuration space when proving the exponential convergence of the DARRT algorithm.

In the holonomic case, Assumption 2.1 is enough to ensure exponential convergence because we have exactly defined the EXTEND and distance functions. When proving the exponential convergence of the DARRT algorithm, we will need to make assumptions about these functions as well.

As we have said, considering the configurations near a particular configuration as interchangeable allows us to make a *tube* out of a single path:

Definition 2.4 (Tube): Let $x', x \in X$. The *tube of radius δ from x' to x*,

$$T_\delta(x', x) = \bigcup_{y \in \pi(x', x)} B_\delta(y), \tag{2.5}$$

is the set of configurations closer than δ to some configuration on the line from x' to x.

Just as larger balls contain smaller balls, larger tubes contain smaller tubes.

Lemma 2.2: For all $x', x \in X$, for all $\zeta > 0$, for all $\delta \leq \zeta$, $T_\delta(x', x) \subseteq T_\zeta(x', x)$.
Proof:

$$T_\delta(x', x) = \bigcup_{y \in \pi(x', x)} B_\delta(y) \subseteq \bigcup_{y \in \pi(x', x)} B_\zeta(y) = T_\zeta(x', x) \tag{2.6}$$

using Lemma 2.1.

■

Now, because X_{free} is open, we can expand any line in X_{free} into a tube. We define the maximum radius of this tube as the radius of locality. This is shown in Figure 2.8.

Definition 2.5 (Radius of Locality): For all $x' \in X_{free}$, for all $x \in U(x')$, $\eta(x', x)$ is the *radius of locality* from x' to x if and only if $T_{\eta(x',x)}(x', x) \subseteq X_{free}$ and $T_\zeta(x', x) \not\subseteq X_{free}$ for all $\zeta > \eta(x', x)$.

Since we can create an open ball around every configuration that lies entirely in free space, we have $\eta(x', x) > 0$ for all $x' \in X_{free}$ and $x \in U(x)$:

Lemma 2.3: For all $x' \in X_{free}$ and all $x \in U(x')$, $\eta(x', x) > 0$.
Proof: By definition of $U(x')$, $\pi(x', x) \subseteq X_{free}$. For all $y \in \pi(x', x)$ let δ_y be the

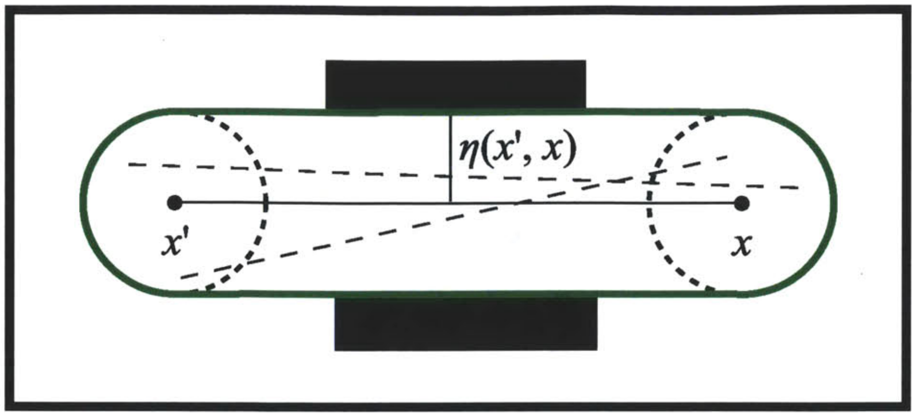


Figure 2.8: Assume we have a point robot navigating in a two dimensional plane. Obstacles are shown as filled black rectangles. If the robot starts at x', it can reach x with a single collision free line segment. The *radius of locality*, $\eta(x', x)$, between x' and x is the shortest distance from that line segment to any obstacle. Because we can draw an open ball around any configuration and guarantee that all configurations within this ball are in free space, $\eta(x', x) > 0$. This allows us to expand the line segment into a convex tube (green solid line). All configurations in this tube are collision free so any path through the tube is collision free. Specifically, if we draw open balls (short dashed) around x' and x of radius $\eta(x', x)$ then a straight line (long dashed) from any configuration in the ball around x' to any configuration in the ball around x is collision free. Therefore any configuration in the ball around x' can reach any configuration in the ball around x with a single straight line segment. This allows us to treat the configurations in the ball of radius $\eta(x', x)$ around x' as interchangeable with x'.

largest number such that $B_{\delta_y}(y) \subseteq X_{free}$. By Assumption 2.1, $\delta_y > 0$ for all y. Choose

$$\delta = \inf_{y \in \pi(x', x)} \delta_y > 0. \tag{2.7}$$

Then let

$$\begin{aligned} T_\delta(x', x) &= \bigcup_{y \in \pi(x', x)} B_\delta(y) && (2.8)\\ &\subseteq \bigcup_{y \in \pi(x', x)} B_{\delta_y}(y) && (2.9)\\ &\subseteq X_{free} && (2.10) \end{aligned}$$

using Lemma 2.1. Since $\eta(x', x)$ is the largest number with $T_{\eta(x', x)} \subseteq X_{free}$, we have $\eta(x', x) \geq \delta > 0$ by Lemma 2.2.

■

The eventual goal is to show that if any path from x' to x exists, for some discretization $\{x_0, ..., x_m\}$ and some δ, if a configuration in $B_\delta(x_j)$ is in the tree, we can guarantee a constant non-zero probability of adding a configuration in $B_\delta(x_{j+1})$ to

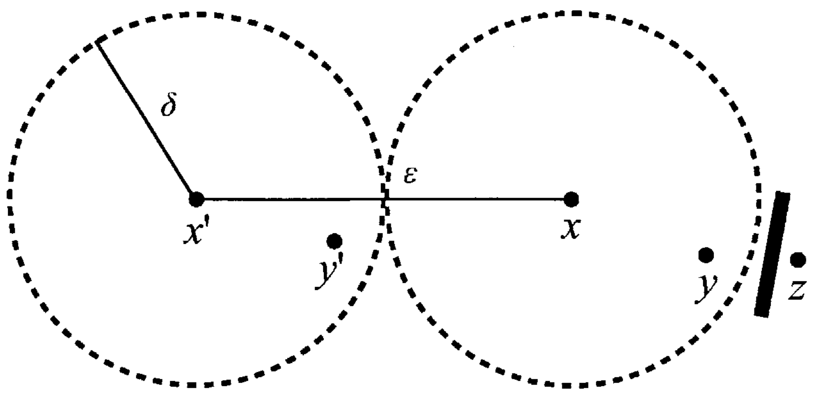


Figure 2.9: Assume we have open balls of radius δ around x' and x and that the straight line path from x' to x is collision free. We would like to be able to specify a distance $\epsilon > 0$ between x' and x and a ball radius $\delta > 0$ that ensures that if there is a configuration y' in the ball around x' in the tree and we sample any configuration y in the ball around x, y' will be the closest configuration in the tree to y. However, such an ϵ and δ cannot be chosen without reference to the current tree. As shown here, if we choose y near the perimeter of the ball around x, there is always some configuration z just outside the perimeter that is closer to y than any configuration in the ball around x'. This configuration might be in the tree and, if there is an obstacle between z and y, then z is not be able to extend all the way to y. Thus we cannot guarantee that y is added to the tree even if y' is already in the tree.

the tree at each iteration as shown in Figure 2.5. This allows us to show that the algorithm should converge exponentially in the number of iterations.

We begin by showing that if the straight line from x' to x is collision free then we can create some discretization of that line $\{x_0, ..., x_m\}$ such that if we have a configuration in $B_\delta(x_j)$ in the tree, we have a constant probability of adding a configuration in $B_\delta(x_{j+1})$. Firstly note that there is no δ that we can choose without reference to the current tree, V, that guarantees that if there is a configuration in $y' \in B_\delta(x')$, and the algorithm samples $y \in B_\delta(x)$, then $y' = \arg\min_{v \in V} \rho(v, y)$. For any $y' \in B_\delta(x')$, we can always create a configuration $z \notin B_\delta(x')$ and a subset of $S \subseteq B_\delta(x)$ such that if y is chosen from S, z is closer to y than y'. This is shown in Figure 2.9. Since $z \notin B_\delta(x')$ we may not be able to guarantee that the path from z to y is collision free. Thus despite sampling in $B_\delta(x)$ and having a configuration in $B_\delta(x')$ in the tree, we may not add a configuration in $B_\delta(x)$ to the tree.

As an aside, note that this problem arises because the RRT algorithm is greedy in its choice of nearest neighbor. If we tested every configuration in the tree for a collision free path to the sampled configuration, as in the case of a probabilistic roadmap [22], we could always guarantee that a configuration sampled from $B_\delta(x)$ would be added to any tree containing a configuration in $B_\delta(x')$.

We solve this problem by not trying to guarantee that $y' = \arg\min_{v \in V} \rho(v, y)$. Instead, we bound how far $z = \arg\min_{v \in V} \rho(v, y)$ is from the path using that y' is in V and the triangle inequality. Specifically, we discretize the straight line segment from x' to x into a sequence $\{x_0, ..., x_m\}$. We then define an *inner radius* δ and a

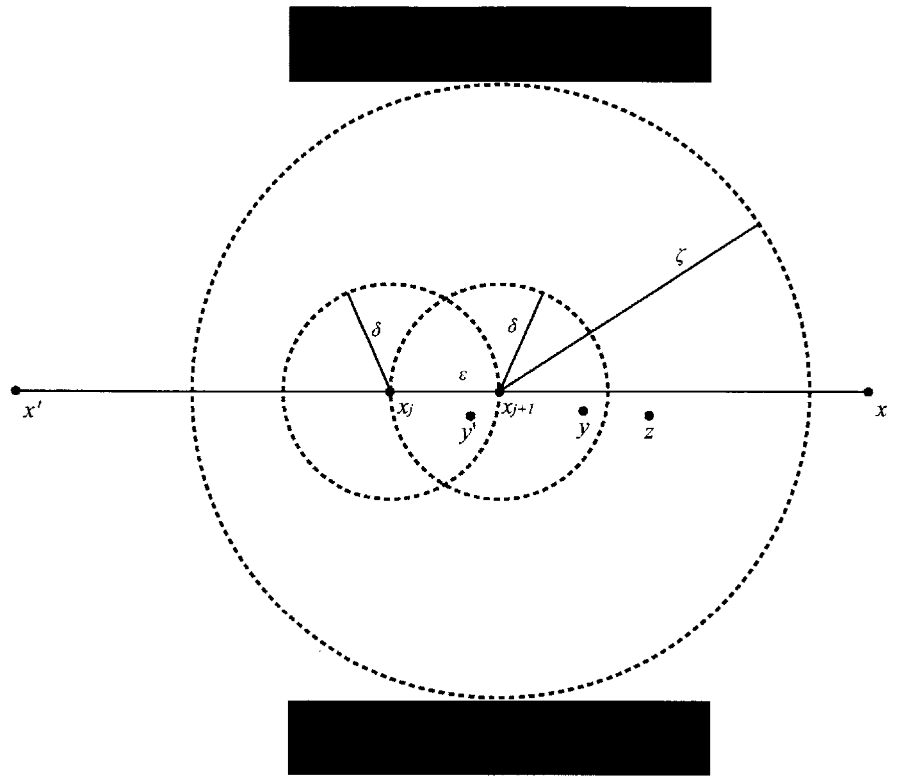


Figure 2.10: The discretization of the straight line path from x' to x. Assume we have open balls of radius δ, the *inner radius*, around x_j and x_{j+1}, and that x_j and x_{j+1} are a distance ϵ apart. Let $y' \in B_\delta(x_j)$ be in the tree and assume we sample $y \in B_\delta(x_{j+1})$. Let z be the closest configuration in the tree to y. Although we cannot guarantee that z is within δ of x_j or x_{j+1}, we *can* put an upper bound, the *outer radius* ζ, on the distance from x_{j+1} to z. With a good choice of δ and ζ, we can ensure that $z \in T_{\eta(x',x)}(x)$ so the path from z to y is collision free. Therefore, we add y to the tree although the path may go through z and not y'.

outer radius ζ as shown in Figure 2.10. Assume there is some configuration y' in the tree V in $B_\delta(x_j)$ (the open ball of inner radius δ around x_j). Now assume that we sample y from $B_\delta(x_{j+1})$ and let $z = \arg\min_{v \in V} \rho(v, y)$. As we said, we cannot guarantee that $z \in B_\delta(x_j)$. However, if the $\{x_j\}$ are close enough together, we *can* guarantee that $z \in T_\zeta(x', x)$ where ζ is the outer radius. If we choose $\zeta \leq \eta(x', x)$ then z is a collision free configuration. Not only that, but $T_\zeta(x', x)$ is convex and we have $z \in T_\zeta(x', x)$ and $y \in T_\zeta(x', x)$. Therefore, the straight line path, $\pi(z, y)$, lies within $T_\zeta(x', x) \subseteq X_{free}$. Thus, even though z may not be in $B_\delta(x_j)$, we still add y to the tree.

We now formally define this discretized path from x' to x and then prove it exists for every set of configurations.

Definition 2.6 (Local Path): For $x', x \in X$ and $\zeta > 0$, a sequence of configurations $\{x_0, ..., x_m\}$ is a *local path from x' to x of outer radius $\zeta > 0$ and inner radius $\delta \in (0, \zeta)$* if and only if $x_0 = x'$, $x_m = x$ and, for all $j < m$, for all $y' \in B_\delta(x_j)$, for all

$y \in B_\delta(x_{j+1})$, for all $z \in X$, if $\rho(z, y) \leq \rho(y', y)$ then $\pi(z, y) \subseteq T_\zeta(x', x)$.

Lemma 2.4: For all $x', x \in X$ and all $\zeta > 0$, a local path from x' to x of outer radius ζ and some associated inner radius δ exists.
Proof: Choose

$$j_{\max} = \left\lceil \frac{\rho(x', x)}{\zeta/4} \right\rceil \tag{2.11}$$

$$\delta = \frac{\rho(x', x)}{j_{\max}} \leq \frac{\zeta}{4}, \tag{2.12}$$

and choose $\{x_0, ..., x_{j_{\max}}\}$ such that for $j \in \{0, ..., j_{\max}\}$,

$$x_j = x' + \frac{j}{j_{\max}}(x - x'). \tag{2.13}$$

This is the line segment from x' to x discretized at intervals of δ so $x_0 = x'$, $x_{j_{\max}} = x$, $x_j \in \pi(x', x)$, and $\rho(x_j, x_{j+1}) = \delta$. Additionally, $B_\delta(x_j) \subseteq T_\delta(x', x) \subset T_\zeta(x', x)$ using Lemma 2.2.

For any $j < j_{\max}$, choose $y' \in B_\delta(x_j)$ and $y \in B_\delta(x_{j+1})$. Consider any z such that

$$\begin{aligned} \rho(z, y) &\leq \rho(y', y) && (2.14)\\ &\leq \rho(y', x_j) + \rho(x_j, x_{j+1}) + \rho(x_{j+1}, y) && (2.15)\\ &< \delta + \rho(x_j, x_{j+1}) + \delta && (2.16)\\ &= 3\delta. && (2.17) \end{aligned}$$

We show $z \in T_\zeta(x', x)$:

$$\begin{aligned} \rho(x_{j+1}, z) &\leq \rho(x_{j+1}, y) + \rho(y, z) && (2.18)\\ &< 4\delta && (2.19)\\ &\leq \zeta. && (2.20) \end{aligned}$$

Now $j < j_{\max}$ so $x_{j+1} \in \pi(x', x)$, giving

$$z \in B_\zeta(x_{j+1}) \subseteq T_\zeta(x', x). \tag{2.21}$$

Therefore $z \in T_\zeta(x', x)$ and $y \in B_\delta(x_{j+1}) \in T_\zeta(x', x)$. Now $T_\zeta(x', x)$ is the tube around the straight line segment from x' to x. Figure 2.8 shows that this is convex (see Lemma 6.15 for a formal proof) so all configurations on the straight line segment from z to y must also be in $T_\zeta(x', x)$. Thus $\pi(z, y) \subseteq T_\zeta(x', x)$ and $\{x_0, ..., x_{j_{\max}}\}$ is a local path from x' to x.

■

From Lemma 2.4, it is a short step to see that if x' can reach x with a collision free straight line path, then we can discretize that path into a sequence of configurations

$\{x_0, ..., x_m\}$ such that, for some $\delta > 0$, if the tree contains a configuration in $B_\delta(x_j)$, there is a constant probability of adding a configuration in $B_\delta(x_{j+1})$.

Lemma 2.5: For all $x' \in X$, for all $x \in U(x')$, for any $\zeta \in (0, \eta(x', x)]$, let $\{x_0, ..., x_m\}$ be a local path from x' to x with outer radius ζ and let δ be the associated inner radius. For all $j < m$, if V_{k-1} contains a configuration in $B_\delta(x_j)$ at iteration $k-1$, the probability of adding a configuration in $B_\delta(x_{j+1})$ to V_k at iteration k is at least $\frac{\mu(B_\delta(x_{j+1}))}{\mu(X)}$.
Proof: Assume we have $y' \in B_\delta(x_j) \cap V_{k-1}$ and we choose $y \in B_\delta(x_{j+1})$. The probability of such a choice is $\frac{\mu(B_\delta(x_{j+1}))}{\mu(X)}$. Let $z = \min_{v \in V_{k-1}} \rho(v, y)$. Then $\rho(z, y) \leq \rho(y', y)$ since $y' \in V_{k-1}$ so by definition of local path $\pi(z, y) \subseteq T_\zeta(x', x) \subseteq T_{\eta(x',x)}(x', x) \subseteq X_{free}$ using Lemma 2.2 and the definition of $\eta(x', x)$. Thus $y \in \text{EXTENDHOLONOMIC}(z, y)$ and y is added to V_k on iteration k.

■

Now let x be some configuration that x' can reach with some collision free path, but not necessarily a straight line one. We need to extend Lemma 2.5 to apply to whole paths rather than just straight lines. We begin by formally defining a path from x' to x.

Definition 2.7 (Path): For all $x', x \in X$, the sequence of configurations $\{x_0, ..., x_t\}$ is a *path from x' to x* if and only if $x_0 = x'$, $x_t = x$, and for all $j \in \{0, ..., t-1\}$, $x_{j+1} \in U(x_j)$.

Now we can formally define what is meant when we say a configuration is "reachable" from another configuration.

Definition 2.8 (Reachable): For all $x', x \in X_{free}$, x is *reachable from x'*, denoted $x \in R(x')$, if and only if there is a path from x' to x.

We showed in Lemma 2.5 that a single straight line can be discretized into a sequence of configurations $\{x_0, ..., x_t\}$ such that for some $\delta > 0$ if the tree has a configuration in $B_\delta(x_j)$, there is a constant probability of adding a configuration in $B_\delta(x_{j+1})$. We now use induction to show this for an entire path. Note that each line segment along the path requires a different outer and inner radius depending on the distance of obstacles from that segment.

Lemma 2.6: For all $x' \in X$, for all $x \in R(x')$, for any $\zeta > 0$, for some $\{\delta_0, ..., \delta_r\} \in (0, \zeta]$, and some sequence $\{x_0, ..., x_r\}$, $x_0 = x'$, $x_r = x$, and for all $j < r$, if a configuration in $B_{\delta_j}(x_j)$ is in the tree V_{k-1} at iteration $k-1$, the probability of adding a configuration in $B_{\delta_{j+1}}(x_{j+1})$ to the tree at iteration k is at least $\frac{\mu(B_{\delta_{j+1}}(x_{j+1}))}{\mu(X)}$.
Proof: Let $\{p_0, .., p_t\}$ be a path from x' to x. Some path exists because $x \in R(x')$. We proceed by induction on the length of the path.

Base Case ($t = 1$)*:* If $t = 1$, then the path is $\{p_0, p_1\}$. Let $\chi = \min(\zeta, \eta(p_0, p_1))$. Let $W_\chi(p_0, p_1) = \{x_0, ..., x_m\}$ be a local path from p_0 to p_1 of outer radius χ and let

$\delta \in (0, \chi]$ be the associated inner radius. $W_\chi(p_0, p_1)$ and δ exist by Lemma 2.4. By definition of local path $x_0 = p_0$ and $x_m = p_1$. Let $\delta_j = \delta$ for all $j \in \{0, ..., m\}$. By Lemma 2.5, for all $j < m$, if a configuration in $B_{\delta_j}(x_j)$ is in V_{k-1} at iteration $k-1$, the probability of adding a configuration in $B_{\delta_{j+1}}(x_{j+1})$ is at least $\frac{\mu(B_{\delta_{j+1}}(x_{j+1}))}{\mu(X)}$.

Induction Step: Let the path from x' to x be $\{p_0, ..., p_t\}$ of length $t+1$. The path $\{p_1, ..., p_t\}$ from p_1 to x is a path of length t. Therefore, by induction, there are some $\{\epsilon_0, ..., \epsilon_s\} > 0$, and some sequence $\{w_0, ..., w_s\}$ such that $w_0 = p_1$, $w_s = p_t$, and for all $j < s$, if a configuration in $B_{\epsilon_j}(w_j)$ is in V_{k-1} at iteration $k-1$, the probability of adding a configuration in $B_{\epsilon_{j+1}}(w_{j+1})$ to V_k at iteration k is at least $\frac{\mu(B_{\epsilon_{j+1}}(w_{j+1}))}{\mu(X)}$. Let $\chi = \min(\epsilon_0, \eta(p_0, p_1))$, let $W_\chi(p_0, p_1) = \{u_0, ..., u_m\}$ be a local path from p_0 to p_1 of outer radius χ, and let δ be the associated inner radius. $W_\chi(p_0, p_1)$ and δ exist by Lemma 2.4. Now consider the sequence of configurations

$$\{x_0, ..., x_r\} = \{u_0, ..., u_m, w_1, ..., w_s\}, \tag{2.22}$$

and let

$$\delta_j = \begin{cases} \delta & \text{if } j \leq m \\ \epsilon_{j-m} & \text{else} \end{cases} \tag{2.23}$$

We have $x_0 = p_0$ by definition of local path and $x_r = x$ by induction. If $j > m$, if a configuration in $B_{\delta_j}(x_j) = B_{\epsilon_{j-m}}(w_{j-m})$ is in V_{k-1} we add a configuration in $B_{\delta_{j+1}}(x_{j+1}) = B_{\epsilon_{j-m+1}}(w_{j-m+1})$ to V_k on iteration k with probability $\frac{\mu(B_{\delta_{j+1}}(x_{j+1}))}{\mu(X)}$ by induction. If $j = m$ then assume a configuration in $B_{\delta_m}(x_m)$ is in V_{k-1}. Now

$$B_{\delta_m}(x_m) = B_\delta(u_m) = B_\delta(p_1) \subseteq B_{\epsilon_0}(w_0) \tag{2.24}$$

because $\delta \leq \chi \leq \epsilon_0$ and $u_m = p_1 = w_0$. Therefore, by induction, we add a configuration in $B_{\epsilon_1}(w_1) = B_{\epsilon_{m+1-m}}(x_{m+1}) = B_{\delta_{m+1}}(x_{m+1})$ to V_k on iteration k with probability $\frac{\mu(B_{\delta_{m+1}}(x_{m+1}))}{\mu(X)}$. If $j < m$, if a configuration in $B_{\delta_j}(x_j) = B_\delta(u_j)$ is in V_{k-1} on iteration $k-1$ then we add a configuration in $B_\delta(u_{j+1}) = B_{\delta_{j+1}}(x_{j+1})$ to V_k on iteration k with probability $\frac{\mu(B_\delta(u_{j+1}))}{\mu(X)} = \frac{\mu(B_{\delta_{j+1}}(x_{j+1}))}{\mu(X)}$ by Lemma 2.5.

■

Since the probability that we advance along the path is independent of iteration, the probability that we have not added a configuration near the end of the path goes down exponentially. This argument is made in LaValle and Kuffner [28] Theorems 6 and 7 but we reproduce it here for convenience.

Theorem 2.7 (Exponential Convergence of the Holonomic RRT Algorithm): Let x_0 be the starting configuration. For all $x \in R(x_0)$, for all $\zeta > 0$, the probability that the tree does not include a configuration in $B_\zeta(x)$ after k iterations is $O(2^{-ak})$ for some positive constant a.

Proof: Choose some $\{\delta_0, ..., \delta_r\} \in (0, \zeta]$, and some sequence $\{x_0, ..., x_r\}$ such that $x_0 = x'$, $x_r = x$, and for all $j < r$, if a configuration in $B_{\delta_j}(x_j)$ is in the tree V_{k-1} at iteration $k-1$, the probability of adding a configuration in $B_{\delta_{j+1}}(x_{j+1})$ to the tree at iteration k is at least $\frac{\mu(B_{\delta_{j+1}}(x_{j+1}))}{\mu(X)}$. We can make such a choice by Lemma 2.6. Let

$$\lambda = \min_{j \in \{1,\ldots,r\}} \frac{\mu(B_{\delta_j}(x_j))}{\mu(X)} > 0. \tag{2.25}$$

By Lemma 2.6, if a configuration in $B_{\delta_j}(x_j)$ is in V_{k-1}, the probability that we add a configuration in $B_{\delta_{j+1}}(x_{j+1})$ to V_k is at least λ. Now $\lambda > 0$, λ does not depend on iteration, and we begin with $x_0 \in B_{\delta_0}(x_0)$ in the tree. Therefore, let us consider each iteration as a Bernoulli distribution in which λ is the probability of a successful outcome. In the worst case, we require r successful outcomes to add a configuration in $B_{\delta_r}(x_r) \subseteq B_\zeta(x)$ to the tree. Note that the success of each outcome is independent; λ is an underestimate of the probability that a sample drawn uniformly at random from the configuration space falls within some ball.

Let $C_1, \ldots, C_k$ be independent and identically distributed random variables whose common distribution is the Bernoulli distribution with parameter λ. C_j is an underestimate of the probability that we add a configuration in $B_{\delta_j}(x_j)$ to the tree given that the tree contains a configuration in $B_{\delta_{j-1}}(x_{j-1})$. The random variable $C = C_1 + \ldots + C_k$ is the number of successes after k iterations. Since each C_i has the Bernoulli distribution, C has a binomial distribution with expected value $E[C] = k\lambda$. Therefore, for any $\gamma \in (0, 1]$,

$$\Pr\left[C < (1-\gamma)k\lambda\right] < e^{-k\lambda\gamma^2/2}. \tag{2.26}$$

Since C is the number of successes after k iterations, we require $C \geq r$. Choosing $\gamma = 1 - \frac{r}{k\lambda}$, for $k > r/\lambda$, we have

$$\Pr[C < r] < \exp\left(-\frac{k\lambda}{2}\left(1 - \frac{r}{k\lambda}\right)^2\right) \tag{2.27}$$

$$= \exp\left(-\frac{k\lambda}{2}\left(1 + \left(\frac{r}{k\lambda}\right)^2 - 2\frac{r}{k\lambda}\right)\right) \tag{2.28}$$

$$= \exp\left(-\frac{k\lambda}{2} + r - \frac{r^2}{2k\lambda}\right). \tag{2.29}$$

Equation 2.29 characterizes the convergence rate of the algorithm in terms of the length of the path, represented by r, and its clearance, represented by λ. A little algebraic manipulation allows us to express it in the form of Definition 2.2:

$$\Pr[C < r] \leq \exp\left(-\frac{k\lambda}{2} + r\right) \tag{2.30}$$

$$= O\left(e^{-k\lambda/2}\right). \tag{2.31}$$

Thus the probability that a configuration in $B_\zeta(x)$ has not been added to the tree after k iterations is $O(2^{-ak})$ for some positive constant a.

■

In Chapter 6, we give a similar proof for the algorithm we present in this thesis.

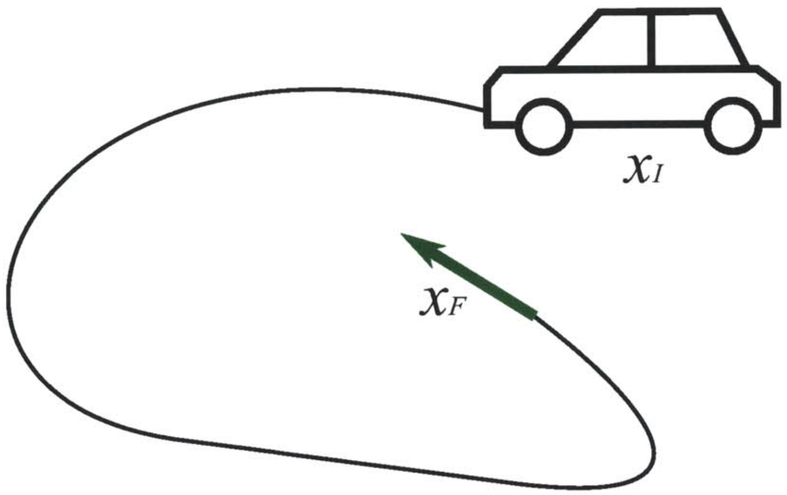


Figure 2.11: To move from the initial position shown to the final position (solid green arrow), the car cannot follow a straight line because it cannot move sideways. Instead, it must use a curved path like the one shown.

Algorithm 2.3 *Input*: x_I, configuration to extend from; x_F, configuration to extend towards; EMPTYSPACEPLANNER, A planner that returns a path from one configuration to another that is executable in the absence of obstacles
Output: A collision free path from x_I towards x_F

```
EXTEND ($x_I, x_F$)
1  $\{x_0, ..., x_m\} \leftarrow$ EMPTYSPACEPLANNER($x_I, x_F$)
2  for $x_i \in \{x_0, ..., x_m\}$
3      if collision($x_i$)
4          return $\{x_0, ..., x_{i-1}\}$
5  return $\{x_0, ..., x_m\}$
```

2.2.3 The RRT Algorithm in Non-Holonomic Spaces

The RRT algorithm can also work in non-holonomic spaces where the system cannot move along any straight line path in the configuration space. For instance, the car shown in Figure 2.11 is non-holonomic. The algorithm outlined in Section 2.2.2 would find paths for the car that the car could not actually execute because it cannot move sideways.

When the system cannot move along any line in the configuration space, the EXTEND function uses what we term an *empty space planner*. An empty space planner, $L(x_I, x_F)$, returns a path from a configuration x_I to a configuration x_F that is executable by the robot in the absence of obstacles. The EXTEND function then checks this path and truncates it to the first collision, as shown in Algorithm 2.3.

This empty space planner is part of the input to the algorithm as it is problem dependent. In Section 2.2.2, it was a straight line path. With non-holonomic systems, like a car, it is often more complicated. For instance, if x_F is a configuration to the left of a car, the empty space planner would return a curve for the car rather than a straight line. This is shown in Figure 2.11.

In this thesis, we work with empty space planners that can return paths all the way from an initial configuration to a final configuration. This is important for manipulation problems because these paths may need to move through low-dimensional subspaces of the configuration space like the subspace in which a robot is grasping an object. However, in some applications, we do not require a full path from the initial to the final configuration. In this situation, it is common to use *local planners* that plan a path only a short way from the initial configuration towards the final configuration. This is most often used when the system is described by a differential equation. By not requiring that the local plan go all the way to the end point, we avoid complicated boundary value problems.

The proof of exponential convergence of the algorithm when we use empty space or local planners has to make some assumptions about those planners. This proof is done in LaValle and Kuffner [28].

2.2.4 RRTConnect Algorithm

The RRTConnect algorithm [23] is a variant of the RRT algorithm that grows two trees, one forward from the starting configuration like the RRT algorithm and one backwards from the goal set, and tries to connect them in the middle. This has proven to be much more efficient in practice.

The RRTConnect algorithm is shown in Algorithm 2.4. This algorithm takes similar input to the RRT algorithm except that the Extend function must be able to extend configurations either backwards or forwards. Namely Extend(x_T, x_S, **True**) extends x_T forwards towards x_S while Extend(x_T, x_S, **False**) extends x_T backwards towards x_S. This distinction is only important in the non-holonomic case.

The advantage of the RRTConnect algorithm is that it finds its way around obstacles more easily because of the attempt to connect the forwards tree to the backwards tree. We give an example of this in Figure 2.12. Figure 2.12a shows an execution trace of the RRT algorithm while Figure 2.12b shows an execution trace of the RRTConnect algorithm for the same samples. In Figure 2.12a, the RRT algorithm:

1. Samples the red configuration (Sample 1)
2. Chooses the start configuration (blue) as the nearest configuration to the red configuration
3. Extends the tree from the start configuration to the red configuration (line 1)
4. Samples the orange configuration (Sample 2)
5. Chooses the red configuration as the nearest configuration to the orange configuration

Algorithm 2.4 *Input*: X, Configuration space; x_0, Starting configuration; X_G, Goal set; ρ, Distance function; EXTEND, Extend function
Output: A path from x_0 into X_G.

```
RRTCONNECT ($X, x_0, X_G, \rho$, EXTEND)
1  $V_a \leftarrow \{x_0\}$, $V_b \leftarrow \{\text{randomConfiguration}(X_G)\}$
2  $F \leftarrow$ True // True when extending forwards
3  while True
4      if $F$
           // Add a goal configuration to the backwards tree
5          $V_b \leftarrow V_b \cup \{\text{randomConfiguration}(X_G)\}$
6      $x_S \leftarrow$uniformRandomConfiguration($X$)
7      $x_T \leftarrow \arg\min_{v \in V_a} \rho(v, x_S)$
8      $\{x_0, ..., x_l\} \leftarrow$ EXTEND($x_T, x_S, F$)
9      $V_a \leftarrow V_a \cup \{x_0, ..., x_l\}$
10     if $l > 0$ // Extend $V_b$ towards $V_a$
11         $x_T \leftarrow \arg\min_{v \in V_b} \rho(v, x_l)$
12         $\{y_0, ..., y_k\} \leftarrow$ EXTEND($x_T, x_l, \neg F$)
13         $V_b \leftarrow V_b \cup \{y_0, ..., y_k\}$
14         if $y_k = x_l$
15             return ExtractPath($V_a, V_b$)
16     swap($V_a, V_b$), $F \leftarrow \neg F$
```

6. Extends the tree from the red configuration to the orange configuration (line 2)

7. Samples the magenta configuration (Sample 3)

8. Chooses a configuration on line 2 as the nearest configuration to the magenta configuration

9. Extends from this configuration towards the magenta configuration, but is truncated because it hits an obstacle (line 3)

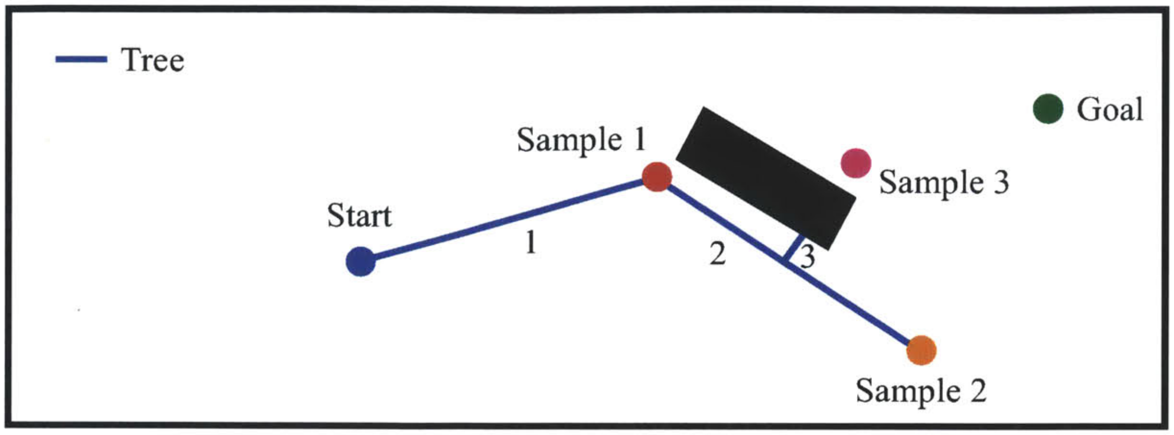
So the RRT algorithm has yet to find a path to the goal. In Figure 2.12b, the RRTCONNECT algorithm:

1. Samples the red configuration (Sample 1)

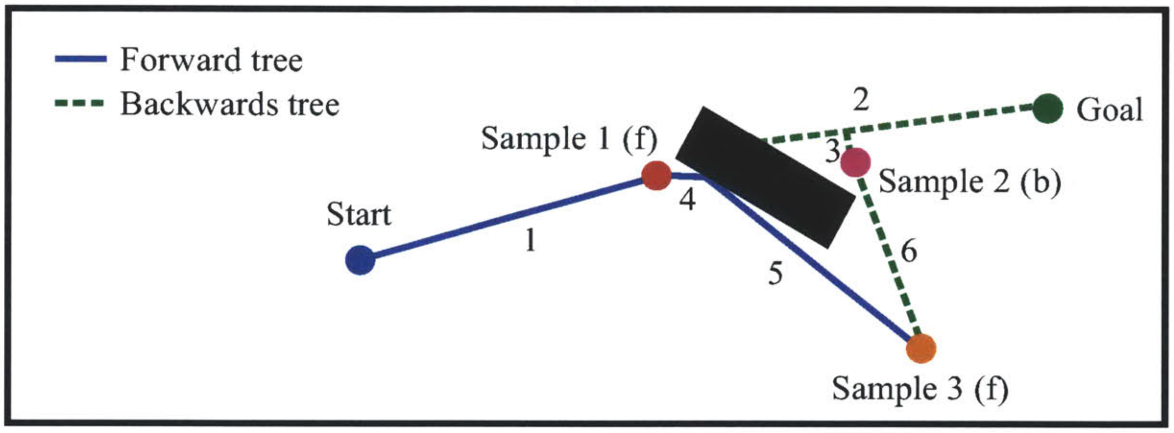
2. Chooses the start configuration (blue) as the nearest configuration to the red configuration

3. Extends the forward tree from the start configuration to the red configuration (line 1)

4. Chooses the goal configuration (green) as the nearest configuration in the backwards tree to the red configuration



(a)



(b)

Figure 2.12: An example of a situation in which the RRTCONNECT algorithm is more efficient than the RRT algorithm. Both figures show a point robot navigating in the two dimensional plane. The robot begins at the blue dot and ends at the green dot and the black rectangle is an obstacle. Figure (a) shows a trace of the RRT algorithm while Figure (b) shows a trace of the RRTCONNECT algorithm. See the text for a full explanation of the traces.

5. Extends backwards from the backwards tree towards the red configuration, but is truncated because it hits an obstacle (line 2)

6. Samples the magenta configuration (Sample 2)

7. Chooses a configuration on line 2 as the closest configuration in the backwards tree to the magenta configuration

8. Extends backwards from this configuration to the magenta configuration (line 3)

9. Chooses the red configuration as the nearest configuration in the forward tree to the magenta configuration

10. Extends the forward tree towards the magenta configuration, but is truncated because it hits an obstacle (line 4)

11. Samples the orange configuration (Sample 3)

12. Chooses a configuration on line 4 as the nearest configuration in the forward tree to the orange configuration

13. Extends this configuration to the orange configuration (line 5)

14. Chooses the magenta configuration as the nearest configuration in the backwards tree to the orange configuration

15. Extends the backwards tree to the orange configuration, connecting the two trees (line 6)

The reason the RRTConnect algorithm is able to find a path quickly in this example while the RRT algorithm requires more time is that when the RRTConnect algorithm extends one tree towards the other, it extends towards the *last* point added to the tree not the nearest point in the tree. This introduces a bias to continue growing into free space. Although the convergence results for the RRTConnect algorithm are the same as for the RRT algorithm, this bias is very helpful in practice.

2.3 Related Work

In this section, we review the literature pertinent to this thesis. Recall that our goal is to solve complicated manipulation problems in which a single object may have to be manipulated multiple times using a different type of manipulation each time. We begin by presenting the previous work in non-prehensile manipulation that we build upon to generate interesting manipulation domains. We then discuss problems and techniques similar to ours, including re-grasping, the navigation among movable obstacles problem, and algorithms for sampling in constrained or multi-modal spaces.

2.3.1 Non-Prehensile Manipulation

In this thesis, we discuss planning for a diverse set of manipulation types. An important aspect of our work is that we can plan for *non-prehensile* manipulation. Non-prehensile manipulation is any form of manipulation in which the object(s) being manipulated are not rigidly attached to the manipulator.

For instance, pushing is a form of non-prehensile manipulation, because the object being pushed is not attached to the robot's end effector. Mason [31, 32, 33] was among the first to discuss pushing for robotic manipulators. Mason's work focuses on dealing with the inherent uncertainty of pushing because the manipulator may not control all degrees of freedom of an object. Additionally, the mechanics of pushing depend on properties that are hard to measure, such as the pressure distribution of an object and the forces of friction on that object. Mason discusses how to predict an object's motion when pushed even when there is some uncertainty about these quantities. Other work [53] uses learning to map robot motions to object motions. In our work, we assume *stable pushing* in which the pusher always remains in contact with the object. We also use two-point pushing, which helps to reduce the uncertainty as, with two-point pushing, a pusher can control all degrees of freedom of an object.

Pushing can also be used as a precursor to grasping. Brost [7] and Dogar and Srinivasa [11] both use a push-grasp. A push grasp is a motion that first "pushes" with the open gripper by moving it parallel to the surface on which by object is resting and then "grasps" by closing the gripper. Brost shows how to decide when the object being push-grasped will roll into the gripper. Dogar and Srinivasa use the push-grasp when an object is in too close contact with the environment for a normal grasp to succeed. For an object in contact with the environment, the push motion can allow a gripper finger to gently separate the object from its contact.

Dogar and Srinivasa [12] expand on the idea of using a push-grasp to separate an object from its environment by allowing the manipulator to sweep some movable objects out of the way while trying to grasp another. They consider a library of four actions: push-grasp, sweep, go-to, and pick-up. The "sweep" action pushes objects with the outside of the hand. Cosgun et al. [9] discuss the related problem of placing an object on a cluttered surface. However, although Dogar and Srinivasa have a library of manipulation actions, they assume that each object or piece of clutter is only moved once using a single type of manipulation action. Cosgun et al. allow multiple objects to be pushed, but they only consider one type of manipulation. Neither work gives any completeness or convergence guarantees. In our work, we have a library of manipulation actions and try to find a plan for a single object that may require many of these actions.

Although we do not explicitly use them in our experiments, other manipulation actions in which the robot is in contact with the object at all times like pulling [26] or pushing a cart [41, 50, 49, 51] should fit within our framework. It should also be possible to extend our approach to more dynamic types of actions like striking and tapping [18, 19], juggling [8, 39, 40], or throwing [42], but this is left for future work.

In all of the work discussed in this section, the focus is on the dynamics and control of a single type of manipulation. When this work addresses planning, it emphasizes planning for this single action type. We take a different view; we build on this existing work to model a set of diverse action types and focus on combining them to generate complex plans.

2.3.2 Re-Grasping

One area of manipulation planning focused on chaining multiple actions for a single object is re-grasping. In the re-grasping task [30, 43], a robot has many choices (sometimes an infinity of choices) for a rigid grasp, and the robot must use more than one grasp to find a solution to the problem. For example, when placing an object on a shelf, a robot might need to adjust the grasp it is using before it can set the object down stably without disturbing anything else on the shelf.

The framework proposed by Siméon et al. [43] for the re-grasping task is the one most closely related to the algorithms we propose in this thesis. They consider manipulation of a single, rigid, six degree-of-freedom movable object. Let O be the configuration space of the object and R be the configuration space of the robot. The full configuration space is $X = O \times R$. Let $X_{free} \subseteq X$ be the subset of X in which neither the object nor the robot is in collision with any obstacles. The authors consider two subsets of X_{free}: CP, the subset of X_{free} in which the object is in a stable placement in the world, and CG, the subset of X_{free} in which the robot holds the object in a valid grasp. These regions allow them to define two types of manipulation paths:

- *Transit paths*: are paths where the robot moves by itself. These paths must be in CP. Note that a path through CP is not necessarily a valid transit path, because the object must remain in the *same* stable position through a transit path. Siméon et al. show that transit paths induce a foliation of CP.

- *Rigid-transfer paths*: are paths where the robot moves while holding the object. These paths must be in CG. Again, not all paths through CG are rigid-transfer paths because the robot's grasp must remain consistent throughout. A grasp "remains consistent" when the position of the robot's end effector does not change relative to the object. Rigid-transfer paths induce a foliation of CG.

A region of interest is $CG \cap CP$, the intersection of CG and CP. This is the region in which the robot can grasp or place the object and all connections between transit and rigid-transfer paths must lie in it. Moreover, Siméon et al. prove the following *reduction property*:

Reduction Property: Any path lying in a connected component of $CG \cap CP$ can be transformed into a finite sequence of transit and rigid-transfer paths.

In other words, within $CG \cap CP$, the connectivity of the space reflects the underlying connectivity of the system: if two points in a connected component of $CG \cap CP$ are connected by any free path (even one that changes the robot's grip in midair or teleports the object to a new position), the robot can in fact move the system between the two points.

Therefore, it is sufficient only to study the inter-connectivity of the connected components of $CG \cap CP$ by transit and rigid-transfer paths. We need not study the intra-connectivity because the reduction property guarantees that once in a connected component of $CG \cap CP$, any other point in that connected component can be reached. Siméon et al. show how to use this to plan first a path of connected components and then each transit or rigid-transfer between connected components. We will call this algorithm PG-map.

Unfortunately, the reduction property relies on the fact that the robot can move a rigidly grasped object instantaneously in any direction. When we consider non-prehensile manipulation, it no longer holds because we might introduce a constraint dependent on the direction of motion. We show a counter-example using a pushing task.

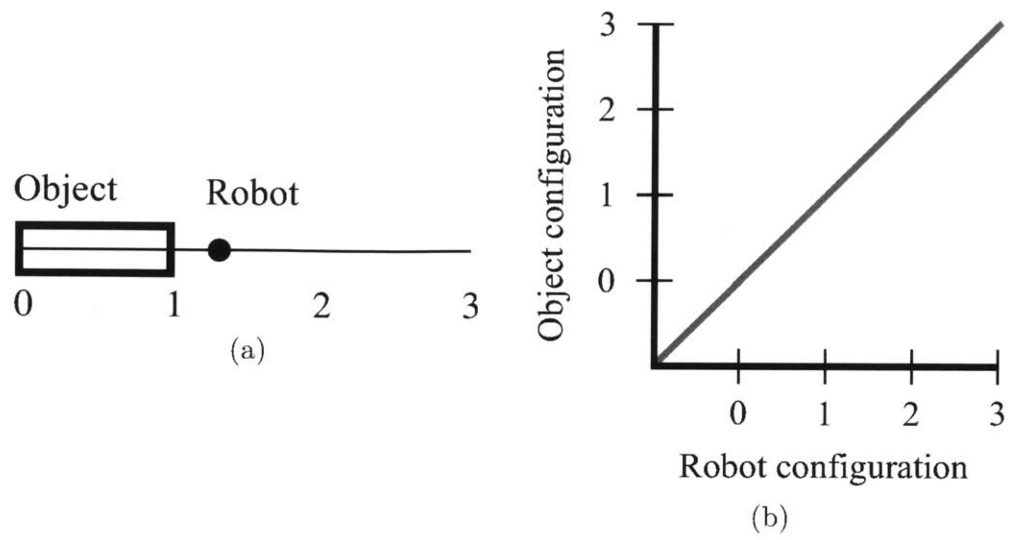


Figure 2.13: A one-dimensional pushing world. (a) The robot (black dot) and object (rectangle) exist on the number line. When the robot is aligned with the left edge of the object, it can push it in the $+x$ direction. (b) The configuration space. CP is the entire region while CG (gray line) is the single line where the robot's center and the object's left edge are aligned. This is a single connected component.

Consider the 1D world shown in Figure 2.13a. If the robot's center is aligned with the left edge of the object, it can push the object in the $+x$ direction. It cannot push the object in the $-x$ direction. Any configuration in which the robot can push the object is a "grasp". All push paths must be in $CG \cap CP$ because the object must also be sitting stably during pushing. The $CG \cap CP$ area is a straight line in configuration space as shown in Figure 2.13b. This has a single connected component so, if the reduction property held, there should be a valid series of transits and pushes from every point in $CG \cap CP$ to every other point. In fact, there is no valid path from, for example, any configuration in which the object is at 1 to any configuration in which the object is at 0 because there is no way for the robot to push the object in the $-x$ direction. Pushing introduces a one-way connectivity.

Therefore, the major leverage proposed by Siméon et al., that the connectivity of $CG \cap CP$ reflects the connectivity of the underlying system, is no longer true once we introduce a non-prehensile action. However, we believe that their idea of using configurations at which rigid-transfer and transit paths intersect as subgoals can be extended to frameworks that include non-prehensile manipulation. We discuss this in Chapter 4.

2.3.3 Navigation Among Movable Obstacles

The navigation among movable obstacles (NAMO) problem [36, 46, 47, 48, 52] is an example of a manipulation problem requiring many manipulation actions. In this problem, there are multiple movable obstacles in the world and a robot navigates among them. Usually, the robot has a goal position, but the movable obstacles do not; the robot only needs to move them if they keep it from reaching the goal. The challenge is that the space is cluttered so that choosing a new position for an obstacle is not necessarily easy.

Most of the work in this domain looks for a plan of transit actions, in which the robot moves alone, and transfer actions, in which the robot moves an obstacle. Usually, it is assumed that only a single transfer action is needed per obstacle and so that each obstacle is only moved once. The focus is on choosing the order in which to move the objects so that they do not obstruct one another rather than on how to move a particular object. In this thesis, we focus on moving a single object that requires many types of manipulation. Future work combining the two approaches could allow a robot to move many objects in a cluttered space using non-prehensile manipulation.

van den Berg et al. [52] approach the NAMO problem somewhat differently. They point out that the canonical representation of the problem as a sequence of transit and transfer actions makes it very hard to devise a probabilistically complete algorithm, because these actions lie in "slices" of the configuration space. There are an infinite number of such slices, but each slice has a probability zero of being sampled. Therefore, they consider the movement of the movable obstacles primarily and then try to find robot motions that can accomplish these obstacle paths.

Specifically, van den Berg et al. look for *valid* obstacle paths. We also use this concept so we review their definition here. An obstacle is *manipulable* if its configuration space representation has some point adjacent to the current connected component of the robot. A path for an obstacle is valid if

1. The path is collision free for the obstacle.

2. The obstacle is manipulable at all points along the path.

van den Berg et al. assume that the obstacle can be rigidly grasped by the robot at any point of contact. Therefore, if the obstacle is "manipulable", the robot can move it any direction. Under this assumption, any valid path has an accompanying collision free robot motion. The algorithm presented by van den Berg et al. first plans valid paths for obstacles and then plans the robot motion for each path. An obstacle may be moved more than once.

We also use the idea of planning for the movable objects first and then finding a robot path that can accomplish this plan. However, van den Berg et al. were only able to demonstrate their algorithm using axis-aligned rectangular obstacles and a robot that can only translate in the plane, because their algorithm requires analytically describing the connected components of the robot's configuration space. Moreover, we do not assume that the robot can rigidly grasp the object at any point of contact or even rigidly grasp it at all, so we need to modify the definition of manipulable. We discuss this further in Chapter 4.

2.3.4 Sampling and Constrained Motion Planning

When working with manipulation, we cannot sample configurations uniformly at random from the space as done on Line 3 of the RRT algorithm because we need to sample

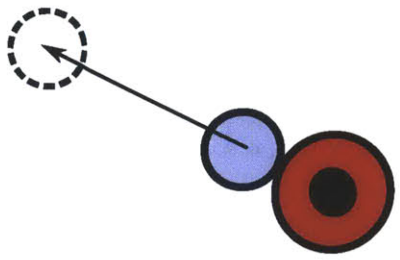


Figure 2.14: The robot (red disc with black center) can push the object (light blue disc) only along the ray (black arrow) connecting the center of the robot to the center of the object.

in spaces that are lower dimensional than the full configuration space. We do this by sampling uniformly at random from the space and then projecting the sample onto a subspace.

Much of the work in sampling other than uniformly from the configuration space has focused on speeding up sampling-based algorithms without changing the exponential convergence properties [6, 16, 24, 25, 54, 55]. These algorithms attempt to boost sampling in narrow passage regions where sampling-based planners tend to have difficulty sampling.

However, there is also work on sampling for constrained motion planning problems in which the constraint is specified as a constraint on the end effector in the work space [4, 5, 45, 56]. In these problems, the constraint defines a lower-dimensional subspace of the full configuration space. For instance, we could define a constraint in which the robot's gripper must remain parallel to the ground at all times for transporting a full pitcher of water. There is zero probability that any such configuration can be sampled uniformly at random from the configuration space. Of this body of work, only Berenson and Srinivasa [4, 5] are able to show that their algorithm is probabilistically complete or exponentially convergent. They take a similar approach to ours so we discuss their work in more detail.

As we do, Berenson and Srinivasa choose a configuration uniformly at random from the space and then project it onto a subspace defined by the constraints of the problem (i.e. the robot's gripper must be parallel to the ground). In their work, Berenson and Srinivasa assume that the constraints are all holonomic constraints on the robot's end effector. With this assumption, they can represent constraints as Task Space Regions or Task Space Region chains. Task Space Regions define a region of the work space to which the end effector is constrained while the chains describe constraints that can be represented as closed kinematic chains. Berenson and Srinivasa use the Jacobian pseudo-inverse method to map samples from the configuration space onto the subspace defined by these constraints. They prove that this sampling covers the constraint space so the algorithm is still probabilistically complete.

The advantage of this representation of constraints is that Berenson and Srinivasa do not require the user to supply the projection functions, but only the constraints on the end effector. Non-prehensile manipulation, however, cannot always be expressed as end effector constraints. Consider a round robot pushing a round object as shown in Figure 2.14. There are two constraints: the robot must contact the object and the robot must push the object along the ray connecting the robot's center to the object's center. In this case, the robot is also the end effector. We can express the constraint that it contact the object as a holonomic constraint since we know the object's position. Once the robot and object begin moving together, however, there is no way to express the directionality constraint of the pushing as a constraint only on the robot's position. We could constrain the robot to only move along a line, but we cannot constrain it to only move along a ray.

Therefore, we require that users provide the full projection functions for the samples rather than just a constraint. We can prove that our algorithm converges exponentially even for projection functions that are not holonomic constraints on the robot's end effector.

2.3.5 Multi-Modal Planning

In Chapter 4, we frame the problem of manipulation with diverse actions as a multi-modal planning problem. Hauser [14] defines a multi-modal planning problem as one in which the system moves between configurations and also among a set of *modes.* The mode space is part of the problem description and each mode describes a set of configurations that all satisfy certain mode-specific constraints. In his initial work, Hauser focused on problems with discrete mode spaces, but low-dimensional mode transitions. For instance, in legged locomotion, the modes are a fixed set of footfalls. The footfalls constrain the feet to be on the ground, and walking must transition through these footfalls.

Subsequently, Hauser and Ng-Throw-Hing [15] extended the work on multi-modal planning to domains with continuous modes. They describe the set of continuous modes as a finite, discrete set of *mode families.* Mode families partition a continuous mode set using a co-parameter that varies to describe each of the different modes. Transitions between modes within a mode family are disallowed; modes must first transition out of the family. An example of a problem with continuous modes is shown in Figure 2.15. In this problem, only one dot can move along the line at a time. The modes in this problem are the dot that is moving and also the position of the two stationary dots. Since the two stationary dots can be located anywhere along the line, there are an infinity of modes. However, we can partition these modes by which dot is moving, giving us three mode families: the blue family, the red family, and the green family. The co-parameter for each the mode family is the position of the stationary dots. For instance, the co-parameter for the blue family is the position of the red and green dots. Additionally, consider moving between two modes in the blue family. This requires moving at least the red or the green dot so we must transition out of the blue family before we can move to a new mode in the blue family.

The multi-modal planning algorithms proposed by Hauser require three pieces of information:

- A mode adjacency graph: This graph should capture every transition between modes, although it may include some impossible transitions. A mode transition

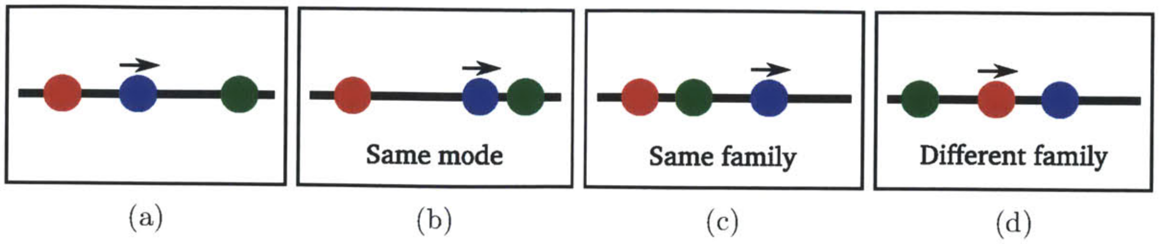


Figure 2.15: An example of a problem with continuous modes, but discrete mode families. Only one dot can move along the line at a time. A mode specifies the moving dot and the exact locations of the stationary dots. A mode family specifies the moving dot with the locations of the stationary dots as co-parameters. Figure (a) shows a mode in the blue family. Figure (b) is in the same mode while figure (c) is in the same mode family but a different mode because the green dot has moved. Figure (d) is a different mode and a different family. Note that to move from the configuration shown in figure (a) to the configuration shown in figure (c), the green dot must move. This requires a moving through configurations in the green family. Modes within the same family cannot transition directly to each other.

graph for the legged locomotion problem with six legs, for example, might have the mode with all feet on the ground adjacent to any mode with just one foot off the ground.

- A method for sampling from mode transitions: Given two adjacent modes, σ and σ', this method should sample a configuration in the transition between the two modes. For example, in legged locomotion, if σ was a mode with one foot in the air and σ' was a mode with all feet on the ground, this function should return a good foot placement for placing all feet on the ground.

- A method for planning within a single mode: The idea is that planning within a single mode is easy so providing a planner to plan a single mode path should not be difficult. In legged locomotion, this would be a plan for placing a foot or lifting one up.

Given this information, all of the explicit multi-modal algorithms perform essentially the same steps. Given a current configuration x and mode σ:

1. Sample an adjacent mode, σ', from the mode adjacency graph

2. Sample a transition configuration, x', from the intersection of σ to σ'

3. Plan a collision free, feasible path within this single mode from x to x'

We focus here on the Random-MMP algorithm proposed by Hauser and Ng-Throw-Hing because it is most similar to our work. Random-MMP is shown in Algorithm 2.5. This algorithm is similar to the RRT algorithm of Algorithm 2.1 except that when it extends the tree, rather than extending towards the sampled configuration, it instead tries to extend towards some mode transition. Hauser and

Algorithm 2.5 *Input*: x_0, Starting configuration; X_G, Goal set; ADJACENTMODE, method for sampling from the mode adjacency graph; TRANSITION, method for sampling a transition between two modes; PLANSINGLEMODE, method for planning an intra-mode path
Output: Tree with path from x_0 into X_G

```
RANDOMMMP($x_0$, $X_G$, ADJACENTMODE, TRANSITION, PLANSINGLEMODE)
1  $T \leftarrow \{(x_0, NULL)\}$
2  while $T \cap X_G = \{\}$
       // $c_i$ is a (configuration, mode) pair
3      $(c_0, ..., c_k) \leftarrow$ EXTENDTREE($T$, ADJACENTMODE, TRANSITION, PLANSINGLEMODE)
4      if $c_0 \neq$ NULL
5          Add the path $c_0 \leftarrow \cdots \leftarrow c_k$ as descendants of $c_0$ in $T$
6  return $T$

EXTENDTREE(T, ADJACENTMODE, TRANSITION, PLANSINGLEMODE)
1  $x \leftarrow$ randomConfiguration()
2  $(x_T, \sigma_T) \leftarrow \arg\min_{(x',\sigma')\in T}$Distance$(x', x)$ // $x_T$ is a configuration and $\sigma_T$ is a mode
3  return PLANMODESWITCH($x_T$, $\sigma_T$, ADJACENTMODE, TRANSITION, PLANSINGLEMODE)

PLANMODESWITCH($x$, $\sigma$, ADJACENTMODE, TRANSITION, PLANSINGLEMODE)
1  $\sigma' \leftarrow$ ADJACENTMODE($\sigma$)
2  $x' \leftarrow$ TRANSITION($\sigma, \sigma'$)
3  if PLANSINGLEMODE($x, x', \sigma$) fails
4      return NULL
5  else
6      return $(x', \sigma')$
```

Ng-Throw-Hing were able to use this algorithm to find paths for a walking robot pushing an object on a table. However, that work required the implementation of complicated mode samplers and a number of heuristics, some of which took substantial pre-processing time. Hauser and Ng-Throw-Hing do not show how to generalize their problem-specific framework to other manipulation problems and, as we show in Chapter 4, the algorithms proposed for multi-modal problems are unable to solve some manipulation problems of interest.

In the next two chapters, we describe our algorithms for planning for manipulation with diverse actions. We build on the work described here, using sampling-based planners similar to the RRT and RANDOM-MMP algorithms, but address problems that neither of these algorithms can solve unmodified.

Chapter 3

Sampling-Based Algorithms for Diverse Action Manipulation

In this section, we present two sampling-based algorithms for diverse action manipulation. We begin by formally defining the Diverse Action MAnipulation (DAMA) problem and showing that the RRT algorithm cannot solve DAMA problems. We then discuss the Diverse Action Rapidly-exploring Random Tree (DARRT) algorithm, a sampling-based algorithm for the DAMA problem. Lastly, we give the DARRT-CONNECT algorithm for DAMA problems, which is based on the RRTCONNECT algorithm.

We do not present results for these algorithms in this chapter as we discuss more algorithms for the DAMA problem in Chapter 4. Therefore, we postpone experimental results until Chapter 5 and theoretical results, including a proof of exponential convergence for DARRT, until Chapter 6.

Some of the work in this chapter was previously discussed in Barry et al. [2, 3].

3.1 Diverse Action Manipulation Problem

We address problems in which we have a robot, a set of movable objects, and a set of *manipulation primitives*. The configuration space for these problems is a cross product space of the robot configuration space, R, and the object configuration spaces, $O_1, ..., O_n$, $X = R \times O_1 \times ... \times O_n$. Manipulation primitives describe the actions the robot can take in the space. For instance, the transit, rigid-transfer, and push manipulations discussed in Chapter 2 are all individual manipulation primitives. More generally, we define a manipulation primitive as a function that returns a *trajectory* from one configuration to another:

Definition 3.1 (Trajectory): Let $x_I, x_F \in X$ be configurations. A function $\tau : [0, 1] \rightarrow X$ is a *trajectory from x_I to x_F* if and only if $\tau(0) = x_I$ and $\tau(1) = x_F$.

Recall that X is the cross product space of the robot and object spaces so trajectories include configurations for objects as well as robots. For instance, a trajectory in which the robot is holding an object in a rigid grasp is a sequence of configurations in which the relative position of the robot's end effector and the object does not change.

Not all types of manipulation can begin at or reach every configuration. For example, transit, in which the robot moves alone, cannot occur from a configuration in which the robot is holding an object. Similarly, rigid-transfer, in which the robot moves with a rigidly attached object, can never reach a configuration in which the robot is not holding an object. Therefore, manipulation primitives are partial functions that operate over a subset of the possible input pairs. Formally:

Definition 3.2 (Manipulation Primitive): A *manipulation primitive* is a deterministic partial function that takes as input an initial configuration x_I and a final configuration x_F and returns a trajectory from x_I to x_F. A primitive p is *applicable* only to pairs of configurations in its domain, denoted $X(p)$:

$$X(p) = \{(x_I, x_F) \in X \times X \big| (x_I, x_F) \text{ is in domain of } p\}. \tag{3.1}$$

The set of initial configurations that a primitive p can be applied to is

$$X_I(p) = \{x_I \in X \big| \exists x_F \in X,\ (x_I, x_F) \in X(p)\}, \tag{3.2}$$

while the set of configurations reachable by primitive p is

$$X_F(p) = \{x_F \in X \big| \exists x_I \in X,\ (x_I, x_F) \in X(p)\}. \tag{3.3}$$

The set of configurations a primitive p is applicable to given an initial configuration x_I is $X_F(p|x_I)$,

$$X_F(p|x_I) = \{x_F \in X \big| (x_I, x_F) \in X(p)\}. \tag{3.4}$$

$X_F(p|x_I)$ may be substantially smaller than $X_F(p)$.

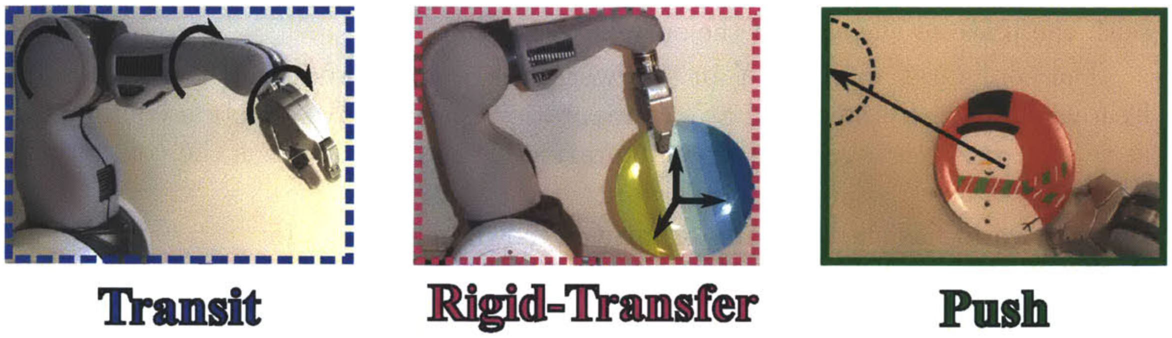
We also make a distinction between primitives that move an object and primitives that move only the robot. This will be helpful when we consider separately the paths objects can take in Chapter 4.

Definition 3.3 (Transit/Transfer Primitive): For primitive p, let $(x_I, x_F) \in X(p)$ and let $\tau = p(x_I, x_F)$. The primitive p is a *transit* primitive if and only if for all $\alpha \in [0, 1]$, the configuration of every object in $\tau(\alpha)$ is the same as its configuration in $\tau(0)$. The primitive p is a transfer primitive if and only if it is not a transit primitive.

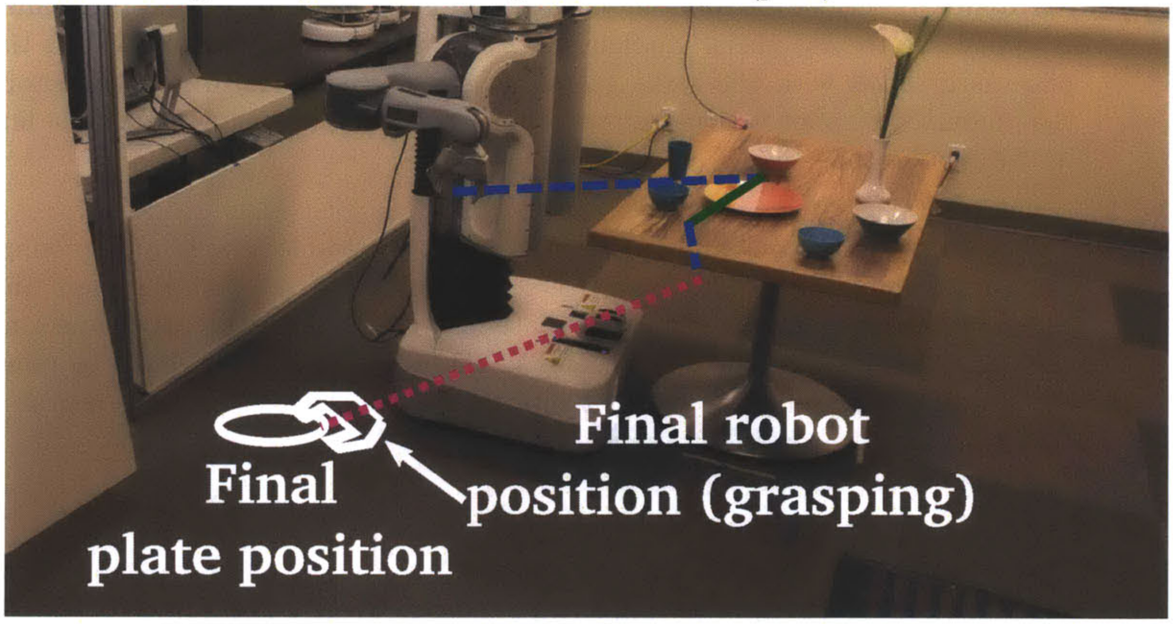
Note that transfer primitives are primitives that move objects on some input, but may not move them on all inputs.

For many primitives both $X_I(p)$ and $X_F(p)$ are lower-dimensional than the full configuration space, X. This fact will keep traditional sampling-based algorithms from being able to solve manipulation problems. We discuss this further in Section 3.2.

Any type of manipulation can be represented as a manipulation primitive provided it is possible to simulate the effect of the primitive. For complicated primitives this



(a) The primitives available in the Push World. The robot can move by itself (`transit`). It can move the rigidly grasped plate (`rigid-transfer`). It can push the plate along the ray connecting the center of the gripper to the center of the plate when the gripper is in two-point contact with the plate, the plate rests on a supporting surface, and the gripper can move along this ray without moving the robot's base (`push`).



(b) A trajectory sequence from the configuration shown in the photograph to the sample shown with the white solid lines. This sequence first `transits` the robot to a pushing configuration (blue dashed), `pushes` the plate to the edge of the table (green solid), `transits` the robot to a grasp (blue dashed), and `rigid-transfers` the plate to the final position (magenta dotted). For simplicity of visualization, we show straight lines in joint space as straight lines for the gripper.

Figure 3.1: An example world in which a robot manipulates a plate.

could require computational integration of equations of motion, but we use simpler primitives in this thesis.

Throughout this chapter, we consider an example world, the Plate World, shown in Figure 3.1, in which a robot manipulates a plate. The robot can push the plate when it is on a support surface or rigidly transfer it when it is grasped. The robot's grippers are too thick to grasp the plate while it is on a surface so the robot must push the plate to the edge of the surface to grasp it.

The configuration space X in this world is the cross product of the robot's configuration space R and the plate's configuration space O. We denote a configuration as $(r, o) \in R \times O$ where r is the configuration of the robot and o is the configuration of the plate.

This world has the following primitives shown in Figure 3.1a:

Transit describes the robot moving alone. `Transit` is applicable to any pair of configurations in which the plate is supported by a surface and does not move. On any applicable input $\big((r_I, o_I), (r_F, o_I)\big)$, `transit` returns a straight line from r_I to r_F in the robot's subspace.

Rigid-transfer describes the robot moving the rigidly attached plate. `Rigid-transfer` is applicable to any initial configuration and final configuration in which the plate is grasped with the same grasp. `Rigid-transfer` returns a trajectory that moves the robot and plate to the final configuration using a straight line in the robot's subspace.

Push describes the robot pushing an object. `Push` is applicable to any initial configuration in which the robot's gripper is in two-point contact with the plate and any final configuration in which the plate has moved along the ray connecting the center of the gripper to the center of the plate and it is possible to move the gripper along this line without moving the robot's base. `Push` returns a trajectory that moves the gripper along this ray.

We give more examples of primitives in Chapter 5.

We generate long trajectories by sequencing the output of several primitives as shown in Figure 3.1b. Because we want to be able to identify trajectories with the primitives that generated them, we use trajectory sequences rather than creating one long trajectory.

Definition 3.4 (Trajectory Sequence): An ordered set of trajectories $\{\tau_0, ..., \tau_l\}$ is a *trajectory sequence* from x_I to x_F if and only if $\tau_0(0) = x_I$, $\tau_l(1) = x_F$ and, for all $i \in \{1, ..., l\}$, $\tau_{i-1}(1) = \tau_i(0)$.

Since each primitive returns a trajectory, a set of primitives generates a trajectory sequence.

Definition 3.5 (Generate): A sequence of primitives $\{p_0, ..., p_l\}$ *generates* a trajectory sequence from x_I to x_F, $\{\tau_0, ..., \tau_l\}$ if and only if for all $i \in \{0, ..., l\}$, $(\tau_i(0), \tau_i(1)) \in X(p_i)$ and $p_i(\tau_i(0), \tau_i(1)) = \tau_i$. A trajectory sequence can be generated by a set of primitives P if there is some sequence of primitives in P that can generate the sequence. The pairs of configurations for which a set of primitives P can generate a trajectory sequence are denoted $X(P)$:

$$X(P) = \left\{ (x_I, x_F) \in X \times X \;\middle|\; \begin{array}{c} \text{some sequence of the primitives in } P \text{ can} \\ \text{generate a trajectory sequence from } x_I \text{ to } x_F \end{array} \right\}. \tag{3.5}$$

The set of initial configurations from which P can generate trajectories is

$$X_I(P) = \{x_I \in X \,|\, \exists x_F \in X,\ (x_I, x_F) \in X(P)\}, \tag{3.6}$$

while the set of configurations to which P can generate trajectories is

$$X_F(P) = \{x_F \in X \,|\, \exists x_I \in X,\ (x_I, x_F) \in X(P)\}. \tag{3.7}$$

The set of configurations to which a primitive set P can generate trajectories given an initial configuration x_I is

$$X_F(P|x_I) = \{x_F \in X \,|\, (x_I, x_F) \in X(P)\}. \tag{3.8}$$

Now that we have formally defined manipulation primitives, trajectories, and trajectory sequences, we can define a manipulation problem.

Definition 3.6 (DAMA Problem): The *Diverse Action MAnipulation* (DAMA) problem is a tuple $\langle R, \{O_1, ..., O_n\}, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G\rangle$ in which R is the configuration space for a robot, $\{O_1, ..., O_n\}$ are the configuration spaces for the movable objects, $\{B_0, ..., B_q\}$ is a set of fixed obstacles, $\{p_0, ..., p_m\}$ is a set of manipulation primitives, x_0 is an initial configuration, and X_G is a set of goal configurations.

The configuration space for a DAMA problem is the cross product space of the robot and object configurations spaces $X = R \times O_1 \times ... \times O_n$.

The goal set for a DAMA problem may be infinite in size, and often is. For example, in manipulation, goal configurations are often specified only for objects. The goal set is any configuration in the full configuration space in which the objects are in their goal configurations.

In most DAMA problems, there are some fixed obstacles in the world with which contact may be permissible. For example, if the robot is pushing an object, the object contacts the table on which it sits, the robot's gripper contacts the object, and the robot's gripper contacts the table. Therefore a primitive can define a set of collisions for which it is *disabling* collision checking. The primitive is responsible for managing this contact. For example, push disables collision checking between the robot gripper and the table so must be careful never to return a trajectory in which the gripper goes through the table.

We define the free space as not allowing any contact:

Definition 3.7 (Free Space): The *free space* for a DAMA problem $\mathcal{P}$, $X_{free}(\mathcal{P})$ is all configurations in which there is no contact between the robot and objects, the robot and the fixed obstacles, or the objects and the fixed obstacles.

We will give a more rigorous treatment of free space in Chapter 6.

A collision free trajectory is one that is either entirely in free space or only has contacts that the primitive allows.

Definition 3.8 (Collision Free): A trajectory τ generated by primitive p is *collision free* in DAMA problem $\mathcal{P}$ if, for all $\alpha \in [0, 1]$, either $\tau(\alpha) \in X_{free}(\mathcal{P})$ or all collisions present in $\tau(\alpha)$ have collision checking disabled by p. A trajectory sequence is *collision free* if all of its trajectories are collision free.

A *solution* to a DAMA problem is a collision free trajectory sequence from x_0 to some configuration in X_G that can be generated by the primitives.

In the next sections, we discuss algorithms for solving DAMA problems based on the RRT [28] and RRTConnect [23] algorithms.

3.2 Diverse Action Rapidly-exploring Random Tree Algorithm

In this section, we present the Diverse Action Rapidly-exploring Random Tree (DARRT) algorithm for solving DAMA problems. This algorithm is based on the RRT algorithm [28], but modified to plan through the low-dimensional subspaces necessary in manipulation. We begin with an example to illustrate the problems with the classic RRT approach and then discuss the DARRT algorithm.

3.2.1 Motivating Example

We begin by motivating our claim that the classic RRT algorithm fails to plan for some manipulation problems. Consider again the Plate World introduced in Section 3.1. For simplicity, in this discussion, we ignore the `rigid-transfer` primitive and assume that we have only a robot arm pushing a plate on a table (the Plate Pushing World). We also assume that all pushing configurations on the table can be reached by the robot. Recall that this world has a configuration space, $X = R \times O$, that is the cross product of a robot configuration space, R, and a plate configuration space, O. In this case, the plate configuration space is two-dimensional as the plate is round so its orientation is irrelevant and we assume it remains on the table. A configuration in this space is denoted $(r, o) \in R \times O$.

Firstly note that although the robot arm is holonomic, we cannot use a holonomic extension like that of Section 2.2.2 because the full configuration space of the robot and plate is non-holonomic. Namely, consider extending from a configuration $x_I = (r_I, o_I)$ to a configuration $x_F = (r_F, o_F)$. If the space was holonomic, we could extend using a straight line in the space. A straight line in the full configuration space corresponds to a line in the robot's subspace and a line in the plate's subspace. While

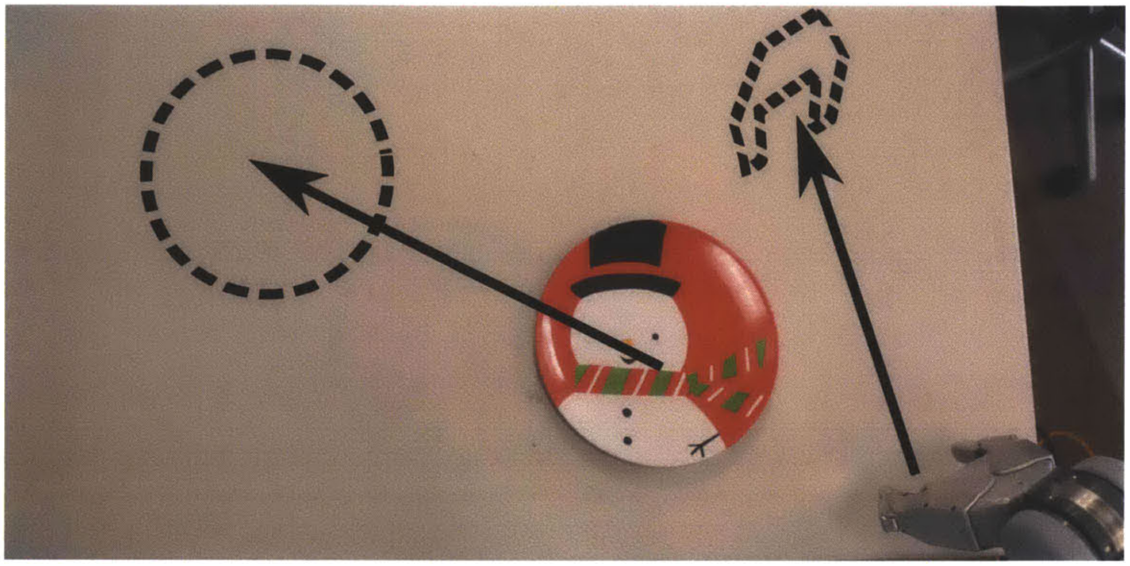
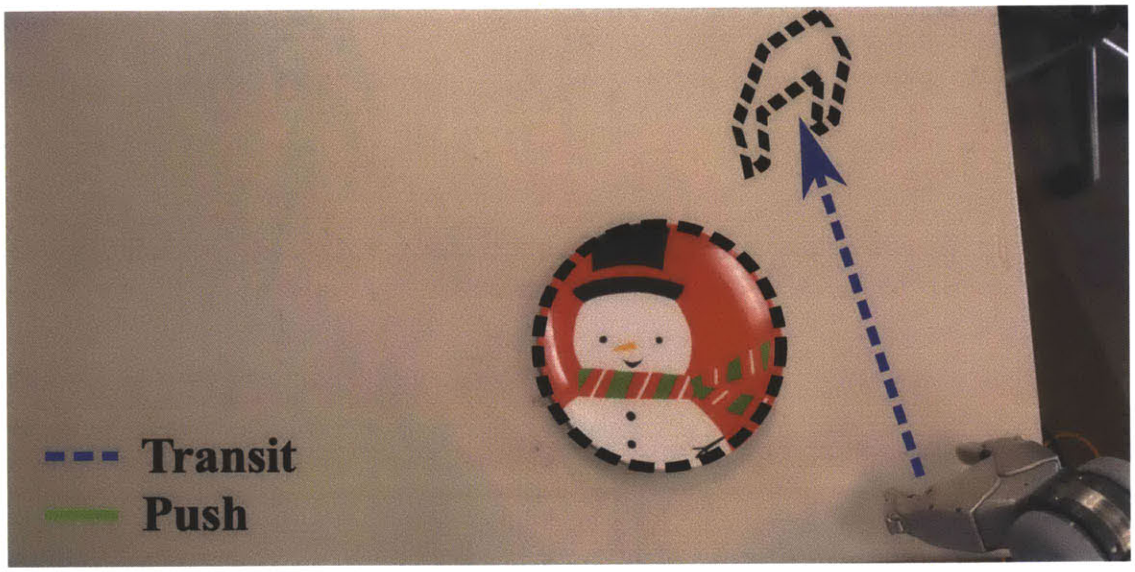


Figure 3.2: A straight line extension in the plate pushing domain. The initial configuration is shown in the photograph while the final configuration is shown by the dashed outlines. This extension cannot be executed because the plate cannot move on its own.

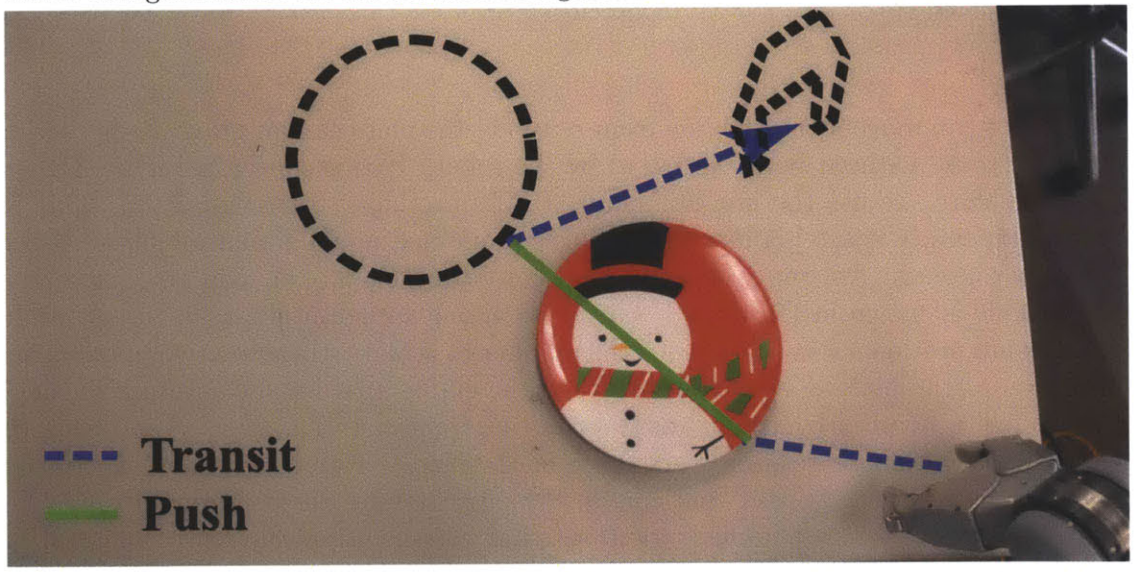
the robot can move along the line from r_I to r_F, the plate cannot move along the line from o_I to o_F without being actuated by the robot. This is shown in Figure 3.2.

Therefore, we use the non-holonomic RRT introduced in Section 2.2.3. This requires an empty space planner that returns plans that could actually be executed in the environment. An empty space planner for our problem of an arm pushing a plate on a table is shown in Figure 3.3. Given an initial configuration $x_I = (r_I, o_I)$ and a final configuration $x_F = (r_F, o_F)$ for the planner to achieve, there are two possibilities. If $o_I = o_F$ (the plate does not move), then the empty space planner returns a single trajectory generated by the `transit` primitive from r_I to r_F. If $o_I \neq o_F$, then the empty space planner returns a sequence of three trajectories. There is exactly one configuration for the gripper that can push the plate along the ray from o_I to o_F. The planner chooses a robot configuration, r_P, that has the gripper in this configuration. Let r'_P be the robot's configuration after pushing the plate from o_I to o_F (we assume the robot arm can reach all pushing configurations on the table). The empty space planner returns a `transit` from (r_I, o_I) to (r_P, o_I), a `push` from (r_P, o_I) to (r'_P, o_F), and a `transit` from (r'_P, o_F) to (r_F, o_F). During `transit`, collision checking between the plate and table is disabled. During `push`, collision checking between the plate and table, table and gripper, and gripper and plate are all disabled.

This empty space planner successfully plans through some of the important lower-level subspaces. For instance, it makes it possible to add to the tree configurations in which the gripper is in two-point contact with the plate. However, with this type of empty space planner, a traditional non-holonomic RRT still fails to find a solution to most manipulation problems. One such scenario is illustrated in Figure 3.4. In this case, a fixed obstacle blocks all direct paths from the robot's initial configuration



(a) If the plate's configuration in the final configuration matches the plate's configuration in the initial configuration, the empty space planner returns a single `transit` from the robot's initial configuration to the robot's final configuration.



(b) If the plate's configuration in the final configuration does not match the plate's configuration in the initial configuration, the empty space planner returns a sequence of three trajectories. There is a single configuration of the robot's gripper that can push the plate along the desired ray to its final configuration. The first trajectory is a `transit` from the robot's initial configuration to this pushing configuration. The second trajectory is a `push` from the plate's initial configuration to its final configuration. The third trajectory is a `transit` from the robot's configuration after `push`ing to its final configuration.

Figure 3.3: The empty space planner in the Plate Pushing World. Initial configurations are shown in the photograph while final configurations are shown with dashed black lines. For ease of visualization, only the gripper's path and only the gripper position in the robot's final configuration are shown, and trajectories that are straight lines in joint space are shown as straight lines for the gripper.

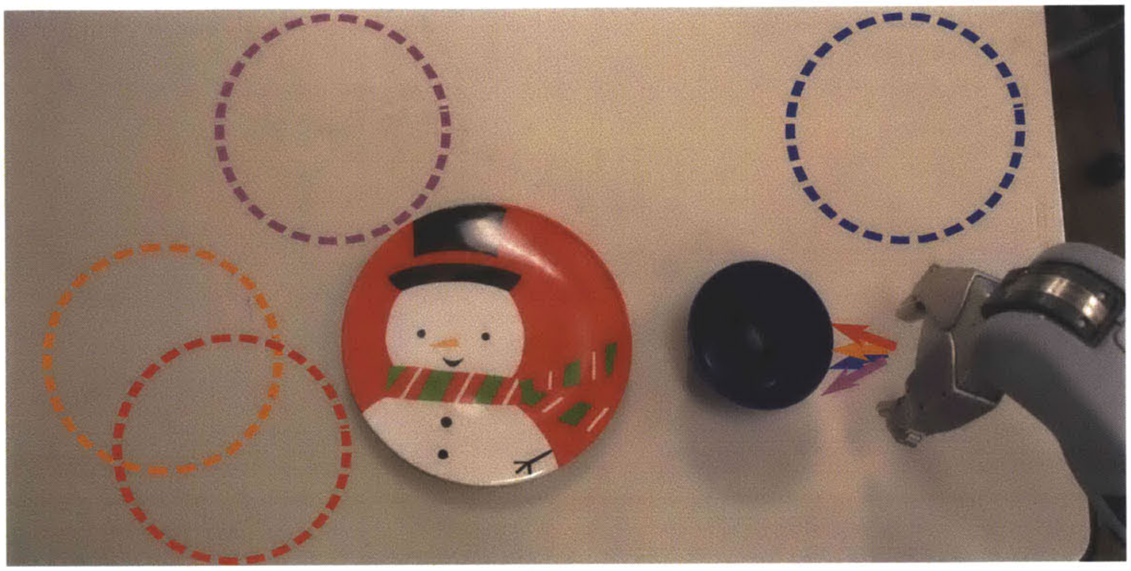


Figure 3.4: An illustration of the failure of the non-holonomic RRT in the Plate Pushing World. The bowl is a fixed obstacle and the initial configuration is shown in the photograph. Every straight line path from the robot's initial configuration to any configuration in which the robot's gripper is in two-point contact with the plate is blocked by the bowl. A number of samples for the plate (the sample for the robot is irrelevant and not shown to avoid clutter) are shown in different colors. The corresponding pieces of path added to the tree are shown as solid lines in the corresponding colors. Because we have zero probability of sampling a configuration in which the plate is in its initial configuration, the first part of the trajectory returned by the empty space planner is always a straight line to a configuration in which the robot's gripper is in two-point contact with the plate. Therefore, with probability one, the algorithm never adds a configuration to the tree in which the robot is pushing the plate.

to any configuration in which the robot's gripper is in two-point contact with the plate. With probability one, the algorithm never samples a configuration in which the plate's sampled position matches its position in the starting configuration passed to the algorithm. Therefore, the empty space planner always returns, as its first primitive, a direct path from the robot's current configuration to a configuration in which its gripper is in two-point contact with the plate. Because all of these paths have an early collision with the bowl, the RRT fails to solve all but the most trivial problems in this domain. This is not simply a situation in which the RRT is slow; the RRT cannot solve the problem shown in Figure 3.4 even given infinite time. It is not complete in this domain because the solution requires moving through the low dimensional subspace in which the robot moves but the plate does not.

The fundamental problem is that the constraints of the primitives lead to the the empty space planner having two distinct solution classes: it can either return a trajectory generated by `transit` or a sequence of `transit-push-transit`. However,

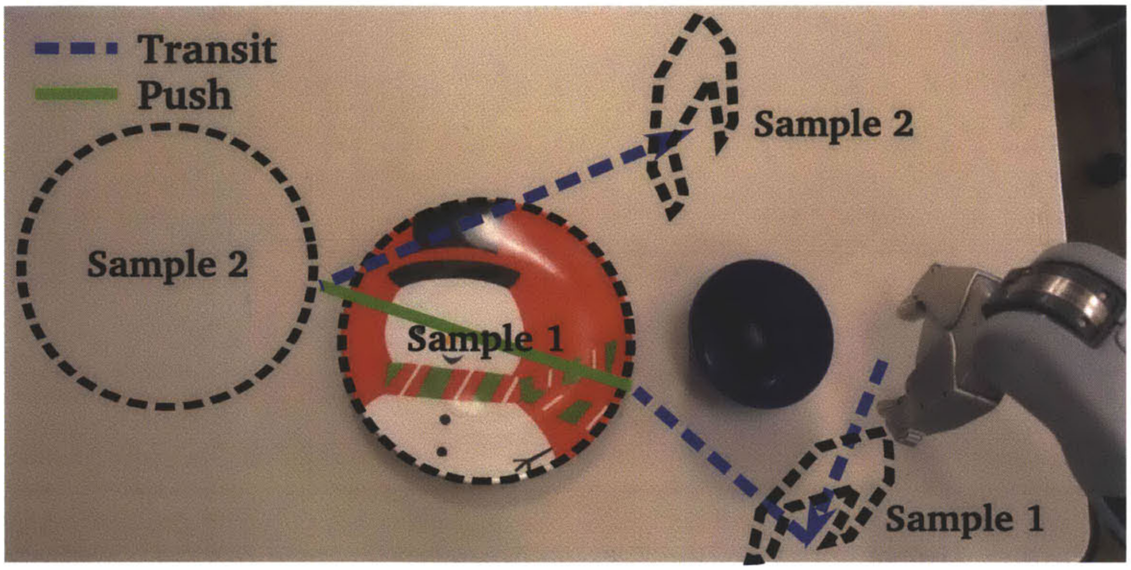


Figure 3.5: In this example, the `transit` projection function was used with Sample 1 followed by the `push` projection function with Sample 2. This allows the robot to move its gripper around the bowl and add configurations to the tree in which it is pushing the plate.

if we sample final configurations uniformly at random from the configuration space, the empty space planner has zero probability of returning the `transit` trajectory. These types of problems are common to manipulation. In almost every manipulation problem, the robot must be near the object in order to manipulate it. Moreover, the types of manipulation themselves can also bifurcate the planner. For instance, consider a primitive that can only move an object to one configuration or along a specific line. Given final configurations sampled uniformly at random from the space, an empty space planner will likely never use this primitive, but it may be necessary to the final solution.

There are many possible methods for circumventing this problem. We could create smarter empty space planners in the manner of explicit multi-modal planning [14]. However, as we discuss in Section 4.1, these planners become more difficult to write as the types of manipulation become more complicated and more inter-dependent. It is hard to imagine a general solution in which the empty space planner is not, in fact, solving the whole problem.

Therefore, our solution is twofold. We use empty space planners to find trajectory sequences the robot can execute in the absence of obstacles, and we also use *projection functions* to ensure that there is some probability that all of the solution classes of the empty space planner can be returned. These functions take as input both the random sample and the configuration in the tree nearest to the random sample and "adjust" the random sample accordingly. For example, in the Plate Pushing World, we could use two projection functions:

Transit Let (r_T, o_T) be the nearest configuration in the tree to the random sample

(r_S, o_S). The `transit` projection function returns (r_S, o_T).

Push Let (r_T, o_T) be the nearest configuration in the tree to the random sample (r_S, o_S). The `push` projection functions returns (r_S, o_S).

Now when we search, we do not immediately try to extend towards the random sample. Instead, we choose a projection function, apply it to the sample, and then use the empty space planner to extend towards the projected configuration. This allows the empty space planner to sometimes return trajectories generated by `transit`, which allows the robot and object to take up different positions relative to each other. This is shown in Figure 3.5.

In the next sections, we explain the DARRT algorithm more fully. In Chapter 6, we show that the combination of the empty space planner and projection functions ensures that DARRT exponentially convergent.

3.2.2 Overview

Pseudo-code for the DARRT algorithm is given in Algorithm 3.1. This algorithm takes as input a DAMA problem, an empty space planner, a set of projection functions, and a distance metric ρ_i for each subspace i. Like an RRT, each iteration begins by choosing a configuration x_S uniformly at random from the full configuration space. We then find the configuration x_T in the tree closest to x_S. We choose a projection function to apply, project x_S to x_F, and extend x_T to x_F. The Extend function is the same Extend function as is used in a non-holonomic RRT.

In the next sections, we discuss our particular choice of distance function for manipulation domains, the empty space planner, and the projection functions. We present the experimental results using this algorithm in Chapter 5 and the theoretical results in Chapter 6.

3.2.3 Distance Function

As we showed in Section 2.2.1, the distance function for an RRT should reflect how well one configuration approximates another. Specifically, if the distance from a configuration x_I to a configuration x_F is small and x_I is in free space, x_F should also be in free space. For manipulation problems, this means that the distance function should consider the distance in each subspace individually. For example, consider again the Plate World of Section 3.1. The full configuration space is the cross product space of the robot's space and the plate's space. If a configuration is in free space, then we can likely move the plate *or* the robot slightly and remain in free space.

Our choice of distance function is shown on Line 4 of Algorithm 3.1. For configuration space $X = R \times O_1 \times ... \times O_n$, let $M_0 = R$ and $M_{i>0} = O_i$. Additionally, for configuration x, let x_i be the projection of configuration x onto subspace M_i. For instance if $x = (r, o_1, ..., o_n)$ the projection of x onto M_0 is r while the projection of x onto $M_{i>0}$ is o_i. We assume that we have distance metrics ρ_i defined for each subspace M_i. We then define the distance from a configuration $x_I \in X$ to a configuration

Algorithm 3.1
Input: $X = R \times O_1 \times ... \times O_n$, Configuration space; $\{B_0, ..., B_q\}$, Fixed obstacles; $\{p_0, ..., p_m\}$, Manipulation primitives; x_0, Initial configuration; X_G, Goal set; L, Empty Space planner; $\{f_0, ..., f_j\}$, Projection functions; $\{\rho_0, ..., \rho_n\}$, Distance metrics for each subspace
Output: Trajectory sequence from x_0 into X_G

```
DARRT ($X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G, L, \{f_0, ..., f_j\}, \{\rho_0, ..., \rho_n\}$)
1  $V \leftarrow \{x_0\}$
2  while $V \cap X_G = \emptyset$
3      $x_S \leftarrow$configuration chosen uniformly at random from $X$
4      $x_T \leftarrow \arg\min_{v \in V} \max_{i \in \{0,...,n\}} \rho_i(v_i, x_{S,i})$
5      $f \leftarrow$ randomChoice($\{f_0, ..., f_j\}$)
6      $x_F \leftarrow f(x_T, x_S)$
7      $\{\tau_0, ..., \tau_l\} \leftarrow$ EXTEND ($x_T, x_F, X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, L$)
8      $V \leftarrow V \cup \bigcup_{\tau \in \{\tau_0,...,\tau_l\}} \bigcup_{\alpha \in [0,1]} \tau(\alpha)$
9  return ExtractTrajectorySequence($V$)

EXTEND ($x_I, x_F, X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, L$)
1  $\{\tau_0, ..., \tau_l\} \leftarrow L(x_I, x_F, \{p_0, ..., p_m\})$
2  for $i \in \{0, ..., l\}$
3      for $\alpha \in [0, 1]$ // Usually discretized in practice
4          if collision($\tau_i(\alpha), \{B_0, ..., B_q\}, X$)
5              return $\{\tau_0, ..., \tau_{i-1}\} \cup$ {$\tau_i$ from 0 to $\alpha$}
6  return $\{\tau_0, ..., \tau_l\}$
```

$x_F \in X$ as the maximum distance in any subspace:

$$\rho(x_I, x_F) = \max_{i \in \{0,...,n\}} \rho_i(x_{I,i}, x_{F,i}). \tag{3.9}$$

The maximum of distance metrics is also a distance metric. In Chapter 6, we show that this distance function allows DARRT to converge exponentially in manipulation domains.

It is interesting to note that this distance metric is not an accurate measure of how far the robot must move between two configurations. Figure 3.6 gives an example in the Plate Pushing World of Section 3.2.1. Here x_I is a configuration in which the robot and plate are far apart while x_F is a configuration in which the plate's and robot's configurations are close to what they are in x_I. To move from configuration x_I to configuration x_F, however, the robot has to move over to the plate, push it, and then move back to nearly its original position. Our distance function considers these two configurations "close," but the actual distance the robot must travel between them is not short. We have not been able to prove that the algorithm is exponentially

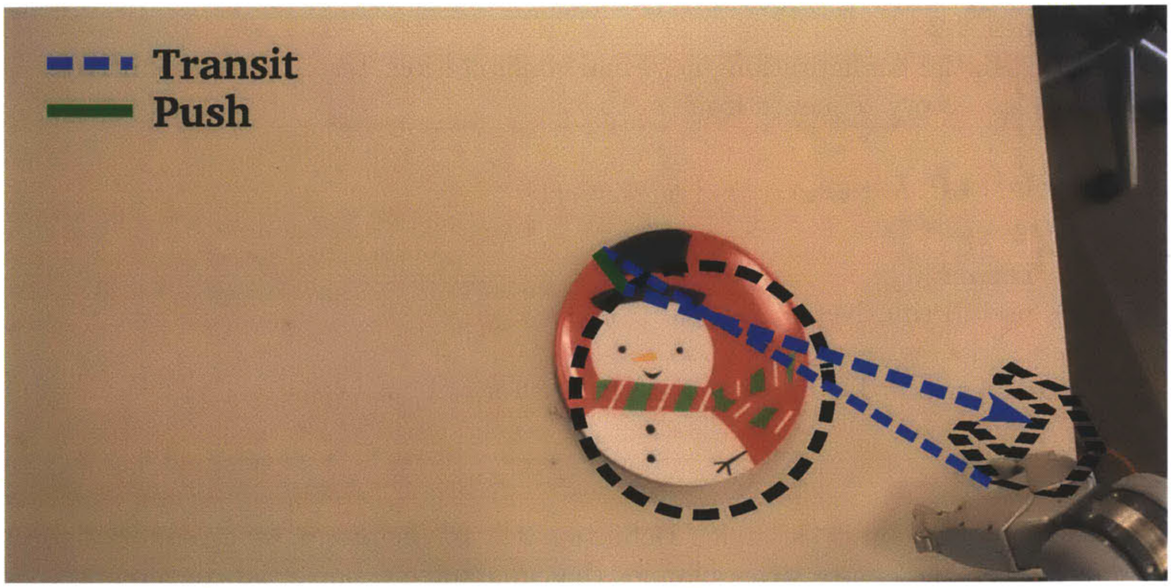


Figure 3.6: The path from the configuration shown in the photograph to the configuration shown with the dashed black lines. Although these two configurations are "close" according to the distance function, this path is long.

convergent if we use a distance function that more accurately reflects the distance the robot travels.

3.2.4 Empty Space Planner

The empty space planner should use the primitives to create a trajectory sequence. This trajectory sequence does not necessarily need to be collision free.

Definition 3.9 (Empty Space Planner): Let $\mathcal{P}$ be a DAMA problem with primitive set P. A function, L, from $X(P)$ to a trajectory sequence is an empty space planner for $\mathcal{P}$ if and only if, for all $x_I, x_F \in X(P)$, $L(x_I, x_F)$ is a trajectory sequence from x_I to x_F that could be generated by P.

This means that if it is possible for the primitives to generate a trajectory from configuration x_I to configuration x_F, the empty space planner must return a trajectory from x_I to x_F. Empty space planners are usually mostly independent of fixed obstacle placement, but they may require some information about support surfaces. Additionally, empty space planners do consider kinematic constraints. The primitives return full trajectories for the robot and objects and in the process must necessarily solve for a full configuration of the robot at every point.

The empty space planner is how the user builds domain knowledge into the algorithm. The planner is dependent on the primitives used so it is part of the input to the DARRT algorithm. In practice, we have found that empty space planners are both easy to write and computationally tractable in many domains. We outline one implementation choice here that worked well in practice, but Algorithm 3.1 only requires

Algorithm 3.2
Input: x_I, Initial configuration; x_F, Final configuration; $\{p_0, ..., p_m\}$, Primitives
Output: Trajectory sequence from x_I to x_F

```
EmptySpacePlanner($x_I, x_F, \{p_0, ..., p_m\}$)
1 if $x_I = x_F$
2     return $\emptyset$
3 $p \leftarrow$ usefulPrimitive($x_I, x_F, \{p_0, ..., p_m\}$)
4 $\{\tau_0, ..., \tau_l\} \leftarrow p$.propagate($x_I, x_F$)
5 return $\{\tau_0, ..., \tau_l\} \cup$ EmptySpacePlanner($\tau_l(1), x_F, \{p_0, ..., p_m\}$)
```

an empty space planner satisfying Definition 3.9. In Chapter 6, we give more rigorous conditions on the empty space planner that guarantee exponential convergence.

In our implementation of the empty space planner, we require that each primitive have some extra information in the form of a **propagate** function. The **propagate** function takes as input an initial configuration x_I and a final configuration x_F and returns either NULL or a trajectory sequence (usually ending with a trajectory generated by the primitive). The first configuration on the trajectory sequence must be x_I, but the last configuration need only be some configuration that can be fed to another primitive's **propagate** function to eventually reach x_F. The user must ensure that the output of **propagate** functions can be chained. A primitive is *useful* in propagating configuration x_I towards configuration x_F if the **propagate** function does not return NULL. The empty space planner then repeatedly searches for a useful primitive[1] and appends the trajectory sequence from that primitive's **propagate** function to its current sequence. This is shown in Algorithm 3.2.

For example, in the Plate World, we use the following **propagate** functions:

Transit The **propagate** function for `transit` returns NULL when transit is not applicable and the trajectory generated by `transit` when `transit` is applicable.

Push The **propagate** function for `push` returns NULL when the plate is in the same position in the initial and final configurations or the plate is not on a support surface in the initial configuration. Otherwise, assume the input is $x_I = (r_I, o_I)$ and $x_F = (r_F, o_F)$. If o_F is on the support surface, let $o_T = o_F$. Otherwise, let o_T be the closest configuration to o_F such that the plate is on the edge of the table. If $o_T = o_I$, the **propagate** function returns NULL. Otherwise, there is one configuration for the gripper that can push the plate along the line from o_I to o_T. The function chooses a configuration for the robot, r_P, with the gripper in this configuration. If it is possible to move the plate in a straight line from

[1]If there are multiple useful primitives for a pair of configurations, the empty space planner chooses one randomly. In Chapter 6, we require that the empty space planner be deterministic. We could require that the empty space planner remember its choices, but in fact, there is zero probability that the empty space planner is called with the same two arguments so there is no need to do this.

o_I to o_T, let r'_P be the robot's configuration after moving the plate along this ray and let $o_P = o_T$. Otherwise, let o_P be the farthest point along this ray that the robot can reach without moving its base, and let r'_P be the configuration for the robot when the plate is at o_P. The function then returns the trajectory generated by `transit` from (r_I, o_I) to (r_P, o_I) and the trajectory generated by `push` from (r_P, o_I) to (r'_P, o_P).

Rigid-Transfer The **propagate** function for `rigid-transfer` returns NULL when the plate is in the same position in the initial and final configurations, the plate is on a support surface and there is no grasp for it that is collision free for the gripper (i.e. the plate is not on the edge of the surface) in the initial configuration, the plate is not grasped in the final configuration, or the grasp in the initial configuration and final configuration do not match. Let r_g be the configuration in which the robot is grasping the plate when the plate is at o_I using the grasp from x_F. The **propagate** function returns a `transit` from (r_I, o_I) to (r_g, o_I) (when $r_g \neq r_I$) and then a `rigid-transfer` from (r_g, o_I) to (r_F, o_F).

Consider creating an empty space plan from the configuration shown in the photograph of Figure 3.1b to the sampled configuration shown with solid white lines. In the initial configuration, `push` is useful. Its **propagate** function returns the trajectory sequence in which the robot first `transits` to a pushing configuration and then `push`es the plate. In the final configuration of this sequence, the plate is on the edge of the table. Thus, in this configuration, `rigid-transfer` is useful. Its **propagate** function returns a `transit` to the grasp and then a `rigid-transfer` to the final configuration. If the first `push` of the plate had not been able to reach the edge of the table, `push` would have continued to be useful, creating more `transit-push` sequences until the plate could be pushed to the edge of the table.

The **propagate** functions are, in a sense, also empty space planners. We let the empty space planner chain them rather than using each **propagate** as a single plan both for ease of implementation and also for the adaptation to the bi-directional planner we discuss in Section 3.3.

The primitives define a set of constraints that govern the trajectories they return. For instance, in the trajectories returned by `transit`, only the robot can change its configuration. In the trajectories returned by `push` (by the actual primitive, not its **propagate** function), the robot and plate always have the same relative configuration. Planning in these constrained spaces creates two challenges: The configuration towards which the algorithm extends should fall within the constraints of some primitive (and not always the same one), and it must be possible to find a configuration at the intersection of two primitives. This is shown in Figure 3.7.

The empty space planner is capable of the latter since it may require many primitives to move from an initial configuration to a final configuration. However, it is possible that the probability of sampling a configuration that the empty space planner can reach is zero. Consider again the Plate World. The empty space planner for this world can only reach configurations in which the plate is supported by the table

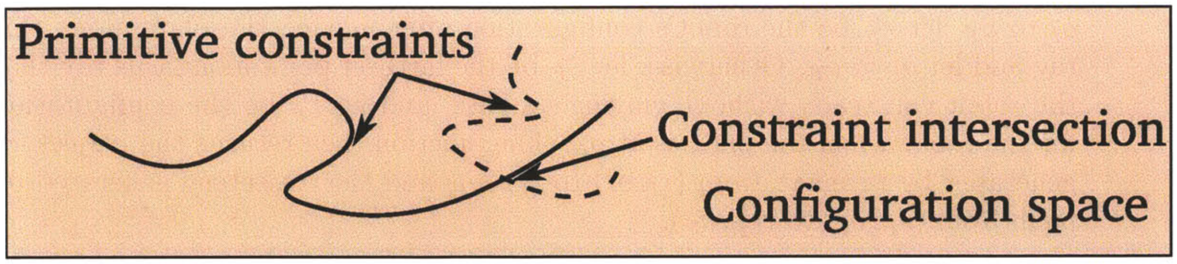


Figure 3.7: Primitives each define a set of constraints that usually are a lower-dimensional subspace of the full configuration space. In this figure, one primitive defines the one-dimensional subset shown by the solid black line and another defines the subset shown by the dashed black line. The search must be able to find configurations in both subsets and also at the (even lower-dimensional) intersection of the sets. We use empty space planners to plan through the intersections and projection functions to guarantee that all samples are in some primitive's constrained subspace.

or grasped. The probability of sampling those configurations is zero. Therefore, we must be able to adjust the final configurations before they are given to the empty space planner. For this, we use projection functions.

3.2.5 Projection Functions

The empty space planner allows the search to move through intersections of constraints as shown. However, it requires samples that fulfill the constraint of some primitive. Moreover, the the empty space planner has many "types" of solutions it can return depending on which constraint the final configuration fulfills. For instance, in the Plate Pushing World, the empty space planner could return a trajectory sequence of only a `transit` or a trajectory sequence of two `transits` and a `push`. Most actual collision free trajectory sequences require multiple `transits` before a `push`. Therefore, we require projection functions that project samples onto the primitive's constraints.

The constraints primitives define depend on the initial configuration, which in turn depends on the primitives earlier in the sequence. The sequence of primitives used to move from one configuration to another is governed by the empty space planner, which returns a sequence of trajectories generated by some sequence of primitives. The sequence of primitives used to generate the trajectory sequence is the *trajectory class*. A necessary condition for completeness is that there is a non-zero probability that the empty space planner return every trajectory class.

Definition 3.10 (Trajectory Class): Let $\mathcal{P}$ be a DAMA problem. The *trajectory class* $C = \{p_0, ..., p_l\}$ is the set of trajectories that can be generated by the sequence of primitives $\{p_0, ..., p_l\}$.

We are interested in the classes of trajectories the empty space planner can return. For trajectory class C, $X(C)$ is the pairs of configurations for which the empty space planner, L, returns a trajectory in C,

$$X(C) = \{(x_I, x_F) | L(x_I, x_F) \in C\}. \tag{3.10}$$

We let the set of initial configurations for which the empty space planner can return a trajectory in class C be

$$X_I(C) = \{x_I \in X | \exists x_F \in X,\ (x_I, x_F) \in X(C)\}, \tag{3.11}$$

while the set of configurations that the empty space planner can reach using a trajectory in class C is,

$$X_F(C) = \{x_F \in X | \exists x_I \in X,\ (x_I, x_F) \in X(C)\}. \tag{3.12}$$

The set of configurations that can be reached from configuration x_I using the empty space planner and only considering trajectories in class C is

$$X_F(C|x_I) = \{x_F \in X | (x_I, x_F) \in X(C)\}. \tag{3.13}$$

For example, in the Plate Pushing World, the empty space planner can return four classes of trajectories: the Transit Class, {`transit`}, the Push Class, {`push`}, the Transit-Push Class, {`transit`, `push`}, and the Transit-Push-Transit Class, {`transit`, `push`, `transit`}. However, if we randomly sample final configurations from the configuration space, the empty space planner has zero probability of ever returning a solution in the Transit, Push, or Transit-Push Classes. We use projection functions instead to ensure some probability of returning all solution classes.

A projection function takes a configuration x_S and an initial configuration x_I and *projects* x_S onto some constrained subspace defined by x_I. Two examples of projection functions were given in Section 3.2.1. A projection function can be any function from $X \times X$ to X.

The projection functions choose the trajectory class that the empty space planner returns. We require that there be some probability of returning every trajectory class.

Definition 3.11 (Projection Function Set): The set of projection functions $\{f_0, ..., f_j\}$ is a *projection function set* for a DAMA problem $\mathcal{P}$ and empty space planner L if and only if for all solution classes C that L can return, for all $x_I \in X_I(C)$, for some f_i, there is probability greater than zero that for x_S chosen uniformly at random from X, $L(x_I, f(x_I, x_S)) \in C$.

As with the empty space planner, Algorithm 3.1 can take as input any set of projection functions that satisfies Definition 3.11. Here we describe a strategy for creating sets of projection functions that works well in practice. In Chapter 6, we treat projection functions more rigorously.

In general, we have found that the ordering of the primitives does not change the constraints on the primitives. The result is that the empty space planner usually has one trajectory class per primitive. Therefore, we create one projection function per primitive. Let $L(x_0)$ be the set of configurations reachable from x_0 using the empty space planner. Then

Definition 3.12 (Primitive Projection Function): For primitive p_i, the *primitive projection function* $f_i(x_I, x_S)$ returns a configuration in $X_F(p) \cap L(x_I)$ or x_S if $X_F(p) \cap L(x_I) = \emptyset$.

Choosing a projection function for a primitive is usually easy. For example, in the Plate World, we have three projection functions:

Transit On input $x_I = (r_I, o_I)$ and $x_S = (r_S, o_S)$, if o_I is not on a support surface (so $x_I \notin X_0(\texttt{transit})$), the function returns x_S. Otherwise, it returns (r_S, o_I).

Push On input $x_I = (r_I, o_I)$ and $x_S = (r_S, o_S)$, if o_I is not on a support surface (so $x_I \notin X_0(\texttt{push})$), this function returns x_S. Otherwise, it first chooses an ending configuration, o_P, for the plate. If (r_I, o_I) is a pushing configuration, o_P is the closest configuration to o_S that can be reached with a single push. Otherwise, o_P is the closest configuration to o_S on the same support surface as the plate in x_I. The robot configuration, r_P, is the pushing configuration in which the robot ends after pushing the plate from o_I to o_P. This function returns (r_P, o_P).

Rigid-transfer function: On input $x_I = (r_I, o_I)$ and $x_S = (r_S, o_S)$, this function first chooses a grasp for the robot. If the robot is grasping the plate in x_I, it chooses this grasp. Otherwise, it chooses a random grasp. It then finds an inverse kinematics solution, r_g. for the robot to use this grasp to hold the plate at o_S. It returns (r_g, o_S).

The definition of the empty space planner and the projection functions are not enough to ensure exponential convergence or even completeness because we cannot guarantee that the distance function allows us to connect the correct configurations. In Chapter 6, we discuss the requirements on the empty space planner and projection functions for exponential convergence. However, it is difficult to verify that a given empty space planner and projection functions meet these requirements. In practice, we have found that using empty space planners and projection functions that fulfill the definitions in this chapter gives good results and are much easier to verify.

3.3 DARRTConnect Algorithm

DARRT is based on the RRT algorithm. Using the empty space planner defined in Section 3.2.4, we can create a version of DARRT based instead on the RRTConnect algorithm.

3.3.1 Motivation

As with the RRT, the majority of the planning time in DARRT is spent finding paths around obstacles. Consider again the Plate World. When the plate is at the edge of

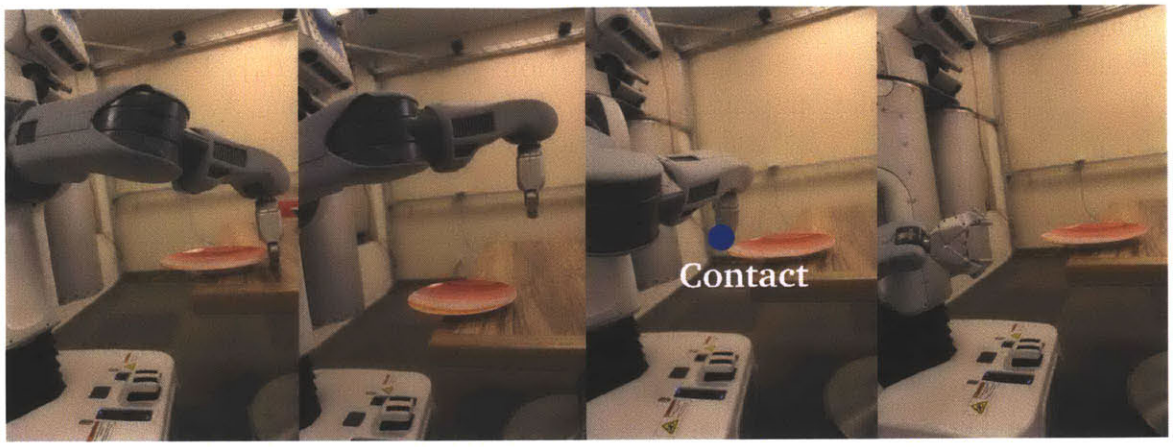


Figure 3.8: When the plate is at the edge of the table the robot can grasp it. However, in trying to move from the pushing configuration (left) to the approach to the grasp (right), the gripper usually contacts the plate.

the table, the robot can grasp it. However, in moving the plate to the edge of the table, the robot must have used the push primitive, which puts its gripper on the far side of the plate from the table edge, as shown in Figure 3.8. During the transition to the grasp, the robot retreats upwards from the push and then moves in a straight line in joint space to the approach to the grasp (the retreat and approach are added to this world in Chapter 5 and make this example clearer). In many cases, there is a collision between the plate and the robot's gripper along this line. Subsequent samples are often near this configuration because the plate is on the boundary of its configurations on the table. However, the configuration in the tree cannot be extended directly to the grasp because of the gripper-plate collision. In order to move around the plate, the gripper must first move off the direct line of the approach-to-grasp, around the plate, and then to the approach-to-grasp configuration. This requires a large number of **`transit`** projection samples before a path is found around the plate.

As with the classic RRT, planning bi-directionally can help alleviate this problem. In this section we present the DARRTConnect algorithm, the bi-directional version of DARRT, based on the RRTConnect algorithm [23].

3.3.2 Algorithm

The DARRTConnect algorithm is shown in Algorithm 3.3. Because our definition of an empty space planner requires that the planner return full plans from an initial configuration x_I to a final configuration x_F, the DARRTConnect algorithm does not require a system of inverse control for primitives. Instead, when extending "backwards" from an ending configuration x_F towards a starting configuration x_I, we simply reverse the order of the arguments to the empty space planner. The empty space planner returns a path from x_I to x_F that we discretize and then check for collisions in the reverse order. The result is a valid path backwards from x_F towards

Algorithm 3.3

Input: $X = R \times O_1 \times ... \times O_n$, Configuration space; $\{B_0, ..., B_q\}$, Fixed obstacles; $\{p_0, ..., p_m\}$, Manipulation primitives; x_0, Initial configuration; X_G, Goal set; L, Empty Space planner; $\{f_0, ..., f_j\}$: Projection functions; $\{\rho_0, ..., \rho_n\}$, Distance metrics for each subspace

Output: Trajectory from x_0 into X_G

```
DARRTConnect ($X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G, L, \{f_0, ..., f_j\}, \{\rho_0, ..., \rho_n\}$)
 1  $V_a \leftarrow \{x_0\}$, $V_b \leftarrow \{$randomConfiguration$(X_G)\}$
 2  $F \leftarrow$ True // True when extending forwards
 3  while True
 4      if $F$
            // Add a goal configuration to the backwards tree
 5          $V_b \leftarrow V_b \cup \{$randomConfiguration$(X_G)\}$
 6      $x_S \leftarrow$configuration chosen uniformly at random from $X$
 7      $x_T \leftarrow \arg\min_{v \in V_a} \max_{i \in \{0,...,n\}} \rho_i(v_i, x_{S,i})$
 8      $f \leftarrow$ randomChoice$(\{f_0, ..., f_j\})$
 9      $x_F \leftarrow f(x_T, x_S, F)$
10      $\{\tau_0, ..., \tau_l\} \leftarrow$ Extend $(x_T, x_F, X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, L, F)$
11      $V_a \leftarrow V_a \cup \bigcup_{\tau \in \{\tau_0,...,\tau_l\}} \bigcup_{\alpha \in [0,1]} \tau(\alpha)$
12      if $l > 0$ or there are configurations in on $\tau_0$ // Extend $V_b$ towards $V_a$
13          $x_T \leftarrow \arg\min_{v \in V_b} \max_{i \in \{0,...,n\}} \rho_i(\tau_l(1)_i, x_{S,i})$
14          $\{\sigma_0, ..., \sigma_k\} \leftarrow$ Extend $(x_T, \tau_l(1), X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, L, \neg F)$
15          $V_b \leftarrow V_b \cup \bigcup_{\sigma \in \{\sigma_0,...,\sigma_k\}} \bigcup_{\alpha \in [0,1]} \sigma(\alpha)$
16          if $\sigma_k(1) = \tau_l(1)$
17              return ExtractTrajectory$(V_a, V_b)$
18      swap$(V_a, V_b)$, $F \leftarrow \neg F$

Extend ($x_T, x_S, X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, L, F$)
    // $x_T$ is the configuration in the tree and $x_S$ is the sampled configuration.
    // $F$ = True when extending forwards, False backwards.
 1  if $F$
 2      $\{\tau_0, ..., \tau_l\} \leftarrow L(x_T, x_S, \{p_0, ..., p_m\})$
 3  else
 4      $\{\tau_0, ..., \tau_l\} \leftarrow L(x_S, x_T, \{p_0, ..., p_m\})$
 5      reverseTrajectories$(\{\tau_0, ..., \tau_l\})$ // Reverse order and every trajectory.
 6  for $i \in \{0, ..., l\}$
 7      for $\alpha \in [0, 1]$ // Usually discretized in practice.
 8          if collision$(\tau_i(\alpha), \{B_0, ..., B_q\}, X)$
 9              return $\{\tau_0, ..., \tau_{i-1}\} \cup \{\tau_i$ from 0 to $\alpha\}$
```

x_I. This is shown in the EXTEND method of Algorithm 3.3 when F is false.

There is one more subtlety in this modification, however, because we use projection functions. We can tell the projection function whether the extension is forwards or backwards. The better the projection function is in the "backwards" direction, the more efficient the algorithm is. However, DARRTCONNECT is still growing a forward tree. Therefore, we do not need to modify the definition of a set of projection functions.

The implementation of backwards projection functions when using the empty space planner in Algorithm 3.2 is similar to that of the forward projection functions. In fact, in many cases, the projection functions we discussed in Section 3.2.5 work almost as well backwards as they do forwards. For instance, we could use the `transit` function unmodified. The `push` function only requires a modification when the final configuration is a pushing configuration. For the `rigid-transfer` function, we should project backwards only when the final configuration is a grasp to create an initial configuration using the same grasp. More generally, for primitive p_i, a reasonable choice for backwards projection function f_i is

$$f_i(x_T, x_S, \mathbf{False}) = \begin{cases} x_S & \text{if } x_F \notin X_F(p_i) \\ \arg\min_{x_I \in X_I(p_i)} \rho(x_I, x_T) & \text{else.} \end{cases} \tag{3.14}$$

These functions can be modified to increase the efficiency of the algorithm.

In the next chapter, we describe a hierarchical algorithm to solve DAMA problems that uses DARRT or DARRTCONNECT as a subroutine. In Chapter 5, we present results for both algorithms and in Chapter 6, we prove that DARRT is exponentially convergent under some assumptions about the configuration space, empty space planner, and projection functions.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

A Hierarchical Approach to Diverse Action Manipulation

In Chapter 3, we presented sampling-based search algorithms for solving the diverse action manipulation problem. However, these algorithms are "flat" in the sense that they search the whole space without attempting to identify and solve smaller parts of the problem first. Manipulation problems lend themselves to more hierarchical algorithms because the planning for each of the manipulation primitives is almost decoupled. For example, the task of picking an object off of a table and placing it on another involves two manipulation primitives: `transit` and `rigid-transfer`. Once we choose the grasp to use, we can plan the `transit` to the grasp and the `rigid-transfer` of the object entirely separately. The caveat is that the grasp must be chosen a priori. The grasp has to be one that the robot can achieve without colliding with obstacles in the world and also one that allows the robot to place the object on the table. The grasp used is a "subgoal" of the problem, and these are not necessarily easy to identify. In this chapter we discuss a hierarchical algorithm for manipulation that attempts to choose good subgoals for each manipulation primitive. We begin by casting the problem as a multi-modal problem. Recall from Section 2.3.5 that multi-modal problems have an extra structure to the configuration space in which configurations can be sorted into modes. We show how to create a hierarchical algorithm that uses this structure to attempt to plan first between large sets of configurations, roughly corresponding to mode families, and then to plan within each set of configurations.

Some of the work in this chapter was previously discussed in Barry et al. [3].

4.1 Manipulation as Multi-Modal Planning

Recall from Section 2.3.5, that a *multi-modal* problem is a tuple $\langle X, \Sigma \rangle$ where X is a configuration space and Σ is a mode space [14]. Each mode $\sigma \in \Sigma$ defines a set of mode-specific constraints that in turn define a set of configurations that satisfy those constraints. A *state* (x, σ) in a multi-modal problem specifies both the current configuration x and the mode σ. For adaptation to continuous-mode problems, such as the DAMA problem, we describe the set of continuous modes as a finite, discrete set of *mode families* [15]. Mode families partition a continuous mode set using a co-parameter that varies to describe each of the different modes. Transitions between modes within a mode family are disallowed; modes must first transition out of the family. An example of a multi-modal problem with continuous modes is shown in Figure 2.15.

The Navigation Among Movable Obstacles (NAMO) problem is an example of a multi-modal manipulation problem. Recall that in this problem there are n movable obstacles and the robot can move by itself or manipulate one movable obstacle at a time. This gives us $n+1$ mode families, one for each movable obstacle and one for the robot moving by itself. As in the example shown in Figure 2.15, the co-parameters of each family are the positions of the stationary obstacles.

We formally define the multi-modal diverse action manipulation problem and then explain why prior techniques in multi-modal planning can fail for some of these problems before presenting our hierarchical algorithm for planning for manipulation.

4.1.1 MM-DAMA Problem

Recall the DAMA problem from Section 3.1. We can automatically create a multi-modal instance of the DAMA problem:

Definition 4.1 (DAMA Problem): Let the DAMA problem be $\langle R, \{O_1, ..., O_n\}, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G\rangle$. For each primitive p_i, we define one *mode family.* A *mode* is an assignment of parameters like grasps or relative configurations to the robot and objects being manipulated and a stationary configuration of the objects not being manipulated. The *MM-DAMA* problem is the DAMA problem augmented by this mode space.

Consider again the Plate World of Section 3.2.1 in which a robot manipulates a plate using the `transit`, `rigid-transfer`, and `push` primitives. This world has three mode families, one for each of the primitives. Within each mode family, there is a set of parameters that we can vary. For instance, in `transit` the robot moves and the plate remains stationary so the co-parameter of the **Transit** family is the plate's configuration. In the **Rigid-Transfer** family, the robot and plate both move but the plate must be grasped, so the co-parameter is the grasp. Similarly, the co-parameter for the **Push** family corresponds to different pushing configurations. If there was also a bowl in the world that could be grasped but not pushed, we would have four mode families: **Transit**, **Rigid-Transfer-Plate**, **Rigid-Transfer-Bowl**, and **Push-Plate**. Note that it is impossible to transition between modes within the same family. For instance, to change the pushing configuration (**Push** family), the robot must pass through **Transit**. Additionally, although we use the mode family formulation because the majority of the modes are continuous, they do not have to be. For instance, we may have a finite number of grasps for the plate. In this case, the **Rigid-Transfer** family has only a finite number of modes.

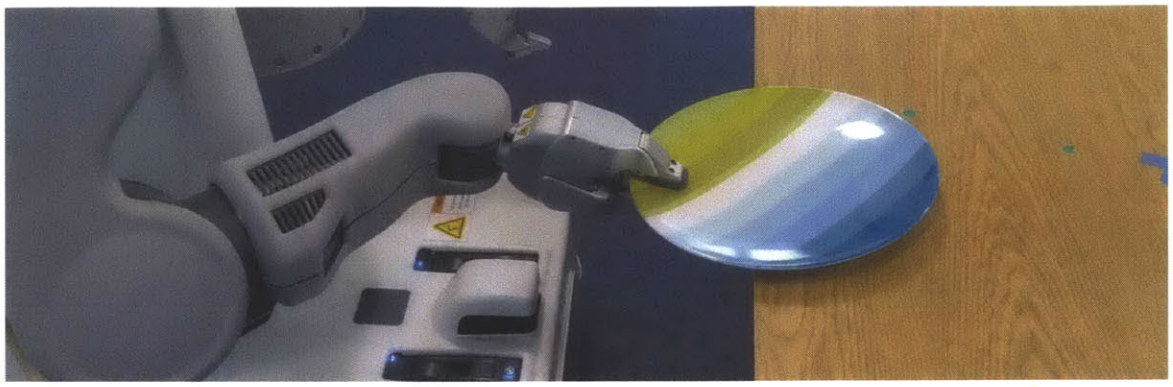


Figure 4.1: We assume the plate can only be grasped at a single position on the table.

4.1.2 Explicit Multi-Modal Planning

Hauser [14] originally proposed the multi-modal framework and also several sampling-based algorithms for solving multi-modal problems. Those algorithms require an extra piece of information specifying not only the modes of the problem but also a high-level mode graph guiding the possible mode transitions. This graph may have transitions that cannot occur in actuality, but it must describe all possible transitions. Recall from Section 2.3.5 that the algorithms rely on the following extension step:

1. Sample an adjacent mode from the mode adjacency graph
2. Sample a transition configuration from the intersection of the two modes
3. Plan a collision free, feasible path within a single mode to reach this transition

However, there are situations in which these algorithms may not be complete. Consider again the problem of pushing an object on a table, but add the additional constraint that there is only one spot on edge of the table at which the robot can grasp the object. This situation is shown in Figure 4.1. The Tool Use domain, for which we present results in Chapter 5, has a constraint like this.

Recall that the primitives in the Plate World are `transit`, `rigid-transfer`, and `push`. The mode families are **Transit**, co-parameterized by the position of the plate, **Rigid-Transfer**, co-parameterized by the grasp used, and **Push**, co-parameterized by the pushing configuration used. The mode adjacency graph for the Plate World is shown in Figure 4.2. The **Transit** and **Push** families are adjacent, as are **Rigid-Transfer** and **Transit**. **Push** and **Rigid-Transfer** are not adjacent because no configuration exists in which the robot and plate are simultaneously in a pushing configuration and a grasping configuration. In other words, the robot must `transit` between any `push` and `rigid-transfer`. Intersections between **Transit** and **Push** are collision free pushing configurations. Intersections between **Transit** and **Rigid-Transfer** are collision free grasping configurations in which the plate is at the special configuration on the edge of the table.

Assume the plate starts on the table. Any path involving rigid transfer must include the following steps:

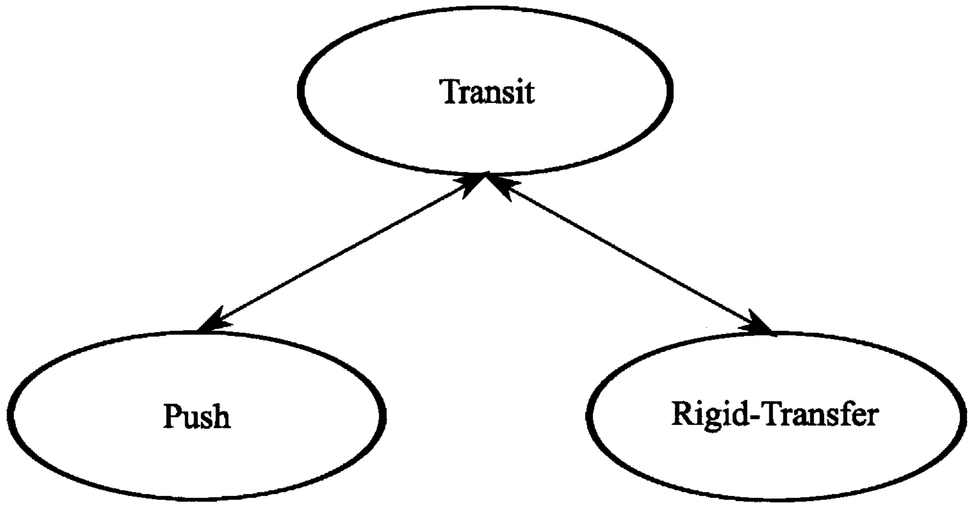


Figure 4.2: The mode adjacency graph in the Plate World.

1. Robot transits to pushing configuration
2. Robot pushes plate to the special configuration on the edge of the table
3. Robot transits to grasping configuration
4. Robot rigid-transfers the plate to the goal

However, we plan single mode pushing paths only when PLANMODESWITCH(x, σ) in Algorithm 2.5 is called with σ=**Push**. In this case, only **Transit** can be returned from ADJACENTMODE because only **Transit** is adjacent to **Push**. Therefore, TRANSITION returns a configuration in the intersection of **Transit** and **Push**, ideally a random position for the plate on the table that can be reached with a single push. Thus all pushing paths are planned such that the plate ends in a random position on the table; the probability that the plate ends at the special point on the edge of the table is zero. Therefore, the algorithm never creates a single mode plan for pushing the plate to that special point on the edge of the table. Thus there is never be a configuration in the tree in which the plate's configuration is at the edge of the table and the transition from **Transit** to **Rigid-Transfer** cannot occur.

Note that we could write the mode graph so that we only sample the plate at the special point on the edge of the table when sampling from the intersection of **Transit** and **Rigid-Transfer** modes. However, this only happens when the single mode planning is for **Transit**, in which the plate cannot move. We require that all points on the table be in the intersection of **Transit** and **Push** so that we can push around obstacles.

An alternative solution to this problem is to define transit-and-push as a single mode family, giving us a "single mode planner" that plans multiple pushes and transits. The intersection of this mode and transit could be configurations with the plate at the special point. However, what if a third primitive needs to be involved? At some point, the single mode planners would become responsible for solving the entire problem because, unlike empty space planners, single mode planners must handle collisions. We discuss this further in Section 4.3.

In most manipulation problems, transfer primitives can only transition to and from transit primitives, but there may be dependency between transfer primitives. Explicit multi-modal planning cannot capture this multi-step dependency. Since our focus is on manipulation problems, we are not able to use the explicit multi-modal algorithms for our problems.

4.2 DARRTH Algorithm

Many solutions to specific manipulation problems implicitly rely on their multi-modal nature. For example, in Section 2.3, we discussed two manipulation problems: re-grasping and NAMO. Both of these problems are subsets of the MM-DAMA problem defined in Section 4.1.1. For re-grasping, there are two mode families: **Rigid-Transfer** and **Transit**. **Transit** is co-parameterized by the location of the object while **Rigid-Transfer** is co-parameterized by the grasp. The NAMO problem with n movable obstacles has $n + 1$ mode families, one for each obstacle, co-parameterized by the robot's grasp and location of the non-moving obstacles, and **Transit**, co-parameterized by the location of the obstacles. The solutions to these problems are usually multi-modal in nature.

Firstly, consider PG-map, which we discussed in Section 2.3.2. Recall that $CG \cap CP$ is the region of configuration space in which the object can sit stably and the robot can grasp it. Therefore, this is the only region in which the robot can change mode families. Since modes within the same family cannot transition to one another, modes can only change within $CG \cap CP$. PG-map first plans a path between connected components of $CG \cap CP$. This gives a high level ordering of mode families since the mode family changes at each component of $CG \cap CP$. However, the reduction property actually allows PG-map to choose the individual *modes* a priori. Each connected component of $CG \cap CP$ may include multiple grasps and multiple object locations, but the reduction property guarantees that any configuration in a connected component of $CG \cap CP$ can reach any other configuration in the same component. The result is that the algorithm can choose any mode in a component of $CG \cap CP$ with which to enter $CG \cap CP$ and any mode in that same component with which to leave it. Thus PG-map can choose grasps and place locations and plan to achieve each of those individually. After planning through the connected components of $CG \cap CP$ the algorithm does not just know the mode family ordering, i.e. first **Transit** then **Rigid-Transfer**, but in fact it knows the mode ordering as well, i.e. `transit` to this grasp then `rigid-transfer` using this grasp. The ability to assign modes a priori is very powerful.

The solutions proposed for NAMO also focus on trying to identify the sequence of modes and then plan for each mode. The only method that does this exactly is the approach proposed by van den Berg et al. [52], which we discussed in Section 2.3.3. van den Berg et al. first choose the obstacles to move, ensuring that there is some actual grasp for the robot, which essentially allows them to choose a mode. After choosing each obstacle and grasp, they plan the individual robot motions.

These algorithms are both examples of an idealized algorithm for solving the MM-DAMA problem:

1. Plan a sequence of modes.

2. Plan for each mode in the sequence individually.

However, PG-map and the version of NAMO proposed by van den Berg et al. both rely on describing connected components of the robot's configuration space to perform these steps. PG-map is able to use the holonomic robot and rigid-grasp aspects of the problem to describe $CG \cap CP$, while van den Berg et al. assume it is possible to analytically describe all connected components of the robot's configuration space. Because we allow non-prehensile manipulation and have potentially high dimensional configuration spaces, we are not able to apply the same leverage to the problem.

When the connected components of the robot's configuration space cannot be described, effective solution methods focus on finding a sequence of mode families in Step 1 rather than a sequence of modes. The difference is that of knowing the correct primitive to apply rather than knowing the exact configuration in which to apply it. For example, if we know that we must find a transition between **Transit** and **Rigid-Transfer**, we know the sequence of mode families. If, however, we know the specific grasp to use in the rigid-transfer, we know the sequence of modes. We also simplify the problem in this way, modifying the algorithm to:

1. Plan a sequence of mode families

2. Plan within each mode family

Moreover, in manipulation, transfer mode families are rarely able to transition to one another because this requires a configuration that could simultaneously be used for two different types of transfer. Therefore, we know that a transit must occur between every type of transfer and the interesting ordering is within the ordering of transfers. Thus, we simplify the algorithm even further:

1. Plan a sequence of transfer mode families

2. Plan each set of transfer and transit trajectories

Throughout the rest of this chapter we assume that the goal only involves a single object, which we refer to as the *goal object*. There may be other objects in the domain that are used as tools. If the domain is uncluttered, the algorithm can be repeated for each object. Otherwise, we can combine our algorithms for manipulation with the NAMO work to first find a candidate order in which to move objects and then plan the manipulations for each object individually.

4.2.1 Finding an Object Path

A common theme in manipulation planning is to plan a path for the objects first and then use information from that plan to guide the search for a full path [36, 37, 44, 46, 47, 48, 52]. In our hierarchical approach to manipulation, we also first try to identify an object path. We then use that path to find a sequence of transfer mode families.

van den Berg et al. [52] give a method for planning a valid object path using two criteria:

1. The path is collision free for the object.

2. The object is manipulable at all points along the path.

van den Berg et al. define "manipulable" to mean that the object is adjacent to the robot's current connected component in configuration space. However, this definition relies on the robot being able to grasp the object at any point of contact, the object being able to sit stably everywhere in its configuration space, and there only being a single object in the domain. Since none of these are necessarily true in our domains, we need to modify the definition of manipulable. Let R_{free} be the the set of configurations for the robot not in contact with any fixed obstacles.

Definition 4.2 (Manipulable): A set of objects are *manipulable from configuration* $(o_{I,1}, ..., o_{I,n})$ *to configuration* $(o_{F,1}, ..., o_{F,n})$ if and only if for some primitive p and robot configurations r_I and r_F, p is applicable to $\big((r_I, o_{I,1}, ..., o_{I,n}), (r_F, o_{F,1}, ..., o_{F,n})\big)$, and the trajectory for the robot returned by p lies entirely within the robot's current connected component of R_{free}.

Now we can update the definition of valid to work with this definition of manipulability:

Definition 4.3 (Valid): A trajectory sequence $\{\tau_0, ..., \tau_l\}$ for a set of objects is *valid* if and only if

1. The sequence is collision free for the objects.

2. For all $i \in \{0, ..., l\}$, the objects are manipulable from $\tau_i(0)$ to $\tau_i(1)$.

Exactly calculating valid paths requires characterizing the connected components of the robot's configuration space and the possible applicable primitives at each configuration. Even were this computationally feasible, it would be very likely harder than solving the original problem. However, we cannot ignore manipulability entirely because the object paths must obey the constraints of the manipulation primitives. For instance, in the Plate World, while the plate is on the table, `push` is the only applicable transfer primitive. The plate cannot rise straight up off the table; it must first be pushed to the edge of the table and, since we are interested in the order of transfer primitives, we want the path for the plate to reflect this constraint.

Thus, we plan object paths using DARRT(CONNECT) but only checking collisions between the objects and the environment. These paths fulfill the first condition of a valid path, but only approximately satisfy manipulability. Although we ensure that there is some applicable primitive, we do not ensure that the corresponding trajectory is within the robot's current connected component.

In general, solving for an object path is much easier than solving for a full path. Although the particular path found for the objects is unlikely to be executable by the robot, the necessity of positioning the objects drives the transfer primitives used so we expect that the sequence of transfer mode families along the object path is informative.

Given an object path, we then convert it to a sequence of transfer mode families and use this to define subgoals.

4.2.2 Manipulation Primitive Subgoals

We use the object path to find a sequence of subgoals. Because we use DARRT(CONNECT) to solve for an object path, the object paths are annotated with the manipulation primitive used. Therefore, we can convert immediately from an object path to a sequence of manipulation primitives. Moreover, since only transfer primitives move an object, the object path consists entirely of transfer primitives.

Therefore, the object path defines a sequence of subgoals $g_0, ..., g_S$ corresponding to the transfer primitives $p_0, ..., p_S$ used along the path. The subgoal g_i is the collision free set of initial configurations for which the primitive p_i is applicable, $X_I(p_i) \cap X_{free}$. We refer to this as a *primitive subgoal.*

Analytically describing $X_I(p_i) \cap X_{free}$ is not necessarily tractable. However, our algorithms do not require an analytical description. For DARRT, we must just be able to decide whether or not a configuration is in $X_I(p_i) \cap X_{free}$. Because we can label trajectories with the primitive that generated them in the empty space planner, when trying to achieve subgoal g_i, we simply check whether the current set of trajectories has any generated by primitive p_i.

When using DARRTCONNECT to achieve a primitive subgoal, we also need to sample from the goal set. To sample from the set $X_I(p_i) \cap X_{free}$, we use the empty space planner to create trajectory sequences from the starting configuration of the current subgoal to a random projected configuration until we find a trajectory τ generated by p_i. If $\tau(0)$ is collision free, we return it as a goal configuration. Otherwise, we sample a new configuration and create another path. Note that we do want to return $\tau(0)$. If we return a later point on the trajectory, we usually must pass through other configurations using p_i (the ones earlier on the trajectory) to reach it. Since these are likely in collision, this removes the usefulness of the goal sampling. The definitions of the empty space planner and projection function set guarantee that a trajectory generated by p_i has some probability of being returned by the empty space planner. In practice, we have found that it is helpful to sample a configuration from the goal set some percentage of the time.

Algorithm 4.1
Input: $X = R \times O_1 \times ... \times O_n$, Configuration space; $\{B_0, ..., B_q\}$, Fixed obstacles; $\{p_0, ..., p_m\}$, Manipulation primitives; x_0, Initial configuration; X_G, Goal set; L, Empty Space planner; $\{f_0, ..., f_j\}$, Projection functions; $\{\rho_0, ..., \rho_n\}$: Distance metrics for each subspace; N, Number of DARRT tries
Output: Trajectory from x_0 into X_G

```
DARRTH(Connect) $(X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G, L, \{f_0, ..., f_j\}, \{\rho_0, ..., \rho_n\})$
1   while no solution has been found
2       $T \leftarrow \emptyset$
        // Only check object collisions
3       $\omega \leftarrow$ DARRT(Connect)$(X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G, L, \{f_0, ..., f_j\}, \{\rho_0, ..., \rho_n\})$
4       $\{p_0, ..., p_k\} \leftarrow$ Transfer primitives on $\omega$
5       $x \leftarrow x_0$, $T \leftarrow \emptyset$
6       for $g \in \{X_I(p_0) \cap X_{free}, ..., X_I(p_k) \cap X_{free}, X_G\}$
7           while no solution and #attempts $< N$
8               $\{\tau_0, ..., \tau_l\} \leftarrow$ DARRT(Connect)$(X, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x, g, L, \{f_0, ..., f_j\}, \{\rho_0, ..., \rho_n\})$
9           if no solution
10              break
11          $T \leftarrow T \cup \{\tau_0, ..., \tau_l\}$
12          $x \leftarrow \tau_i(1)$
13  return $T$
```

4.2.3 DARRTH(Connect) Algorithm

Pseudo-code for DARRTH(Connect) is shown in Algorithm 4.1. We first generate an object path using DARRT(Connect) but only checking collisions for the objects. We then identify the sequence of transfer primitives used along this path and, for each primitive, run DARRT(Connect) until we achieve a configuration in which that primitive is applicable. Lastly, we solve for the final goal set.

Because we only approximate the validity of object paths, we cannot guarantee that the sequence of transfer mode families found from the object path is correct. Therefore, if we are unable to find a solution for a subgoal, we do eventually restart the entire algorithm. Thus we must specify two restart conditions: The time after which to restart a DARRT(Connect) run and the number of DARRT(Connect) runs after which to restart the entire algorithm. We chose these numbers empirically for each problem, but found that we rarely required more than one iteration of DARRTH(Connect).

In Chapter 5, we show that DARRTH(Connect) is usually significantly more efficient than its flat counterpart. In Chapter 6, we prove that DARRTH(Connect) is complete under certain assumptions about the primitives and configuration space.

4.3 DARRT as a Multi-Modal Planner

While we are interested in the multi-modal DAMA problem primarily as guiding the hierarchical algorithm, DARRT itself can be viewed as a multi-modal algorithm. In this case, however, the modes are not the primitives, but the trajectory classes used during planning. At each iteration, DARRT uses a projection function to choose the "mode" (i.e. trajectory class) and then extends in that mode. The single mode planners are, in fact, just different cases within the empty space planner. They are different from previously proposed single mode planners because they do not attempt to find collision free plans, but we can still prove exponential convergence for the algorithm.

Chapter 5

Diverse Action Manipulation Experiments

In this chapter, we present results for the DARRT, DARRTConnect, DARRTH, and DARRTHConnect algorithms from Chapters 3 and 4 on a set of problems from two domains. We show that DARRTConnect is almost always more efficient than DARRT and that the hierarchical algorithms usually perform better than their flat counterparts.

We implemented the algorithms on two domains, the Plate Domain and the Tool Use Domain. The implementation was built on the Open Motion Planning Library (OMPL) [10]. In both domains, we used the Willow Garage PR2 robot [13], planning for one of the robot's seven degrees of freedom arms and its three degree of freedom base for a total of ten degrees of freedom in the robot subspace. Videos of the robot executing in these domains can be found on the website[1].

Some of the results in this chapter were previously given in Barry et al. [2, 3].

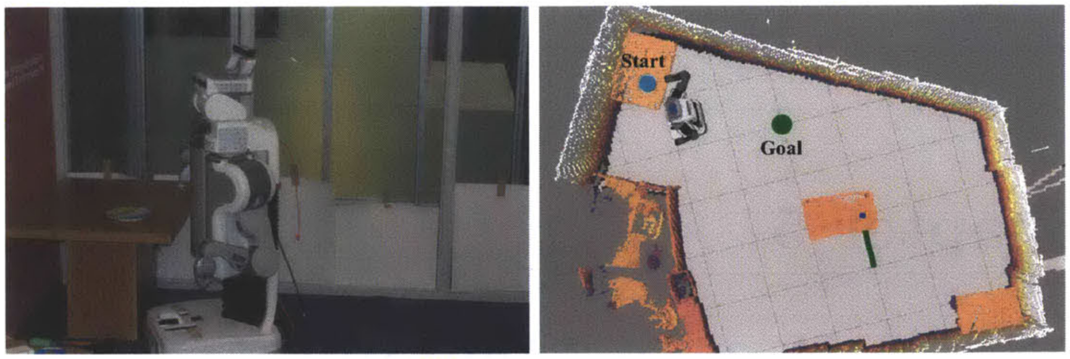
5.1 Plate Domain

The plate domain was discussed in Section 3.1. In this domain, there is a single movable object, the plate. The plate is round so we ignored its yaw dimension in the search. This gives us a fifteen dimensional search space. In this domain, the plate began on a table. The PR2 had to maneuver to the table, push the plate to the edge of the table, grasp the plate, and transfer it to somewhere else in the domain. We solved five different problem instances in this domain, altering the robot's and plate's starting configurations and the plate's goal configuration. These are shown in Figures 5.1 and Figure 5.2.

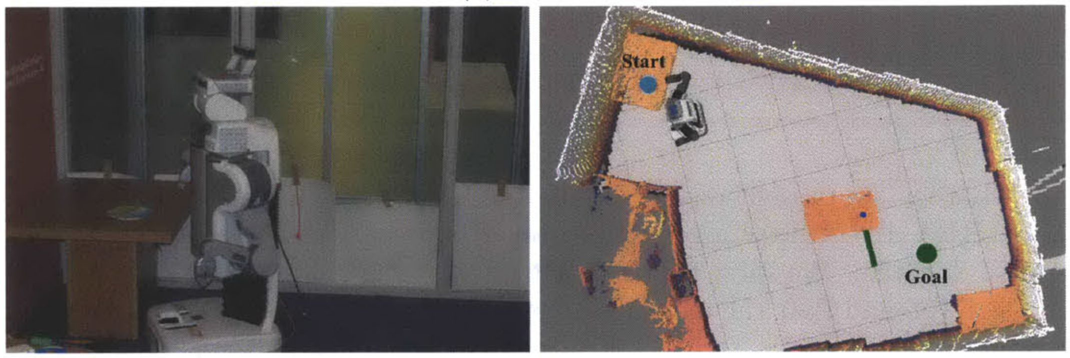
5.1.1 Implementation Details

The primitives we implemented in this domain are similar to the primitives discussed in Chapter 3. Recall from Chapter 3 that for each primitive we need to define the

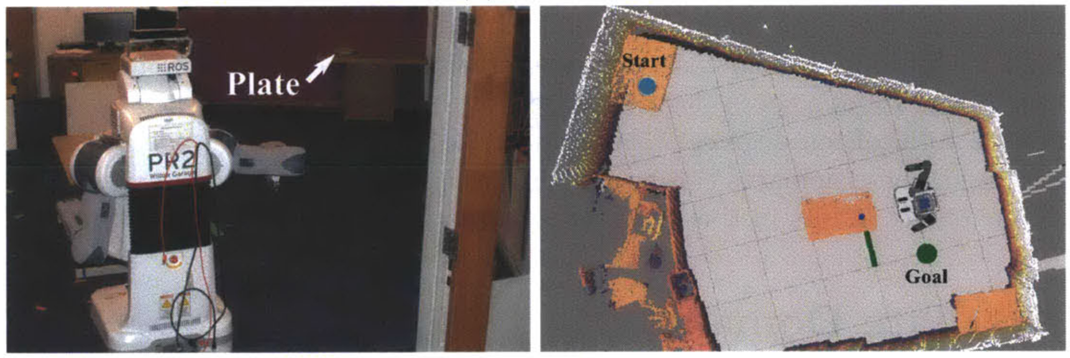
[1] `http://people.csail.mit.edu/jbarry/pr2/darrt`



(a) World 0



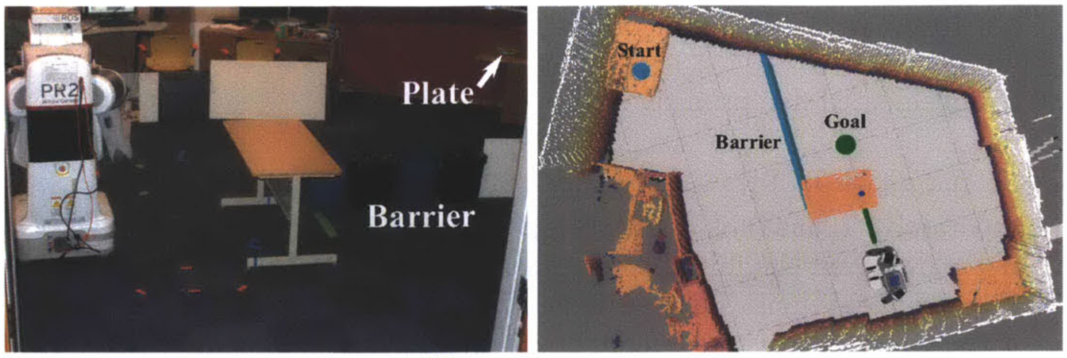
(b) World 1



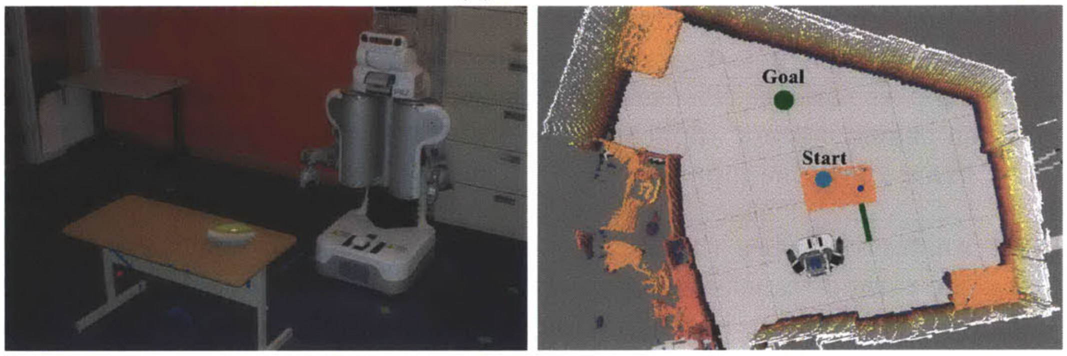
(c) World 2

Figure 5.1: Instances of Worlds 0-2 in the Plate Domain. The goal configuration was specified only for the plate and is shown by the location of the green disc. Goals were a sphere of 2cm radius around the shown location. The left image shows the initial configuration in reality while the right image shows the world in which the algorithm plans. In each problem, the robot and plate began at the shown configuration. The robot had to maneuver to the table, push the plate to the edge of the table, grasp it, and then transfer it to the shown goal location. The collision checking in this world was done on the dense three-dimensional map shown. The origin of this map was under the center table; the y axis is shown in green and the z axis in blue (the x axis is not visible).

following:



(a) World 3



(b) World 4

Figure 5.2: Instances of Worlds 3-4 in the Plate Domain. The goal configuration was specified only for the plate and is shown by the location of the green disc. Goals were a sphere of 2cm radius around the shown location. The left image shows the initial configuration in reality while the right image shows the world in which the algorithm plans. In each problem, the robot and plate began at the shown configuration. The robot had to maneuver to the table, push the plate to the edge of the table, grasp it, and then transfer it to the shown goal location. The collision checking in this world was done on the dense three-dimensional map shown. The origin of this map was under the center table; the y axis is shown in green and the z axis in blue (the x axis is not visible). In World 3, an extra obstacle was added. The robot is not allowed to collide with the barrier.

Trajectory The trajectory that the primitive itself returns as part of its definition.

Applicability The domain of the function.

Collision The collisions for which the primitive disables collision checking during its trajectory.

Propagate The function used by the empty space planner to chain the primitives. This function will return a sequence of trajectories each generated by some primitive. The trajectory returned by the primitive is usually included in this sequence but the **propagate** function *not* the same as the primitive function. Recall that applicability applies only to the trajectory returned by the primitive not to the **propagate** function.

We implemented five primitives:

Transit describes the PR2 robot moving by itself. The base and the arm are moved separately. `Transit` is applicable to any configurations in which the plate has not moved and the initial configuration is not a pushing configuration (in this case, `retreat` must be used before `transit` is applicable). `Transit` returns a trajectory that is a straight line in joint space between the two robot arm configurations and a straight line in Cartesian space between the two robot base configurations. The `transit` function randomly chooses whether to move the arm or base first. `Transit` disables collision checking between the plate and its support surface. The `transit` **propagate** function returns NULL exactly when `transit` is not applicable and otherwise returns the trajectory generated by `transit` on the input.

Rigid-transfer describes the PR2 robot moving the rigidly grasped plate. Like `transit`, rigid-transfer moves the base and the arm separately. `Rigid-transfer` is applicable only to initial and final configurations in which the plate is rigidly grasped in the same grasp. `Rigid-transfer` returns a straight line in joint space between the robot arm configurations and a straight line in Cartesian space between the robot base configurations. The `rigid-transfer` function randomly chooses whether to move the arm or base first. `Rigid-transfer` disables collision checking between the plate and the robot gripper. The `rigid-transfer` **propagate** function returns NULL unless the plate is grasped in the initial configuration. If the plate is grasped, the **propagate** function returns a `rigid-transfer` trajectory that moves the plate to its final configuration using the grasp of the initial configuration.

Approach/Retreat describe the PR2 robot moving its gripper in a straight line. The PR2's base does not move. `Approach` is applicable in situations defined by the `push` and `pickup` primitives and returns a straight line in Cartesian space for the gripper between two gripper configurations. `Retreat` is applicable to any initial configuration in which the robot and plate are in a pushing configuration and any final configuration in which the gripper has moved directly upwards ($+z$) and there is an inverse kinematics solution for every point along the line in Cartesian space from the gripper's initial configuration to its final configuration. `Retreat` returns a straight line in the upwards ($+z$) direction for the gripper in Cartesian space. `Approach` and `retreat` both disable collision checking between the robot's gripper and the plate, the plate and the table, and the robot's gripper and the table. The **propagate** function for `approach` always returns NULL (the **propagate** functions of `push` and `pickup` use the trajectories returned by `approach`). The **propagate** function for `retreat` returns NULL unless the robot and plate are in a pushing configuration and otherwise returns the trajectory generated by `retreat`.

Push describes the PR2 robot pushing the plate on the table. `Push` is applicable to any initial configuration in which the plate is on a support surface and the robot is in two point contact with the plate and any final configuration in which the plate is on a support surface, the robot is still in two point contact with the plate, the robot and plate have moved along the ray connecting the center of the gripper to the center of the plate, and the gripper can move along this ray without moving the robot's base. `Push` returns a straight line in Cartesian space for the gripper from its initial to final configuration. `Push` disables collision checking between the robot's gripper and the plate, the plate and the table, and the robot's gripper and the table. The `push` **propagate** function returns NULL if the plate does not begin on a support surface or the plate does not move. Otherwise, let the input to the `push` **propagate** function be $\big((r_T, o_T), (r_S, o_S)\big)$. The `push` **propagate** function calculates the configuration, o_P, for the plate on the table closest to o_S. If $o_P = o_T$, the function returns NULL. The `push` **propagate** function solves an inverse kinematics problem to find a configuration r_P for the robot in which it can `push` the plate from o_T to o_P. It then solves for moving the gripper in a straight line in Cartesian space without moving the base along the ray connecting the center of the gripper to the center of the plate. If there is no inverse kinematics solution for some point on this ray, only the piece of the ray before the first point at which there is no inverse kinematics solution is returned. Let the final configuration of the robot and plate after pushing be (r'_P, o'_P) ($o'_P = o_P$ if there was an inverse kinematics solution for every point along the ray). The **propagate** function also finds an approach configuration r_A above r_P from which the robot can move the gripper downwards in a straight line. The `push` **propagate** function returns three trajectories: a `transit` to the approach-to-push configuration (this is both a base move and an arm move), an `approach` in the downwards ($-z$) direction into the initial pushing configuration, and a `push` to (r'_P, o'_P).

Pickup describes the PR2 robot closing its gripper to grasp the plate and lifting it straight up. `Pickup` is applicable to any initial configuration in which the robot is grasping the plate and the plate is resting on a support surface and any final configuration in which the gripper and plate have moved directly up from their initial configuration ($+z$) and there is an inverse kinematics solution along every point on the line in Cartesian space between the initial and final configurations of the gripper. `Pickup` returns a straight line for the gripper in the upwards ($+z$) direction. `Pickup` disables collision checking between the robot's gripper and the plate, the plate and the table, and the robot's gripper and the table. The `pickup` **propagate** function returns NULL unless the center of the plate is at the edge of the table. When the plate is at the edge of the table, the **propagate** function chooses a grasp for the plate. We use a discrete set of 200 grasps for the plate, placed evenly around its perimeter. The `pickup` **propagate** function discards any grasps in collision with the table. Let r_g be the robot's configuration in the grasp. The **propagate** function also chooses an approach-to-grasp configuration r_A from which the robot can move the gripper in straight line in Cartesian space with no rotation to r_g. The **propagate** function returns three trajectories: a `transit` (both arm and base) to the approach-to-grasp configuration, an `approach` to the grasp, and a `pickup`.

Examples of each of these primitives can be seen in Figure 5.3. The empty space planner chains these primitives as described in Chapter 3.

The `pickup`, `approach`, `retreat`, and `push` primitives all require the gripper to "move in a straight line in Cartesian space." To accomplish this, we discretize the line in Cartesian space and then solve an inverse kinematics problem for every point on the line. We assume that the gripper's movement between these points is a line in Cartesian space. In practice, we have found that discretizing the line every 3cm works well.

The `pickup`, `approach`, and `retreat` primitives are used to regulate collision checking. In theory, these should be folded into the `rigid-transfer` and `transit` functions. For instance, the **propagate** function for `transit` should retreat when necessary. As a matter of implementation and exposition, it is easier to consider them their own primitives, but they do not require separate projection functions and `pickup` does not become a subgoal for the object path.

To simplify the implementation for this domain, we considered projection functions as defining "active spaces": either the robot moved or the plate moved. Using this view, we also considered the distance function in only one space or the other. This breaks the theoretical guarantees, but worked well in practice because we had only a single object. In Section 5.2, we discuss a domain with multiple objects for which this approach no longer works.

We used two projection functions in this domain corresponding to whether the robot or object should move. Let the input to the function be $x_T = (r_T, o_T)$ and $x_S = (r_S, o_S)$. Then

Transit Returns (r_T, o_S).

Identity Returns (r_S, o_S).

These functions had the same return for a backwards or a forwards extension.

When planning backwards, we allowed the robot to approach and pick up the plate in mid-air. During collision checking, we treated any configuration in which the plate was unsupported as invalid. This allowed us to calculate the grasp during the **propagate** rather than have a `rigid-transfer` projection function. However, it also meant that, with these projection functions, the backwards extension would never place the plate on the table because the probability of sampling the plate on the table is zero. When connecting the backwards tree towards the forward tree, however, we could add configurations in which the plate was on the table to the backward tree.

For the subspace distance metrics we used:

Plate We treated the plate as a six degree of freedom object and added its Euclidean translational distance to the angular distance.

Domain	**DARRT**		**DARRTConnect**		Mode-Specified (s)
	DARRT (s)	DARRTH (s)	DARRTCONNECT (s)	DARRTHCONNECT (s)	(Lower Bound)
World 0	12 (15)	20 (15, 10)	**11** (15)	19 (15, 10)	13
World 1	42 (60)	40 (15, 10)	34 (30)	**28** (15, 10)	14
World 2	142 (90)	48 (15, 10)	98 (30)	**36** (15, 10)	19
World 3	1004 (200)	203 (30, 10)	436 (120)	**61** (30, 10)	36
World 4	411 (90)	214 (60, 3)	**165** (60)	240 (60, 3)	–

Table 5.1: Overall planning time (wall clock time in seconds) averaged over 50 runs in the Plate Domain. DARRTH is the hierarchical algorithm using DARRT as a flat planner and DARRTHCONNECT is the hierarchical algorithm using DARRT-CONNECT as a flat planner. For DARRT(CONNECT) the number shown in parenthesis is the number of seconds after which the algorithm was restarted. For DARRTH(CONNECT) the numbers in parenthesis are (seconds after which each subgoal was restarted, maximum tries per subgoal). For the forward planners, we sampled from the goal configurations 10% of the time. For the bi-directional planners, we added a goal configuration to the backwards tree for every 20 non-goal configurations added to the backwards tree.

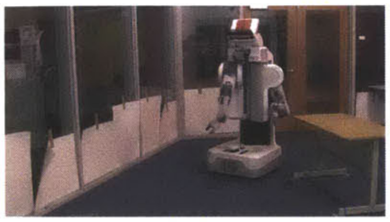
Robot In the arm subspace we used the Euclidean distance that the wrist moved (we ignored the angular distance). In the base subspace, we used the Euclidean distance plus one-tenth the angular distance. The full distance function in the robot subspace was the sum of the distances in the arm subspace and the base subspace.

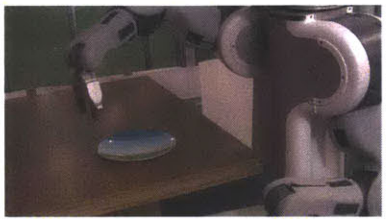
In this domain, rather than always choose the maximum distance in either subspace, we instead first chose the projection function. If we used the Identity projection functions, we used the distance in the object's subspace. If we used the Transit projection function, we used the distance in the robot's subspace.

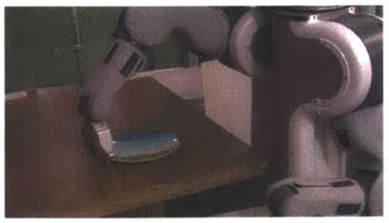
We also needed a method for breaking ties in the distance function. This is because many configurations in the tree will have the object or robot in the same configuration. For instance, the first extension is almost a `transit` for the robot. All of the configurations along this `transit` have the plate in the same configuration. Ties in distance the plate's subspace were decided based on how far the robot would have to travel to its initial contact with the plate. We broke further ties in the object's subspace and ties in the robot's subspace randomly. In theory, all ties should be broken randomly, but the slight increase in greediness in the distance function appeared to be helpful in practice.

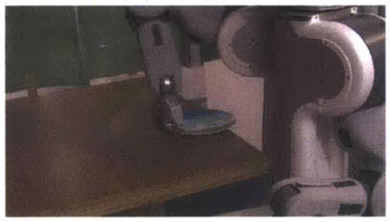
5.1.2 Results

The results in the Plate Domain are shown in Table 5.1. All times are wall clock time in seconds. We did not make any attempt to streamline the set up of these problems so these times include only planning time, not any time required to initialize the algorithm. For the hierarchical domains, the set up time of each sub-problem is not included. The averages are taken over 50 runs; the time for each run is given in Appendix B.1. An execution trace is shown in Figure 5.3.


(a) `Transit`


(b) `Approach`

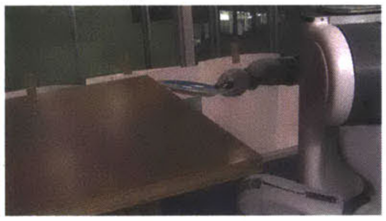

(c) Subgoal 1


(d) `Push`

(e) `Retreat`

(f) `Approach`

(g) `Pickup`


(h) Subgoal 2

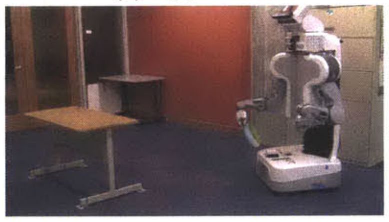

(i) `Rigid-transfer`

Figure 5.3: An example execution in the Plate Domain. (a) The robot `transits` to an approach-to-push configuration. (b) The approach-to-push configuration. The gripper will descend into the pushing configuration using the `approach` primitive. (c) The pushing configuration. By achieving this configuration, the robot achieves the first subgoal. (d) The robot `push`es the plate to the edge of the table. (e) The gripper rises in a straight line using the `retreat` primitive. (f) The robot in the approach-to-grasp configuration. The gripper will move in a straight line to the grasp configuration using the `approach` primitive. (g) The grasp configuration. (h) `Pickup` lifts the plate straight up. This becomes a `rigid-transfer`, achieving the second subgoal. (i) The robot `rigid-transfer`s the plate to its final destination.

For comparison's sake, to show that the planning times are within the realm of reason, we also found an approximate lower bound on planning time using a planner for which we specified the exact modes by hand. For the mode-specified planner, we gave the robot base and arm positions at every mode switch along the path and then used out-of-the-box planners to find a plan for the base or arm alone. We gave the mode-specified planner at least 11 waypoints in each problem: 1) base position at table, 2) arm position at approach to pushing, 3) arm position while pushing, 4) arm position after pushing, 5) arm position at retreat from pushing, 6) base position for picking up the object, 7) arm position for approaching the grasp, 8) arm position in the grasp, 9) arm position after lifting the object, 10) base position at the goal pose, and 11) arm position at the goal pose. Note that the base and arm planners are planning in three and seven dimensional spaces respectively while the DARRT

Domain	DARRTH				DARRTHConnect			
	Object	S1 (Transit)	S2 (Push)	S3 (Rigid-Transfer)	Object	S1	S2	S3
World 0	7	1	7	5	3	1	12	3
World 1	6	1	7	26	3	1	12	12
World 2	6	13	9	21	4	7	16	9
World 3	7	23	8	166	3	9	8	42
World 4	10	24	174	5	4	32	201	3

Table 5.2: Time taken to solve each subgoal. The Object column is the time taken to plan the object path while S1, S2 and S3 are the times taken to plan the intermediate subgoals using the flat planner. (Times are rounded to the nearest second so the sum of the results may not exactly equal the total time reported in Table 5.1.)

and DARRTH variants make an entire plan in fifteen dimensional space. This mode-specified planner is shown in the last column of Table 5.1. We did not run it on World 4 as that was a world designed to show a weakness of the hierarchical planner.

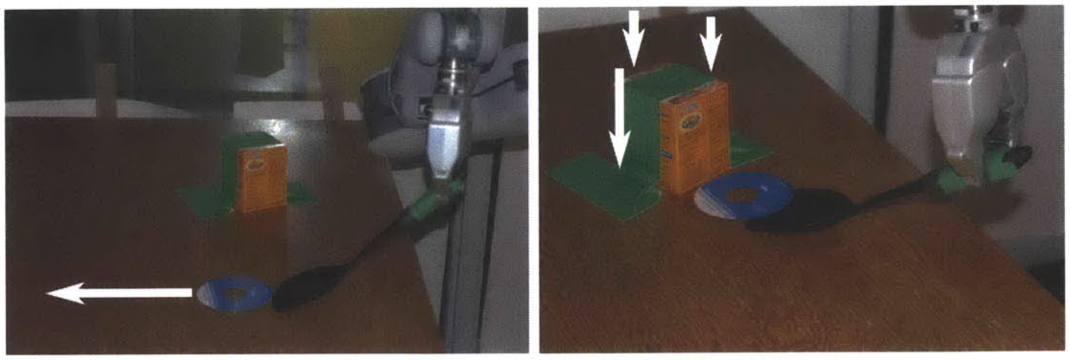
We restarted the sampling-based algorithms after a certain amount of time. This time is shown in parentheses in Table 5.1. For the hierarchical algorithms, this time was used for each sub-problem (i.e. we ran the sub-problem for this amount of time and if there was not a solution, we restarted the sub-problem). This time was picked empirically. We also report the number of tries for each sub-problem before restarting the entire hierarchical algorithm. We discuss this further in Section 5.3.

DARRTH(Connect) identified three subgoals in these domains: `push`, `rigid-transfer` and achieving the actual goal pose. These subgoals were the same for every problem as they all required the same high-level sequence of transfer primitives. The time taken for each subgoal, as well as the time required for planning the object path, is shown for the hierarchical planners in Table 5.2. The subgoals are also marked in the execution trace in Figure 5.3.

We discuss these results farther in Section 5.3.

5.2 Tool Use Domain

The Tool Use domain is an example of using the DARRT(H)(Connect) algorithms for tool use. In this domain, there are two movable objects: a small CD and a spatula. Goals are specified for the CD. The CD is too small and breakable to be grasped by the robot so the only way for the robot to transfer it is to push it or use the spatula. However, if the robot tries to just slide the spatula under the CD, the CD slides away as shown in Figure 5.4a. In order to slide the spatula under the CD, the CD must be resting against a fixed object. In this case, we have a box on the table that the CD can rest against. There are only four positions on the table that allow the CD to rest against the box as shown in Figure 5.4b. The robot must transit to the table supporting the CD, push it to one of these four configurations, transit to the table supporting the spatula, grasp the spatula, return to the table supporting the CD, use the spatula to pick up the CD, and finally transfer the CD to its goal



(a) (b)

Figure 5.4: Using the spatula to lift the CD in the Tool Use Domain. (a) If the CD is not resting against a fixed obstacle, the spatula pushes it instead of lifting it. (b) There are four configurations for the CD on the table where its rests against the box (the one where the CD is and the three configurations shown with white arrows). When the CD rests against the block, the spatula can slide underneath it and lift it up.

location. Because there are only a finite number of configurations for the CD in which the spatula can be used, this is an example of a domain that cannot be solved with explicit multi-modal planning.

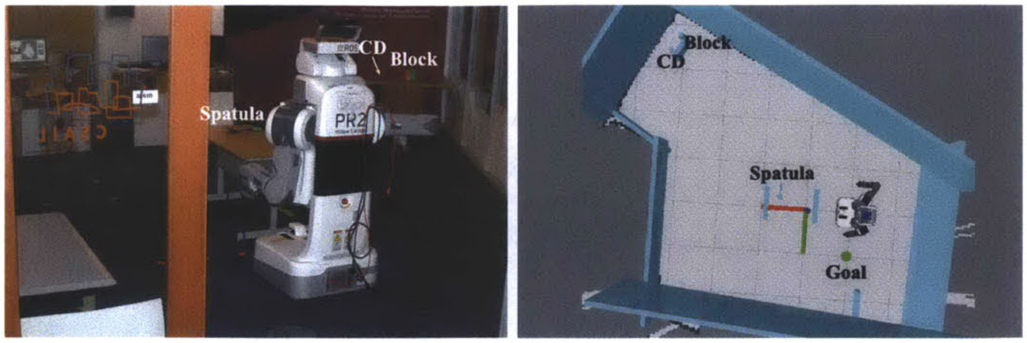
The CD is always parallel to the ground and is round. Therefore, it has only the three translational degrees of freedom. The spatula is a full six degree of freedom object. This gives us a nineteen dimensional configuration space.

We solved four different problem instances in this domain, altering the spatula's starting configuration and the CD's goal configuration. These are shown in Figures 5.5 and 5.6.

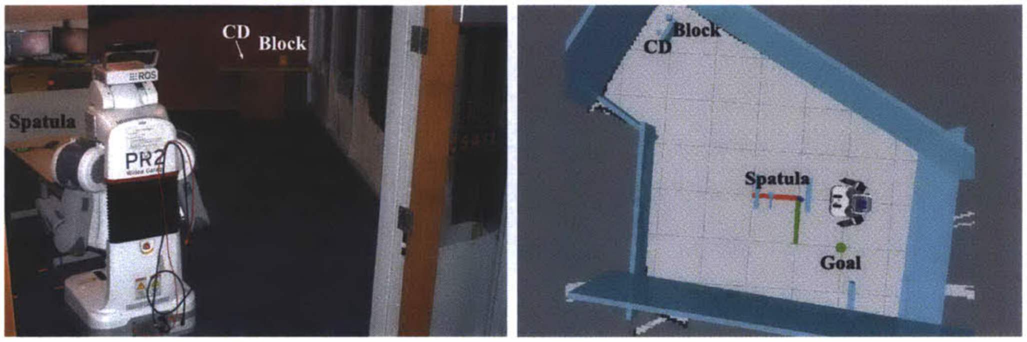
5.2.1 Implementation Details

A configuration in this world is denoted (r, d, s) where r is the configuration of the robot, d is the configuration of the CD, and s is the configuration of the spatula. Recall that for each primitive we need to define the trajectory it returns, its applicability, the collisions for which it disables collision checking, and its **propagate** function. We used seven primitives in this domain:

Transit describes the PR2 robot moving by itself. The base and the arm are moved separately. `Transit` is applicable to any configurations in which the CD and spatula have not moved and the initial configuration is not a pushing configuration (in this case, `retreat` must be used before `transit` is applicable). `Transit` returns a trajectory that is a straight line in joint space between the two robot arm configurations and a straight line in Cartesian space between the two robot base configurations. The `transit` function randomly chooses whether to move the arm or base first. `Transit` disables collision checking between the CD and its support surface and the spatula and its support surface. The `transit`



(a) World 0

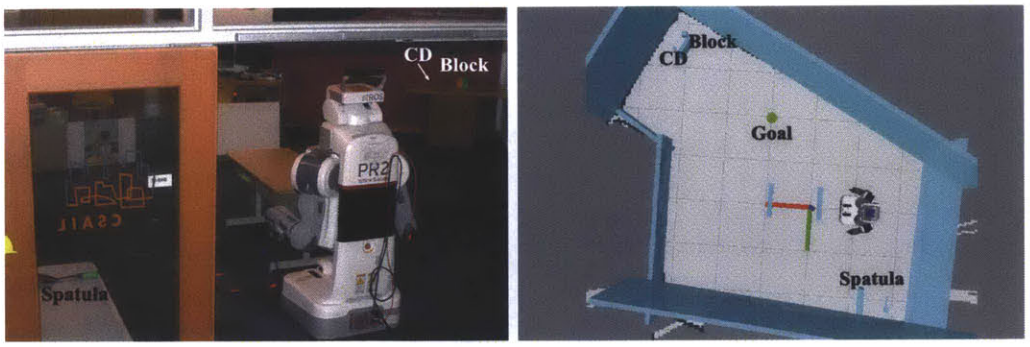


(b) World 1

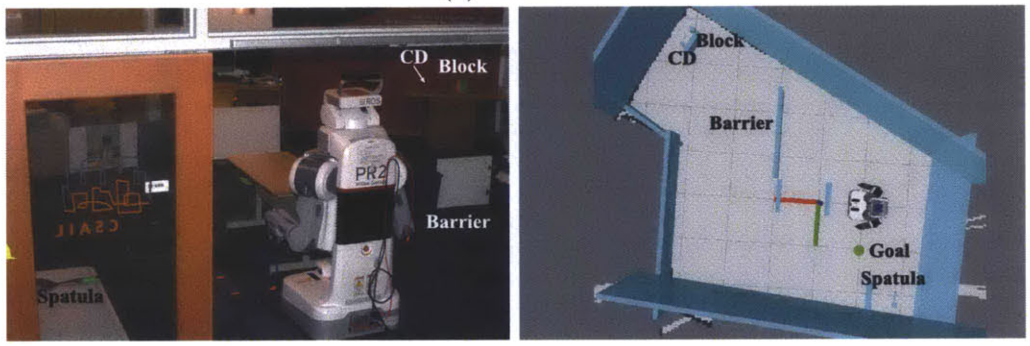
Figure 5.5: Instances of Worlds 0-1 in the Tool Use Domain. The goal configuration was specified only for the CD and is shown by the location of the green disc. Goals were a sphere of 2cm radius around the shown location. The left image shows the initial configuration in reality while the right image shows the world in which the algorithm plans. In each problem, the robot, CD, and spatula began at the shown configuration. The robot had to maneuver to the CD's table, push the CD to the block, maneuver to the spatula's table, grasp the spatula, return to the CD's table, use the spatula to lift the CD, and transfer the CD to its goal configuration. Note that the collision checking in this domain was done against the several rectangular boxes shown in blue rather than using the full collision map of the Plate World. The origin was under the center table; the x axis is shown in red, the y axis in green and the z axis in blue.

propagate function returns NULL exactly when `transit` is not applicable and otherwise returns the trajectory generated by `transit`.

Approach/Retreat describes the PR2 robot moving its gripper in a straight line. The PR2's base does not move. `Approach` is applicable in situations defined by the `push-CD` and `pickup-spatula` primitives, and returns a straight line in Cartesian space for the gripper between two gripper configurations. `Retreat` is applicable to any initial configuration in which the robot and CD are in a pushing configuration and any final configuration in which the gripper has moved directly upwards ($+z$) and there is an inverse kinematics solution for every



(a) World 2



(b) World 3

Figure 5.6: Instances of worlds 2-3 in the Tool Use Domain. The goal configuration was specified only for the CD and is shown by the location of the green disc. Goals were a sphere of 2cm radius around the shown location. The left image shows the initial configuration in reality while the right image shows the world in which the algorithm plans. In each problem, the robot, CD, and spatula began at the shown configuration. The robot had to maneuver to the CD's table, push the CD to the block, maneuver to the spatula's table, grasp the spatula, return to the CD's table, use the spatula to lift the CD, and transfer the CD to its goal configuration. Note that the collision checking in this domain was done against the several rectangular boxes shown in blue rather than using the full collision map of the Plate World. The origin was under the center table; the x axis is shown in red, the y axis in green and the z axis in blue. World 3 has an extra obstacle. The robot was not allowed to collide with the barrier.

point along the line in Cartesian space from the gripper's initial configuration to its final configuration, and the CD and spatula have not moved. `Retreat` returns a straight line in the upwards ($+z$) direction for the gripper in Cartesian space. `Approach` and `retreat` both disable collision checking between the CD and its table, the spatula and its table, the robot's gripper and the nearest object, and the robot's gripper and the nearest table. The **propagate** function for `approach` always returns NULL (the **propagate** functions of `push-CD` and `pickup-spatula` use the trajectories returned by `approach`). The **propagate** function for `retreat` returns NULL unless the robot and CD are in a pushing configuration and otherwise returns the trajectory generated by `retreat`.

Push-CD describes the PR2 robot pushing the CD on the table. `Push-CD` is applicable to any initial configuration in which the CD is on a support surface and the robot is in two point contact with the CD and any final configuration in which the CD is on a support surface, the robot is still in two point contact with the CD, the robot and CD have moved along the ray connecting the center of the gripper to the center of the CD, the gripper can move along this ray without moving the robot's base, and the spatula has not moved. `Push-CD` returns a straight line in Cartesian space for the gripper from its initial configuration to its final configuration. `Push-CD` disables collision checking between the CD and its table, the spatula and its table, the robot's gripper and the CD, and the robot's gripper and the CD's table. The `push-CD` **propagate** function returns NULL if the CD does not begin and end on a support surface, the CD does not move, the spatula does move, or the robot is grasping the spatula. Otherwise, let the input to the `push` **propagate** function be $\big((r_T, d_T, s_T), (r_S, d_S, s_S)\big)$. The `push` **propagate** function calculates the configuration, d_P, for the CD on the table closest to d_S. If $d_P = d_T$, the function returns NULL. The `push-CD` **propagate** function solves an inverse kinematics problem to find a configuration r_P for the robot in which it can `push-CD` the CD from d_T to d_P. It then solves for moving the gripper in a straight line in Cartesian space without moving the base along the ray connecting the center of the gripper to the center of the CD. If there is no inverse kinematics solution for some point on this ray, only the piece of the ray before the first point at which there is no inverse kinematics solution is returned. Let the final configuration of the robot, CD, and spatula after pushing be (r'_P, d'_P, s_T) ($d'_P = d_P$ if there was an inverse kinematics solution for every point along the ray). The **propagate** function also finds an approach configuration r_A above r_P from which the robot can move the gripper downwards in a straight line. The `push-CD` **propagate** function returns three trajectories: the `transit` to the approach-to-push configuration (this is both a base move and an arm move), an `approach` in the downwards ($-z$) direction into the initial pushing configuration, and a `push-CD` to (r'_P, d'_P, s_T).

Rigid-transfer-spatula describes the PR2 robot moving the rigidly grasped spatula without the CD on it. `Rigid-transfer-spatula` is applicable only to initial and final configurations in which the spatula is rigidly grasped in the same grasp and the CD is not balanced on the spatula in the initial or final configurations. `Rigid-transfer-spatula` returns a straight line in joint space between the robot arm configurations and a straight line in Cartesian space between the robot base configurations. The `rigid-transfer-spatula` function randomly chooses whether to move the arm or base first. `Rigid-transfer-spatula` disables collision checking between the robot gripper and the spatula and the CD and its table. The `rigid-transfer-spatula` **propagate** function returns NULL exactly when `rigid-transfer-spatula` is not applicable and otherwise returns the trajectory generated by `rigid-transfer-spatula`.

Transfer-CD describes the PR2 robot moving the CD balanced on the spatula. In order that the CD remain balanced on the spatula, the paddle of the spatula must remain parallel to the ground. `Transfer-CD` is applicable to any initial configuration in which the CD is balanced on the spatula and any final configuration in which the CD is balanced on the spatula and the robot is grasping the spatula using the same grasp as in the initial configuration. `Transfer-CD` returns a trajectory that is a straight line in Cartesian space to the robot's final configuration for the arm and a straight line in Cartesian space to the robot's final configuration for the base. As with `transit` and `rigid-transfer-spatula`, the order of the arm and base moves is randomized. `Transfer-CD` disables collision checking between the spatula and the CD and the robot's gripper and the spatula. The `transfer-CD` **propagate** function returns NULL exactly when `transfer-CD` is not applicable and otherwise returns the trajectory generated by `transfer-CD`.

Pickup-spatula describes the PR2 closing its gripper to grasp the spatula and lifting it straight up. `Pickup-spatula` is applicable to any initial configuration in which the robot is grasping the spatula, the CD is in one of the four configurations in which the spatula can be used, and the spatula is resting on a support surface and any final configuration in which the spatula and gripper have moved directly upwards ($+z$) along a line in Cartesian space for which there is an inverse kinematics solution at every point and the CD has not moved. It returns a straight line for the gripper in the upwards ($+z$) direction. `Pickup-spatula` disables collision checking between the spatula and its table, the CD and its table, the robot's gripper and the spatula, and the robot's gripper and the spatula's table. The `pickup-spatula` **propagate** function always returns NULL unless the spatula is resting on the table in the initial configuration and grasped in the final configuration and the CD is in one of the four configurations in which the spatula can be used in the final configuration. The `pickup-spatula` **propagate** function returns up to seven trajectories: a `transit` (both arm and base) to an approach-to-push configuration, an `approach` in the downwards ($-z$) direction to the pushing configuration, a `push-CD` to push the CD to its final configuration, a `retreat` from pushing, a `transit` (both arm and base) to the approach-to-grasp, an `approach` to the grasp, which is a straight line in the downwards ($-z$) direction for the gripper in Cartesian space, and a `pickup-spatula`. If the CD is already in the correct position, the **propagate** function will not include the pushing trajectories. If it is not possible to push the CD all the way to its final configuration without moving the base, the **propagate** function will only return up to the pushing trajectory. We defined only one grasp for the spatula so the **propagate** function always uses that grasp.

Use-spatula describes the PR2 using the spatula to lift the CD off the table. This primitive is shown in Figure 5.7. `Use-spatula` is applicable to initial configurations in which the robot is grasping the spatula, the CD is at one of the four positions in which the spatula can be used, and the spatula is above the CD at an angle and final configurations in which the CD is balanced on the spatula and has moved straight upwards ($+z$). The `use-spatula` primitive returns: a straight line in the downwards ($-z$) direction for the gripper in Cartesian space with the spatula held at an angle, an approach along the line towards the CD in Cartesian space that slides the spatula under the CD, an angled trajectory that results in the paddle of the gripper parallel to the floor and the CD resting entirely on the paddle, and a lift that lifts the spatula and CD together. The primitive is only applicable when there are inverse kinematic solutions for all of these movements. `Use-Spatula` disables collision checking between the robot's gripper and the spatula, the robot's gripper and the CD's table, the spatula and the CD's table, the spatula and the CD, the spatula and the block, the robot's gripper and the CD, the robot's gripper and the block, the CD and its table, and the CD and the block. The `use-spatula` **propagate** function returns NULL unless the CD begins on a support surface and ends on the spatula. Otherwise, the **propagate** function returns up to nine trajectories: a `transit` (base and arm) to an approach-to-pushing configuration for the CD, an `approach` in the downwards ($-z$) direction to the pushing configuration, a `push-CD` to one of the four configurations for the CD from which the spatula can be used (the **propagate** function chooses one randomly), a `retreat` from the push, a `transit` (base and arm trajectories) to an approach-to-grasp configuration for the spatula, an `approach` in the downwards ($-z$) direction to the grasping configuration, a `pickup-spatula`, a `transit` (base and arm trajectories) to the use-spatula configuration, and finally the `use-spatula` trajectory. All nine trajectories are only returned if necessary. If the CD is already in a location at which the spatula can be used, there will be no pushing trajectories. Similarly, if the robot is holding the spatula, there will be no pushing or picking trajectories. If the `push-CD` fails to push the CD all the way to its location, only those trajectories are returned.

Note that the long **propagate** functions of `pickup-spatula` and `use-spatula` have trajectory sequences that are the returns of other **propagate** functions. For instance, the first four trajectories on the `pickup-spatula` **propagate** function return are the same as the trajectories on `push-CD`. Therefore, implementing these functions is not difficult.

As in the Plate Domain, the `pickup-spatula`, `use-spatula`, `approach`, and `retreat` primitives are used to regulate collisions. In theory, these should be folded into the `rigid-transfer-spatula`, `transfer-CD`, and `transit` functions. As a matter of implementation and exposition, it is easier to consider them their own primitives, but they do not require separate projection functions and `pickup` and `use-spatula` do not become a subgoal for the object path.

We used four projection functions in this domain. On input $x_T = (r_T, d_T, s_T)$ and $x_S = (r_S, d_S, s_S)$:

Transit returns (r_S, d_T, s_T) if the spatula and CD are resting on the table in d_T and s_T. Otherwise, it chooses a random configuration, d_P, for the CD on its initial

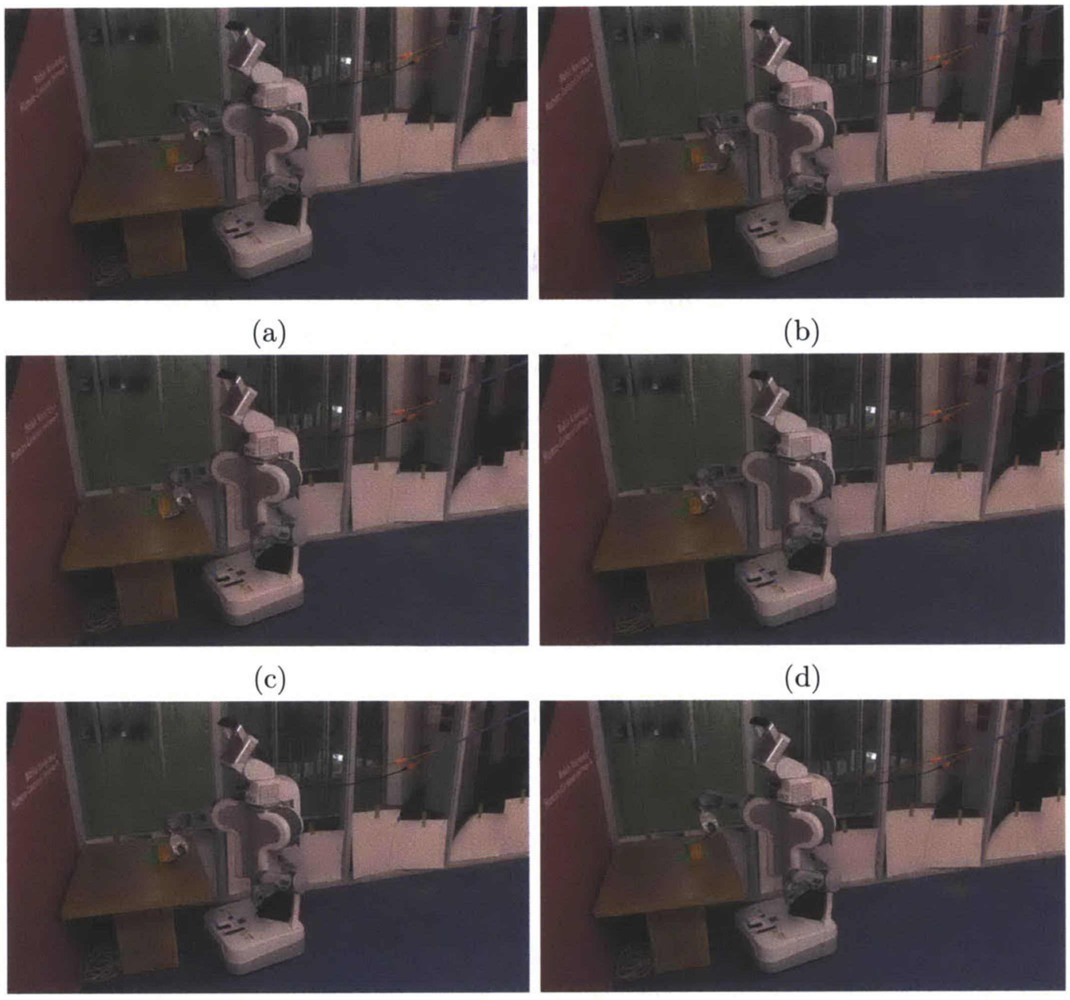

(a) (b) (c) (d) (e) (f)

Figure 5.7: The `use-spatula` primitive. (a)-(b) The first trajectory is straight down with the spatula held at an angle. (c)-(d) The second trajectory slides the spatula under the CD with the spatula still held at an angle. (e) The third trajectory angles downward so that the paddle of the spatula is parallel to the ground. (f) The final trajectory lifts the CD.

table and returns (r_S, d_P, s_0) where s_0 is the configuration of the spatula in x_0, the starting configuration of the search.

Push chooses a random configuration for the CD on its initial table, d_P. It returns (r_S, d_P, s_0) where s_0 is the configuration of the spatula in x_0, the starting configuration.

Rigid-transfer-spatula first chooses a position, d_P, for the CD in one of the four configurations from which the spatula can be used. If d_T is one of these configurations it uses that. It then finds an inverse kinematics solution for the robot, r_g, at which it is grasping the spatula and the spatula is in s_S (there is only one grasp for the spatula). It returns (r_g, d_P, s_S).

Transfer-CD chooses a configuration for the spatula, s_g, for which the CD is at d_S (note that d_S is three dimensional so we can assume the angle is parallel to the floor) and then a configuration for the robot, r_g, to grasp the spatula at s_g. It returns (r_g, d_S, s_g).

These functions had the same return for a backwards or a forwards extension.

The spatula is projected to its initial configuration in the Transit and Push projection functions because there is no `place-spatula` primitive. Therefore, there is no way to put the spatula back down once the robot is holding it. Putting it back in its original configuration allows the backwards extension to possibly connect to the forward tree even if the extension includes the `pickup-spatula` primitive.

Similarly, the projection functions and **propagate** functions are such that the spatula cannot be grasped until the CD is properly positioned. This is because once the robot is holding the spatula, it can no longer push the CD.

It is possible, especially when extending backwards, that the empty space planner does fail to create any plan. In this case, the algorithm adds no new configurations to the tree and moves on to the next iteration.

For the subspace distance metrics we used:

Spatula The Euclidean translational distance plus one half the angular distance.

CD The Euclidean translational distance.

Robot In the robot's base space, we used the two-dimensional Euclidean translation plus one-tenth the angular distance. In the arm space, we used the average change in joint angle. The overall distance function returned the maximum of these two distances.

Our overall distance function was the maximum distance in any subspace. We broke ties randomly.

We found that generating primitive goals for DARRTHConnect (the only algorithm that generated a large number of primitive goals) in this domain was a substantial factor in running time. We addressed this in two ways:

Firstly, we only added new goal configurations to the backwards tree once for every 500 non-goal configurations added to the backwards tree. Because we add complete paths, the trees usually had tens of thousands of configurations. This resulted in adding tens of goal configurations (usually between 20 and 100). Because we only add valid goal configurations, this was easily sufficient for solving the problem.

Secondly, we also checked if configurations added to the forwards tree satisfied the goal condition. This is because we may add a configuration that uses the goal primitive without connecting to the backwards tree. For instance, when achieving `push-CD`, the backwards tree grows backwards from some instances of `push-CD`. However, the forwards tree may add a different instance of `push-CD` and that also qualifies as achieving the goal.

These modifications were also present when running DARRTConnect, but likely had little effect on running time.

Domain	DARRT		DARRTConnect	
	DARRT (s)	DARRTH (s)	DARRTCONNECT (s)	DARRTHCONNECT (s)
World 0	206 (45)	51 (15, 100)	39 (30)	**31** (15, 100)
World 1	1072 (200)	130 (45, 100)	101 (45)	**65** (30, 100)
World 2	773 (200)	127 (45, 100)	514 (200)	**41** (45, 100)
World 3	8282 (500)	312 (60, 100)	10166 (300)	**162** (45, 100)

Table 5.3: Overall planning time (wall clock time in seconds) averaged over 50 runs in the Tool Use domain problems. DARRTH is the hierarchical algorithm using DARRT as a flat planner and DARRTHCONNECT is the hierarchical algorithm using DARRTCONNECT as a flat planner. For DARRT(CONNECT) the number shown in parenthesis is the number of seconds after which the algorithm was restarted. For DARRTH(CONNECT) the numbers in parenthesis are (seconds after which each subgoal was restarted, maximum tries per subgoal). For the forward planners, we sampled from the goal configurations 10% of the time. For the bi-directional planners, we added one goal configuration to the backwards tree for every 500 non-goal configurations added.

Domain	DARRTH					DARRTHConnect				
	Object	S1	S2	S3	S4	Object	S1	S2	S3	S4
World 0	10	6	14	2	20	4	4	10	1	11
World 1	13	6	82	6	23	5	5	35	9	11
World 2	12	5	103	5	1	5	4	27	5	1
World 3	13	64	175	33	27	7	38	58	33	26

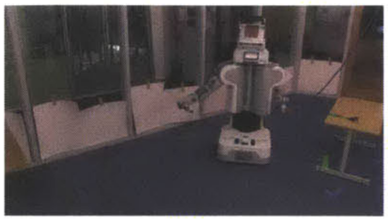
Table 5.4: Time taken to solve each subgoal. The Object column is the time taken to plan the object path while S1, S2, S3, and S4 are the times taken to plan the intermediate subgoals using the flat planner. (Times are rounded to the nearest second so the sum of the results may not exactly equal the total time reported in Table 5.3.)

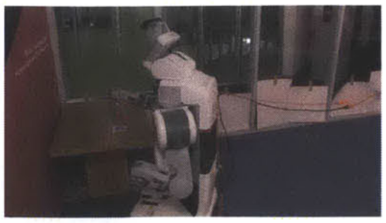
5.2.2 Results

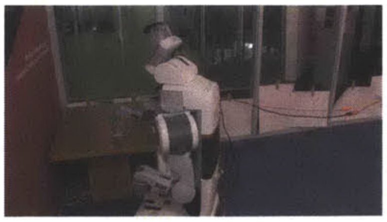
The results in this domain are shown in Table 5.3. All times are wall clock time in seconds. As in the Plate Domain, these times do not include set up times. The averages are taken over 50 runs; the time for each run is given in Appendix B.2. An execution trace is shown in Figure 5.8.

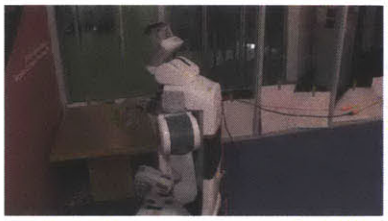
DARRTH(CONNECT) identified four subgoals in these domains: `push`, `rigid-transfer-spatula`, `transfer-CD`, and achieving the actual goal pose. These subgoals were the same for every problem as they all required the same high-level sequence of transfer primitives. The time taken for each subgoal, as well as the time required for planning the object path, is shown for the hierarchical planners in Table 5.4. The subgoals are also marked in the execution trace in Figure 5.8.

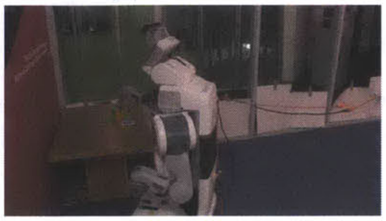
We restarted the sampling-based algorithms after a certain amount of time. This time is shown in parentheses in Table 5.3. For the hierarchical algorithms, this time

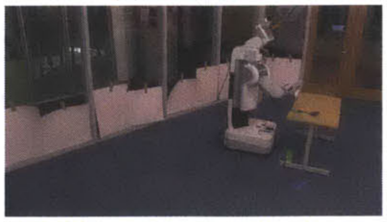

(a) `Transit`

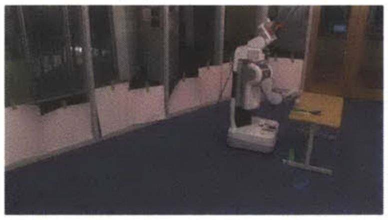

(b) `Approach`

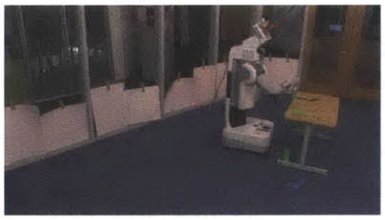

(c) Subgoal 1

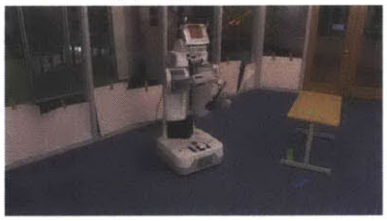

(d) `Push-CD`

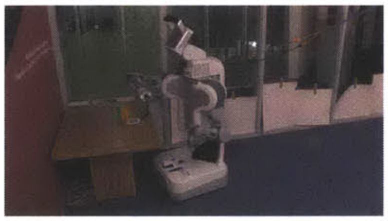

(e) `Retreat`

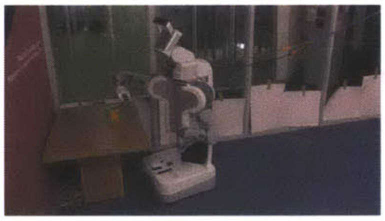

(f) `Approach`


(g) `Pickup-spatula`


(h) Subgoal 2


(i) `Rigid-transfer-spatula`


(j) `Use-spatula`


(k) Subgoal 3

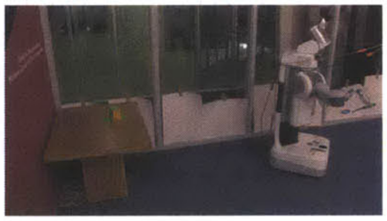

(l) `Transfer-CD`

Figure 5.8: An example execution in the Tool Use Domain. (a) The robot `transits` to an approach-to-push configuration. (b) The approach-to-push configuration. The gripper will descend into the pushing configuration using the `approach` primitive. (c) The pushing configuration. By achieving this configuration, the robot achieves the first subgoal. (d) The robot uses `push-CD` to push the CD to the block. (e) The gripper rises in a straight line using the `retreat` primitive. (f) The robot in the approach-to-grasp configuration for the spatula. The gripper will move in a straight line to the grasp configuration using the `approach` primitive. (g) The grasp configuration. (h) `Pickup-spatula` lifts the spatula straight up. This becomes a `rigid-transfer-spatula`, achieving the second subgoal. (i) The robot uses `rigid-transfer-spatula` to transfer the spatula to the table with the CD on it. (j) The first configuration on `use-spatula`. See Figure 5.7 for the execution of `use-spatula`. (k) The transition from `use-spatula` to `transfer-CD`. This is the third subgoal. (l) The robot uses `transfer-CD` to move the CD to its final goal location.

was used for each sub-problem (i.e. we ran the sub-problem for this amount of time and if there was not a solution, we restarted the problem). This time was picked empirically. We also report the number of tries for each sub-problem before restarting the hierarchical algorithms. We discuss this further in Section 5.3.

5.3 Discussion

The bi-directional planner is almost always faster than the forward planner, as expected, and the hierarchical planners are faster than their flat counterparts. Except in World 4, which we will discuss in detail, DARRTHConnect is not unreasonably slower than the mode-specified planner in the Plate Domain, even though the latter has much more information.

5.3.1 Problem Difficulty

The problems are numbered roughly according to their difficulty. In the Plate Domain, the difficulty is governed by how complicated the path to configurations at which the robot can manipulate the plate is and then how complicated the path from those configurations to the goal configuration is. In World 0, both are simple. In World 1, the robot begins at the table with the plate, but has to navigate around the central table while transferring the plate to the goal location. In World 2, the robot has to navigate around the central table both to originally reach the plate and then also to reach the goal location. In World 3, we remove one of the routes around the central table. Finally, in World 4, the robot must move all the way around the table to pick up the plate, but can reach the plate from the side of the table closest to the initial robot configuration. This world was set up to show a weakness of the hierarchical planner and is discussed further in Section 5.3.3.

In the Tool Use world, the robot and CD always begin in the same initial configuration so the difficulty is mostly governed by the path from the table supporting the CD to a configuration from which the robot can pick up the spatula. In World 0, there are many collision free paths that the empty space planner might return from a configuration near the table supporting the CD to the table supporting the spatula and back. In World 1, there are a small number of collision free paths with the robot to the left of the table that the empty space planner can return. In Worlds 2 and 3 there are no such paths. In these worlds, the path to and from the table with the spatula on it must involve multiple calls to the empty space planner to move around the central table. World 3 is harder than World 2 because one possible route around the central table has been removed. Additionally, the CD goal configuration for World 3 also requires the robot to navigate around the central table.

5.3.2 Forward vs Bi-Directional Planners

The RRTConnect algorithm is usually preferred in practice to the RRT algorithm for its increase in efficiency. Our results confirm this for the DARRT(H)Connect planners. The bi-directional planners are nearly always faster than their forward counterparts especially on more complicated problems.

The Tool Use Domain Worlds 2 and 3 appear somewhat anomalous in this regard. The DARRT and DARRTCONNECT times are close in World 2 and DARRT is faster in World 3. In fact, we suspect that DARRT and DARRTCONNECT should be closer in time in the Tool Use Domain than they are in the Plate Domain because the distance function is less informative about how far the robot will actually travel. Therefore, when connecting the trees, DARRTCONNECT is less likely to pick a configuration to which there will be a collision free path because it is less likely to pick a configuration to which there is a short path. Using one distance function during sampling and another, more informative one, when connecting the two trees could be an interesting direction of future research.

However, in Tool Use World 1, DARRTCONNECT does substantially better than DARRT. In this world, there were a few configurations for the robot in which it could reach the spatula on the table without moving around the table. The bi-directionality of DARRTCONNECT made it easier for it to find these configurations when connecting the two trees. Often a forwards path from the CD table would contact the table supporting the spatula at a configuration from which the robot could reach the spatula. The DARRTCONNECT planner would always try to pick up the spatula from these configurations (the inverse kinematic solvers had a bias towards solutions that did not move the base) when extending the backwards tree towards the newly added configuration. The forwards planner, however, might add these configurations to the tree but then rarely choose them as the nearest configuration in the tree to a sample. The forwards planner usually found solutions that required navigating around the table.

In Tool Use World 3, both algorithms took much longer than the reset time they were given. In this world, it is likely that solving the problem within 300 or 500 seconds was so rare that any advantage the bi-directionality could give was masked by the improbability of solving the problem in the allowed time.

5.3.3 Flat vs Hierarchical Planners

In most cases, the hierarchical planners outperform their flat counterparts. The notable exceptions are the Plate Worlds 0 and 4.

Plate World 0 is a trivial domain for which a single goal sample can solve the problem depending on the path chosen by the empty space planner. Although the four sub-problems are also easy, solving them individually takes more time than solving the single problem.

World 4 was chosen to illustrate a weakness of the hierarchical planners. In this domain it is possible to reach a `push` configuration from the wrong side of the table. In this case, the robot's starting configuration when solving the second subgoal (pick up the plate) is actually worse than the original starting configuration. It is clear from the amount of time taken by DARRTH(CONNECT) on the second subgoal (pick up the plate) that it fell into this trap often.

Except in the world designed to be difficult for them, the hierarchical algorithms consistently out-perform their flat counterparts and DARRTHConnect plans only a factor of about two slower than the mode-specified planner. The leverage is owing primarily to three factors:

Targeted Restarts As is common with sampling-based planners, we restart DARRT(H)(Connect) after a specified time period. With DARRT(Connect), we can do no better than restarting from the starting configuration. With DARRTH(Connect), however, we restart from the most recent subgoal. For example, lifting the plate off of the table is a difficult problem. If the move from the table to the goal configuration is also difficult, DARRT(Connect) might restart after having solved for lifting up the plate but before being able to transfer it to its goal location. DARRTH(Connect), however, by its automatic choice of good subgoals, restarts from the configuration in which the plate has been lifted.

Nearest Neighbor Calculation We used a linear search for nearest neighbor. As the trees grew, this search became a limiting factor in efficiency. In the Tool Use domain, this was less noticeable because we used a very fast distance function. However, the distance function we used for the Plate Domain involved a forward kinematics calculation. The flat searches necessarily have larger trees so this calculation could take a substantial amount of time. This was especially noticeable in World 3 where the final transfer of the plate to the goal was a difficult task. Finding a path that lifted the plate off the table creates a large tree. In the flat planner, these configurations were all still in the tree when finding a transfer to the plate's goal position. The hierarchical algorithms, however, started searching for the transfer with trees with only a single configuration.

Focused Projection Functions The projection functions also limited the available primitives for the hierarchical planners during some sub-problems. This was only true for the Tool Use Domain, as we only had two projection functions in the Plate Domain. Consider achieving the `transfer-CD` primitive from a configuration in which the robot is holding the spatula. When the `transit` projection function is called, the resulting configuration with the spatula back on the table in its initial configuration cannot be achieved from any configuration in the tree because there is no way to place the spatula back down. Therefore, the robot will never try to `transit` (the `transit` projection function might still be called but no configurations will be added to the tree). Similarly, once the CD has been lifted, the robot will never `transit` or `rigid-transfer`. This saves a large number of inverse kinematics calculations.

One possible weakness of the hierarchical planner not shown here is that it can create impossible subgoals for itself. For instance, in the Tool Use Domain, it could push the CD to a side of the box that was not reachable with the spatula. It is for this reason that we have to restart the entire hierarchical planner and not just each sub-problem.

5.3.4 Reset Times

We chose the reset times empirically based on how difficult the domain appeared and usually a few test trials. While we tried to choose fair times, we purposely did not optimize for the reset time. In practice, we would not run the same problem over and over again to choose a good reset time.

We usually gave DARRT and DARRTConnect longer reset times than DARRTH and DARRTHConnect. For the hierarchical algorithms, the reset time is for each sub-problem while for the flat searches the reset time is for the entire search. Therefore, it makes sense that we have to give DARRT and DARRTConnect more time before restarting. However, there is the concern that all that matters is the number of restarts; that each algorithm actually solves the problem quickly sometimes and not at all other times. In this case, giving DARRT and DARRTConnect longer reset times would bias the results against them unfairly. An analysis shows that this does not appear to be the case.

The ratio of times between the flat planner and hierarchical planner is too high for it to be caused by the reset time in the Tool Use Domain Worlds 2 and 3. Similarly, the ratio for the running times of DARRTConnect to DARRTHConnect is too high to be entirely due to the reset time in the Plate Domain Worlds 2 and 3 while the ratio for the running times for DARRT to DARRTH is too high in the Tool Use Domain World 1.

We can check that giving the flat and hierarchical planners the same reset time would not nullify the results in the remaining problems using the full trial results given in Appendix B. Consider, for instance, DARRT and DARRTH in Plate Domain World 2. In this world, DARRTH is given a reset time of 15 seconds while DARRT had a reset time of 90 seconds. These results show that DARRT was able to solve the problem in 15 seconds only in 7 trials of the 50. Each trial requires, on average one restart at 90 seconds. Additionally, the 15 second planner only solves the problem approximately every 7 times ($50/7 = 7\frac{1}{7}$). Therefore, the remaining six solutions found for the 90 second planner would also trigger a restart. This indicates an average of 13 restarts for the 15 second planner, for a running time of about 195 with 15 second restarts. The same type of analysis shows that decreasing the running time of DARRT to 30 seconds in Plate Domain World 3 does not improve its performance or DARRT to 15 seconds in Tool Use Domain World 0. (It is harder to verify that reducing the reset time of DARRTConnect to 30 seconds from 45 seconds in Tool Use Domain World 1 might not improve its performance because those two reset times are so close.)

We should point out here that it is easier to choose good reset times for the hierarchical planner because the sub-problems are short. We can almost always be sure that 60 seconds will be enough to solve the problem. In easy domains 15 seconds is enough. This is reflected in that the reset times for the hierarchical planners are the same across three problems in the Plate Domain.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Exponential Convergence of the Search Algorithms

Recall that we defined exponential convergence in Definition 2.2: A sampling algorithm *converges exponentially* if the probability that the algorithm returns a solution when one exists is $1 - O(2^{-ak})$ after k samples for some positive constant a.

In Chapter 3, we presented the DARRT algorithm for manipulation and in Chapter 4, we presented the DARRTH(CONNECT) algorithm. In this chapter, we prove that, under some assumptions about the configuration space, empty space planner, projection functions, and distance function, the DARRT and DARRTH(CONNECT) algorithms are both exponentially convergent.

The proof that the DARRT algorithm converges exponentially is in two pieces. We first provide a high-level proof that makes some strong assumptions about the configuration space, empty space planner, projection functions, and distance function. We then show that there are manipulation domains with interesting manipulation primitives in which these assumptions hold. The proof that the DARRTH(CONNECT) algorithm is exponentially convergent is more straightforward.

6.1 Exponential Convergence of the DARRT Algorithm

In proving exponential convergence of the DARRT algorithm, we use a more general version of the algorithm than the one given in Chapter 3 so that we can more easily state the assumptions we make. This version of the DARRT algorithm takes the distance function and the method for sampling the space as inputs rather than using the choices we made in Chapter 3. We can recover the algorithm of Chapter 3 by using those choices as inputs to the algorithm discussed here. We begin by formally defining the input for the DARRT algorithm, and then we show that it is exponentially convergent.

6.1.1 DARRT Input

The DARRT algorithm is designed to search in a product space of a robot and a set of objects. As we discussed in Section 3.2.3, we need distance functions in each subspace as well as a distance function for the full product space. In fact, the subspaces must have distance metrics defined on them, although we do not require that the distance function defined for the full space be a metric. We begin by defining metric.

Definition 6.1 (Metric): For a space X, a *metric* is a function $\rho(x', x) : X \times X \to \mathbb{R}$ that satisfies

1. $\rho(x', x) \geq 0$
2. $\rho(x', x) = 0$ if and only if $x' = x$
3. $\rho(x', x) = \rho(x, x')$
4. $\forall x'' \in X$, $\rho(x', x) \leq \rho(x', x'') + \rho(x'', x)$

Point 4 is the *triangle inequality.*

The configuration space for the DARRT algorithm is the product space of any spaces with metrics defined on them. In general, this consists of one robot space and several object spaces.

Definition 6.2 (DARRT Configuration Space): A configuration space for the DARRT algorithm, $X = M_0 \times M_1 \times M_2 \times ... \times M_n$, is the cross product of several subspaces. Each of these subspaces must have a metric ρ_i defined for it. For $x \in X$, x_i denotes the projection of x onto subspace i. Similarly, for subset $B \subseteq X$,

$$B_i = \{m \in M_i | \exists x \in B \text{ with } x_i = m\} \qquad (6.1)$$

denotes the projection of the entire set onto subspace i.

One problem with manipulation is that we allow contact between the robot and objects (but usually not overlap), but the RRT algorithm requires that the free space be open, which does not allow contact. We resolve this issue by requiring that the empty space planner handle contact between the robot and objects if contact is required. Specifically, the empty space planner is responsible for keeping the plans in a set of *restricted* configurations. This restricted configuration space does not depend on the fixed obstacles but only on the robot, objects, and ways of manipulating the objects. Therefore, the empty space planner is still independent of the obstacle placement.

Definition 6.3 (Restricted Configuration Space): The *restricted configuration space* $X_{restricted} \subseteq X$ is a set of allowed configurations in X.

For example, consider a domain consisting of a round robot and a round object. The robot can contact the round object so that it can grip it and move it. It cannot overlap with the object. The restricted configuration space is all configurations in which the robot and object are not in contact or only in contact along their borders. The empty space planner must only return plans within this space. When the algorithm does collision checking for the empty space plan, it does not check for collisions between the robot and object, but only between the robot and fixed obstacles and the object and fixed obstacles.

We can think of the restricted configuration space as the actual configuration space for the algorithm. We sample only from this space and all plans lie within this space. We define it separately from the DARRT configuration space because it is helpful to keep in mind that it is a subset of a cross product space.

Another advantage of the restricted configuration space is that it makes explicit the structure of the free space. This space is the set of configurations in which the robot does not contact any obstacles, the objects do not contact any obstacles, and the configuration is in the restricted configuration space. Therefore, the free space is the product of the free spaces within each subspace intersected with the restricted space. This will be important in our definition of tube for the DARRT algorithm and also in the example given in Section 6.2.3.

Definition 6.4 (DARRT Free Space): Let the DARRT configuration space be $X = M_0 \times ... \times M_n$ with restricted configuration space $X_{restricted}$. The *free space for the ith subspace* is denoted as $M_{i,free}$. The *product free space* is

$$X_A = M_{0,free} \times ... \times M_{n,free}. \tag{6.2}$$

The *DARRT free space* is the intersection of the product free space and the restricted configuration space,

$$X_{free} = X_{restricted} \cap X_A. \tag{6.3}$$

The main point of the DARRT algorithm is a way to "adjust" samples based on the nearest configuration in the tree. As we discussed in Chapter 3, we accomplish this using *projection functions* that project the sample onto lower dimensional subspaces.

Definition 6.5 (Projection Function): A *projection function* $f(x', x) : X \times X \to X$ takes two configurations, x' and x, and projects x onto some subspace, $P(x') \subseteq X$. The subspace $P(x')$ may be dependent on the first argument to f.

As we showed in Chapter 3, in manipulation we can no longer assume that a straight line from one configuration to another represents an executable trajectory. To create these executable trajectories we use "empty space planners." An empty space planner returns a trajectory:

Definition 6.6 (Trajectory): Let $x', x \in X$. A function $\tau : [0, 1] \to X$ is a *trajectory from x' to x* if and only if $\tau(0) = x'$ and $\tau(1) = x$. A configuration $p \in X$ is *on the trajectory* τ, denoted $p \in \tau$, if there is some $t \in [0,1]$ with $p = \tau(t)$. For all $a \in [0,1]$, for all $b \in [a,1]$, the *sub-trajectory* $\tau[a,b]$ of τ from a to b is defined, for $t \in [0,1]$ as

$$\tau[a,b](t) = \tau\left(a + (b-a)t\right). \tag{6.4}$$

An empty space planner is a function that returns a trajectory from x' to x that lies entirely in $X_{restricted}$:

Definition 6.7 (Empty Space Planner): A deterministic function L is a *empty space planner* if and only if it returns a trajectory, $L(x',x)$, from x' to x and, for all $t \in [0,1]$, $L(x',x)(t) \in X_{restricted}$. We denote the set of configurations returned by the empty space planner as

$$\pi(x',x) = \bigcup_{t\in[0,1]} L(x',x)(t). \tag{6.5}$$

In the proof of exponential convergence for the RRT algorithm in holonomic spaces given in Section 2.2.2, the empty space planner was the line segment between two configurations.

Given a trajectory returned by the empty space planner, we must only consider the part of the trajectory that lies entirely in X_{free}. This is done by the EXTEND function in Algorithm 3.1, but we define it more formally here.

Definition 6.8 (Truncated Collision Free Trajectory): Let L be an empty space planner, let $x' \in X_{free}$, and let $x \in X$. A trajectory $\tau(x',x)$ is a *truncated collision free trajectory* from x' to x if and only if $\tau(0) = x'$, $\tau(x',x)$ is a sub-trajectory of $L(x',x)$, and, for all $t \in [0,1]$, $\tau(x',x)(t) \in X_{free}$.

We also require that if the empty space planner returns a collision free trajectory, that entire trajectory is added to the tree. This requirement defines the control functions for the DARRT algorithm.

Definition 6.9 (DARRT Control Function): The function $u(x',x)$ is a *DARRT control function* if and only if it returns a truncated collision free trajectory on all input and $u(x',x) = L(x',x)$ whenever $L(x',x)(t) \in X_{free}$ for all $t \in [0,1]$.

The EXTEND function of Algorithm 3.1 is a DARRT control function.

Now we are ready to describe the input to the DARRT algorithm. The free space and empty space planner are input implicitly in the control function while the restricted configuration space is input implicitly in the SAMPLECONFIGURATIONSPACE method. The input is:

- SAMPLECONFIGURATIONSPACE: A method for sampling from $X_{restricted}$.
- $x_0 \in X_{free}$: Starting configuration.

Algorithm 6.1
Input: SampleConfigurationSpace, A method for sampling from $X_{restricted}$; x_0, Starting configuration; X_G, Goal set; ρ, Distance function; u, Control function; F, Set of projection functions
Output: A graph containing from x_0 into X_G.

```
DARRT(SampleConfigurationSpace, $x_0, X_G, \rho, u, F$)
1  $V_0 \leftarrow \{x_0\}$, $k \leftarrow 1$
2  while $V_{k-1} \cap X_G = \emptyset$
3      $x \leftarrow$ SampleConfigurationSpace()
4      $x' \leftarrow \arg\min_{v \in V_{k-1}} \rho(v, x)$
5      $f \leftarrow$ uniformRandomChoice($\{f_0, ..., f_{|F|}\}$)
6      $\tau \leftarrow u(x', f(x', x))$
7      $V_k \leftarrow V_{k-1} \cup \bigcup_{t \in [0,1]} \tau(t)$
8      $k \leftarrow k + 1$
9  return $V_{k-1}$
```

- $X_G \subseteq X$: Set of goal configurations.
- $\rho(x', x) : X \times X \rightarrow \mathbb{R}$: Returns the distance from x' to x. This need not be a metric.
- u: A DARRT control function.
- $F = \{f_0, ..., f_{|F|}\}$: A set of projection functions.

The code for the DARRT algorithm using this input is shown in Algorithm 6.1.

In the next section, we show that under some assumptions about the free space, the projection functions, the distance function, and the empty space planner, the DARRT algorithm converges exponentially. We then prove that these assumptions hold in two manipulation domains. The descriptions for these domains are given in Sections 6.2.2 and 6.2.3. The reader may wish to read these descriptions before proceeding to the next section to form an idea of the formal description of DARRT manipulation domains.

6.1.2 DARRT Analysis

This section is very similar in form to Section 2.2.2. We let the configuration space be $X = M_0 \times ... \times M_n$ with distance metrics $\rho_0, ..., \rho_n$. Throughout n will be the number of subspaces.

We are looking for paths in the space from some configuration x' to some configuration x. These paths consist of segments provided by the empty space planner (in Section 2.2.2 these segments were straight lines). The set of configurations a configuration can reach using only the empty space planner is the *locally reachable* set:

Definition 6.10 (Locally Reachable): For $x' \in X$, the *locally reachable* set of configurations from x',

$$U(x') = \left\{x \in X \middle| \pi(x', x) \subseteq X_{free}\right\}, \tag{6.6}$$

is the set of configurations for which L returns a collision free trajectory from x'. By definition of DARRT control function, $x \in U(x')$ if and only if $x = u(x', x)(1)$.

Recall that the convergence argument for the RRT algorithm in holonomic spaces required the free space to be open. Because the objects and robot might each contact fixed obstacles individually, the DARRT algorithm requires that the free space in each subspace be open. We define an "open ball" on the full configuration space as the union of open balls in each subspace:

Definition 6.11 (Open Ball): Let $x \in X$ be a configuration in a DARRT configuration space. An *open ball of radius δ around x* is the set

$$B_\delta(x) = \left\{x' \in X_{restricted} \middle| \forall i \in \{0, .., n\},\ \rho_i(x, x') < \delta\right\}, \tag{6.7}$$

consisting of open balls of radius δ in each subspace.

Note that the ball consists only of configurations in the restricted space. This will be important in allowing contact between the robot and objects while maintaining an open free space.

Balls of larger radius contain balls of smaller radius:

Lemma 6.1: For all $x \in X$, for all $\zeta > 0$, for all $\delta \leq \zeta$, $B_\delta(x) \subseteq B_\zeta(x)$.
Proof:

$$\begin{aligned}
B_\delta(x) &= \left\{x' \in X_{restricted} \middle| \forall i \in \{0, .., n\},\ \rho_i(x, x') < \delta\right\} && (6.8)\\
&= X_{restricted} \cap \left\{x' \in X \middle| \forall i \in \{0, ..., n\},\ \rho_i(x, x') < \delta\right\} && (6.9)\\
&\subseteq X_{restricted} \cap \left\{x' \in X \middle| \forall i \in \{0, ..., n\},\ \rho_i(x, x') < \zeta\right\} && (6.10)\\
&= \left\{x' \in X_{restricted} \middle| \forall i \in \{0, .., n\},\ \rho_i(x, x') < \delta\right\} && (6.11)\\
&= B_\zeta(x). && (6.12)
\end{aligned}$$

■

For the DARRT algorithm, the assumption that X_{free} is open is:

Assumption 6.1: For all $x \in X_{free}$, for some $\delta > 0$, $B_\delta(x) \subseteq X_{free}$.

Consider a situation in which we have a robot and an object. We allow contact but not overlap between the robot and object. In this case, the restricted space is

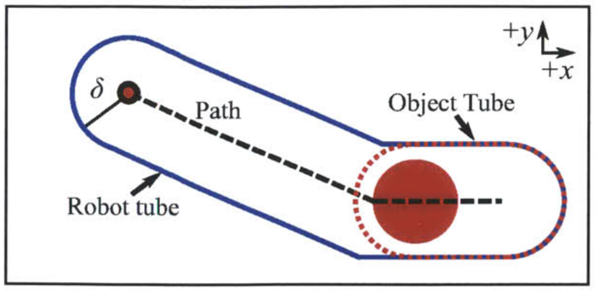


Figure 6.1: Tubes should be cross products of subspace tubes, not a union of open balls. This figure shows the tube around a path (black dashed) taken by a point robot (black-outlined red disc) and a disc object (red disc) using the pushing dynamics of Section 6.2.3. The robot is in free space anywhere inside the tube around the path it takes (solid blue) while the object is in free space anywhere inside the tube around the path it takes (dotted red). The tube in the full space is not the union of the cross product balls along the path in the full space (a much smaller set of points), but rather the cross product of the tubes in each space.

any configuration in which the robot and object do not overlap. The open balls in this space only contain configurations in which the robot and object do not overlap because they only contain configurations in the restricted state space. Therefore, although we allow contact, the open balls are still within the free space because they do not contain every configuration within δ of some configuration but only every configuration in the restricted space.

As with the holonomic case, we can turn the empty space planner's trajectories into tubes. However, the open ball is defined individually in each subspace. Therefore, the tube should be the cross product of the tube in each subspace rather than the tube in the full subspace. This is shown in Figure 6.1. We still require that the entire tube be in $X_{restricted}$.

Definition 6.12 (Tube): Let $x, x' \in X$. The *tube of radius δ from x' to x*,

$$T_\delta(x', x) = X_{restricted} \cap \left(\left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_0 \right) \times \left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_1 \right) \times \ldots \left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_n \right) \right), \tag{6.13}$$

is the set of configurations with all components less than δ from the path from x' to x.

Note that in general

$$T_\delta(x', x) \neq \left(\left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_0 \right) \times \left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_1 \right) \times \ldots \times \left(\bigcup_{p \in \pi(x', x)} (B_\delta(p))_n \right) \right) \tag{6.14}$$

even though $B_\delta(x)$ is limited to $X_{restricted}$. This is because it is possible that

$$B_\delta(x) \subset (B_\delta(x))_0 \times ... \times (B_\delta(x))_n. \tag{6.15}$$

For example, assume subspace 0 represents a point robot and subspace 1 represents a disc object. $X_{restricted}$ is all configurations in which the robot is not in the interior of the object. Let x be a configuration in which the robot and object are in contact along the border of the object. Then $B_\delta(x)$ is the configurations near x where the robot and object are not in collision. However

$$(B_\delta(x))_0 = \{m \in M_0 | \rho_0(x_0, m) < \delta\} \tag{6.16}$$

because for every robot configuration near x there is *some* object configuration near x such that the object and robot are not in collision. Similarly,

$$(B_\delta(x))_1 = \{m \in M_1 | \rho_1(x_1, m) < \delta\}. \tag{6.17}$$

Therefore

$$(B_\delta(x))_0 \times (B_\delta(x))_1 \not\subset X_{restricted}. \tag{6.18}$$

Thus intersecting the cross product with $X_{restricted}$ in Equation 6.14 is necessary.

Just as balls of larger radius contain balls of smaller radius, tubes of larger radius contain tubes of smaller radius:

Lemma 6.2: For all $x', x \in X$, for all $\zeta > 0$, for all $\delta \leq \zeta$, $T_\delta(x', x) \subseteq T_\zeta(x', x)$.
Proof:

$$\begin{aligned} T_\delta(x', x) &= X_{restricted} \cap \left(\left(\bigcup_{p \in \pi(x',x)} (B_\delta(p))_0 \right) \times ... \times \left(\bigcup_{p \in \pi(x',x)} (B_\delta(p))_n \right) \right) && (6.19) \\ &\subseteq X_{restricted} \cap \left(\left(\bigcup_{p \in \pi(x',x)} (B_\zeta(p))_0 \right) \times ... \times \left(\bigcup_{p \in \pi(x',x)} (B_\zeta(p))_n \right) \right) && (6.20) \\ &= T_\zeta(x', x) && (6.21) \end{aligned}$$

using Lemma 6.1.

■

Now assume we have some $x' \in X_{free}$ and some $x \in U(x')$. Then the trajectory from x' to x returned by the empty space planner is collision free. Moreover, Assumption 6.1 should give us that this trajectory can be expanded into a tube without leaving the free space. The radius of this expansion is the radius of locality. The radius of locality in the holonomic case is shown in Figure 2.8.

Definition 6.13 (Radius of Locality): For all $x' \in X_{free}$, for all $x \in U(x')$, $\eta(x', x)$ is the *radius of locality* from x' to x if and only if $T_{\eta(x',x)}(x', x) \subseteq X_{free}$ and $T_\zeta(x', x) \not\subseteq X_{free}$ for all $\zeta > \eta(x', x)$.

Since each configuration along $\pi(x', x)$ can be expanded a little bit in each subspace, we should have that the whole path can be expanded in each subspace. We first need to note that if a ball is in free space the projection of the ball onto a subspace is in the free space of that subspace.

Lemma 6.3: For all $x \in X$ and all $\delta > 0$, if $B_\delta(x) \subseteq X_{free}$ then $(B_\delta(x))_i \subseteq M_{i,free}$.
Proof: We proceed by contradiction. Assume there is some $m \in (B_\delta(x))_i$ such that $m \notin M_{i,free}$. Since $m \in (B_\delta(x))_i$ there is some $y \in B_\delta(x)$ with $y_i = m$. Then $y \notin M_{0,free} \times ... \times M_{n,free}$ so $y \notin X_{free}$.

■

Now we can show that $\eta(x', x) > 0$.

Lemma 6.4: For all $x' \in X_{free}$ and all $x \in U(x')$, $\eta(x', x) > 0$.
Proof: By definition of $U(x')$, $\pi(x', x) \subseteq X_{free}$. For all $y \in \pi(x', x)$ let δ_y be the largest number such that $B_{\delta_y}(y) \subseteq X_{free}$. By Assumption 6.1, $\delta_y > 0$ for all y. Choose

$$\delta = \inf_{y \in \pi(x',x)} \delta_y > 0. \tag{6.22}$$

Consider $p \in T_\delta(x', x)$ and $i \in \{0, ..., n\}$. Then there is some $y \in \pi(x', x)$ with $p_i \in (B_\delta(y))_i \subseteq (B_{\delta_y}(y))_i$ using Lemma 6.1. Now $B_{\delta_y}(y) \subseteq X_{free}$ so by Lemma 6.3, $p_i \in M_{i,free}$. Therefore $p \in M_{0,free} \times ... \times M_{n,free}$. Since $p \in X_{restricted}$ by definition of $T_\delta(x', x)$, $p \in (M_{0,free} \times ... \times M_{n,free}) \cap X_{restricted} = X_{free}$. Thus $T_\delta(x', x) \subseteq X_{free}$. Since $\eta(x', x)$ is the largest number with $T_{\eta(x',x)} \subseteq X_{free}$, we have $\eta(x', x) \geq \delta > 0$ by Lemma 6.2.

■

In the holonomic case, we defined a local path between two configurations as a set of configurations around which we could draw open balls and guarantee that the presence of a configuration in one of the balls in the tree gave a constant probability of adding a configuration in the next ball to the tree as shown in Figure 2.5. Here, however, we know less about the empty space planner. Specifically, we cannot be certain that a ball is the correct "shape" to draw around a configuration because the objects may not be able to move instantaneously in any direction. Therefore, rather than define a sequence of configurations around which we put open balls, we define a sequence of sets $\{W^0, ..., W^m\}$ such that the presence of a configuration in the tree in one set guarantees a constant probability of adding a configuration in some later set to the tree. An example of a domain where open balls are not sufficient is given in Section 6.2.3.

Assume we have some configuration y' in the tree in subspace W^j. As we discussed in Section 2.2.2, we may not be able to guarantee that if we sample in W^{j+1}, we necessarily add this sample to the tree. Instead, we may need to sample from some set that depends on y'. Putting these requirements together allows us to define a local path in the more general case.

Definition 6.14 (Local Path): For $x', x \in X_{restricted}$ and $\delta > 0$, a set of subspaces $\{W^0, ..., W^m\}$ and an associated probability $\lambda \in (0, 1]$ is a *local path from x' to x of* *radius δ and sampling probability λ* if and only if

1. For some $\epsilon \in (0, \delta]$, $B_\epsilon(x') \subseteq W^0$
2. $W^m \subseteq B_\delta(x)$
3. For all $j < m$, for all $y' \in W^j$, for some $S^j(y')$, for all $y \in S^j(y')$ for all $z \in X$, if $\rho(z, y) \leq \rho(y', y)$ then for some $f \in F$, $f(z, y) \in \bigcup_{l>j} W^l$ and $\pi(z, f(z, y)) \subseteq T_\delta(x', x)$.
4. For all $j < m$, for all $y' \in W^j$, the probability of sampling from $S^j(y')$ is at least λ.

In point 1, we use ϵ because W^0 may not contain all of $B_\delta(x')$ but does need to contain some open ball around x'. In point 3, $S^j(y')$ is usually in W^{j+1} but does not have to be because we apply a projection function to the configuration sampled from $S^j(y')$.

In the holonomic case we could prove a local path existed (Lemma 2.4) because we knew the EXTEND function. Here, we must assume that such a path exists.

Assumption 6.2: For all $x', x \in X$ and all $\delta > 0$, a local path from x' to x of radius δ and sampling probability greater than zero exists.

Assumptions 6.1 and 6.2 are the only two assumptions necessary for proving exponential convergence. Assumption 6.1 is an assumption about the free space that is independent of the empty space planner and projection functions while Assumption 6.2 is an assumption about the empty space planner and projection functions that is independent of the free space. Assumption 6.1 (as with Assumption 2.1) is something we need to ensure is true in the definition of the free space and is usually satisfied by not allowing contact between the moving components and the fixed obstacles. Assumption 6.2 is a guideline for writing the empty space planner and projection functions.

From Assumption 6.2, we can show that if the empty space planner returns a collision free path from x' to x, we can find sets $\{W^0, ..., W^m\}$ with $x' \in W^0$ and $x \in W^m$ such that if a configuration in W^j is in the tree, we have a constant probability of adding a configuration in $\bigcup_{l>j} W^l$.

Lemma 6.5: For all $x' \in X$, for all $x \in U(x')$, for any $\delta \in (0, \eta(x', x)]$, let $\{W^0, ..., W^m\}$ be a local path from x' to x of radius δ and let $\lambda > 0$ be the associated sampling probability. For all $j < m$, if V_{k-1} contains a configuration in W^j at iteration $k-1$, the probability of adding a configuration in $\bigcup_{l>j} W^l$ at iteration k is at least $\lambda/|F|$.

Proof: Assume we have $y' \in W^j \cap V_{k-1}$ at iteration $k-1$ and that we sample in $y \in S^j(y')$. The probability of such a sample is at least λ by definition of local path. Let

$$z = \arg\min_{v \in V_{k-1}} \rho(v, y) \tag{6.23}$$

be the configuration returned on Line 4 of the DARRT algorithm. Then since $y' \in V_{k-1}$, we have $\rho(z,y) \leq \rho(y',y)$. Therefore by definition of local path, there is some $f \in F$ with $f(z,y) \in \bigcup_{l>j} W^l$ and $\pi(z,f(z,y)) \subseteq T_\delta(x',x) \subseteq T_{\eta(x',x)}(x',x) \subseteq X_{free}$ using Lemma 6.2. Assume we choose this f on Line 5 of the DARRT algorithm. The probability of such a choice is $1/|F|$ since we choose a projection from F uniformly at random. Since the two events are independent, the probability that we both choose $y \in S^j(y')$ and f is $\lambda/|F|$. Since $\pi(z,f(z,y)) \subseteq X_{free}$, $f(z,y) = u(z,f(z,y))(1)$ so $f(z,y) \in \bigcup_{l>j} W^l$ is added to V_k on iteration k.

■

Now we need to consider paths that are not necessarily single applications of the empty space planner. In the holonomic case, this was paths consisting of multiple line segments. In this case, they are combinations of trajectories returned by the empty space planner.

Definition 6.15 (Path): For all $x', x \in X$, the sequence of configurations $\{x^0, ..., x^t\}$ is a *path from* x' *to* x if and only if $x^0 = x'$, $x^t = x$, and for all $j \in \{0, ..., t-1\}$, $x^{j+1} \in U(x^j)$.

A configuration x is reachable from a configuration x' if there is a path from x' to x:

Definition 6.16 (Reachable): For all $x', x \in X_{free}$, x is *reachable from* x', denoted $x \in R(x')$, if and only if there is a path from x' to x.

We showed in Lemma 6.5 that if $x \in U(x')$ then we can create some sequence $\{W^0, ..., W^m\}$ such that if a configuration in W^j is in the tree, there is a constant probability of adding a configuration in $\bigcup_{l>j} W^l$. Now we use induction to show that this is true for any configuration x' can reach.

Lemma 6.6: For all $x' \in X$, for all $x \in R(x')$, for any $\zeta > 0$, for some $\gamma > 0$, for some $\epsilon > 0$, and some sequence of subspaces $W = \{W^0, ..., W^r\}$, $B_\epsilon(x') \subseteq W^0$, $W^r \subseteq B_\zeta(x)$, and for all $j \in \{0, ..., r-1\}$, if a configuration in W^j is in V_{k-1} at iteration $k-1$, the probability of adding a configuration in $\bigcup_{l>j} W^l$ to the tree at iteration k is at least γ and γ is independent of k.
Proof: Let $\{x^0, .., x^t\}$ be a path from x' to x. Some path exists because $x \in R(x')$. We proceed by induction on the length of the path.

Base Case ($t = 1$)*:* If $t = 1$, the path is $\{x^0, x^1\}$. Let $\delta = \min\left[\zeta, \eta(x^0, x^1)\right]$. Let $\{W^0, ..., W^m\}$ be a local path from x^0 to x^1 of radius δ with sampling probability $\lambda > 0$. By Assumption 6.2, $\{W^0, ..., W^m\}$ and λ exist. By definition of local path, $W^m \subseteq B_\delta(x^1) \subseteq B_\zeta(x^1)$ and for some $\epsilon > 0$, $B_\epsilon(x^0) \subseteq W^0$. Let $\gamma = \lambda/|F| > 0$. Clearly γ is independent of k. Lemma 6.5 gives us that if V_{k-1} contains a configuration in W^j at iteration $k-1$ then the probability we add a configuration in $\bigcup_{l>j} W^l$ to V_k at iteration k is at least γ.

Induction Step: Let the path be $\{x^0, ..., x^t\}$ of length $t+1$. The path $\{x^1, ..., x^t\}$

from x^1 to x is of length t so by induction there are some sequence of subspaces $\{W^0, ..., W^m\}$, some $\gamma' > 0$, and some $\delta > 0$ such that $W^m \subseteq B_\zeta(x^t)$, $B_\delta(x^1) \subseteq W^0$, and, if a configuration in W^j is in V_{k-1} at iteration $k-1$ then a configuration in $\bigcup_{l>j} W^l$ is added to V_k with probability at least γ'. Let $\{P^0, ..., P^p\}$ be a local path from x^0 to x^1 of radius $\min\left[\delta, \zeta, \eta(x^0, x^1)\right]$ with associated sampling probability $\lambda > 0$. $\{P^0, ..., P^p\}$ and λ exist by Assumption 6.2. Consider the sequence $\{Q^0, ..., Q^{p+m+1}\} = \{P^0, ..., P^p, W^1, ..., W^m\}$. By induction, $W^m \subseteq B_\zeta(x^t)$. By definition of local path, for some $\epsilon > 0$, $B_\epsilon(x^0) \in P^0$. Assume V_{k-1} contains a configuration in Q^j and consider the probability of adding a configuration in $\bigcup_{l>j} Q^l$. If $j \geq p+1$, by induction this probability is at least $\gamma' > 0$ and independent of k. Assume $j = p$. By definition of local path, $Q^p = P^p \subseteq B_\delta(x^1) \subseteq W^0$. Thus by induction, if a configuration in $Q^p \subseteq W^0$ is in V_{k-1} at iteration $k-1$, a configuration in $\bigcup_{l>0} W^l = \bigcup_{l>p} Q^l$ is added to V_k at iteration k with probability at least γ'. If $j \leq p-1$,

$$\Pr\left(V_k \cap \bigcup_{l>j} Q^l \neq \emptyset \middle| V_{k-1} \cap Q^j \neq \emptyset \text{ and } j \leq p-1\right) \geq \Pr\left(V_k \cap \bigcup_{j<l\leq p} Q^l \neq \emptyset \middle| V_{k-1} \cap P^j \neq \emptyset\right) \tag{6.24}$$

$$= \Pr\left(V_k \cap \bigcup_{l>j} P^l \neq \emptyset \middle| V_{k-1} \cap P^j \neq \emptyset\right) \tag{6.25}$$

$$\geq \lambda/|F| \tag{6.26}$$

using Lemma 6.5. Therefore, if V_{k-1} contains a configuration in Q^j, the probability we add a configuration in $\bigcup_{l>j} Q^l$ is at least $\min\left[\lambda/|F|, \gamma'\right] > 0$ and is independent of k.

■

Thus the probability that we advance along the path is independent of iteration. The proof of exponential convergence is almost identical to the holonomic case (Theorem 2.7).

Theorem 6.7 (Exponential Convergence of the DARRT algorithm): For all $x \in R(x_0)$, for all $\delta > 0$, the probability that the tree does not include a configuration in $B_\delta(x)$ after k iterations is $O(2^{-ak})$ for some positive constant a.

Proof: By Lemma 6.6, for some $\lambda > 0$ and independent of k and for some sequence of subspaces $\{W^0, ..., W^r\}$, $x_0 \in W^0$, $W^r \subseteq B_\delta(x)$, and for all $j < r$, if V_{k-1} contains a configuration in W^j, the probability of adding a configuration in $\bigcup_{l>j} W^l$ is at least λ. Let us consider each iteration as a Bernoulli distribution in which λ is the probability of a successful outcome. Since we begin with $x_0 \in W^0$ in the tree, in the worst case, we require r successful outcomes to add a configuration in $W^r \subseteq B_\delta(x)$ to the tree.

Let $C_1, ..., C_k$ be independent and identically distributed random variables whose common distribution is the Bernoulli distribution with parameter λ. The random variable $C = C_1 + ... + C_k$ is the number of successes after k iterations. Since each

C_i has the Bernoulli distribution, C has a binomial distribution with expected value $E[C] = k\lambda$. Therefore, for any $\gamma \in (0, 1]$,

$$\Pr\left[C < (1-\gamma)k\lambda\right] < e^{-kp\gamma^2/2}. \tag{6.27}$$

Since C is the number of successes after k iterations, we require $C \geq r$. Choosing $\gamma = 1 - \frac{r}{k\lambda}$, for $k > r/\lambda$, we have

$$\begin{aligned}
\Pr[C < r] &< \exp\left(-\frac{k\lambda}{2}\left(1 - \frac{r}{k\lambda}\right)^2\right) && (6.28)\\
&= \exp\left(-\frac{k\lambda}{2}\left(1 + \left(\frac{r}{k\lambda}\right)^2 - 2\frac{r}{k\lambda}\right)\right) && (6.29)\\
&= \exp\left(-\frac{k\lambda}{2} + r - \frac{r^2}{2k\lambda}\right) && (6.30)\\
&\leq \exp\left(-\frac{k\lambda}{2} + r\right) && (6.31)\\
&= O\left(e^{-k\lambda/2}\right). && (6.32)
\end{aligned}$$

Thus the probability that a configuration in $B_\delta(x)$ has not been added to the tree after k iterations is $O(2^{-ak})$ for some positive constant a.

■

Assumptions 6.1 and 6.2 are strong. In the next section, we show examples of two manipulation domains that satisfy these assumptions.

6.2 Examples

In this section we give two examples of manipulation domains that fulfill the assumptions necessary for exponential convergence of the DARRT algorithm. We first explain the notation we use and give a few lemmas about product spaces. The proofs of these lemmas can be found in Appendix A.

6.2.1 Preliminaries: Notation and Cross Product Spaces

Notation: All of the following examples are of a single round robot and object operating in a two dimensional plane. We use X to denote the full configuration space, but c to denote an individual configuration to avoid confusion with the coordinate x. Throughout we let R be the space representing the robot, O be the space representing the object, and $X = R \times O$ be the full configuration space. R and O are both be two dimensional while X is four dimensional. We denote a configuration in the full space $c^j = (c^j_R, c^j_O) \in X$. The notation $c^j_R = (x^j_R, y^j_R)$ refers to the robot's configuration while $c^j_O = (x^j_O, y^j_O)$ refers to the object's configuration. The j superscript is used to indicate that these are for configuration c^j.

There are a number of lemmas about product spaces that are useful to many of the examples. Most are intuitive, but formal proofs can be found in Appendix A.

Recall that if $X = M_0 \times ... \times M_n$ and $B \subseteq X$ then

$$B_i = \{m \in M_i | \exists c \in B \text{ s.t. } c_i = m\}.$$

Lemma 6.8: For all $B \subseteq M_0 \times ... \times M_n$, $B \subseteq B_0 \times ... \times B_n$.

Lemma 6.9: For some non-empty sets $Q_i \subseteq M_i$, let $B = Q_0 \times ... \times Q_n$. For any subsets $W_i \subseteq Q_i$, $W_0 \times ... \times W_n \subseteq B$.

Lemma 6.10: For some non-empty sets $Q_i \subseteq M_i$, let $B = Q_0 \times ... \times Q_n$. Then $B_i = Q_i$ for $i \in \{0, ..., n\}$.

Corollary 6.11: For all $c \in X$, for all $\delta > 0$, let $X = M_0 \times ... \times M_n$ where each M_i has distance metric ρ_i. Define

$$X_\delta(c) = \{c' \in X \,|\forall i \in \{0, ..., n\},\ \rho_i(c, c') < \delta\}. \tag{6.33}$$

Then $X_\delta(c) = (X_\delta(c))_0 \times ... \times (X_\delta(c))_n$.

Corollary 6.12: For all $c', c \in X$, for all $\delta > 0$, define

$$Q_\delta(c', c) = \left(\bigcup_{p \in \pi(c', c)} (X_\delta(p))_0 \right) \times ... \times \left(\bigcup_{p \in \pi(c', c)} (X_\delta(p))_n \right). \tag{6.34}$$

Then $Q_\delta(c', c) = (Q_\delta(c', c))_0 \times ... \times (Q_\delta(c', c))_n$.

Corollary 6.13: For all $c', c \in X$, for all $q \in \pi(c', c)$, for all $i \in \{0, ..., n\}$, for all $\delta > 0$, $(X_\delta(q))_i \subseteq (Q_\delta(c', c))_i$.

Lemma 6.14: Let $B = B_0 \times ... \times B_n \subseteq M_0 \times ... \times M_n$. If $\mu_i(B_i) > 0$ and we choose a configuration at random from $M_0 \times ... \times M_n$, then the probability of sampling from B is $\prod_{i=0}^{n} \frac{\mu_i(B_i)}{\mu_i(M_i)} > 0$.

Our last lemma formally proves that the tube around a line segment in the plane is convex. This can be seen graphically in Figure 2.8.

Lemma 6.15: For $i \in \{0, ..., n\}$, let M_i represent a plane with the Euclidean distance metric

$$\rho_i\left((x_i^\alpha, y_i^\alpha), (x_i^a, y_i^a)\right) = \sqrt{(x^\alpha - x^a)^2 + (y^\alpha - y^a)^2}, \tag{6.35}$$

and let

$$\tau_i(t) = \left(x_i^a + (x_i^b - x_i^a)t, y_i^a + (y_i^b - y_i^a)t\right) \tag{6.36}$$

describe a straight line in subspace M_i from c_i^a to c_i^b. Similarly, let

$$\sigma_i(t) = \left(x_i^\alpha + (x_i^\beta - x_i^\alpha)t, y_i^\alpha + (y_i^\beta - y_i^\alpha)t\right) \tag{6.37}$$

describe a straight line from c_i^α to c_i^β. Let

$$\left(\pi(c^a, c^b)\right)_i = \bigcup_{t\in[0,1]} \tau_i(t), \tag{6.38}$$

and let

$$\left(Q_\delta(c^\alpha, c^\beta)\right)_i = \bigcup_{t\in[0,1]} \{m \in M_i \,|\rho_i(\sigma(t), m) < \delta\}. \tag{6.39}$$

If $c_i^a \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$ and $c^b \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$, then $\left(\pi(c^a, c^b)\right)_i \subseteq \left(Q_\delta(c^\alpha, c^\beta)\right)_i$.

We are now ready to proceed with the examples.

6.2.2 Point Rigid Transfer

We start with the simplest possible manipulation example. We have a point robot and a point object in a two dimensional plane. The point robot can move in any direction instantaneously, but the object can only move when its coordinate is the same as the robot's coordinate. When the robot and object are in the same place, the object's motion mimics the robot's motion. The robot can occupy the same location as the object without moving it. This domain is shown in Figure 6.2. Because the object does not have to move with the robot, the empty space planner allows the robot to occupy the same position as the object even when the robot is moving alone. This allows us to set $X_{restricted} = X$.

We first describe the projection functions, sampling function, distance functions, free space, and empty space planner needed for this example. We then prove that Assumptions 6.1 and 6.2 hold.

Projection Functions

We require only two projection functions, one allowing the robot to move alone and the identity:

- $f_R(c', c) = (c_R, c'_O)$
- $f_I(c', c) = c$.

Distance Functions

We use the Euclidean distance metric for the robot space R and the object space O. Our full distance function ρ is the maximum distance in either subspace. For

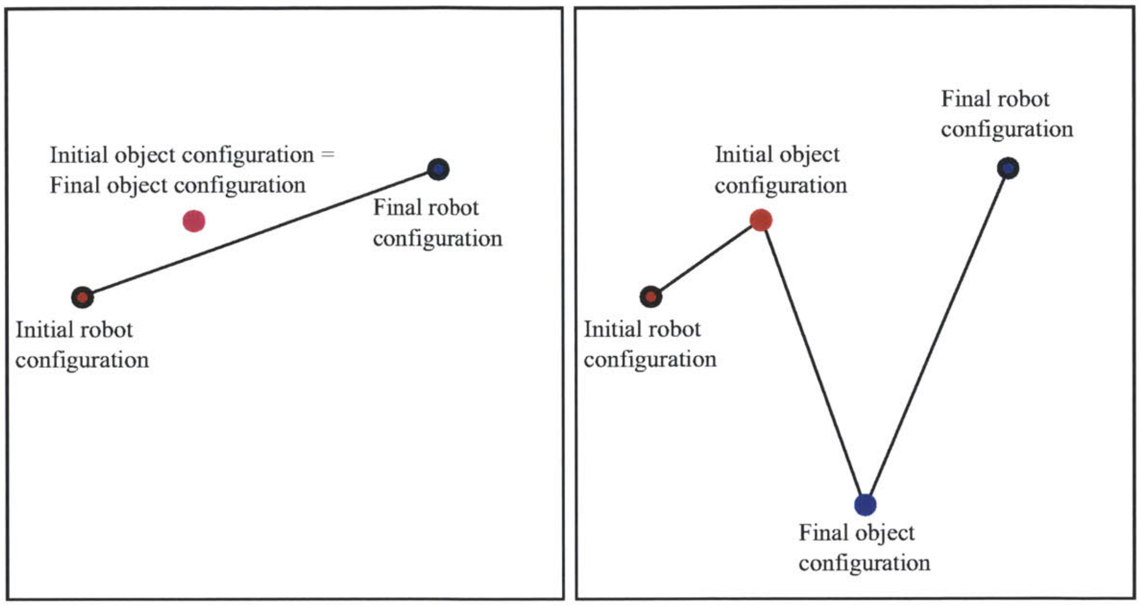

Initial object configuration =
Final object configuration
Final robot configuration
Initial robot configuration
(a)
Final robot configuration
Initial object configuration
Initial robot configuration
Final object configuration
(b)

Figure 6.2: A point robot and a point object. The robot can move the object when they occupy the same point. The robot is shown with a black outline while the object is a solid disc. The trajectories are shown from the red configuration to the blue configuration. (a) If the object does not move, the robot travels in a single straight line from its starting configuration (red) to its ending configuration (blue). (b) If the object moves, the robot takes a trajectory of three straight lines. It first transits to the object's position and transfers the object to the object's final configuration. Once the object is in place, the robot transits itself to its final configuration.

$c', c \in X$,

$$\rho_R(c'_R, c_R) = \sqrt{(x_R - x'_R)^2 + (y_R - y'_R)^2} \tag{6.40}$$

$$\rho_O(c'_O, c_O) = \sqrt{(x_O - x'_O)^2 + (y_O - y'_O)^2} \tag{6.41}$$

$$\rho(c', c) = \max\Big[\rho_R(c'_R, c_R), \rho_O(c'_O, c_O)\Big]. \tag{6.42}$$

The Euclidean distance metric is known to be a metric so ρ_R and ρ_O are full metrics with symmetry and the triangle inequality.

Free Space

Let C_{obs} be all points on an obstacle or on the boundary of an obstacle or on the boundary of the space. We define $M_{i,free}$ as the configurations for which component

i is not in contact with the obstacles:

$$M_{R,free} = \left\{ r \in R \,\middle|\, \inf_{p \in C_{obs}} \rho_R(r,p) > 0 \right\} \tag{6.43}$$

$$M_{O,free} = \left\{ o \in O \,\middle|\, \inf_{p \in C_{obs}} \rho_O(o,p) > 0 \right\} \tag{6.44}$$

$$X_{restricted} = X \tag{6.45}$$

$$X_{free} = M_{R,free} \times M_{O,free} \tag{6.46}$$

where Equation 6.46 follows from Equations 6.43- 6.45. We have $X_{restricted} = X$ because we allow the robot and object to occupy the same space without forcing the robot to move the object.

Note that since $X_{restricted} = X$, $B_\delta(c) = X_\delta(c)$ and $T_\delta(c', c) = Q_\delta(c', c)$ in the notation of Section 6.2.1. Therefore, we can apply the lemmas in Section 6.2.1 directly in our proofs of Assumptions 6.1 and 6.2.

We also require that R and O are both convex.

Sampling the Space

Since $X = X_{restricted}$, SampleConfigurationSpace chooses a random configuration from X.

Empty Space Planner

Lastly, we define the empty space planner. If the object does not move, the empty space planner moves the robot in a straight line to its final destination. If the object does move, the planner moves the robot to the object, moves the object to its final destination, and then moves the robot to its final destination as shown in Figure 6.2. More formally, on input (c', c), the planner returns the following:

$c'_O = c_O$ Move the robot in a single straight line from c'_R to c_R. This trajectory is

$$\tau(t) = \left(\left(x'_R + (x_R - x'_R)t, y'_R + (y_R - y'_R)t \right), c'_O \right). \tag{6.47}$$

$c'_O \neq c_O$ We move the robot in a straight line from c'_R to c'_O, then from c'_O to c_O, and then from c_O to c_R. The object moves in a straight line from c'_O to c_O during the second segment. The trajectory is

$$\tau(t) = \begin{cases} \left(\left(x'_R + (x'_O - x'_R)3t, y'_R + (y'_O - y'_R)3t\right), c'_O\right) & \text{if } t \leq \frac{1}{3} \\ \left(\left(x'_O + (x_O - x'_O)(3t-1), y'_O + (y_O - y'_O)(3t-1)\right), \right. \\ \quad \left. \left(x'_O + (x_O - x'_O)(3t-1), y'_O + (y_O - y'_O)(3t-1)\right)\right) & \text{if } \frac{1}{3} < t \leq \frac{2}{3} \\ \left(\left(x_O + (x_R - x'_O)(3t-2), y_O + (y_R - y_O)(3t-2)\right), c_O\right) & \text{if } \frac{2}{3} < t \leq 1 \end{cases} \tag{6.48}$$

Because these trajectories consist only of straight lines within R or O, and R and O are convex, we can guarantee that they always remain within $X_{restricted} = X$. Therefore, this is an empty space planner.

Proof that Assumption 6.1 Holds

We first show that X_{free} is open.

Theorem 6.16 (Assumption 6.1 Holds): For all $c \in X_{free}$, for some $\delta > 0$, $B_\delta(c) \subseteq X_{free}$.
Proof: Choose $c \in X_{free}$. Let

$$\delta_R = \inf_{p \in C_{obs}} \rho_R(c_R, p) \tag{6.49}$$
$$\delta_O = \inf_{p \in C_{obs}} \rho_O(c_O, p) \tag{6.50}$$
$$\delta = \frac{1}{2} \min\left[\delta_R, \delta_O\right]. \tag{6.51}$$

By definition of X_{free}, $\delta_R > 0$ and $\delta_O > 0$ so $\delta > 0$. Choose $c' \in B_\delta(c)$. By the triangle inequality for any $p \in C_{obs}$,

$$\begin{aligned} \rho_R(p, c'_R) &\geq \rho_R(p, c_R) - \rho_R(c'_R, c_R) && (6.52)\\ &> \delta_R - \delta && (6.53)\\ &\geq \frac{1}{2}\delta_R && (6.54)\\ &> 0 && (6.55)\\ \rho_O(p, c'_O) &\geq \rho_O(p, c_O) - \rho_O(c'_O, c_O) && (6.56)\\ &> \delta_O - \delta && (6.57)\\ &\geq \frac{1}{2}\delta_O && (6.58)\\ &> 0. && (6.59) \end{aligned}$$

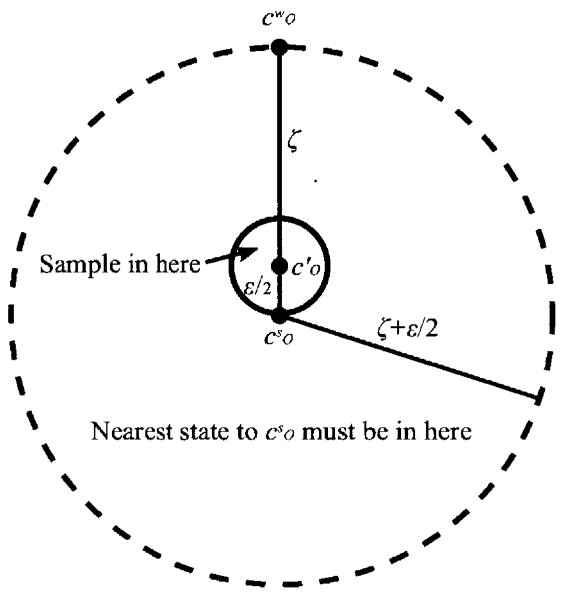


Figure 6.3: Assume we have a configuration c^w_O for the object in the tree that is a distance of ζ from c'_O and we sample a new configuration c^s_O for the object a distance of $\epsilon/2$ from c'_O. Then the nearest configuration in the tree to c^s_O could be as far as $\zeta + \epsilon$ from c'_O.

Therefore $c' \in M_{R,free} \times M_{O,free} = X_{free}$ so $B_\delta(c) \subseteq X_{free}$.

■

Proof that Assumption 6.2 Holds

Now we show that Assumption 6.2 holds. We begin by showing it holds when the robot moves alone. This is similar to Lemma 2.4, but made slightly more complicated by the fact that we must take into account the object's configuration as well as the robot's even though the object is not moving.

This proof diverges from Lemma 2.4 because the configuration we sample is *not* necessarily the configuration we try to extend towards. Assume we know there is a collision free path from c' to c where $c'_O = c_O$. We place open balls in the robot's subspace around configurations along the path the robot takes from c'_R to c_R, similar to Figure 2.5. As in Lemma 2.4, if we make the radius of these balls small enough and place them close enough together, we can guarantee that, if we have a configuration in one ball in the tree and we sample a configuration in the next ball, we add that configuration to the tree.

The problem is how to sample the object's configuration as we do this. Unlike the robot's configuration, the *sampled* configuration for the object is *not* the configuration we try to extend towards. Specifically, assume we sample $c^s \in X$ and that the nearest configuration in the tree is $c^n \in X$. Then, because we are considering the case in which the robot moves by itself, we extend not towards $c^s = (c^s_R, c^s_O)$ but towards (c^s_R, c^n_O). Thus we must have that c^n_O is sufficiently close to c'_O. We accomplish this by assuming we start close to c'_O and sampling the configuration of the object *very* close to c'_O so that even c^n_O must be quite close to c'_O. Assume we start with a configuration in the tree, c^w, in which the object is a distance ζ from c'_O and we sample within a radius of $\frac{\epsilon}{2}$ from around c'_O. Then c^n_O could be a distance of $\zeta + \epsilon$ from c'_O as shown in Figure 6.3. Therefore, we allow the ball around c'_O in which the object might be found to grow as we move along the path. With proper choices of the size of the ball around the robot, the spacing of these balls, ζ and ϵ , we can always ensure that the possible locations for the object remain within some outer radius.

Lemma 6.17: For $c', c \in X$, let $c'_O = c_O$. For all $\eta > 0$, for some sets $\{W^0, ..., W^m\}$, $B_{\eta/6}(c') \subseteq W^0$, $W^m \subseteq B_\eta(c)$, and for all $j < m$, for all $c^w \in W^j$, for some $S^j(c^w)$, for all $c^s \in S^j(c^w)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^w, c^s)$, $f_R(c^n, c^s) \in W^{j+1}$, and $\pi(c^n, f_R(c^n, c^s)) \subseteq T_\eta(c', c)$, and for some $\lambda^R_\eta(c', c) > 0$, the probability of sampling from $S^j(c^w)$ is at least $\lambda^R_\eta(c', c)$.

Proof: We define

$$\zeta = \frac{\eta}{2} \tag{6.60}$$
$$\delta = \frac{\eta}{6} \tag{6.61}$$
$$j_{\max} = \left\lceil \frac{\rho(c', c)}{\eta/6} \right\rceil \tag{6.62}$$
$$d = \frac{\rho(c', c)}{j_{\max}} \leq \frac{\eta}{6} \tag{6.63}$$
$$\epsilon = \frac{d}{\rho(c', c)} \left(\eta - (\zeta + \delta)\right). \tag{6.64}$$

For $0 \leq j \leq j_{\max}$, we also define

$$r^j = \left(x'_R + \frac{jd}{\rho(c', c)}(x_R - x'_R), y'_R + \frac{jd}{\rho(c', c)}(y_R - y'_R)\right) \tag{6.65}$$
$$W^j_R = \left\{q \in R \,\middle|\, \rho_R(r^j, q) < \delta\right\} \tag{6.66}$$
$$W^j_O = \{o \in O \,|\, \rho_O(c'_O, o) < \zeta + j\epsilon\} \tag{6.67}$$
$$W^j = W^j_R \times W^j_O. \tag{6.68}$$

We first show that $B_\delta(c') \subseteq W^0$ and $W^{j_{\max}} \subset B_\eta(c)$.

Since $\delta < \zeta$ and $r^0 = c'_R$ and using Corollary 6.11, we have

$$\begin{aligned} B_\delta(c') &= (B_\delta(c'))_R \times (B_\delta(c'))_O && (6.69)\\ &\subseteq \left\{q \in R \,\middle|\, \rho_R(r^0, q) < \delta\right\} \times (B_\zeta(c'))_O && (6.70)\\ &= W^0. && (6.71) \end{aligned}$$

Now

$$\begin{aligned}
\zeta + j\epsilon \le \zeta + j_{\max}\frac{d}{\rho(c',c)}\left(\eta - (\zeta+\delta)\right) &= \zeta + j_{\max}\frac{\rho(c',c)/j_{\max}}{\rho(c',c)}\left(\eta - (\zeta+\delta)\right) && (6.72)\\
&= \eta - \delta && (6.73)\\
&< \eta. && (6.74)
\end{aligned}$$

Additionally

$$\begin{aligned}
r^{j_{max}} &= \left(x'_R + \frac{j_{\max}d}{\rho(c',c)}(x_R - x'_R), y'_R + \frac{j_{\max}d}{\rho(c',c)}(y_R - y'_R)\right) && (6.75)\\
&= \left(x'_R + \frac{j_{\max}\rho(c',c)/j_{\max}}{\rho(c',c)}(x_R - x'_R), y'_R + \frac{j_{\max}\rho(c',c)/j_{\max}}{\rho(c',c)}(x_R - x'_R)\right) && (6.76)\\
&= (x_R, y_R) && (6.77)\\
&= c_R. && (6.78)
\end{aligned}$$

Therefore

$$\begin{aligned}
W^{j_{\max}} &\subseteq (B_\delta(c))_R \times (B_\eta(c'))_O && (6.79)\\
&= (B_\delta(c))_R \times (B_\eta(c))_O && (6.80)\\
&\subset (B_\eta(c))_R \times (B_\eta(c))_O && (6.81)\\
&= B_\eta(c) && (6.82)
\end{aligned}$$

again using Corollary 6.11 and the fact that $c'_O = c_O$.

Now we show that $W^j \subset T_\eta(c',c)$ as this makes the rest of the proof easier. Since $c'_O = c_O$,

$$\pi(c',c) = \bigcup_{t\in[0,1]} \left(\left(x'_R + (x_R - x'_R)t, y'_R + (y_R - y'_R)t\right), c'_O\right). \qquad (6.83)$$

Let $c^j = (r^j, c'_O)$. Using that $0 \le j \le j_{\max}$,

$$\begin{aligned}
c^j &= \left(\left(x'_R + \frac{jd}{\rho(c',c)}(x_R - x'_R), y'_R + \frac{jd}{\rho(c',c)}\right), c'_O\right) && (6.84)\\
&= \left(\left(x'_R + \frac{j\rho(c',c)/j_{\max}}{\rho(c',c)}(x_R - x'_R), y'_R + \frac{j\rho(c',c)/j_{\max}}{\rho(c',c)}(y_R - y'_R)\right), c'_O\right) && (6.85)\\
&= \left(\left(x'_R + \frac{j}{j_{\max}}(x_R - x'_R), y'_R + \frac{j}{j_{\max}}(y_R - y'_R)\right), c'_O\right) && (6.86)\\
&\in \pi(c',c). && (6.87)
\end{aligned}$$

Therefore, using that $\zeta \leq \zeta + j\epsilon < \eta$,

$$\begin{aligned}
W^j &= \left(B_\delta(c^j)\right)_R \times \left(B_{\zeta+j\epsilon}(c^j)\right)_O && (6.88)\\
&\subset \left(B_\eta(c^j)\right)_R \times \left(B_\eta(c^j)\right)_O && (6.89)\\
&\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.90)\\
&= T_\eta(c',c) && (6.91)
\end{aligned}$$

using Corollaries 6.12 and 6.13.

Now we show that for all $j < j_{\max}$, for all $c^w \in W^j$, for some $S^j(c^w)$, for all $c^s \in S^j(c^w)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^w, c^s)$ then $f_R(c^n, c^s) \in W^{j+1}$, $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^w)$ is always greater than zero. For $0 \leq j < j_{\max}$ and $c^w \in W^j$, we choose

$$\begin{aligned}
S^j_R(c^w) &= \left\{r \in R \,\middle|\, \rho_R(r^{j+1}, r) < \delta\right\} && (6.92)\\
S^j_O(c^w) &= \left\{o \in O \,\middle|\, \rho_O(c'_O, o) < \frac{\epsilon}{2}\right\} && (6.93)\\
S^j(c^w) &= S^j_R(c^w) \times S^j_O(c^w). && (6.94)
\end{aligned}$$

Now $\delta > 0$ so $S^j_R(c^w)$ is an open ball of radius $\delta > 0$ in R. Therefore $p_R = \frac{\mu_R(S^0(c'))}{\mu_R(R)} = \frac{\mu_R(S^j(c^w))}{\mu_R(R)}$ is independent of j or c^w. Now we also have that

$$\begin{aligned}
\epsilon &= \frac{d}{\rho(c',c)}(\eta - (\zeta + \delta)) && (6.95)\\
&= \frac{d}{\rho(c',c)}\left(\eta - \frac{\eta}{2} - \frac{\eta}{6}\right) && (6.96)\\
&= \frac{d\eta}{3\rho(c',c)} && (6.97)\\
&> 0 && (6.98)
\end{aligned}$$

so $S^j_O(c^w)$ is an open ball of radius $\frac{\epsilon}{2} > 0$ in O. Therefore $p_O = \frac{\mu_O(S^0(c'))}{\mu_O(O)} = \frac{\mu_O(S^j(c^w))}{\mu_O(O)}$ is independent of j or c^w. Therefore, the probability of sampling from $S^j(c^w)$ is $\lambda^R_\eta(c', c) = p_O p_R > 0$ by Lemma 6.14 regardless of j or c^w.

Assume we choose $c^s \in S^j(c^w)$. By the triangle inequality,

$$\begin{aligned}
\rho_R(c^w_R, c^s_R) &\leq \rho_R(c^w_R, r^j) + \rho_R(r^j, r^{j+1}) + \rho_R(r^{j+1}, c^s_R) && (6.99)\\
&< \delta + \Bigg(\left(x'_R + \tfrac{(j+1)d}{\rho(c',c)}(x_R - x'_R) - \left(x'_R + \tfrac{jd}{\rho(c',c)}(x_R - x'_R)\right)\right)^2 \\
&\qquad + \left(y'_R + \tfrac{(j+1)d}{\rho(c',c)}(y_R - y'_R) - \left(y'_R + \tfrac{jd}{\rho(c',c)}(y_R - y'_R)\right)\right)^2\Bigg)^{1/2} + \delta && (6.100)\\
&= 2\delta + \sqrt{\left(\frac{d}{\rho(c',c)}(x_R - x'_R)\right)^2 + \left(\frac{d}{\rho(c',c)}(y_R - y'_R)\right)^2} && (6.101)
\end{aligned}$$

$$= 2\delta + \sqrt{\left(\frac{d}{\rho(c',c)}\right)^2 ((x_R - x'_R)^2 + (y_R - y'_R)^2)} \tag{6.102}$$

$$= 2\delta + \sqrt{\left(\frac{d}{\rho(c',c)}\right)^2 \rho(c',c)^2} \tag{6.103}$$

$$= 2\delta + d, \tag{6.104}$$

and

$$\rho_O(c^w_O, c^s_O) \leq \rho_O(c^w_O, c'_O) + \rho_O(c'_O, c^s_O) \tag{6.105}$$

$$< \zeta + j\epsilon + \frac{\epsilon}{2} \tag{6.106}$$

$$= \zeta + \left(j + \frac{1}{2}\right)\epsilon. \tag{6.107}$$

Now for some $c^n \in X$ let

$$\rho(c^n, c^s) \leq \rho(c^w, c^s) \tag{6.108}$$

$$= \max\left[\rho_R(c^w_R, c^s_R), \rho_O(c^w_O, c^s_O)\right] \tag{6.109}$$

$$< \max\left[2\delta + d, \zeta + \left(j + \frac{1}{2}\right)\epsilon\right] \tag{6.110}$$

$$\leq \max\left[\frac{\eta}{3} + \frac{\eta}{6}, \zeta + \left(j + \frac{1}{2}\right)\epsilon\right] \tag{6.111}$$

$$= \max\left[\frac{\eta}{2}, \frac{\eta}{2} + \left(j + \frac{1}{2}\right)\epsilon\right] \tag{6.112}$$

$$= \zeta + \left(j + \frac{1}{2}\right)\epsilon. \tag{6.113}$$

Let $c^f = f_R(c^n, c^s) = (c^s_R, c^n_O)$. We first show that $c^f \in W^{j+1}$. Consider

$$\rho_O(c'_O, c^f_O) = \rho_O(c'_O, c^n_O) \tag{6.114}$$

$$\leq \rho_O(c'_O, c^s_O) + \rho_O(c^s_O, c^n_O) \tag{6.115}$$

$$< \frac{\epsilon}{2} + \zeta + \left(j + \frac{1}{2}\right)\epsilon \tag{6.116}$$

$$= \zeta + (j+1)\epsilon \tag{6.117}$$

so $c^f_O \in W^{j+1}_O$. Additionally

$$\rho_R\left(r^{j+1}, c^f_R\right) = \rho_R\left(r^{j+1}, c^s_R\right) < \delta \tag{6.118}$$

so $c^f_R \in W^{j+1}_R$. Therefore $c^f \in W^{j+1}_R \times W^{j+1}_O = W^{j+1} \subseteq T_\eta(c', c)$.

Now we show that $\pi(c^n, c^f) \subseteq (T_\eta(c', c))_O$. Firstly, $c^f_O = c^n_O \in W^{j+1}_O \subseteq (T_\eta(c', c))_O$.

Additionally, using that $j < j_{\max}$,

$$\begin{aligned}
\rho_R(r^{j+1}, c^n_R) &\leq \rho_R(r^{j+1}, c^s_R) + \rho_R(c^s_R, c^n_R) && (6.119)\\
&< \delta + \zeta + \left(j + \frac{1}{2}\right)\epsilon && (6.120)\\
&< \delta + \zeta + j_{\max}\frac{d}{\rho(c', c)}(\eta - (\zeta + \delta)) && (6.121)\\
&= \delta + \zeta + j_{\max}\frac{\rho(c', c)/j_{\max}}{\rho(c', c)}(\eta - \zeta - \delta) && (6.122)\\
&= \delta + \zeta + (\eta - \zeta - \delta) && (6.123)\\
&= \eta. && (6.124)
\end{aligned}$$

Since $j < j_{\max}$, $j + 1 \leq j_{\max}$ and we showed in Equation 6.87 that $r^j \in (\pi(c', c))_R$ for all $j \in \{0, ..., j_{\max}\}$. Thus $c^n_R \in (T_\eta(c', c))_R$. We already have $c^n_O = c^f_O \in (T_\eta(c', c))_O$ so

$$c^n \in (T_\eta(c', c))_R \times (T_\eta(c', c))_O = T_\eta(c', c) \qquad (6.125)$$

by Corollary 6.12. Thus, $c^n \in T_\eta(c', c)$ and $c^f \in T_\eta(c', c)$. Now $c^n_O = c^f_O$ so

$$\begin{aligned}
\pi(c^n, c^f) &= \bigcup_{t\in[0,1]}\left(\left(x^n_R + (x^f_R - x^n_R)t, y^n_R + (x^f_R - x^n_R)t\right), c^n_O\right) && (6.126)\\
&\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O && (6.127)\\
&= T_\eta(c', c) && (6.128)
\end{aligned}$$

using Lemma 6.15 and Corollary 6.12.

■

The case in which the robot and object move together is actually closer to the case of the point robot by itself. We do modify the proof slightly to account for the fact that the robot and object may not start or end in exactly the same place, but the fact that we are, in this case, extending towards the sampled configuration makes the proof less involved than that of Lemma 6.17.

Lemma 6.18: For $c', c \in X$, assume $c'_O \neq c_O$, $c'_R = c'_O$ and $c_R = c_O$. Then for all $\eta > 0$, for some sets $\{W^0, ..., W^m\}$, for some $\epsilon \in (0, \eta]$, $B_\epsilon(c') \subseteq W^0$, $W^m \subseteq B_\eta(c^e)$, and for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$, $f_I(c^n, c^s) \in W^{j+1}$, and $\pi(c^n, f_I(c^n, c^s)) \subseteq T_\eta(c', c)$, and for some $\lambda^O_\eta(c', c) > 0$, the probability of sampling from $S^j(c^j)$ is at least $\lambda^O_\eta(c', c)$.
Proof: Let

$$\begin{aligned}
j_{\max} &= \left\lceil \frac{\rho(c', c)}{\eta/4} \right\rceil && (6.129)\\
\delta &= \frac{\rho(c', c)}{j_{\max}} \leq \frac{\eta}{4} && (6.130)
\end{aligned}$$

and for $0 \le j \le j_{\max}$, define

$$\begin{aligned} c^j &= \left(\left(x'_O + (x_O - x'_O)\frac{j}{j_{\max}}, y'_O + (y_O - y'_O)\frac{j}{j_{\max}} \right), \left(x'_O + (x_O - x'_O)\frac{j}{j_{\max}}, y'_O + (y_O - y'_O)\frac{j}{j_{\max}} \right) \right) && (6.131) \\ W^j &= B_\delta(c^j). && (6.132) \end{aligned}$$

Now $c^0 = c'$ and $c^{j_{\max}} = c$ so clearly $B_\delta(c') = W^0$ and $W^{j_{\max}} = B_\delta(c) \subseteq B_\eta(c)$.

For any $j < j_{\max}$, choose $c^w \in W^j$. We define

$$S^j(c^w) = B_\delta(c^{j+1}) = W^{j+1}. \tag{6.133}$$

Now $S^j(c^w) = (B_\delta(c^{j+1}))_R \times (B_\delta(c^{j+1}))_O$ by Corollary 6.11 and $(B_\delta(c^{j+1}))_R$ is just an open ball of radius δ in subspace R. Therefore $p_R = \frac{\mu_R(B_\delta(c'))}{\mu_R(R)} = \frac{\mu_R\left((B_\delta(c^{j+1}))\right)_R}{\mu_R(R)} > 0$ is the probability we sample from $(S^j(c^w))_R$ and is independent of c^w or j. Similarly, $p_O = \frac{\mu_O\left((B_\delta(c'))_O\right)}{\mu_O(O)} > 0$ is the probability we sample from $(S^j(c^w))_O$. Therefore by Lemma 6.14, the probability of sampling from $S^j(c^w)$ is $\lambda^O_\eta(c', c) = p_R p_O > 0$.

Now we show that $W^j \subseteq T_\eta(c', c)$. We have

$$\pi(c', c) = \bigcup_{t \in [0,1]} \left((x'_O + (x_O - x'_O)t, y'_O + (y_O - y'_O)t), (x'_O + (x_O - x'_O)t, y'_O + (y_O - y'_O)t) \right). \tag{6.134}$$

Now for $0 \le j \le j_{\max}$, $c^j \in \pi(c', c)$. Therefore

$$\begin{aligned} W^j &= B_\delta(c^j) \subseteq B_\eta(c^j) && (6.135) \\ &= \left(B_\eta(c^j)\right)_R \times \left(B_\eta(c^j)\right)_O && (6.136) \\ &\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O && (6.137) \\ &= T_\eta(c', c) && (6.138) \end{aligned}$$

using Corollaries 6.11, 6.12 and 6.13.

Now let $c^s \in S^j(c^w)$ and consider any c^n such that

$$\begin{aligned} \rho(c^n, c^s) &\le \rho(c^w, c^s) && (6.139) \\ &= \max\left[\rho_R(c^w_R, c^s_R), \rho_O(c^w_O, c^s_O)\right] && (6.140) \\ &\le \max\Big[\rho_R(c^w_R, c^j_R) + \rho(c^j_R, c^{j+1}_R) + \rho_R(c^{j+1}_R, c^s_R), \\ &\qquad \rho_O(c^w_O, c^j_O) + \rho_O(c^j_O, c^{j+1}_O) + \rho_O(c^{j+1}, c^s_O)\Big] && (6.141) \\ &< \delta + \Bigg(\left(\left(x'_O + \tfrac{(j+1)\delta}{\rho(c',c)}(x_O - x'_O)\right) - \left(x'_O + \tfrac{jd}{\rho(c',c)}(x_O - x'_O)\right)\right)^2 + \\ &\qquad \left(\left(y'_O + \tfrac{(j+1)\delta}{\rho(c',c)}(y_O - y'_O)\right) - \left(y'_O + \tfrac{jd}{\rho(c',c)}(y_O - y'_O)\right)\right)^2\Bigg)^{1/2} + \delta && (6.142) \end{aligned}$$

$$
\begin{aligned}
&= 2\delta + \sqrt{\left(\tfrac{\delta}{\rho(c',c)}(x_O - x'_O)\right)^2 + \left(\tfrac{\delta}{\rho(c',c)}(y_O - y'_O)\right)^2} && (6.143)\\
&= 2\delta + \sqrt{\left(\frac{\delta}{\rho(c',c)}\right)^2 ((x_O - x'_O)^2 + (y_O - y'_O)^2)} && (6.144)\\
&= 2\delta + \sqrt{\left(\frac{\delta}{\rho(c',c)}\right)^2 \rho(c',c)^2} && (6.145)\\
&= 3\delta. && (6.146)
\end{aligned}
$$

We first show that $c^n \in T_\eta(c',c)$. We have

$$
\begin{aligned}
\rho_R(c_R^{j+1}, c_R^n) &\le \rho_R(c_R^{j+1}, c_R^s) + \rho_R(c_R^s, c_R^n) && (6.147)\\
&< 4\delta && (6.148)\\
&\le \eta. && (6.149)
\end{aligned}
$$

Similarly

$$
\begin{aligned}
\rho_O(c_O^{j+1}, c_O^n) &\le \rho_O(c_O^{j+1}, c_O^s) + \rho_O(c_O^s, c_O^n) && (6.150)\\
&< 4\delta && (6.151)\\
&\le \eta. && (6.152)
\end{aligned}
$$

Now $j < j_{\max}$ so $c^{j+1} \in \pi(c',c)$. Therefore

$$
\begin{aligned}
c^n &\in \left(B_\eta(c^{j+1})\right)_R \times \left(B_\eta(c^{j+1})\right)_O && (6.153)\\
&\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.154)\\
&= T_\eta(c',c) && (6.155)
\end{aligned}
$$

using Corollaries 6.11, 6.12 and 6.13.

Consider $f_I(c^n, c^s) = c^s \in S^{j+1} = W^{j+1} \subseteq T_\eta(c',c)$. Firstly let $c^b = (c_O^n, c_O^n)$ and $c^e = (c_O^s, c_O^s)$. Recall that $c_R^{j+1} = c_O^{j+1}$ and that $\rho_R = \rho_O$. Then

$$
\begin{aligned}
\rho(c^{j+1}, c^b) &= \max\Big[\rho_R(c_R^{j+1}, c_O^n), \rho_O(c_O^{j+1}, c_O^n)\Big] && (6.156)\\
&= \max\Big[\rho_O(c_O^{j+1}, c_O^n), \rho_O(c_O^{j+1}, c_O^n)\Big] && (6.157)\\
&= \rho_O(c_O^{j+1}, c_O^n) && (6.158)\\
&< \eta && (6.159)
\end{aligned}
$$

by Equation 6.152. Similarly, using that $c^s \in B_\delta(c^{j+1})$,

$$
\begin{aligned}
\rho(c^{j+1}, c^e) &= \max\Big[\rho_R(c_R^{j+1}, c_R^e), \rho_O(c_O^{j+1}, c_O^s)\Big] && (6.160)\\
&= \max\Big[\rho_O(c_O^{j+1}, c_O^s), \rho_O(c_O^{j+1}, c_O^s)\Big] && (6.161)\\
&= \rho_O(c_O^{j+1}, c_O^s) && (6.162)
\end{aligned}
$$

$$< \delta \tag{6.163}$$
$$< \eta. \tag{6.164}$$

Therefore

$$\begin{aligned} c^b, c^e \in B_\eta(c^{j+1}) &= \left(B_\eta(c^{j+1})\right)_R \times \left(B_\eta(c^{j+1})\right)_O && (6.165)\\ &\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.166)\\ &= T_\eta(c',c) && (6.167)\end{aligned}$$

using Corollaries 6.11, 6.12 and 6.13. Now consider

$$\begin{aligned} \pi(c^n, c^s) \;=\; & \bigcup_{t\in[0,1]} \left(\left(x_R^n + (x_R^b - x_R^n)t, y_R^b + (y_R^b - y_R^n)t\right), c_O^n\right) \\ & \cup \bigcup_{t\in[0,1]} \left(\left(x_O^b + (x_R^e - x_R^b)t, y_R^b + (y_R^e - y_R^b)t\right), \right. \\ & \qquad \left. \left(x_O^n + (x_O^s - x_O^n)t, y_O^n + (y_O^s - y_O^n)t\right)\right) && (6.168)\\ & \cup \bigcup_{t\in[0,1]} \left(\left(x_R^e + (x_R^s - x_R^e)t, y_O^s - (y_R^e - y_O^s)t\right), c_O^s\right) \\ \subseteq \; & (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.169)\\ = \; & T_\eta(c',c) && (6.170)\end{aligned}$$

using Lemma 6.15 and Corollary 6.12.

■

Now we have shown that Assumption 6.2 holds for each possible segment of the trajectories returned by the empty space planner and we must just show that it holds for whole trajectories. The proof of this is similar to Lemma 6.6.

Theorem 6.19 (Assumption 6.2 Holds): For all $\eta > 0$, for all $c', c \in X$, for some $\lambda \in (0,1]$ and a set of subspaces $W_\delta(c',c) = \{W^0, ..., W^m\}$, for some $\epsilon > 0$, $B_\epsilon(c') \in W^0$, $W^m \subseteq B_\eta(c)$, and for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$ then for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$, and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least λ.

Proof: Firstly assume $c'_O = c_O$. Then this follows directly from Lemma 6.17.

Now assume $c'_O \neq c_O$. Let $c^p = ((x'_O, y'_O), c'_O)$ and let $c^e = ((x_O, y_O), c_O)$. Then

$$\pi(c', c) = \pi(c', c^p) \cup \pi(c^p, c^e) \cup \pi(c^e, c) \tag{6.171}$$

and $c^p \in \pi(c', c)$ and $c^e \in \pi(c^b, c^e)$. Because $c_O^e = c_O$, by Lemma 6.17, we can choose $\{D^0, ..., D^{d_{\max}}\}$ such that for some $\delta_D \in (0, \eta]$, $B_{\delta_D}(c^e) \subseteq D^0$, $D^{d_{\max}} \subseteq B_\eta(c)$, and for all $j < d_{\max}$, for all $c^j \in D^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$,

$f_R(c^n, c^s) \in D^{j+1}$, $\pi(c^n, f_R(c^n, c^s)) \subseteq T_\eta(c^e, c) \subset T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^R_\eta(c^e, c)$.

Since $c^p_R = c^p_O$ and $c^e_R = c^e_O$, by Lemma 6.18, we can choose $\{P^0, ..., P^{p_{\max}}\}$ such that for some $\delta_P \in (0, \delta_D]$, $B_{\delta_P}(c^p) \subseteq P^0$, $P^{p_{\max}} \subseteq B_{\delta_D}(c^e)$, and for all $j < p_{\max}$, for all $c^j \in P^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$, for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} P^l$, $\pi(c^n, f(c^n, c^s)) \subseteq T_{\delta_D}(c^p, c^e) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^O_{\delta_D}(c^p, c^e)$.

Lastly, since $c'_O = c^p_O$, Lemma 6.17 gives that we can choose $\{A^0, ..., A^{a_{\max}}\}$ such that for some $\delta_A \in (0, \delta_P]$, $B_{\delta_A}(c') \subseteq A^0$, $A^{a_{\max}} \subseteq B_{\delta_P}(c^p)$, and for all $j < a_{\max}$, for all $c^j \in A^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$, $f_R(c^n, c^s) \in A^{j+1}$, $\pi(c^n, f_R(c^n, c^s)) \subseteq T_{\delta_P}(c', c^p) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^R_{\delta_P}(c', c^p)$.

Now let

$$\{W^0, ..., W^m\} = \{A^0, ..., A^{a_{\max}-1}, P^0, ..., P^{p_{\max}-1}, D^0, ..., D^{d_{\max}}\}. \tag{6.172}$$

We have $B_{\delta_A}(c') \subseteq W^0$ and $W^m \subseteq B_\eta(c)$. Moreover, $A^{a_{\max}} \subseteq B_{\delta_P}(c^p) \subseteq P^0$ and $P^{p_{\max}} \subseteq B_{\delta_D}(c^e) \subseteq D^0$. Therefore for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$ then for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$ and the probability of sampling from $S^j(c^j)$ is at least $\min\left[\lambda^R_{\delta_P}(c', c^p), \lambda^O_{\delta_D}(c^p, c^e), \lambda^R_\eta(c^e, c)\right] > 0$.

■

Thus Assumption 6.2 holds for the case of a point robot and object. We now do a more complicated example with a non-prehensile manipulation primitive.

6.2.3 Disc Pushing

In this example, we consider a point robot pushing a disc of radius O_R. The robot can only push the disc in the four cardinal directions from four points on the disc as shown in Figure 6.4. Note that the object can move to a new configuration that is not along a line parallel to a principal axis, but this requires the robot push the object twice (Figure 6.4f). For simplicity, we allow the robot and object to overlap. This example requires projection functions that respect the interaction between the robot and object as well as projections onto the robot's space alone.

Projection Functions

We use three projection functions. Either the robot moves by itself or it pushes the object parallel to one of the principal axes.

- $f_R(c', c) = (c_R, c'_O)$
- $f_x(c', c) = (c_R, (x_O, y'_O))$
- $f_y(c', c) = (c_R, (x'_O, y_O))$

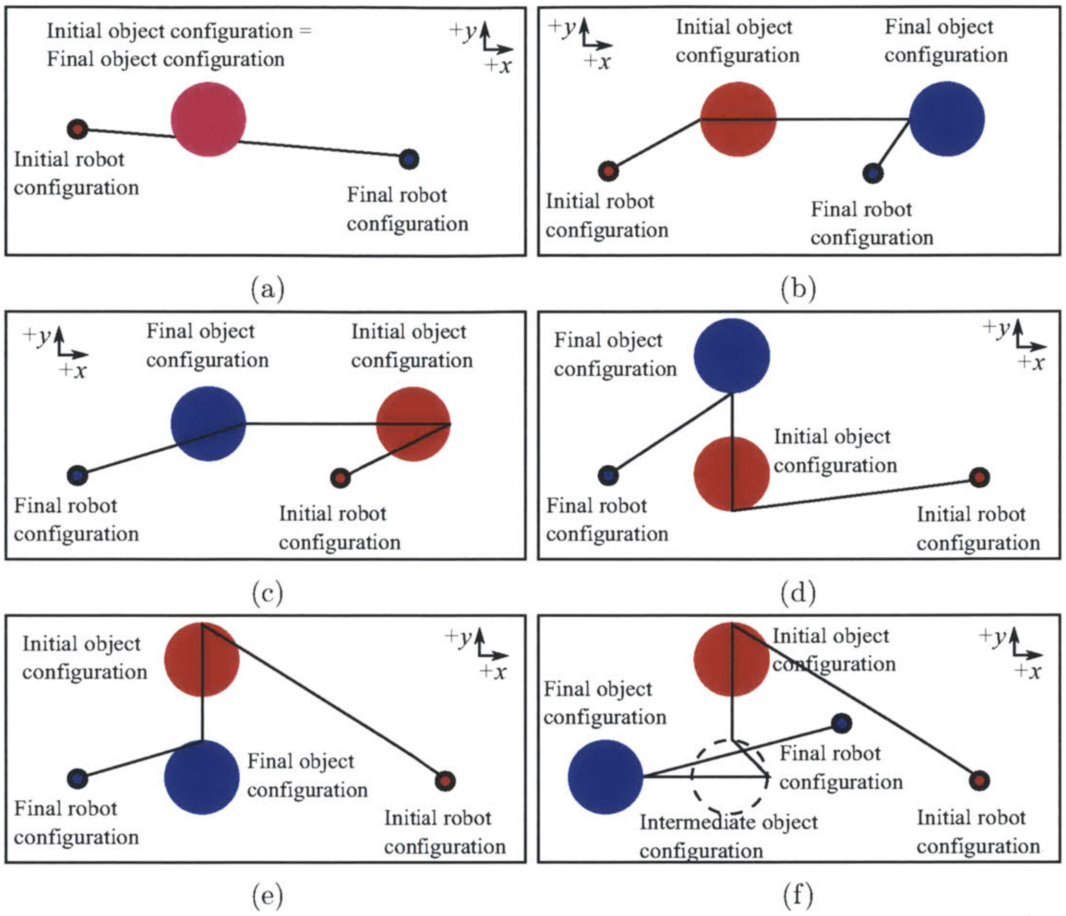

Initial object configuration = Final object configuration
+y
+x
Initial robot configuration
Final robot configuration
(a)
+y
+x
Initial object configuration
Final object configuration
Initial robot configuration
Final robot configuration
(b)
+y
+x
Final object configuration
Initial object configuration
Final robot configuration
Initial robot configuration
(c)
Final object configuration
+y
+x
Initial object configuration
Final robot configuration
Initial robot configuration
(d)
Initial object configuration
+y
+x
Final object configuration
Final robot configuration
Initial robot configuration
(e)
Initial object configuration
+y
+x
Final object configuration
Final robot configuration
Intermediate object configuration
Initial robot configuration
(f)

Figure 6.4: A robot pushing a disc. The robot is a point shown as a filled disc outlined in black and the object is a solid disc. Trajectories are shown from the red configuration to the blue configuration. (a) When the object's initial configuration is the same as its final configuration, the robot moves in a single straight line from its initial configuration to its final configuration. The robot and object are allowed to overlap. (b) When the object's final configuration has a larger x value than its initial configuration, the robot can push in the $+x$ direction by moving to the smallest x coordinate on the object and pushing in the $+x$ direction. (c) When the object's final configuration has a smaller x value than its initial configuration, the robot can push in the $-x$ direction by moving to the smallest x coordinate on the object and pushing in the $-x$ direction. (d) When the object's final configuration has a larger y value than its initial configuration, the robot can push in the $+y$ direction by moving to the smallest y coordinate on the object and pushing in the $+y$ direction. (e) When the objects' final configuration has a smaller y value than its initial configuration, the robot can push in the $-y$ direction by moving to the largest y coordinate on the object and pushing in the $-y$ direction. (f) If the object's final configuration is not along a line parallel to a principal axis from its initial configuration, the trajectory from initial configuration to final configuration requires the robot to push the object twice (the object's position after the first push is shown as a dashed outline). The empty space planner only plans for single pushes; trajectories like these can be created using multiple instances of the empty space planner.

Distance Functions

We use the Euclidean distance metric for the robot space R and the object space O. Our full distance function ρ is the maximum distance in either subspace. For $c', c \in X$

$$\rho_R(c'_R, c_R) = \sqrt{(x_R - x'_R)^2 + (y_R - y'_R)^2} \tag{6.173}$$

$$\rho_O(c'_O, c_O) = \sqrt{(x_O - x'_O)^2 + (y_O - y'_O)^2} \tag{6.174}$$

$$\rho(c', c) = \max\Big[\rho_R(c'_R, c_R), \rho_O(c'_O, c_O)\Big]. \tag{6.175}$$

The Euclidean distance metric is known to be a metric so ρ_R and ρ_O are full metrics with symmetry and the triangle inequality. The maximum of two metrics is also a metric so ρ itself is a metric, but this is not be necessary for the proofs.

Free Space

Let C_{obs} be all points on an obstacle or on the boundary of an obstacle or on the boundary of the space. We define $M_{i,free}$ as the configurations for which component i is not in contact with the obstacles:

$$M_{R,free} = \{r \in R | \inf_{p \in C_{obs}} \rho_R(r, p) > 0\} \tag{6.176}$$

$$M_{O,free} = \{o \in O | \inf_{p \in C_{obs}} \rho_O(o, p) > O_R\} \tag{6.177}$$

$$X_{restricted} = X \tag{6.178}$$

$$X_{free} = M_{R,free} \times M_{O,free} \tag{6.179}$$

where Equation 6.179 follows from Equations 6.176- 6.178. We have $X_{restricted} = X$ because we allow overlap between the robot and object.

Note that since $X_{restricted} = X$, $B_\delta(c) = X_\delta(c)$ and $T_\delta(c', c) = Q_\delta(c', c)$ in the notation of Section 6.2.1. Therefore we can apply the lemmas in Section 6.2.1 directly in the proofs of Assumptions 6.1 and 6.2.

We also require that R and O are both convex.

Sampling the Space

Since $X = X_{restricted}$, SAMPLECONFIGURATIONSPACE need only select a random configuration from X.

Empty Space Planner

Now we define the empty space planner. The empty space planner always operates on input of the form $(c', f(c', c))$ for $c', c \in X$ and $f \in F$. Therefore, the empty space planner need only be defined for the domain

$$\left\{(c', c) \in X_{free} \times X \,\middle|\, \exists f \in F, c^i \in X \text{ s.t. } f(c', c^i) = c\right\} = \tag{6.180}$$
$$\left\{(c', c) \in X_{free} \times X \,\middle|\, c'_O = c_O \text{ or } y'_O = y_O \text{ or } x'_O = x_O\right\}. \tag{6.181}$$

This means the empty space planner plans for at most one push. If the object's initial and final configurations are the same, the empty space planner just returns a straight line in the robot's subspace from the robot's initial configuration to its final configuration. This is shown in Figure 6.4a. If the object's final configuration has a larger x value(smaller x value)(larger y value)(smaller y value) than its initial configuration, the empty space planner plans to move the robot to the smallest x coordinate(largest x coordinate)(smallest y coordinate)(largest y coordinate) on the object, plans a push in a straight line in the $+x(-x)(+y)(-y)$ direction, and finally plans a move to the robot's final configuration. This is shown in Figure 6.4b(Figure 6.4c)(Figure 6.4d)(Figure 6.4e).

Specifically, on input (c', c), the planner returns the following:

$c'_O = c_O$ Move the robot in a single straight line from c'_R to c_R. This trajectory is

$$\tau(t) = \left(\left(x'_R + (x_R - x'_R)t, y'_R + (y_R - y'_R)t\right), c'_O\right). \tag{6.182}$$

$y'_O = y_O$ In this case, we must decide whether the push the object right or left. Recall that the radius of the object is O_R.

$x'_O < x_O$ The robot moves in a single straight line to the smallest x coordinate on the object. It then pushes the object directly right to x_O and finally moves in a straight line to its final configuration. The trajectory is

$$\tau(t) = \begin{cases} \left(\left(x'_R + (x'_O - O_R - x'_R)3t, y'_R + (y'_O - y'_R)3t\right), c'_O\right) & \text{if } t \leq \frac{1}{3} \\ \left(\left(x'_O - O_R + (x_O - x'_O)(3t-1), y'_O\right), \left(x'_O + (x_O - x'_O)(3t-1), y'_O\right)\right) & \text{if } \frac{1}{3} < t \leq \frac{2}{3} \\ \left(\left(x_O - O_R + (x_R - (x_O - O_R))(3t-2), y'_O + (y_R - y'_O)(3t-2)\right), c_O\right) & \text{if } \frac{2}{3} < t \leq 1 \end{cases} \tag{6.183}$$

$x'_O > x_O$ The robot moves in a single straight line to the largest x coordinate on the object. It then pushes the object directly left to x_O and finally moves in a straight line to its final configuration. The trajectory is

$$\tau(t)=\begin{cases}\left(\left(x'_R+(x'_O+O_R-x'_R)3t,y'_R+(y'_O-y'_R)3t\right),c'_O\right) & \text{if } t\leq\frac{1}{3}\\ \left(\left(x'_O+O_R+(x_O-x'_O)(3t-1),y'_O\right),\left(x'_O+(x_O-x'_O)(3t-1),y'_O\right)\right) & \text{if } \frac{1}{3}<t\leq\frac{2}{3}\\ \left(\left(x_O+O_R+(x_R-(x_O+O_R))(3t-2),y'_O+(y_R-y'_O)(3t-2)\right),c_O\right) & \text{if } \frac{2}{3}<t\leq 1\end{cases} \tag{6.184}$$

$x'_O = x_O$ In this case, we must decide whether the push the object up or down. Recall that the radius of the object is O_R.

$y'_O < y_O$ The robot moves in a single straight line to the smallest y coordinate on the object. It then pushes the object directly upwards to y_O and finally moves in a straight line to its final configuration. The trajectory is

$$\tau(t)=\begin{cases}\left(\left(x'_R+(x'_O-x'_R)3t,y'_R+(y'_O-O_R-y'_R)3t\right),c'_O\right) & \text{if } t\leq\frac{1}{3}\\ \left(\left(x'_O,y'_O-O_R+(y_O-y'_O)(3t-1)\right),\left(x'_O,y'_O+(y_O-y'_O)(3t-1)\right)\right) & \text{if } \frac{1}{3}<t\leq\frac{2}{3}\\ \left(\left(x'_O+(x_R-x'_O)(3t-2),y_O-O_R+(y_R-(y_O-O_R))(3t-2)\right),c_O\right) & \text{if } \frac{2}{3}<t\leq 1\end{cases} \tag{6.185}$$

$y'_O > y_O$ The robot moves in a single straight line to the largest y coordinate on the object. It then pushes the object directly downwards to y_O and finally moves in a straight line to its final configuration. The trajectory is

$$\tau(t)=\begin{cases}\left(\left(x'_R+(x'_O-x'_R)3t,y'_R+(y'_O+O_R-y'_R)3t\right),c'_O\right) & \text{if } t\leq\frac{1}{3}\\ \left(\left(x'_O,y'_O+O_R+(y_O-y'_O)(3t-1)\right),\left(x'_O,y'_O+(y_O-y'_O)(3t-1)\right)\right) & \text{if } \frac{1}{3}<t\leq\frac{2}{3}\\ \left(\left(x'_O+(x_R-x'_O)(3t-2),y_O+O_R+(y_R-(y_O+O_R))(3t-2)\right),c_O\right) & \text{if } \frac{2}{3}<t\leq 1\end{cases} \tag{6.186}$$

Because these trajectories consist only of straight lines within R or O, R and O are convex, and the starting configuration is within free space (so the object must be more than O_R from any boundary), we can guarantee that they always remain within $X_{restricted} = X$. Therefore, this is an empty space planner.

Proof that Assumption 6.1 Holds

We first show that X_{free} is open.

Theorem 6.20 (Assumption 6.1 Holds): For all $c \in X_{free}$, for some $\delta > 0$, $B_\delta(c) \subseteq X_{free}$.

Proof: Choose $c \in X_{free}$. Let

$$\begin{aligned} \delta_R &= \inf_{p \in C_{obs}} \rho_R(c_R, p) && (6.187)\\ \delta_O &= \inf_{p \in C_{obs}} \rho_O(c_O, p) - O_R && (6.188)\\ \delta &= \frac{1}{2}\min\left[\delta_R, \delta_O\right]. && (6.189)\\ & && (6.190)\end{aligned}$$

By definition of X_{free}, $\delta_R > 0$ and $\delta_O > 0$ so $\delta > 0$. Choose $c' \in B_\delta(c)$. By the triangle inequality for any $p \in C_{obs}$,

$$\begin{aligned} \rho_R(p, c'_R) &\geq \rho_R(p, c_R) - \rho_R(c'_R, c_R) && (6.191)\\ &> \delta_R - \delta && (6.192)\\ &\geq \frac{1}{2}\delta_R && (6.193)\\ &> 0 && (6.194)\\ \rho_O(p, c'_O) &\geq \rho_O(p, c_O) - \rho_O(c'_O, c_O) && (6.195)\\ &> \delta_O + O_R - \delta && (6.196)\\ &\geq \frac{1}{2}\delta_O + O_R && (6.197)\\ &> O_R. && (6.198)\end{aligned}$$

Therefore $c' \in M_{R,free} \times M_{O,free} = X_{free}$ so $B_\delta(c) \subseteq X_{free}$.

■

Proof that Assumption 6.2 Holds

The proof that Assumption 6.2 holds when the robot moves by itself in the case of a point robot and object is given in Lemma 6.17. Since the trajectory the robot and object take when the object does not move are the same in this example, the proof that Assumption 6.2 holds for this example is identical.

The proof that Assumption 6.2 holds for the trajectories in which the robot pushes the object is slightly more complicated. We only do the proof in the case that $x'_O = x_O$ and $y'_O < y_O$ as it is clear that the other three cases are analogous.

We are trying to create a sequence of regions $\{W^0, ..., W^m\}$ such that if there is a configuration $c^j \in W^j$ in the tree, there is a constant probability of adding a configuration in $\bigcup_{l>j} W^l$. In the case where the moving components could move instantaneously in any direction (Lemmas 2.4, 6.17, and 6.18), we used open balls for the W^j. However, in this case, the object cannot move instantaneously in any direction even when the robot is in contact with it. We need the shape of the W^j to reflect this restriction. For instance, assume that there is a collision free path for the object from c'_O to c_O where $x'_O = x_O$ and $y'_O < y_O$. Then, if we have some configuration in the tree at $(x^j_O, y^j_O) \in W^j_O$ where x^j_O is near x'_O and $y'_O \leq y^j_O \leq y_O$, we want to add a configuration with a y value of at least $y_O^j + s$ for some finite s. This allows us to move "up" along the path. However, the x value of the added configuration only needs to be near x'_O. Thus it makes more sense for the W_O^j to be rectangular.

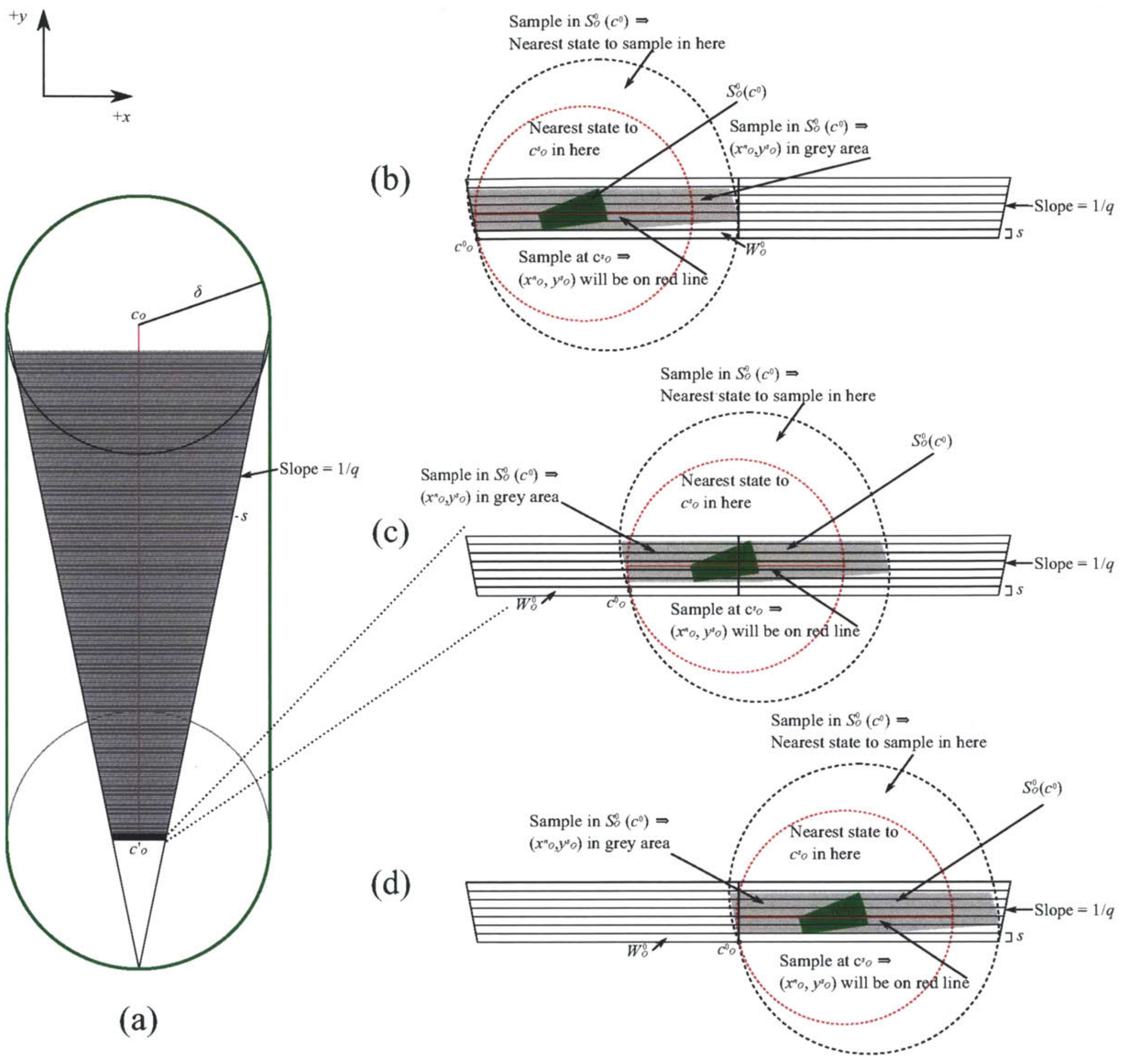
In fact, our W_O^j's are trapezoidal. This is because when extending from a configuration in the tree, we do not extend directly towards the sampled configuration. Assume we have some configuration $c_O^j \in W_O^j$ in the tree and we sample a configuration c_O^s. Let c_O^n be any configuration no farther from c_O^s than c_O^j. In the case of $x'_O = x_O$ and $y'_O < y_O$, we try to extend towards $f_y(c^n, c^s) = (c_R^s, (x_O^n, y_O^s))$. Therefore, we both need to ensure that $y_O^s > y_O^j + s$ *and* that x_O^n is within the tube of radius δ from c'_O to c_O. This raises the same problem we had in Section 6.2.2: If x_O^j is a distance of ζ from x'_O then x_O^n could be as far as $\zeta + \epsilon$ from x'_O (see Figure 6.3). Therefore, the width of the W_O^j's needs to expand as we move upwards. The amount it can expand by is governed by the slope of the line connecting $(x'_O, y'_O - \delta)$ to $(x'_O + \delta, y_O)$ as shown in Figure 6.5.

Now assume we have some c_O^j in one of these trapezoidal W_O^j. We need to sample a configuration c_O^s that ensures that for any c_O^n no farther from c_O^s than c_O^j, $(x_O^n, y_O^s) \in \bigcup_{l>j} W_O^l$. Assume the distance from the bottom line of W_O^j to the bottom line of W_O^{j+1} is s. We always want $y_O^s - y_O^j \geq s$. Let ρ be the distance from c_O^j to c_O^n. Assume c_O^j is near the left edge of W_O^j as shown in part (b) of Figure 6.5. Let the amount the bottom line of each W_O^j expands be $2q_x$. If the sampled configuration is a distance of $y_O^s - y_O^j$ above c_O^j we must ensure that $x_O^n > x_O^s - \rho > -q_x/s(y_O^s - y_O^j)$. This gives us an upper bound on the angle between c_O^j and c_O^s while the need to move a distance of at least s gives us a lower bound. Putting the two bounds together gives us that the region from which we sample, $S_O^j(c_O^j)$, should be a rectangle in polar coordinates. This region is shown in green in Figure 6.5. As we move c_O^j right (along the $+x$ axis) in W_O^j, the same relationship holds until we reach the center with $x_O^j = x'_O$. Once we cross the center line, we must ensure that $x_O^s < x_O^j$. Therefore, S_O^j is split into two cases: one when $x_O^j \leq x'_O$ and one when $x_O^j > x'_O$. We only analyze the $x_O^j \leq x'_O$ case since the other one is simply the mirror image.

There is one more technicality: we must make sure that the final W^j is within $B_\delta(c)$. We do this by setting $W^m = B_\delta(c)$ and ensuring that s is small enough that it is not possible to "miss" $B_\delta(c)$ entirely.

Figure 6.5 shows the important geometric quantities of the following lemma visually.

Lemma 6.21: For $c', c \in X$, let $x'_O = x_O = x'_R = x_R$, $y'_O < y_O$, $y'_R = y'_O - O_R$ and $y_R = y_O - O_R$. Let $c^e = (c'_R, c_O)$. Then for all $\eta > 0$, for some sequence $\{W^0, ..., W^m\}$, for some $\epsilon \in (0, \eta]$, $B_\epsilon(c') \subseteq W^0$, $W^m \subseteq B_\eta(c^e)$, and for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$, for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$, and for some $\lambda_\eta^O(c', c) > 0$, the probability of sampling from $S^j(c^j)$ is at least $\lambda_\eta^O(c', c)$.



+y
+x
(a)
c_O
δ
Slope = 1/q
s
c'_O
(b)
(c)
(d)
Sample in $S^0_O(c^0) \Rightarrow$ Nearest state to sample in here
$S^0_O(c^0)$
Nearest state to c^s_O in here
Sample in $S^0_O(c^0) \Rightarrow$ (x^n_O, y^s_O) in grey area
Slope = 1/q
s
c^0_O
W^0_O
Sample at $c^s_O \Rightarrow$ (x^n_O, y^s_O) will be on red line

Figure 6.5: Assume there is a collision free path for the object from c'_O to c_O (magenta line). (a) Because the space is open, we can put a tube (green solid line) of radius δ around this path and guarantee that anywhere in the tube is collision free. Within the tube, the W^j_O are outlined in gray. The line bordering them has a slope of $1/q$ and W^j_O is a distance of s below W^{j+1}_O in the y direction. Assume we have a configuration at $c^0_O \in W^0$ with $x^0_O \leq x'_O$. Figures (b)-(d) show close-ups of $\{W^0_O, ..., W^6_O\}$ with c^0_O in a different place in W^0_O in each. $S^0_O(c^0)$ is shown as a green area. Regardless of the location of c^0_O, S^0_O is always the same size so it always has the same measure. If a configuration c^s_O is sampled in S^0_O, let $c^n_O \in O$ be any configuration no farther from c^s_O than c^0_O. Then c^s_O is within the dotted black line. This dotted black line is clearly within the outer green tube so c^n_O is in free space. If $y^n_O > y^s_O$ then c^n_O is already in $\bigcup_{l>0} W^l_O$. Otherwise, (x^n_O, y^s_O) falls into the gray region, which is clearly contained in $\bigcup_{l>0} W^l_O \subset (T_\eta(c', c))_O$. Therefore, there is a collision free path from c^n_O to (x^n_O, y^s_O). For example, if we choose c^s_O at the location shown by the red dot then the configurations no farther from c^s_O than c^0_O are all within the red dashed circle. Choose c^n_O from within this circle. If $y^n_O > c^s_O$ then $c^n_O \in \bigcup_{l>0} W^l_O$. Otherwise, (x^n_O, y^s_O) is along the red line, which is contained in $\bigcup_{l>0} W^l_O$. Therefore, if we start with a configuration in W^0_O, we have a constant probability of adding a configuration in $\bigcup_{l>0} W^l$.

Proof: Firstly note that

$$\rho(c', c) = \max\Big[\rho_R(c'_R, c_R), \rho_O(c'_O, c_O)\Big] \tag{6.199}$$
$$= \max\Big[y_R - y'_R, y_O - y'_O\Big] \tag{6.200}$$
$$= y_R - y'_R \tag{6.201}$$
$$= y_O - y'_O. \tag{6.202}$$

Let

$$\delta = \min\left[\eta, \frac{\rho(c', c)}{2}\right] \tag{6.203}$$
$$q = \frac{\delta}{\rho(c', c) + \delta} \tag{6.204}$$
$$s = \frac{q^2\delta}{6} \tag{6.205}$$
$$j_{\max} = \left\lceil \frac{\rho(c', c) - q\delta}{s} \right\rceil + 1. \tag{6.206}$$

Note that $q \leq \frac{1}{3} < 1$. For $0 \leq j < j_{\max}$, we define

$$W_R^j = \left\{ r \in R \,\middle|\, \rho_R(c'_R, r) < (j+1)\frac{qs}{2} \right\} \tag{6.207}$$
$$W_O^j = \left\{ (x, y) \in O \,\middle|\, \begin{array}{c} y'_O + s\left(j - \frac{1}{2}\right) < y \leq y'_O + s\left(j + \frac{1}{2}\right) \text{ and} \\ x'_O - q(y - (y'_O - \delta)) < x < x'_O + q(y - (y'_O - \delta)) \end{array} \right\} \tag{6.208}$$
$$W^j = W_R^j \times W_O^j, \tag{6.209}$$

and we let $W^{j_{\max}} = B_\delta(c^e) \subseteq B_\eta(c^e)$. In Figure 6.5, the W^j are shown outlined by gray lines. Each W^j is trapezoidal in shape. The top and bottom of the trapezoid are parallel to the x axis and separated along the y axis by a distance of s. The top of each the trapezoid is longer than the bottom; the slope of the line connecting the bottom right corner of the trapezoid to the top right is $1/q$ while the slope of the line connecting the bottom left to the top left is $-1/q$.

We first show that for some $\epsilon \in (0, \eta]$, $B_\epsilon(c') \subseteq W^0$. We choose $\epsilon = \frac{qs}{2} < \eta$ so $W_R^0 = (B_\epsilon(c'))_R$. Choose $(x_O^b, y_O^b) \in (B_\epsilon(c'))_O$. Then $y'_O - \frac{qs}{2} < y^b < y'_O + \frac{qs}{2}$ and $x'_O - \frac{qs}{2} < x^b < x'_O + \frac{qs}{2}$. Now $q \leq \frac{1}{3}$ so

$$x'_O - q(y^b - (y'_O - \delta)) < x'_O - q\left(y'_O - \frac{qs}{2} - (y'_O - \delta)\right) \tag{6.210}$$
$$= x'_O - q\delta + \frac{q^2 s}{2} \tag{6.211}$$
$$< x'_O - q\frac{q^2\delta}{6} + \frac{qs}{2} \tag{6.212}$$
$$= x'_O - qs + \frac{qs}{2} \tag{6.213}$$

$$= x'_O - \frac{qs}{2} \tag{6.214}$$
$$< x^b \tag{6.215}$$

and

$$x'_O + q(y^b - (y'_O - \delta)) > x'_O + q\left(y'_O - \frac{qs}{2} - (y'_O - \delta)\right) \tag{6.216}$$
$$= x'_O + q\delta - \frac{q^2 s}{2} \tag{6.217}$$
$$> x'_O + q\frac{q^2\delta}{6} - \frac{qs}{2} \tag{6.218}$$
$$= x'_O + qs - \frac{qs}{2} \tag{6.219}$$
$$= x'_O + \frac{qs}{2} \tag{6.220}$$
$$> x^b. \tag{6.221}$$

Therefore
$$x'_O - q(y^b - (y'_O - \delta)) < x^b < x'_O + q(y^b - (y'_O - \delta)). \tag{6.222}$$
Since $\frac{qs}{2} < \frac{s}{2}$, we have $y'_O - \frac{s}{2} < y^b \leq y'_O + \frac{s}{2}$. Therefore $(x^b, y^b) \in W^0_O$ so $(B_\epsilon(c'))_O \subseteq W^0_O$. Thus

$$W^0 = W^0_R \times W^0_O \tag{6.223}$$
$$\supseteq (B_\epsilon(c'))_R \times (B_\epsilon(c'))_O \tag{6.224}$$
$$= B_\epsilon(c') \tag{6.225}$$

using Lemma 6.9 and Corollary 6.11.

We also show that $W^j \subseteq T_\eta(c', c)$ for all j as that is helpful for the rest of the proof. Firstly consider $W^{j_{\max}} = B_\delta(c^e)$. Then

$$W^{j_{\max}}_R = (B_\delta(c^e))_R \tag{6.226}$$
$$= (B_\delta(c'))_R \tag{6.227}$$
$$\subseteq (B_\eta(c'))_R \tag{6.228}$$
$$\subseteq (T_\eta(c', c))_R \tag{6.229}$$

by Corollary 6.13 and

$$W^{j_{\max}}_O = (B_\delta(c^e))_O \tag{6.230}$$
$$= (B_\delta(c))_O \tag{6.231}$$
$$\subseteq (B_\eta(c'))_O \tag{6.232}$$
$$\subseteq (T_\eta(c', c))_O \tag{6.233}$$

also by Corollary 6.13. Therefore

$$\begin{aligned} W^{j_{\max}} &= W_R^{j_{\max}} \times W_O^{j_{\max}} && (6.234)\\ &\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.235)\\ &= T_\eta(c',c) && (6.236)\end{aligned}$$

using Lemma 6.9 and Corollary 6.12. Now consider $j < j_{\max}$. Let $r \in W_R^j$. Then using that $q = \frac{\delta}{\rho(c',c)+\delta}$ and $j+1 \le j_{\max}$,

$$\begin{aligned} \rho_R(c'_R, r) &< (j+1)\frac{qs}{2} && (6.237)\\ &\le \left(\left\lceil \frac{\rho(c',c)-q\delta}{s}\right\rceil + 1\right)\frac{qs}{2} && (6.238)\\ &\le \left(\frac{\rho(c',c)-q\delta}{s} + 1 + 1\right)\frac{qs}{2} && (6.239)\\ &= \frac{1}{2}\frac{\delta}{\rho(c',c)+\delta}(\rho(c',c)-q\delta) + qs && (6.240)\\ &< \frac{\delta}{2} + \frac{q^3\delta}{6} && (6.241)\\ &< \delta && (6.242)\\ &\le \eta && (6.243)\end{aligned}$$

so $W_R^j \subseteq (B_\eta(c'))_R \subseteq (T_\eta(c',c))_R$ by Corollary 6.13. Now let $c_O^j \in W_O^j$. Then using that $\rho(c',c) = y_O - y'_O$ and $q < 1$,

$$\begin{aligned} y_O^j &\le y'_O + s\left(j + \frac{1}{2}\right) && (6.244)\\ &\le y'_O + s\left(\left\lceil \frac{\rho(c',c)-q\delta}{s}\right\rceil + \frac{1}{2}\right) && (6.245)\\ &\le y'_O + s\left(\frac{\rho(c',c)-q\delta}{s} + 1 + \frac{1}{2}\right) && (6.246)\\ &= y'_O + \rho(c',c) - q\delta + \frac{3}{2}s && (6.247)\\ &= y_O - q\delta + \frac{3}{2}s && (6.248)\\ &= y_O - q\delta + \frac{q^2\delta}{4} && (6.249)\\ &< y_O. && (6.250)\end{aligned}$$

Now, by our choice of c' and c we have

$$\pi(c',c) = \bigcup_{t\in[0,1]} \left(\left(x'_O+(x_O-x'_O)t, y'_O-O_R+(y_O-y'_O)t\right), \left(x'_O+(x_O-x'_O)t, y'_O+(y_O-y'_O)t\right)\right) \tag{6.251}$$
$$= \bigcup_{t\in[0,1]} \left(\left(x'_O, y'_O - O_R + (y_O - y'_O)t\right), \left(x'_O, y'_O + (y_O - y'_O)t\right)\right) \tag{6.252}$$
$$= \left\{c^t \in X \,\middle|\, x^t_R = x^t_O = x'_O \text{ and } y'_O \le y^t_O \le y_O \text{ and } y^t_R = y^t_O - O_R\right\}. \tag{6.253}$$

Therefore, if $y^j_O \ge y'_O$, there is some $c^t \in \pi(c',c)$ with $x^t_O = x'_O$ and $y^t_O = y^j_O$. Then

$$\rho_O(c^t_O, c^j_O) = \left|x'_O - x^j_O\right| \tag{6.254}$$
$$< q(y^j_O - (y'_O - \delta)) \tag{6.255}$$
$$< q(y_O - y'_O + \delta) \tag{6.256}$$
$$= \frac{\delta}{\rho(c',c)+\delta}(\rho(c',c)+\delta) \tag{6.257}$$
$$= \delta \tag{6.258}$$
$$\le \eta. \tag{6.259}$$

Therefore $c^j_O \in (B_\eta(c^t))_O \subseteq T_\eta(c',c)$ by Corollary 6.13. If $y^j < y'_O$ then $j = 0$ and $y'_O - y^0_O < \frac{s}{2} = \frac{q^2\delta}{12}$. Therefore

$$\rho_O(c'_O, c^0_O) = \sqrt{(x'_O - x^0_O)^2 + (y'_O - y^0_O)^2} \tag{6.260}$$
$$< \sqrt{(q(y^0_O - (y'_O - \delta)))^2 + (y'_O - y^0_O)^2} \tag{6.261}$$
$$= \sqrt{q^2(\delta - (y'_O - y^0_O))^2 + (y'_O - y^0_O)^2} \tag{6.262}$$
$$= \sqrt{q^2\delta^2 - 2q^2\delta(y'_O - y^0_O) + (1+q^2)(y'_O - y^0_O)^2} \tag{6.263}$$
$$= \sqrt{q^2\delta^2 - (y'_O - y^0_O)(2q^2\delta - (1+q^2)(y'_O - y^0_O))} \tag{6.264}$$
$$< \sqrt{q^2\delta^2 - (y'_O - y^0_O)\left(2q^2\delta - (1+q^2)\frac{q^2\delta}{12}\right)} \tag{6.265}$$
$$= \sqrt{q^2\delta^2 - (y'_O - y^0_O)q^2\delta\left(2 - \frac{1+q^2}{12}\right)} \tag{6.266}$$
$$< \sqrt{q^2\delta^2 - (y'_O - y^0_O)q^2\delta\left(2 - \frac{1}{6}\right)} \tag{6.267}$$
$$< q\delta \tag{6.268}$$
$$< \delta \tag{6.269}$$
$$\le \eta \tag{6.270}$$

so $c_O^0 \in (B_\eta(c'))_O \subseteq (T_\eta(c',c))_O$ by Corollary 6.13. Therefore $W_O^j \subseteq (T_\eta(c',c))_O$ and we have

$$\begin{aligned} W^j &= W_R^j \times W_O^j && (6.271)\\ &\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O && (6.272)\\ &= T_\eta(c',c) && (6.273)\end{aligned}$$

by Lemma 6.9 and Corollary 6.12.

Now we are ready to show that for all $j < j_{\max}$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$ then for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$, $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is greater than zero. Let $j < j_{\max}$ and let $c^j \in W^j$. Our $S_O^j(c^j)$ are rectangles in polar coordinates:

$$S_R^j(c^j) = \left\{ r \in R \,\middle|\, \rho_R(c_R^j, r) < \frac{qs}{2} \right\} \qquad (6.274)$$

$$S_O^j(c^j) = \begin{cases} \left\{ (x,y) \in O \,\middle|\, \begin{array}{c} \frac{q\delta}{4} < \rho_O(c_O^j, (x,y)) < \frac{q\delta}{2} \text{ and} \\ \frac{2q}{3} < \frac{y - y_O^j}{\rho_O(c_O^j,(x,y))} < \frac{2q}{1+q^2} \text{ and } x \geq x_O^j \end{array} \right\} & \text{if } x_O^j \leq x'_O \\ \left\{ (x,y) \in O \,\middle|\, \begin{array}{c} \frac{q\delta}{4} < \rho_O(c_O^j, (x,y)) < \frac{q\delta}{2} \text{ and} \\ \frac{2q}{3} < \frac{y - y_O^j}{\rho_O(c_O^j,(x,y))} < \frac{2q}{1+q^2} \text{ and } x \leq x_O^j \end{array} \right\} & \text{else} \end{cases} \qquad (6.275)$$

$$S^j(c^j) = S_R^j(c^j) \times S_O^j(c^j). \qquad (6.276)$$

Now $\mu_R(S_R^j(c^j))$ is just the measure of an open ball of radius $\frac{qs}{2}$. Therefore, $\frac{\mu_R(S_R^j(c^j))}{\mu_R(R)} = \frac{\mu_R(S_R^0(c'))}{\mu_R(R)} = p_R$ is independent of c^j or j. Now consider $\mu_O(S_O^j(c^j))$. For $c_O^s \in O$, define

$$\theta(c_O^s, c_O^j) = \tan^{-1}\left(\frac{y_O^s - y_O^j}{x_O^s - x_O^j}\right). \qquad (6.277)$$

Then

$$S_O^j(c^j) = \begin{cases} \left\{ c_O^s \in O \,\middle|\, \begin{array}{c} \frac{q\delta}{4} < \rho_O(c_O^j, c_O^s) < \frac{q\delta}{2} \text{ and} \\ \frac{2q}{3} < \sin(\theta(c_O^s, c_O^j)) < \frac{2q}{1+q^2} \text{ and } 0 \leq \theta \leq \frac{\pi}{2} \end{array} \right\} & \text{if } x_O^j \leq x'_O \\ \left\{ c_O^s \in O \,\middle|\, \begin{array}{c} \frac{q\delta}{4} < \rho_O(c_O^j, c_O^s) < \frac{q\delta}{2} \text{ and} \\ \frac{2q}{3} < \sin(\theta(c_O^s, c_O^j)) < \frac{2q}{1+q^2} \text{ and } \frac{\pi}{2} \leq \theta \leq \pi \end{array} \right\} & \text{else.} \end{cases} \qquad (6.278)$$

Now $q < 1$ so $0 < \frac{2q}{3} < q < q\frac{2}{1+q^2} < 1$. Therefore $S_O^j(c^j)$ is just a box in polar coordinates with a difference in radius of $\frac{q\delta}{4}$ and a difference in angle of $\sin^{-1}\left(\frac{2q}{1+q^2}\right) - \sin^{-1}\left(\frac{2q}{3}\right)$. Examples of this shape are shown in Figure 6.5. The measure of this box is greater than zero and independent of j and c^j. Let $p_O = \frac{\mu_O(S_O^0(c'))}{\mu_O(O)} = \frac{\mu_O(S_O^j(c^j))}{\mu_O(O)}$ be the probability of sampling from $S^j_O(c^j)$. By Lemma 6.14, the probability of sampling from $S^j(c^j)$ is $\lambda^O_\eta(c', c) = p_R p_O > 0$.

Now assume $x^j_O \leq x'_O$ (the case $x^j_O > x'_O$ is analogous so we only analyze $x^j_O \leq x'_O$) and assume we choose $c^s \in S^j(c^j)$. Now $0 < q < 1$ so

$$\begin{aligned}
\rho_R(c^j_R, c^s_R) &< \frac{qs}{2} && (6.279)\\
&< \frac{s}{2} && (6.280)\\
&= \frac{q^2\delta}{12} && (6.281)\\
&< \frac{q\delta}{4} && (6.282)\\
&< \rho_O(c^j_O, c^s_O). && (6.283)
\end{aligned}$$

Therefore

$$\begin{aligned}
\rho(c^j, c^s) &= \max\left[\rho_R(c^j_R, c^s_R), \rho_O(c^j_O, c^s_O)\right] && (6.284)\\
&= \rho_O(c^j_O, c^s_O) && (6.285)\\
&< \frac{\delta q}{2}. && (6.286)
\end{aligned}$$

Let c^n be any configuration with

$$\begin{aligned}
\rho(c^n, c^s) &\leq \rho(c^j, c^s) && (6.287)\\
&< \frac{\delta q}{2}. && (6.288)
\end{aligned}$$

We first show that $c^n_R \in (T_\eta(c', c))_R$. Since $c^j_R \in W^j_R$ and $j < j_{\max}$, we have

$$\rho_R(c^j_R, c'_R) < (j+1)\frac{qs}{2}. \qquad (6.289)$$

The triangle inequality gives

$$\begin{aligned}
\rho_R(c^n_R, c'_R) &\leq \rho_R(c^n_R, c^s_R) + \rho_R(c^s_R, c^j_R) + \rho_R(c^j_R, c'_R) && (6.290)\\
&< \frac{q\delta}{2} + \frac{qs}{2} + (j+1)\frac{qs}{2} && (6.291)\\
&\leq \frac{q\delta}{2} + \left(\left\lceil \frac{\rho(c', c) - q\delta}{s} \right\rceil + 2\right)\frac{qs}{2} && (6.292)\\
&\leq \frac{q\delta}{2} + \left(\frac{\rho(c', c) - q\delta}{s} + 1 + 2\right)\frac{qs}{2} && (6.293)\\
&= \frac{q}{2}(\rho(c', c) + \delta - q\delta) + \frac{3}{2}qs && (6.294)\\
&< \frac{1}{2}\frac{\delta}{\rho(c', c) + \delta}(\rho(c', c) + \delta) + \frac{3}{2}qs && (6.295)
\end{aligned}$$

$$= \frac{\delta}{2} + \frac{q^3\delta}{4} \tag{6.296}$$

$$< \delta \tag{6.297}$$

$$\leq \eta. \tag{6.298}$$

Therefore

$$c_R^n \in (B_\eta(c'))_R \subseteq (T_\eta(c',c))_R \tag{6.299}$$

by Corollary 6.13.

Now we show that $c_R^s \in W_R^{j+1}$. Consider

$$\rho_R(c_R^s, c_R') \leq \rho_R(c_R^s, c_R^j) + \rho_R(c_R^j, c_R') \tag{6.300}$$

$$< \frac{qs}{2} + (j+1)\frac{qs}{2} \tag{6.301}$$

$$= (j+2)\frac{qs}{2}. \tag{6.302}$$

So if $j+1 < j_{\max}$, $c_R^s \in W_R^{j+1} \subseteq (T_\eta(c',c))_R$. If $j+1 = j_{\max}$ then

$$\rho_R(c_R^s, c_R') < (j_{\max}+1)\frac{qs}{2} \tag{6.303}$$

$$= \left(\left\lceil \frac{\rho(c',c) - q\delta}{s} \right\rceil + 1 + 1\right)\frac{qs}{2} \tag{6.304}$$

$$\leq \left(\frac{\rho(c',c) - q\delta}{s} + 1 + 2\right)\frac{qs}{2} \tag{6.305}$$

$$= \frac{q}{2}(\rho(c',c) - q\delta) + \frac{3}{2}qs \tag{6.306}$$

$$< \frac{1}{2}\frac{\delta}{\rho(c',c)+\delta}(\rho(c',c)+\delta) + \frac{q^3\delta}{4} \tag{6.307}$$

$$= \frac{\delta}{2} + \frac{q^3\delta}{4} \tag{6.308}$$

$$< \delta. \tag{6.309}$$

Therefore $c_R^s \in (B_\delta(c'))_R = W_R^{j_{\max}} \subseteq (T_\eta(c',c))_R$.

Now we show some properties of c_O^s. This allows us both to show that $c_O^n \in (T_\eta(c',c))_O$ and also that the final configuration is in $\bigcup_{l>j} W^l$.

We first show the y coordinate of c_O^s is between y_O' and y_O. We have

$$y_O^s = y_O^j + (y_O^s - y_O^j) \tag{6.310}$$

$$\leq y_O' + s\left(j + \frac{1}{2}\right) + \rho_O(c_O^j, c_O^s)\left(\frac{y_O^s - y_O^j}{\rho_O(c_O^j, c_O^s)}\right) \tag{6.311}$$

$$< y_O' + s\left(j + \frac{1}{2}\right) + \frac{q\delta}{2}\frac{2q}{1+q^2} \tag{6.312}$$

$$\leq y'_O + s\left(\left\lceil\frac{\rho(c',c)-q\delta}{s}\right\rceil + \frac{1}{2}\right) + \frac{q^2\delta}{1+q^2} \tag{6.313}$$
$$\leq y'_O + s\left(\frac{\rho(c',c)-q\delta}{s} + 1 + \frac{1}{2}\right) + \frac{q^2\delta}{1+q^2} \tag{6.314}$$
$$= y'_O + \left(y_O - y'_O - q\delta + \frac{3}{2}s\right) + \frac{q^2\delta}{1+q^2} \tag{6.315}$$
$$= y_O - \left(q\delta - \frac{q^2\delta}{1+q^2} - \frac{3}{2}s\right) \tag{6.316}$$
$$= y_O - q\delta\left(1 - \frac{q}{1+q^2} - \frac{q}{4}\right). \tag{6.317}$$

Now the maximum value of $\frac{q}{1+q^2}$ is $\frac{1}{2}$ and, if $q < 1$, the maximum value of $\frac{q}{4}$ is $\frac{1}{4}$. Therefore

$$y^s_O < y_O - q\delta\left(1 - \frac{1}{2} - \frac{1}{4}\right) \tag{6.318}$$
$$< y_O. \tag{6.319}$$

We also have that

$$y^s_O = y^j_O + (y^s_O - y^j_O) \tag{6.320}$$
$$> y'_O + s\left(j - \frac{1}{2}\right) + \rho(c^s_O, c^j_O)\left(\frac{y^s_O - y^j_O}{\rho(c^s_O, c^j_O)}\right) \tag{6.321}$$
$$> y'_O - \frac{s}{2} + \frac{q\delta}{4}\frac{2q}{3} \tag{6.322}$$
$$= y'_O - \frac{s}{2} + \frac{q^2\delta}{6} \tag{6.323}$$
$$= y'_O + \frac{s}{2} \tag{6.324}$$
$$> y'_O. \tag{6.325}$$

Therefore,

$$y'_O < y^s_O < y_O. \tag{6.326}$$

Now $(\pi(c',c))_O = \{(x,y) \in O | x = x'_O \text{ and } y'_O \leq y \leq y_O\}$. Thus we can choose $c^t_O \in (\pi(c',c))_O$ with $y^t = y^s_O$ and $x^t_O = x'_O$.

Now we show that $c^n_O \in (T_\eta(c',c))_O$. We first bound the distance from c^t_O to c^s_O and then use the triangle inequality to bound the distance from c^t_O to c^n_O. We have

$$\rho_O(c^t_O, c^s_O) = |x'_O - x^s_O| \tag{6.327}$$
$$= \max\left[x'_O - x^s_O, x^s_O - x'_O\right]. \tag{6.328}$$

Now

$$x^s_O = x^j_O + (x^s_O - x^j_O) \tag{6.329}$$

$$= x^j_O \pm \rho_O(c^j_O, c^s_O)\left(\frac{\sqrt{\rho_O(c^j_O, c^s_O)^2 - (y^s_O - y^j_O)^2}}{\rho_O(c^j_O, c^s_O)}\right) \tag{6.330}$$

$$= x^j_O \pm \rho_O(c^j_O, c^s_O)\sqrt{1 - \left(\frac{y^s_O - y^j_O}{\rho_O(c^j_O, c^s_O)}\right)^2}. \tag{6.331}$$

Now we also require $x^s_O \geq x^j_O$ so we must choose the +, giving us

$$x^s_O = x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{1 - \left(\frac{y^s_O - y^j_O}{\rho_O(c^j_O, c^s_O)}\right)^2} \tag{6.332}$$

$$> x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{1 - \left(\frac{2q}{1+q^2}\right)^2} \tag{6.333}$$

$$= x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{\frac{(1+q^2)^2 - 4q^2}{(1+q^2)^2}} \tag{6.334}$$

$$= x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{\frac{1 - 2q^2 + q^4}{(1+q^2)^2}} \tag{6.335}$$

$$= x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{\frac{(1-q^2)^2}{(1+q^2)^2}} \tag{6.336}$$

$$= x^j_O + \rho_O(c^j_O, c^s_O)\frac{1-q^2}{1+q^2}. \tag{6.337}$$

In order to complete this lower bound, we need a lower bound on x^j_O. We have

$$x^j_O > x'_O - q(y^j_O - (y'_O - \delta)) \tag{6.338}$$

and we showed in Equation 6.248 that $y^j_O \leq y_O - q\delta + \frac{3}{2}s$. Therefore

$$x^j_O > x'_O - q\left(y_O - q\delta + \frac{3}{2}s - y'_O + \delta\right) \tag{6.339}$$

$$= x'_O - \frac{\delta}{\rho(c', c) + \delta}(y_O - y'_O + \delta) + q^2\delta - \frac{3}{2}sq \tag{6.340}$$

$$= x'_O - \delta + q^2\delta - \frac{3}{2}sq. \tag{6.341}$$

Therefore

$$x^s_O > x'_O - \delta + q^2\delta - \frac{3}{2}sq + \rho_O(c^j_O, c^s_O)\frac{1-q^2}{1+q^2}. \tag{6.342}$$

Now we also have that $x_O^j \leq x'_O$ giving a loose upper bound for x_O^s of

$$x_O^s < x_O^j + \rho_O(c_O^j, c_O^s) \leq x'_O + \rho_O(c_O^j, c_O^s). \tag{6.343}$$

Therefore

$$\rho_O(c_O^t, c_O^s) = \max\Big[x'_O - x_O^s, x_O^s - x'_O\Big] \tag{6.344}$$
$$< \max\left[\delta + \frac{3}{2}sq - q^2\delta - \rho_O(c_O^j, c_O^s)\frac{1-q^2}{1+q^2}, \rho_O(c_O^j, c_O^s)\right]. \tag{6.345}$$

Now we chose δ so that $q \leq \frac{1}{3}$, giving

$$\delta + \frac{3}{2}sq - q^2\delta - \rho_O(c_O^j, c_O^s)\frac{1-q^2}{1+q^2} > \delta(1-q^2) - \rho_O(c_O^j, c_O^s) \tag{6.346}$$
$$> \delta(1-q^2) - \frac{q\delta}{2} \tag{6.347}$$
$$\geq \delta\left(1 - \frac{1}{9} - \frac{1}{6}\right) \tag{6.348}$$
$$= \frac{13}{18}\delta \tag{6.349}$$
$$> \frac{1}{6}\delta \tag{6.350}$$
$$\geq \frac{q\delta}{2} \tag{6.351}$$
$$> \rho_O(c_O^j, c_O^s). \tag{6.352}$$

Therefore

$$\rho_O(c_O^t, c_O^s) < \delta + \frac{3}{2}sq - q^2\delta - \rho_O(c_O^j, c_O^s)\frac{1-q^2}{1+q^2}. \tag{6.353}$$

These properties allow us to put a bound on where c_O^n can be. Recall $\rho_O(c_O^n, c_O^s) \leq \rho_O(c_O^j, c_O^s)$. Then using the triangle inequality and that $q < 1$,

$$\rho_O(c_O^t, c_O^n) \leq \rho_O(c_O^t, c_O^s) + \rho_O(c_O^s, c_O^n) \tag{6.354}$$
$$< \delta + \frac{3}{2}sq - q^2\delta - \rho_O(c_O^j, c_O^s)\frac{1-q^2}{1+q^2} + \rho_O(q_O^j, q_O^s) \tag{6.355}$$
$$= \delta + \frac{3}{2}sq - q^2\delta - \rho_O(c_O^j, c_O^s)\left(\frac{1-q^2-1-q^2}{1+q^2}\right) \tag{6.356}$$
$$= \delta + \frac{3}{2}sq - q^2\delta + \rho_O(c_O^j, c_O^s)\frac{2q^2}{1+q^2} \tag{6.357}$$
$$< \delta + \frac{q^3\delta}{4} - q^2\delta + \delta\frac{q^3}{1+q^2} \tag{6.358}$$
$$= \delta + q^2\delta\left(\frac{q}{4} + \frac{q}{1+q^2} - 1\right) \tag{6.359}$$

$$< \delta + q^2\delta\left(\frac{1}{4}+\frac{1}{2}-1\right) \tag{6.360}$$
$$= \delta - \frac{q^2\delta}{4} \tag{6.361}$$
$$< \delta \tag{6.362}$$
$$\leq \eta. \tag{6.363}$$

Therefore $c^n_O \in (B_\eta(c^t))_O \subseteq (T_\eta(c', c))_O$ by Corollary 6.13. Combining this with Equation 6.299 and using Corollary 6.12 gives us that $c^n \in T_\eta(c', c)$.

Now we show that with some choice of f, the final configuration is in $\bigcup_{l>j} W^l$. Firstly consider choosing the projection function f_y. Then

$$c^f = f_y(c^n, c^s) = (c^s_R, (x^n_O, y^s_O)). \tag{6.364}$$

We first consider the x coordinate of c^f_O and show that $x'_O - q(y^s_O - (y'_O - \delta)) < x^n_O < x'_O + q(y^s_O - (y'_O - \delta))$. Let

$$\phi = \frac{y^s_O - y^j_O}{\rho_O(c^j_O, c^s_O)}. \tag{6.365}$$

Recall from Equation 6.332 that

$$x^s_O = x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{1 - \left(\frac{y^s_O - y^j_O}{\rho_O(c^j_O, c^s_O)}\right)^2} \tag{6.366}$$
$$= x^j_O + \rho_O(c^j_O, c^s_O)\sqrt{1-\phi^2}. \tag{6.367}$$

Recall $\frac{2q}{3} < \phi < \frac{2q}{1+q^2} \leq 1$. Additionally, $x^n_O - x^s_O \geq -\rho_O(c^n_O, c^s_O) \geq -\rho_O(c^j_O, c^s_O)$. Thus

$$x^n_O = (x^n_O - x^s_O) + (x^s_O - x^j_O) + x^j_O \tag{6.368}$$
$$\geq x^j_O - \rho(c^n_O, c^s_O) + \rho_O(c^j_O, c^s_O)\sqrt{1-\phi^2} \tag{6.369}$$
$$\geq x^j_O - \rho(c^j_O, c^s_O) + \rho_O(c^j_O, c^s_O)\sqrt{1-\phi^2} \tag{6.370}$$
$$> x^j_O - \rho_O(c^j_O, c^s_O)\left(1 - \sqrt{1 - \frac{2q}{1+q^2}\phi}\right) \tag{6.371}$$
$$= x^j_O - \rho_O(c^j_O, c^s_O)\left(1 - \sqrt{1 - \left(\frac{2q\phi + 2q^3\phi}{1+q^2} - q^2\left(\frac{2q}{1+q^2}\right)\phi\right)}\right) \tag{6.372}$$
$$> x^j_O - \rho_O(c^j_O, c^s_O)\left(1 - \sqrt{1 - \frac{2q\phi + 2q^3\phi}{1+q^2} + q^2\phi^2}\right) \tag{6.373}$$
$$= x^j_O - \rho_O(c^j_O, c^s_O)\left(1 - \sqrt{1 - 2q\phi + q^2\phi^2}\right) \tag{6.374}$$
$$= x^j_O - \rho_O(c^j_O, c^s_O)\left(1 - \sqrt{(1-q\phi)^2}\right) \tag{6.375}$$
$$= x^j_O - \rho_O(c^j_O, c^s_O)(1 - (1 - q\phi)) \tag{6.376}$$

$$= x_O^j - q\rho_O(c_O^j, c_O^s)\frac{(y_O^s - y_O^j)}{\rho_O(c_O^j, c_O^s)} \tag{6.377}$$
$$= x_O^j - q(y_O^s - y_O^j) \tag{6.378}$$
$$> x'_O - q(y_O^j - (y'_O - \delta)) + qy_O^j - qy_O^s \tag{6.379}$$
$$= x'_O - q(y_O^s - (y'_O - \delta)). \tag{6.380}$$

Using that $x_O^j \leq x'_O$ and $y_O^s > y'_O$, we also have

$$x_O^n = (x_O^n - x_O^s) + (x_O^s - x_O^j) + x_O^j \tag{6.381}$$
$$\leq x_O^j + 2\rho_O(c_O^j, c_O^s) \tag{6.382}$$
$$< x'_O + q\delta \tag{6.383}$$
$$= x'_O + q(y'_O - (y'_O - \delta)) \tag{6.384}$$
$$< x'_O + q(y_O^s - (y'_O - \delta)). \tag{6.385}$$

Additionally,

$$y_O^s = y_O^j + \rho_O(c_O^j, c_O^s)\frac{y_O^s - y_O^j}{\rho_O(c_O^j, c_O^s)} \tag{6.386}$$
$$> y_O^j + \frac{q\delta}{4}\left(\frac{2q}{3}\right) \tag{6.387}$$
$$> y'_O + s\left(j - \frac{1}{2}\right) + \frac{q^2\delta}{6} \tag{6.388}$$
$$= y'_O + s\left(j + \frac{1}{2}\right). \tag{6.389}$$

Now consider $\bigcup_{l>j} W_O^l$. We have that

$$y'_O + s\left((j_{\max} - 1) + \frac{1}{2}\right) = y'_O + s\left(\left\lceil\frac{\rho(c', c) - q\delta}{s}\right\rceil + \frac{1}{2}\right) \tag{6.390}$$
$$\leq y'_O + s\left(\frac{\rho(c', c) - q\delta}{s} + \frac{1}{2}\right) \tag{6.391}$$
$$= y'_O + (y_O - y'_O) - q\delta + \frac{1}{2}s \tag{6.392}$$
$$= y_O - q\delta + \frac{1}{2}s \tag{6.393}$$

so

$$\bigcup_{j<l\leq j_{\max}} W_O^l = \left(B_\eta(c^b)\right)_O \cup \bigcup_{j<l<j_{\max}} \left\{ (x,y) \in O \,\middle|\, \begin{array}{c} y'_O+s\left(l-\frac{1}{2}\right)<y\leq y'_O+s\left(l+\frac{1}{2}\right) \text{ and} \\ x'_O-q(y-(y'_O-\delta))<x<x'_O+q(y-(y'_O-\delta)) \end{array} \right\} \tag{6.394}$$

$$= \left(B_\eta(c^b)\right)_O \cup \left\{ (x,y) \in O \,\middle|\, \begin{array}{c} y'_O+s\left(j+\frac{1}{2}\right)<y\leq y_O-q\delta+\frac{1}{2}s \text{ and} \\ x'_O-q(y-(y'_O-\delta))<x<x'_O+q(y-(y'_O-\delta)) \end{array} \right\}. \tag{6.395}$$

Thus if $y_O^s \leq y_O - q\delta + \frac{1}{2}s$, $(x_O^n, y_O^s) \in \bigcup_{l>j} W_O^l$. Now assume $y_O^s > y_O - q\delta + \frac{1}{2}s$ and recall that $y_O^s < y_O$ so $0 < y_O - y_O^s < q\delta - \frac{1}{2}s$. We show that $(x_O^n, y_O^s) \in (B_\eta(c))_O$:

$$\rho_O(c_O, (x_O^n, y_O^s)) = \sqrt{(x_O - x_O^n)^2 + (y_O - y_O^s)^2} \tag{6.396}$$

$$= \sqrt{(x'_O - x_O^n)^2 + (y_O - y_O^s)^2} \tag{6.397}$$

$$< \sqrt{(q(y_O^s - (y'_O - \delta)))^2 + (y_O - y_O^s)^2} \tag{6.398}$$

$$= \sqrt{\left(\frac{\delta}{y_O - y'_O + \delta}(y_O - y'_O + \delta) + q(y_O^s - y_O)\right)^2 + (y_O - y_O^s)^2} \tag{6.399}$$

$$= \sqrt{\delta^2 + 2q\delta(y_O^s - y_O) + (1+q^2)(y_O - y_O^s)^2} \tag{6.400}$$

$$= \sqrt{\delta^2 + (y_O - y_O^s)((1+q^2)(y_O - y_O^s) - 2q\delta)} \tag{6.401}$$

$$< \sqrt{\delta^2 + (y_O - y_O^s)\left((1+q^2)\left(q\delta - \frac{1}{2}s\right) - 2q\delta\right)} \tag{6.402}$$

$$< \sqrt{\delta^2 + (y_O - y_O^s)((1+q^2)q\delta - 2q\delta)} \tag{6.403}$$

$$= \sqrt{\delta^2 + (y_O - y_O^s)q\delta(1+q^2-2)} \tag{6.404}$$

$$= \sqrt{\delta^2 - (y_O - y_O^s)q\delta(1-q^2)} \tag{6.405}$$

$$\leq \delta \tag{6.406}$$

$$\leq \eta \tag{6.407}$$

so if $y_O^s > y_O - q\delta + \frac{1}{2}s$, $(x_O^n, y_O^s) \in (B_\eta(c'))_O = W_O^{j_{\max}} \subset (T_\eta(c', c))_O$. Thus

$$(x_O^n, y_O^s) \in \bigcup_{l>j} W_O^l \subseteq (T_\eta(c', c))_O. \tag{6.408}$$

Therefore, there is some $k \in \{j+1, \ldots, j_{\max}\}$ for which $(x_O^n, y_O^s) \in W_O^k$. We also showed that $c_R^s \in W_R^{j+1} \subset W_R^{j+2} \subset \ldots \subset W_R^{j_{\max}}$. Therefore, $c_R^s \in W_R^k$. Thus $(c_R^s, (x_O^n, y_O^s)) \in W_R^k \times W_O^k = W^k \subseteq \bigcup_{l>j} W^l$.

Now assume $y^n_O \leq y^s_O$ and that we choose $f = f_y$. Let

$$c^f = f_y(c^n, c^s) = (c^n_R, (x^n_O, y^s_O)). \tag{6.409}$$

We will show that $\pi(c^n, c^f) \subseteq T_\eta(c', c)$. Let $c^p = ((x^n_O, y^n_O - O_R), c^n_O)$ and let $c^v = ((x^n_O, y^s_O - O_R), (x^n_O, y^s_O))$. Note that $c^p_O = c^n_O \in (T_\eta(c', c))_O$ and $c^v_O = c^f_O \in (T_\eta(c', c))_O$. Let

$$c^t_O = \arg \inf_{p \in (\pi(c',c))_O} \rho_O(p, c^n_O). \tag{6.410}$$

We must have $\rho_O(c^n_O, c^t_O) < \eta$ since $c^n_O \in (T_\eta(c', c))_O$. Let

$$c^t_R = (x^t_O, y^t_O - O_R) \in (\pi(c', c))_R. \tag{6.411}$$

Then

$$\begin{aligned}
\rho_R(c^p_R, (x^t_O, y^t_O - O_R)) &= \sqrt{(x^n_O - x^t_O)^2 + (y^n_O - y^t_O)^2} &(6.412)\\
&= \rho_O(c^n_O, c^t_O) &(6.413)\\
&< \eta &(6.414)
\end{aligned}$$

so $c^p_R \in (T_\eta(c', c))_R$. Similarly, let

$$c^z_O = \arg \inf_{p \in (\pi(c',c))_O} \rho_O(p, c^f_O). \tag{6.415}$$

We must have $\rho_O(c^f_O, c^z_O) < \eta$ because $c^f_O \in (T_\eta(c', c))_O$. Let

$$c^z_R = (x^z_O, y^z_O - O_R) \in (\pi(c', c))_R. \tag{6.416}$$

Then

$$\begin{aligned}
\rho_R(c^v_R, c^z_R) &= \sqrt{(x^n_O - x^z_O)^2 + (y^s_O - y^z_O)^2} &(6.417)\\
&= \rho_O(c^f_O, c^z_O) &(6.418)\\
&< \eta &(6.419)
\end{aligned}$$

so $c^v_R \in (T_\eta(c', c))_R$. Then

$$\begin{aligned}
\pi(c^n, c^p) &= \bigcup_{t \in [0,1]} \left(\left(x^n_R + (x^p_R - x^n_R)t, y^n_R + (y^p_R - y^n_R)t \right), c^n_O \right) &(6.420)\\
&\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O &(6.421)\\
&= T_\eta(c', c) &(6.422)
\end{aligned}$$

using Lemma 6.15 and Corollary 6.12. Similarly,

$$\begin{aligned}\pi(c^v, c^f) &= \bigcup_{t\in[0,1]} \left(\left(x_O^v + (x_R^f - x_O^v)t, y_O^v + (y_R^f - y_R^v)t\right), c_O^f\right) && (6.423)\\ &\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O && (6.424)\\ &= T_\eta(c', c) && (6.425)\end{aligned}$$

using Lemma 6.15 and Corollary 6.12. Now consider

$$\begin{aligned}\pi(c^p, c^v) &= \bigcup_{t\in[0,1]} \Big(\left(x_R^p + (x_R^v - x_R^p)t, y_R^p + (y_R^v - y_R^p)\right), \\ &\qquad \left(x_O^p + (x_O^v - x_O^p)t, y_O^p + (y_O^v - y_O^p)t\right)\Big) && (6.426)\\ &\subseteq \left(\bigcup_{t\in[0,1]} \left(x_R^p + (x_R^v - x_R^p)t, y_R^p + (y_R^v - y_R^p)\right)\right) \times \\ &\qquad \left(\bigcup_{t\in[0,1]} \left(x_O^p + (x_O^v - x_O^p)t, y_O^p + (y_O^v - y_O^p)t\right)\right) && (6.427)\\ &\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O && (6.428)\\ &= T_\eta(c', c) && (6.429)\end{aligned}$$

using Lemmas 6.8 and 6.15 and Corollary 6.12. Since $y_O^n \le y_O^s$,

$$\begin{aligned}\pi(c^n, c^f) &= \pi(c^n, c^p) \cup \pi(c^p, c^v) \cup \pi(c^v, c^f) && (6.430)\\ &\subseteq T_\eta(c', c). && (6.431)\end{aligned}$$

Therefore if $y_O^n \le y_O^s$, $f_y(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$.

Now assume $y_O^n > y_O^s$ and that we choose f_R, giving $c^f = f_R(c^n, c^s) = (c_R^s, c_O^n)$. We have already shown that $c_R^s \in W_R^{j+1}$. We now show that $c_O^n \in \bigcup_{l>j} W_O^l$. Recall that

$$x'_O - q(y_O^s - (y'_O - \delta)) < x_O^n < x'_O + q(y_O^s - (y'_O - \delta)).$$

If $y_O^n > y_O^s$ then

$$x'_O - q(y_O^n - (y'_O - \delta)) < x_O^n < x'_O + q(y_O^n - (y'_O - \delta)). \qquad (6.432)$$

By Equation 6.389, $y_O^n > y'_O + s\left(j + \frac{1}{2}\right)$. Therefore, using Equation 6.395, if $y^n \le y_O - q\delta + \frac{1}{2}s$ then $c_O^n \in \bigcup_{l>j} W^l$. Consider if $y^n > y_O - q\delta + \frac{1}{2}s$. Firstly assume

$y^n_O \leq y_O$. Then

$$\sqrt{(x_O - x^n_O)^2 + (y_O - y^n_O)^2} = \sqrt{(x'_O - x^n_O)^2 + (y_O - y^n_O)^2} \tag{6.433}$$
$$< \sqrt{(q(y^n_O - (y'_O - \delta)))^2 + (y_O - y^n_O)^2} \tag{6.434}$$
$$< \sqrt{(q(y^n_O - (y'_O - \delta)))^2 + (y_O - y^n_O)^2} \tag{6.435}$$
$$= \sqrt{\left(\frac{\delta}{\rho(c',c)+\delta}(y_0 - y'_O + \delta) - q(y_O - y^n_O)\right)^2 + (y_O - y^n_O)^2} \tag{6.436}$$
$$= \sqrt{(\delta - q(y_O - y^n_O))^2 + (y_O - y^n_O)^2} \tag{6.437}$$
$$= \sqrt{\delta^2 - 2q\delta(y_O - y^n_O) + q^2(y_O - y^n_O)^2 + (y_O - y^n_O)^2}. \tag{6.438}$$
$$= \sqrt{\delta^2 + (y_O - y^n)((1+q^2)(y_O - y^n_O) - 2q\delta)} \tag{6.439}$$
$$\leq \sqrt{\delta^2 + (y_O - y^n_O)\left((1+q^2)\left(q\delta - \frac{1}{2}s\right) - 2q\delta\right)} \tag{6.440}$$
$$\leq \sqrt{\delta^2 - (y_O - y^n_O)q\delta(1-q^2)} \tag{6.441}$$
$$< \delta \tag{6.442}$$
$$\leq \eta \tag{6.443}$$

since $q < 1$. Now assume $y^n_O \geq y_O$. We have already shown that $c^n_O \in (T_\eta(c',c))_O$. Therefore, if $y^n_O \geq y_O$, we must have $c^n_O \in (B_\eta(c))_O$. Thus $c^n_O \in \bigcup_{l>j} W^l_O$ so there is some $k \in \{j+1, ..., j_{\max}\}$ with $c^n_O \in W^k_O$. Since $c^s_R \in W^{j+1}_R \subset W^{j+2}_R \subset ... \subset W^{j_{\max}}_R$, $c^s_R \in W^k_R$. Then $c^f = (c^s_R, c^n_O) \in W^k_R \times W^k_O = W^k \subseteq \bigcup_{l>j} W^l$. Now using that $c^n \in T_\eta(c',c)$ and $c^f \in T_\eta(c',c)$,

$$\pi(c^n, c^f) = \bigcup_{t\in[0,1]} \left(\left(x^n_R + (x^f_R - x^n_R)t, y^n_R + (y^f_R - y^n_R)t\right), c^n_O\right) \tag{6.444}$$
$$\subseteq (T_\eta(c',c))_R \times (T_\eta(c',c))_O \tag{6.445}$$
$$= T_\eta(c',c). \tag{6.446}$$

Thus there is some choice of f that gives us $f(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c',c)$.

■

Now we have shown that Assumption 6.2 holds for each possible segment of the trajectories returned by the empty space planner and we must just show that it holds for whole trajectories. The proof of this is similar to Lemmas 6.6 and 6.19.

Theorem 6.22 (Assumption 6.2 Holds): For all $\eta > 0$, for all $c', c \in X$, for some $\lambda \in (0,1]$ and a sequence of subspaces $W_\delta(c',c) = \{W^0, ..., W^m\}$, for some $\epsilon > 0$, $B_\epsilon(c') \subseteq W^0$, $W^m \subseteq B_\eta(c)$, and for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \le \rho(c^j, c^s)$ then for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least λ.

Proof: Firstly assume $c'_O = c_O$. Then this follows directly from Lemma 6.17.

Now assume $c'_O \neq c_O$, $x'_O = x_O$, and $y'_O < y_O$. Let $c^p = \Big((x'_O, y'_O - O_R), c'_O\Big)$ and let $c^e = \Big((x'_O, y_O - O_R), c_O\Big)$. Then

$$\pi(c', c) = \pi(c', c^p) \cup \pi(c^p, c^e) \cup \pi(c^e, c), \tag{6.447}$$

and $c^p \in \pi(c', c)$ and $c^e \in \pi(c^b, c^e)$. Because $c^e_O = c_O$, by Lemma 6.17, we can choose $\{D^0, ..., D^{d_{\max}}\}$ such that for some $\delta_D \in (0, \eta]$, $B_{\delta_D}(c^e) \subseteq D^0$, $D^{d_{\max}} \subseteq B_\eta(c)$, and for all $j < d_{\max}$, for all $c^j \in D^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \le \rho(c^j, c^s)$, $f_R(c^n, c^s) \in D^{j+1}$, $\pi(c^n, f_R(c^n, c^s)) \subseteq T_\eta(c^e, c) \subset T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^R_\eta(c^e, c)$. Now let $c^b = (c^p_R, c_O)$. Also by Lemma 6.17, we can choose we can choose $\{Q^0, ..., Q^{q_{\max}}\}$ such that for some $\delta_Q \in (0, \delta_D]$, $B_{\delta_Q}(c^b) \subseteq D^0$, $Q^{q_{\max}} \subseteq B_{\delta_D}(c^e)$, and for all $j < q_{\max}$, for all $c^j \in Q^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \le \rho(c^j, c^s)$, $f_R(c^n, c^s) \in D^{j+1}$, $\pi(c^n, f_R(c^n, c^s)) \subseteq T_{\delta_D}(c^b, c^e)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^R_{\delta_D}(c^b, c^e)$. Now $\delta_D \le \eta$ so

$$\begin{aligned} T_{\delta_D}(c^b, c^e) &\subseteq T_\eta(c^b, c^e) && (6.448)\\ &= \bigcup_{t\in[0,1]} \left(B_\eta\left((x^b_R + (x^e_R - x^b_R)t, y^b_R + (y^e_R - y^b_R)t)\right)\right)_R \times (B_\eta(c))_O && (6.449)\\ &= \bigcup_{t\in[0,1]} \left(B_\eta(x'_O, y'_O - O_R + (y_O - y'_O)t)\right)_R \times (B_\eta(c))_O. && (6.450)\end{aligned}$$

Now for all $t \in [0,1]$, $(x'_O, y'_O - O_R + (y_O - y'_O)t) \in (\pi(c', c))_R$ so $(B_\eta(x'_O, y'_O - O_R + (y_O - y'_O)t))_R \subseteq (T_\eta(c', c))_R$ by Corollary 6.13. Therefore

$$\begin{aligned} T_{\delta_D}(c^b, c^e) &\subseteq (T_\eta(c', c))_R \times (B_\eta(c))_O && (6.451)\\ &\subseteq (T_\eta(c', c))_R \times (T_\eta(c', c))_O && (6.452)\\ &= T_\eta(c', c) && (6.453)\end{aligned}$$

using Corollaries 6.12 and 6.13.

By Lemma 6.21, we can choose $\{P^0, ..., P^{p_{\max}}\}$ such that for some $\delta_P \in (0, \delta_Q]$, $B_{\delta_P}(c^p) \subseteq P^0$, $P^{p_{\max}} \subseteq B_{\delta_Q}(c^b)$, and for all $j < p_{\max}$, for all $c^j \in P^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \le \rho(c^j, c^s)$, for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} P^l$, $\pi(c^n, f(c^n, c^s)) \subseteq T_{\delta_D}(c^p, c^e) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^O_{\delta_Q}(c^p, c^e)$. Lastly, we choose $\{A^0, ..., A^{a_{\max}}\}$ such that for some $\delta_A \in (0, \delta_P]$, $B_{\delta_A}(c') \subseteq A^0$, $A^{a_{\max}} \subseteq B_{\delta_P}(c^p)$, and for all $j < a_{\max}$, for all $c^j \in A^j$, for some $S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \le \rho(c^j, c^s)$, $f_R(c^n, c^s) \in A^{j+1}$, $\pi(c^n, f_R(c^n, c^s)) \subseteq$

$T_{\delta_P}(c', c^p) \subseteq T_\eta(c', c)$, and the probability of sampling from $S^j(c^j)$ is at least $\lambda^R_{\delta_P}(c', c^p)$.

Now let

$$\{W^0, ..., W^m\} = \{A^0, ..., A^{a_{\max}-1}, P^0, ..., P^{p_{\max}-1}, Q^0, ..., Q^{q_{\max}-1}, D^0, ..., D^{d_{\max}}\}. \tag{6.454}$$

We have $B_{\delta_A}(c') \subseteq W^0$ and $W^m \subseteq B_\eta(c)$. Moreover, $A^{a_{\max}} \subseteq B_{\delta_P}(c^p) \subseteq P^0$, $P^{p_{\max}} \subseteq B_{\delta_Q}(c^b) \subseteq Q^0$, and $Q^0 \subseteq B_{\delta_D}(c^e) \subseteq D^0$. Therefore for all $j < m$, for all $c^j \in W^j$, for some $S^j(c^j)$, for all $c^s \in S^j(c^j)$, for all $c^n \in X$, if $\rho(c^n, c^s) \leq \rho(c^j, c^s)$ then for some $f \in F$, $f(c^n, c^s) \in \bigcup_{l>j} W^l$ and $\pi(c^n, f(c^n, c^s)) \subseteq T_\eta(c', c)$ and the probability of sampling from $S^j(c^j)$ is at least $\min\left[\lambda^R_{\delta_P}(c', c^p), \lambda^O_{\delta_Q}(c^p, c^b), \lambda^R_{\delta_D}(c^b, c^e), \lambda^R_\eta(c^e, c)\right] > 0$.

The other three cases for $c'_O \neq c_O$ are similar.

■

Thus the DARRT algorithm is complete in the disc pushing domain.

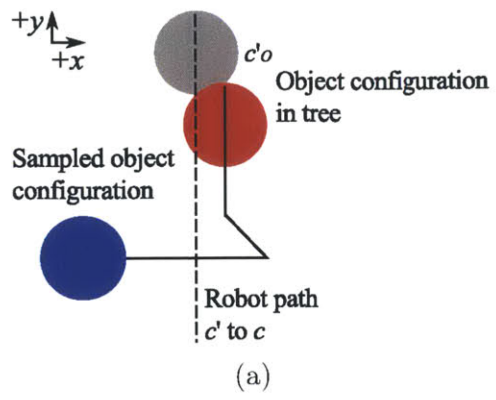
6.2.4 In Defense of Projection Functions

The main conclusion of the proof given in Section 6.1 and the example domains is that we require projection functions in manipulation domains. We argued this in Section 3.2.1 using the example that, without projection functions, the robot could never adjust its position relative to the objects. The example in Section 6.2.3 shows why we need projection functions for each type of manipulation. Assume we did not use the f_x and f_y projection functions, but rather just f_R and f_I, the projection functions used in Section 6.2.2. Now consider a known collision free path from c' to c and assume the object's path is entirely parallel to the y axis (almost all paths contain some vertical segment so assume we just take one of these segments). Assume we sample some configuration c^s near c' and use the f_I projection function, and let c^n be the nearest configuration in the tree to c^s. Then $y^s_O \neq y^n_O$ with probability 1. Therefore, the robot has to use two pushes, as in Figure 6.4f to move from c^s to c^n and we cannot argue the robot's path is near the path from c' to c. This is shown in Figure 6.6a.

A subtler example is a disc robot pushing a disc object. The robot can contact the object at any point but can only push it along the ray connecting the center of the robot to the center of the object. Now consider a path from c' to c and assume for simplicity's sake that the object moves vertically. Let c^s be a sample near c' and let c^n be the nearest configuration in the tree. The probability that $y^s_O = y^n_O$ is zero. In fact, we cannot constrain the angle between c^n_O and c^s_O at all if we use a Euclidean distance metric. Therefore, the robot's path might be as much as the radius of the object from the path from c'_R to c_R. This is shown in Figure 6.6b.

6.3 Exponential Convergence of DARRTH(Connect)

We can also prove that the DARRTH(Connect) algorithm, discussed in Chapter 4, is exponentially convergent. This is a simpler proof, as it is clearer that each iteration of the algorithm has a probability of success that is independent of the iteration.



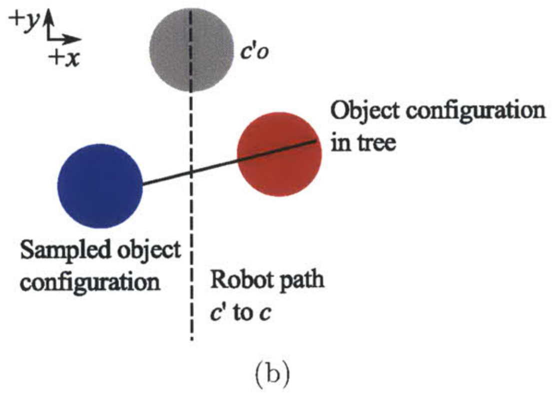


Figure 6.6: The vertical line from c' to c (gray disc and dashed line) is collision free for the robot and object. (a) In the example of Section 6.2.3, the robot can only push the object parallel to the x or y axis. The blue disc shows the object's sampled configuration and the red disc is the nearest configuration in the tree. The robot's path to push the red disc to the blue disc (solid line) is very different from its path in moving from c' to c because the y coordinates of the red and blue disc are different. (b) Even if the robot is able to push the object in any direction, the robot's path from the configuration in the tree to the sampled configuration (solid line) may still differ from its path from c'_R to c_R by as much as the radius of the object.

However, there are situations in which DARRTH(Connect) is not complete. Firstly, it could always generate a sequence of subgoals that the lower-level planners cannot solve. We require that it have some probability of generating *achievable subgoals*:

Definition 6.17 (Achievable Subgoals): A sequence $\{g_0, ..., g_m\}$ is *achievable* by an algorithm A if and only if for all $i \in \{1, ..., m\}$, there is some non-zero probability that A terminates in a configuration in g_{i-1} from which there is a collision free path to a configuration in g_i.

Assumption 6.3: The probability that the sequence of subgoals created on Line 6 of DARRTH(Connect) is achievable by DARRT(Connect) is non-zero and constant with respect to the number of DARRTH(Connect) iterations.

Secondly, DARRTH(Connect) only allows its lower-level planner, DARRT(Connect) to run for a finite amount of time. We showed in Section 6.1 that DARRT is exponentially convergent and therefore finds a solution, if one exists, if given infinite time. However, DARRTH(Connect) cannot allow DARRT(Connect) an infinite amount of time. Therefore, we must assume that the time it is given is enough to find a solution if one exists.

Assumption 6.4: If there is a collision free path from a starting configuration into a goal set, there is a constant probability that DARRT(Connect) finds a solution before DARRTH(Connect) terminates the algorithm.

Under Assumptions 6.3 and 6.4, the DARRTH(Connect) algorithm is exponentially convergent.

Theorem 6.23 (Exponential Convergence of the DARRTH(Connect) **Algorithm):**The probability that DARRTH(Connect) terminates after k iterations if a solution exists is $O(2^{-ak})$ for some positive constant a.
Proof: The probability that DARRTH(Connect) terminates on iteration k is the probability that the sequence of subgoals it produces is achievable and the probability that DARRT(Connect) reaches a configuration for each subgoal from which there is a collision free path to the next subgoal. By Assumptions 6.3 and 6.4, these are both non-zero and independent of iteration. Therefore, by the same argument used in Theorems 2.7 and 6.7, DARRTH(Connect) converges exponentially.

■

We have found that Assumptions 6.3 and 6.4 tend to hold in practice.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Conclusion

In this thesis we presented the Diverse Action Manipulation problem and four algorithms, DARRT, DARRTConnect, DARRTH, and DARRTHConnect, for DAMA problems. These algorithms can plan for problems with multiple, non-prehensile manipulation actions. We showed that all DAMA problems are multi-modal and designed the DARRTH and DARRTHConnect to take advantage of this structure to solve the problem hierarchically.

7.1 Summary of Contributions

The contributions of this thesis are:

We formally defined the diverse action manipulation (DAMA) problem and described two sampling-based algorithms, the Diverse Action Rapidly-exploring Random Tree (DARRT) algorithm and the DARRTConnect algorithm, to solve it (Chapter 3). These algorithms are based on the RRT and RRTConnect algorithms for non-holonomic domains in that they use empty space planners to return complete paths from an initial configuration to a final configuration in the absence of obstacles. Rather than sample uniformly from the configuration space, these algorithms use user-defined projection functions to modify samples. The projection functions ensure that the empty space planner can always return a plan from one configuration to another. These modifications allow DARRT and DARRTConnect to plan for manipulation problems on which classic sampling-based algorithms fail.

We also presented the DARRTH and DARRTHConnect algorithms for the DAMA problem (Chapter 4). We showed that any DAMA problem can be represented as a multi-modal problem and used this to motivate our hierarchical algorithms (Chapter 4). The hierarchical algorithms break each manipulation problem into the sub-problems of achieving poses from which a particular manipulation action is possible. The combination of the shorter planning horizons and the targeted attempts to find collision free configurations for specific manipulation actions make these algorithms more efficient than their flat counterparts.

We implemented the DARRT, DARRTConnect, DARRTH, and DARRTH-Connect algorithms and showed all four could successfully plan in two complicated manipulation domains (Chapter 5). As expected, DARRTCONNECT is more efficient than DARRT and the hierarchical algorithms perform better than the flat searches.

Lastly, we gave a proof of exponential convergence for the DARRT algorithm (Chapter 6). This proof provides guidelines for writing good empty space planners and projection functions for DARRT by formally expressing the assumptions necessary for exponential convergence. We also proved that there are two manipulation domains in which these assumptions hold, one of which has a non-prehensile manipulation primitive.

7.2 Future Work

There are many future avenues for research. These include:

Primitives for DARRTH: In theory, we should be able to identify the primitives DARRTH(CONNECT) needs for each subgoal and only use those primitives in that subgoal. For instance, in both the Plate World and Use Spatula World, the first subgoal is to achieve a pushing configuration. This only requires `transit`. As we discussed in Chapter 5, the projection functions do this partially, but we could try to do it more explicitly. This has some similarities to explicit multi-modal planning [14, 15].

Planning for Uncertainty: In this thesis, we assumed manipulation primitives were deterministic. Even with the primitives used here, that was often a bad assumption. For instance, when pushing a plate in Chapter 5, the robot gripper often only made single point contact with the plate. In this situation, the plate would rotate away from the gripper, usually causing the execution to fail. In the Tool Use Domain, the robot sometimes grasped the spatula incorrectly so that it was at an angle. Even a small angle at the grasp caused the paddle of the spatula to be translated from what the plan expected. The result was that the spatula would fail to slide under the CD or that the CD would drop off.

It is possible that some feedback during execution could fix some of these problems. The pushing and tool use actions were both open loop; there was no visual or tactile feedback that could allow the robot to correct online. Feedback would also allow us to employ a re-planning technique, re-planning from the current configuration when the error becomes too high.

On the other hand, analysis of the pushing showed that much of the time the error was in the perception of the plate's location. In this case, online correction and re-planning cannot fix the problem. Instead, the algorithm should explicitly plan for uncertainty. For example, a more certain action should be preferred to a less certain one. In the Plate World, it is also possible for the robot to grasp the plate by using one gripper to press on one end and tilt it up. This tilt action is successful more often than pushing the plate, but requires a lot of room as both arms must be able to access the plate. A planner that could represent uncertainty should pick the tilt action when it could and use the push action as a fall-back.

A more ambitious line of research is to represent uncertainty explicitly in the manipulation primitives, allowing each primitive to return a set of possible configurations or even a distribution over these configurations. The planner should then search for the plan with the highest probability of success. Possibly previous work on adapting sampling-based planning to uncertain domains [1, 34, 35, 38] could also be extended to the DARRT algorithm.

Empty Space Planner and Projection Functions: We assumed that the empty space planner and projection functions were an input to the algorithm. We outlined a choice of implementation for them in Chapter 3, but many key aspects are still user-defined. A method for learning good empty space planners and projection functions or using a symbolic planner to automatically generate empty space plans and projection functions would make the planner easier to use.

Cluttered Domains: In this thesis, we focused on uncluttered domains. This allowed us to assume that we had a goal pose defined only for a single object since if we had goals for multiple objects we could run the algorithm for each one. In a cluttered domain, we would need to consider carefully the order in which to move each object. It might be possible to combine the work in the navigation among movable obstacles problem [36, 46, 47, 48, 52] with our work to solve DAMA problems in cluttered domains.

Dynamics: All of the manipulation primitives considered in this thesis only allowed an object to move when it was in contact with the robot. In theory, more dynamic actions like throwing or shoving should just require an empty space planner that can describe them. In practice, our current work assumes that any time an object is in contact with a surface, it is resting stably on it. Introducing stability calculations into the configuration validity check would impact efficiency. Additionally, the current algorithms require an empty space planner that can plan a trajectory between any two configurations. In a system described by a differential equation, this is impractical. It might not be difficult to adapt DARRT to such a situation, but DARRTConnect relies on the empty space plans ending at the specified final configuration.

Restricted Configuration Spaces: We were able to prove exponential convergence for the DARRT planner, but only with some strong assumptions about the empty space planner and projection functions. Specifically, we have not given an example of a domain in which contact between the robot and some objects is sometimes disallowed but we can prove Assumptions 6.1 and 6.2 hold. The problem is creating an empty space planner that manages contact between the robot and object and also fulfills Assumption 6.2.

For example, consider again the Disc Pushing domain of Section 6.2.3, but this time assume the robot is not allowed to penetrate the object. When designing the empty space planner, we must choose how the robot moves around the object. Assume that the robot moves around the object by moving up along the $+y$ axis if its y

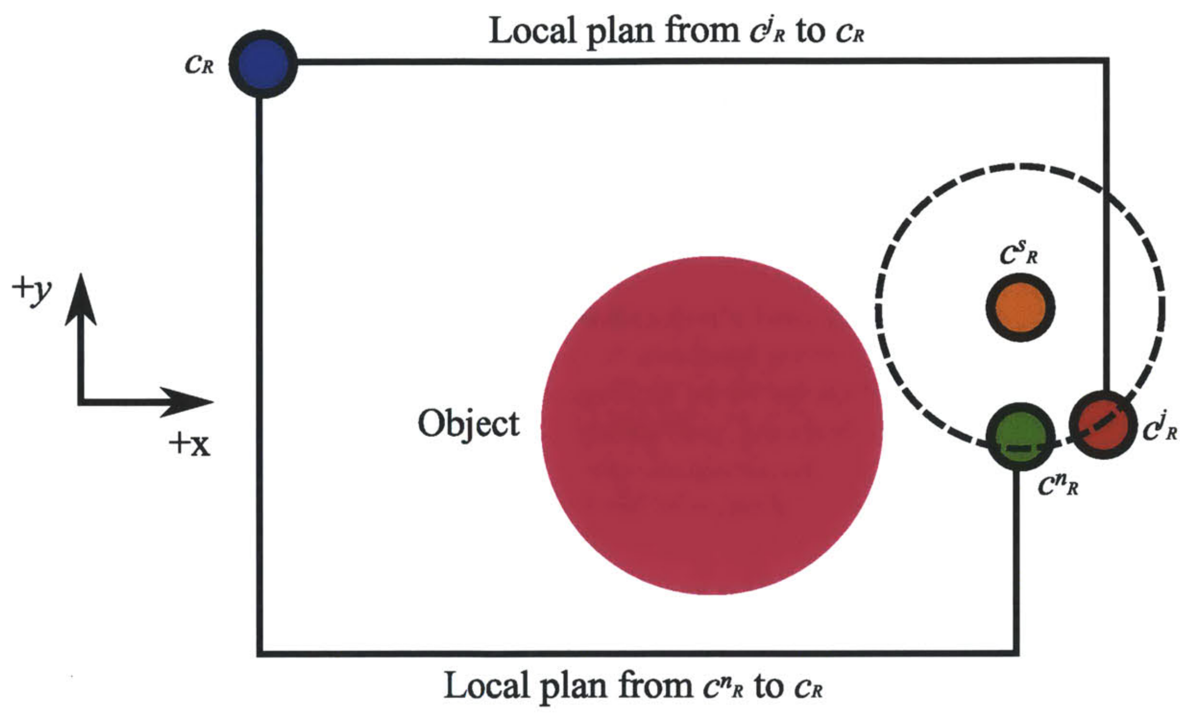

Local plan from c^j_R to c_R
c_R
$+y$
$+x$
Object
c^s_R
c^j_R
c^n_R
Local plan from c^n_R to c_R

Figure 7.1: A poor choice for a empty space planner that avoids contact with the object. When the robot is at or above the midpoint of the object on the y axis, the empty space planner plans a path that moves up the y axis and then to the final configuration. Otherwise, the empty space planner plans a path that moves down the y axis. Here we assume c^j_R (red disc with black outline) and c^n_R (green disc with black outline) are both in the tree and that we know the path from c^j_R to c_R is collision free. Assume we sample a configuration at c^s_R (orange disc with black outline). Then c^n_R is the closest configuration in the tree to c^s_R. We would like to use the openness of the space to argue that the empty space plan from c^n_R to c_R must be collision free if the empty space plan from c^j_R to c_R is collision free, but $y^n_R < y^j_R$ so the empty space planner plans a empty space plan that goes in the $-y$ direction first.

coordinate above the y midpoint of the object and down otherwise. Now assume we have a configuration c^j_R in the tree where the robot is at the midpoint of the object and that we know the empty space plan from c^j_R to c_R is collision free. We would like the argue that the empty space plans from configurations near c^j_R to c_R are very close to the empty space plan from c^j_R to c_R and thus, using the openness of the space, must also be collision free. Specifically, we must be able to sample a configuration c^s_R in such a way that for all configurations c^n_R no farther from c^s_R than c^j_R, the empty space plan from c^n_R to c_R is near the empty space plan from c^j_R to c_R. If $x^s_R = x^j_R$ and $y^s_R > y^j_R$, we can guarantee this is the case. However, we have a zero probability of sampling from that set. If we allow any variation in x^s_R, the nearest configuration c^n_R to c^s_R might have a y coordinate less than y^j_R as shown in Figure 7.1. The empty space plan from c^n_R to c_R goes down the $-y$ axis first and is therefore not near the empty space plan from c^j_R to c_R.

We could argue that the midpoint of the robot is a zero measure set so we do not have to worry about including configurations in the tree that share a y coordinate with the object. However, this then creates regions for sampling in which the size of the region depends on the distance from the midpoint. This size could be arbitrarily small and we cannot guarantee that it is not dependent on iteration. Similarly, we cannot allow the size of the regions in which we sample to depend on the configurations in the tree without introducing a dependence on iteration. If the probability of advancing along the path depends on iteration, we lose the guarantee of exponential convergence.

Alternatively, we could require that the robot always move the same direction around the object but this would result in very different paths if the path barely contacts the object or if it does not contact the object. One solution that does work is to always have the empty space plan approach the object and move around it regardless of whether the more direct path would have been collision free. However, this is a very inefficient empty space planner.

Object/Robot Collisions: Similarly, requiring the empty space planner to manage all collisions between the robot and objects seems unnecessary. When the robot transits, for instance, the objects should just be treated as other obstacles in the environment. However, we have yet to be able to devise a method for allowing the empty space planner to manage contact between the robot and objects only when contact is necessary (i.e. grasping). If we simply define a tube as

$$T_\delta(c', c) = \bigcup_{p \in \pi(c', c)} B_\delta(p) \tag{7.1}$$

then the free space can have any structure we want provided it remains open. This allows us to have a Boolean variable that, when true, disallows contact between the robot and object and, when false, assumes that the empty space planner manages contact between the robot and object. This is commonly done using straight line approaches and retreats in manipulation. By positioning the robot a small distance from the object and then moving in a straight line until contact is made, we guarantee that although we are not doing a collision check between the robot and object - they will be in contact after all - the contact that is made is the kind we want.

Unfortunately, if we use Equation 7.1 for the tube, we are ignoring some important structure of the free space. Namely, if a path is collision free for the robot, the robot never collides with stationary obstacles along this path regardless of the object's configuration. This allows us to disconnect the position of the robot and object during the convergence proof and is important in Lemma 6.21. The problem is that in that proof, we use the maximum distance in the robot's or object's subspace as the total distance. This means that we can put an upper bound on the distance from the nearest configuration in the tree to the sampled configuration but that upper bound is the *same* for the robot and object. Therefore, the sampled configuration cannot be far from the robot's current position or the object's current position, but the robot moves to its sampled configuration while the object moves to some projection of its sampled configuration. Thus we cannot maintain a constant relative configuration between the robot and object.

One way around this problem might be to have the projection function also modify the robot's configuration when it modifies the object's. We have not yet fully explored this avenue of research.

Infinite Types of Manipulation: Our current proof of exponential convergence also only works for manipulation in which there are a finite number of choices. For instance, in the Disc Pushing example, the robot can push the object in four directions. This required four projection functions. If we allow the robot to push the object in any direction (as we did in Chapters 3 and 4), we need an infinity of projection functions and have zero probability that the correct one is chosen.

One possibility is to use that the free space is open so that it is never necessary for the robot to push the object at a single, precise angle, but instead there is always a small range of angles that would work. Then we could argue that if we randomly choose a pushing angle, there is some probability that it falls within this range. This seems a promising approach to this problem, but it is unclear whether it works in the general case. Can we argue that an open free space must always prevent specificity to such a degree that randomly choosing from an infinite number of projection functions still gives a non-zero probability of choosing a "right" one?

Assumptions for DARRTH: Lastly, our assumptions for DARRTH, especially Assumption 6.4, are broad. We give no guidance for how to choose running times for the lower level planners that fulfill this assumption. While in practice we have found that it is not difficult to choose good reset times, more work in this direction could make the hierarchical planner more effective.

Appendix A

Proofs

In this appendix, we give those proofs that were left out of the main text.

Lemma 6.8: For all $B \subseteq M_0 \times ... \times M_n$, $B \subseteq B_0 \times ... \times B_n$.
Proof: Let $(m_0, ..., m_n) \in B$. Then $m_i \in B_i$ for all i by definition. Therefore, $(m_0, ..., m_n) \in B_0 \times ... \times B_n$ so $B \subseteq B_0 \times ... \times B_n$.

■

Lemma 6.9: For some non-empty sets $Q_i \subseteq M_i$, let $B = Q_0 \times ... \times Q_n$. For any subsets $W_i \subseteq Q_i$, $W_0 \times ... \times W_n \subseteq B$.
Proof: Let $(m_0, ..., m_n) \in W_0 \times ... \times W_n$. Then $m_i \in W_i \subseteq Q_i$ so $(m_0, ..., m_n) \in Q_0 \times ... \times Q_n = B$.

■

Lemma 6.10: For some non-empty sets $Q_i \subseteq M_i$, let $B = Q_0 \times ... \times Q_n$. Then $B_i = Q_i$ for $i \in \{0, ..., n\}$.
Proof: For all $j \in \{0, ..., n\} \setminus \{i\}$, choose $q_j \in Q_j$. This choice can be made because $Q_j \neq \emptyset$. Then for all $m \in Q_i$,

$$(q_0, ..., q_{i-1}, m, q_{i+1}, ..., q_n) \in B \tag{A.1}$$

so $Q_i \subseteq B_i$. Now assume $Q_i \subset B_i$. Then there is some $m \in B_i$ with $m \notin Q_i$. By definition of B_i, there is an element $b \in B$ with $b_i = m$. However $b \notin Q_0 \times ... \times Q_{i-1} \times Q_i \times Q_{i+1} \times ... \times Q_n = B$ so we have a contradiction. Therefore, $B_i = Q_i$.

■

Corollary 6.11: For all $c \in X$, for all $\delta > 0$, let $X = M_0 \times ... \times M_n$ where each M_i has distance metric ρ_i. Define

$$X_\delta(c) = \{c' \in X \,|\forall i \in \{0, ..., n\},\ \rho_i(c, c') < \delta\}. \tag{A.2}$$

Then $X_\delta(c) = (X_\delta(c))_0 \times ... \times (X_\delta(c))_n$.

Proof: We have that

$$X_\delta(c) = \{c' \in X \,|\rho_0(c_0, c'_0) < \delta \text{ and } \rho_1(c_1, c'_1) < \delta \text{ and ... and } \rho_n(c_n, c'_n) < \delta\} \quad (A.3)$$
$$= \{m_0 \in M_0 \,|\rho_0(c_0, m_0) < \delta\} \times ... \times \{m_n \in M_n \,|\rho_n(c_n, m_n) < \delta\} \quad (A.4)$$
$$= (X_\delta(c))_0 \times ... \times (X_\delta(c))_O \quad (A.5)$$

where the last step follows by Lemma 6.10.

■

Corollary 6.12 For all $c', c \in X$, for all $\delta > 0$, define

$$Q_\delta(c', c) = \left(\bigcup_{p \in \pi(c',c)} (X_\delta(p))_0 \right) \times ... \times \left(\bigcup_{p \in \pi(c',c)} (X_\delta(p))_n \right). \quad (A.6)$$

Then $Q_\delta(c', c) = (Q_\delta(c', c))_0 \times ... \times (Q_\delta(c', c))_n$.
Proof: This follows directly from Lemma 6.10.

■

Corollary 6.13: For all $c', c \in X$, for all $q \in \pi(c', c)$, for all $i \in \{0, ..., n\}$, for all $\delta > 0$, $(X_\delta(q))_i \subseteq (Q_\delta(c', c))_i$.
Proof: By Lemma 6.10,

$$(Q_\delta(c', c))_i = \bigcup_{p \in \pi(c',c)} (X_\delta(p))_i \supseteq (X_\delta(q))_i \,. \quad (A.7)$$

■

Lemma 6.14: Let $B = B_0 \times ... \times B_n \subseteq M_0 \times ... \times M_n$. If $\mu_i(B_i) > 0$ and we choose a configuration at random from $M_0 \times ... \times M_n$, then the probability of sampling from B is $\prod_{i=0}^n \frac{\mu_i(B_i)}{\mu_i(M_i)} > 0$.
Proof: The probability that a configuration sampled uniformly at random from M_i is in B_i is $\frac{\mu_i(B_i)}{\mu_i(M_i)}$. The probability of sampling from B_i and B_j is independent if $i \neq j$ so the probability of sampling a configuration in $B_0 \times ... \times B_n = B$ is $\prod_{i=0}^n \frac{\mu_i(B_i)}{\mu_i(M_i)}$.

■

Lemma 6.15: For $i \in \{0, ..., n\}$, let M_i represent a plane with the Euclidean distance metric

$$\rho_i\left((x_i^\alpha, y_i^\alpha), (x_i^a, y_i^a)\right) = \sqrt{(x^\alpha - x^a)^2 + (y^\alpha - y^a)^2}, \quad (A.8)$$

and let

$$\tau_i(t) = \left(x_i^a + (x_i^b - x_i^a)t, y_i^a + (y_i^b - y_i^a)t\right) \quad (A.9)$$

describe a straight line in subspace M_i from c_i^a to c_i^b. Similarly, let

$$\sigma_i(t) = \left(x_i^\alpha + (x_i^\beta - x_i^\alpha)t, y_i^\alpha + (y_i^\beta - y_i^\alpha)t\right) \tag{A.10}$$

describe a straight line from c_i^α to c_i^β. Let

$$\left(\pi(c^a, c^b)\right)_i = \bigcup_{t\in[0,1]} \tau_i(t), \tag{A.11}$$

and let

$$\left(Q_\delta(c^\alpha, c^\beta)\right)_i = \bigcup_{t\in[0,1]} \left\{m \in M_i \,|\rho_i(\sigma(t), m) < \delta\right\}. \tag{A.12}$$

If $c_i^a \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$ and $c^b \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$, then $\left(\pi(c^a, c^b)\right)_i \subseteq \left(Q_\delta(c^\alpha, c^\beta)\right)_i$.

Proof: The easiest way to see this is to realize from Figure 2.8 that $\left(Q_\delta(c^\alpha, c^\beta)\right)_i$, the tube around the line from c_i^α to c_i^β is convex. However, we do a formal proof. Let

$$t^a = \arg\min_{t\in[0,1]} \rho_i(\tau_i(0), \sigma_i(t)) \tag{A.13}$$

$$t^b = \arg\min_{t\in[0,1]} \rho_i(\tau_i(1), \sigma_i(t)). \tag{A.14}$$

Since $\tau_i(0) = c_i^a \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$ and $\tau_i(1) = c_i^b \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$, we must have

$$\rho_i(\tau_i(0), \sigma_i(t^a)) < \delta \tag{A.15}$$

$$\rho_i(\tau_i(1), \sigma_i(t^b)) < \delta. \tag{A.16}$$

For $t \in [0, 1]$, we need to show that

$$\min_{s\in[0,1]} \rho_i(\tau_i(t), \sigma_i(s)) = \min_{s\in[0,1]} \rho_i\Big(\left(x^a + (x^b - x^a)t, y^a + (y^b - y^a)t\right), \left(x^\alpha + (x^\beta - x^\alpha)s, y^\alpha + (y^\beta - y^\alpha)s\right)\Big) \tag{A.17}$$

is less than δ. We choose $s = (1-t)t^a + tt^b$. Note that since $t \in [0, 1]$, $t^a \in [0, 1]$, and $t^b \in [0, 1]$ so $s \in [0, 1]$. Then

$$\min_{s\in[0,1]} \rho_i\Big(\tau_i(t), \sigma_i(s)\Big) \leq \rho_i\Big(\left(x^a+(x^b-x^a)t, y^a+(y^b-y^a)t\right), \left(x^\alpha+(x^\beta-x^\alpha)\big((1-t)t^a+tt^b\big), y^\alpha+(y^\beta-y^\alpha)\big((1-t)t^a+tt^b\big)\right)\Big) \tag{A.18}$$

$$\begin{aligned}
&\rho_i\Bigg(\bigg(\Big(1-t\Big)x^a+tx^b,\Big(1-t\Big)y^a+ty^b\bigg),\\
=\;&\quad\bigg(\Big(1-\big((1-t)t^a+tt^b\big)\Big)x^\alpha+\Big((1-t)t^a+tt^b\Big)x^\beta,\\
&\qquad\Big(1-\big((1-t)t^a+tt^b\big)\Big)y^\alpha+\Big((1-t)t^a+tt^b\Big)y^\beta\bigg)\Bigg)
\end{aligned}\tag{A.19}$$

$$\begin{aligned}
&\rho_i\Bigg(\bigg(\Big(1-t\Big)x^a+tx^b,\Big(1-t\Big)y^a+ty^b\bigg),\\
=\;&\quad\bigg(\Big((1-t)(1-t^a)+t(1-t^b)\Big)x^\alpha+\Big((1-t)t^a+tt^b\Big)x^\beta,\\
&\qquad\Big((1-t)(1-t^a)+t(1-t^b)\Big)y^\alpha+\Big((1-t)t^a+tt^b\Big)y^\beta\bigg)\Bigg)
\end{aligned}\tag{A.20}$$

$$\begin{aligned}
&\rho_i\Bigg(\bigg(\Big(1-t\Big)x^a+tx^b,\Big(1-t\Big)y^a+ty^b\bigg),\\
=\;&\quad\bigg(\Big(1-t\Big)\Big((1-t^a)x^\alpha+t^ax^\beta\Big)+t\Big((1-t^b)x^\alpha+t^bx^\beta\Big),\\
&\qquad\Big(1-t\Big)\Big((1-t^a)y^\alpha+t^ay^\beta\Big)+t\Big((1-t^b)y^\alpha+t^by^\beta\Big)\bigg)\Bigg)
\end{aligned}\tag{A.21}$$

$$\begin{aligned}
=\;&\Bigg(\bigg((1-t)\Big(x^a-\big((1-t^a)x^\alpha+t^ax^\beta\big)\Big)+t\Big(x^b-\big((1-t^b)x^\alpha+t^bx^\beta\big)\Big)\bigg)^2+\\
&\bigg((1-t)\Big(y^a-\big((1-t^a)y^\alpha+t^ay^\beta\big)\Big)+t\Big(y^b-\big((1-t^b)y^\alpha+t^by^\beta\big)\Big)\bigg)^2\Bigg)^{1/2}
\end{aligned}\tag{A.22}$$

$$\begin{aligned}
\le\;&\sqrt{\bigg((1-t)\Big(x^a-\big((1-t^a)x^\alpha+t^ax^\beta\big)\Big)\bigg)^2+\bigg((1-t)\Big(y^a-\big((1-t^a)y^\alpha+t^ay^\beta\big)\Big)\bigg)^2}+\\
&\sqrt{\bigg(t\Big(x^b-\big((1-t^b)x^\alpha+t^bx^\beta\big)\Big)\bigg)^2+\bigg(t\Big(y^b-\big((1-t^b)y^\alpha+t^by^\beta\big)\Big)\bigg)^2}
\end{aligned}\tag{A.23}$$

$$\begin{aligned}
=\;&(1-t)\sqrt{\Big(x^a-(x^\alpha+(x^\beta-x^\alpha)t^a)\Big)^2+\Big(y^a-(y^\alpha+(y^\beta-y^\alpha)t^a)\Big)^2}+\\
&t\sqrt{\Big(x^b-(x^\alpha+(x^\beta-x^\alpha)t^b)\Big)^2+\Big(y^b-(y^\alpha+(y^\beta-y^\alpha)t^b)\Big)^2}
\end{aligned}\tag{A.24}$$

$$= (1-t)\rho_i\left(\tau_i(0),\sigma_i(t^a)\right)+t\rho_i\left(\tau_i(1),\sigma_i(t^b)\right) \tag{A.25}$$

$$< (1-t)\delta+t\delta \tag{A.26}$$

$$= \delta \tag{A.27}$$

where we used the triangle inequality from Equation A.22 to Equation A.23. Therefore $\tau_i(t) \in \left(Q_\delta(c^\alpha, c^\beta)\right)_i$ so $\left(\pi(c^a, c^b)\right)_i \subseteq \left(Q_\delta(c^\alpha, c^\beta)\right)_i$.

■

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Tables

In this appendix, we give the running times for all 50 trials in all domains.

B.1 Plate Domain

Trial	World 0		World 1		World 2	
	DARRT	DARRTConnect	DARRT	DARRTConnect	DARRT	DARRTConnect
1	24.8530082703	12.1420135498	73.1299514771	42.0676498413	249.893295	146.159210205
2	25.2604675293	25.0727005005	13.7393503189	24.6021938324	293.295013	140.922454834
3	1.39321279526	28.9283447266	113.751396179	6.79439735413	27.727839	80.7767028809
4	6.58768558502	15.3374214172	5.4530415535	13.9056224823	616.887329	31.1870918274
5	34.8496665955	4.04470157623	78.2549591064	6.70723724365	229.391235	111.829048157
6	10.5147733688	13.938741684	119.639266968	16.6826686859	34.819622	87.3661804199
7	6.9623632431	10.944934845	11.4734802246	80.6182861328	125.461395	81.9426956177
8	8.52929210663	1.57593238354	25.2328624725	3.72853970528	103.840454	165.623916626
9	6.36330938339	10.4848861694	14.1433954239	74.1298294067	110.530426	135.882659912
10	2.5048365593	6.56440258026	23.4186458588	5.14542293549	103.713783	16.28723526
11	5.85524654388	25.1558418274	88.903755188	19.3326358795	56.023216	165.524490356
12	7.23881959915	10.0918493271	11.8443155289	6.91189146042	342.727875	8.84900283813
13	7.91177129745	11.7331800461	17.1807689667	49.118019104	200.344696	51.200843811
14	11.1774396896	5.62956094742	48.4593162537	59.9919242859	549.797974	106.101425171
15	30.4044265747	8.80497550964	9.48311424255	74.9314346313	32.650772	69.6569519043
16	21.4986953735	4.03668451309	17.613407135	72.1267166138	106.684357	15.3522491455
17	14.2782850266	5.1648106575	9.73727035522	30.9724788666	205.284531	18.8743476868
18	2.69238615036	8.57702159882	18.3255405426	10.2306470871	13.920184	41.6143951416
19	4.00860548019	6.89589118958	14.7979955673	43.8586807251	33.711994	20.3964366913
20	3.57081007957	21.5716667175	38.7284088135	15.9595556259	47.907951	165.553771973
21	21.5688858032	1.82173407078	76.3397674561	46.0464286804	151.041046	14.4170322418
22	3.96931648254	52.037361145	107.836196899	3.38854384422	300.340302	11.6623382568
23	4.26936101913	6.67952632904	6.56545066833	64.5071563721	23.714552	208.241821289
24	8.13865947723	5.72200727463	252.50302124	18.6352138519	29.979578	76.7551574707
25	8.94091320038	5.43474340439	15.1236419678	57.5368003845	22.757980	16.1933917999
26	34.6880874634	15.5247335434	15.1384954453	74.0634078979	32.839401	202.743713379
27	20.4043388367	25.7725257874	88.0833435059	22.6156997681	66.574669	82.5219116211
28	3.16124796867	4.28062057495	13.2173423767	11.613699913	119.151337	21.8832569122
29	6.55652999878	26.4314022064	12.5391941071	12.778676033	383.560425	139.846054077
30	6.45152378082	8.33593273163	21.430524826	50.3813972473	109.697693	49.0341949463
31	7.31854343414	4.7619805336	20.5888805389	24.4047489166	189.968063	388.658416748
32	11.3752279282	20.3968238831	76.703125	28.1214790344	44.682709	140.810211182
33	4.94772624969	17.7540454865	17.268995285	20.6296539307	41.860809	75.3992462158
34	10.0191383362	18.434961319	76.8541641235	12.7614784241	191.846359	300.830688477
35	16.5032730103	19.5638713837	9.04199504852	78.7243270874	144.127747	11.45429039
36	6.41973495483	3.32170748711	6.60248088837	24.5229492188	26.909597	110.356903076
37	18.3793029785	5.81441640854	9.80319786072	20.9650363922	132.933868	20.7336673737
38	8.07968616486	7.17969083786	7.42400979996	22.4163188934	83.398178	298.345977783
39	2.51591491699	11.4777317047	46.6809654236	29.6807250977	19.122221	77.8001251221
40	27.8308448792	3.33804440498	8.23838329315	21.6507816315	265.117950	30.568769455
41	20.5548667908	2.11848163605	42.6840248108	48.3655166626	377.317657	115.546722412
42	7.54402828217	5.99009132385	67.6187973022	29.9604053497	53.312374	54.084941864
43	4.62053966522	10.7399549484	51.864276886	20.4585571289	32.026329	198.312072754
44	4.80425071716	4.71338653564	83.1271286011	4.14812231064	30.237673	87.2501831055
45	7.31790399551	5.49134540558	5.69688224792	19.5693855286	48.990459	41.8829841614
46	24.7185401917	8.72538757324	189.450073242	22.8749008179	20.776333	27.6178474426
47	36.4168319702	2.40024852753	5.81682348251	6.48738527298	189.714615	208.83732605
48	8.1096868515	13.3253202438	7.25219297409	98.4884567261	68.492538	26.8554801941
49	8.12162685394	8.4445734024	10.0322265625	10.3282060623	291.365051	55.2532501221
50	6.02952289581	6.77755069733	19.449678421	115.472412109	147.848297	135.318115234

Table B.1: Full results for DARRT and DARRTConnect in Plate Domain Worlds 0-2.

Trial	World 3		World 4	
	DARRT	DARRTConnect	DARRT	DARRTConnect
1	367.583892822	210.611572266	255.725891113	15.4208374023
2	1579.33691406	193.164245605	213.909637451	11.6602878571
3	2626.99829102	174.053771973	36.8034553528	160.407226562
4	1750.4621582	715.125976562	3690.42211914	67.4789428711
5	140.23260498	385.847747803	247.963043213	165.55897522
6	558.485168457	100.527709961	298.79586792	30.642539978
7	1366.82739258	431.820556641	54.431224823	28.0328865051
8	1545.48608398	646.145446777	137.739562988	301.178222656
9	979.06262207	546.159057617	406.419555664	140.871765137
10	474.736907959	79.0073852539	105.601539612	111.674285889
11	689.461914062	562.632080078	20.778137207	1202.10192871
12	1244.3416748	59.7328414917	659.324645996	95.5570983887
13	2860.0625	697.594482422	391.571685791	281.907958984
14	1536.46118164	998.137451172	140.656173706	148.39932251
15	714.623657227	376.095458984	721.561523438	224.211212158
16	385.794433594	1020.36578369	351.111175537	167.585968018
17	899.578552246	435.392181396	624.843200684	173.573989868
18	1980.51416016	113.851020813	126.640625	153.254257202
19	1530.95275879	517.042785645	186.806182861	20.3062171936
20	1115.7824707	71.8495101929	73.9352645874	78.9277648926
21	520.011535645	530.716003418	309.613342285	752.55657959
22	181.343505859	229.849655151	204.655593872	68.4412536621
23	1678.1229248	447.050079346	105.188583374	117.597724915
24	483.055786133	442.75769043	387.530273438	140.246231079
25	1071.29772949	270.146057129	864.330383301	42.1528778076
26	467.253936768	673.564025879	571.908691406	40.4016647339
27	377.86151123	312.00958252	853.254150391	244.103271484
28	138.536392212	406.053222656	624.840576172	72.2246780396
29	995.585021973	446.509246826	111.679298401	324.235229492
30	649.400878906	835.720092773	1610.47180176	20.6438312531
31	2390.11401367	645.548522949	624.893432617	41.927570343
32	158.082061768	533.000488281	417.511016846	174.619384766
33	756.917724609	48.7792129517	668.773376465	363.991241455
34	1266.79040527	780.175170898	215.712005615	84.8227996826
35	544.521728516	325.512451172	283.296386719	32.968952179
36	307.009399414	355.85357666	345.435089111	192.707992554
37	314.2628479	914.446411133	59.9098892212	118.498603821
38	1551.80310059	76.009513855	17.3104076385	11.7908306122
39	1896.92236328	593.433410645	224.63885498	134.351104736
40	374.479309082	119.24407959	372.879180908	73.0841598511
41	1038.3560791	22.9310703278	69.5134735107	133.628219604
42	960.861206055	85.0711975098	12.1451253891	10.9917583466
43	189.159255981	61.1502380371	571.496398926	46.4175224304
44	1972.78955078	790.879516602	48.8462486267	108.626022339
45	38.9730796814	1320.57421875	89.8367614746	135.536056519
46	588.40625	1294.51367188	197.476089478	320.197631836
47	1322.61669922	112.915489197	120.623580933	89.0702362061
48	2115.43334961	187.160049438	770.010559082	253.180953979
49	1342.33959961	184.864730835	570.314575195	337.763580322
50	149.555038452	425.860870361	468.7472229	189.515106201

Table B.2: Full results for DARRT and DARRTConnect in Plate Domain Worlds 3-4.

World 0 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	10.0581483841	0.666657805443	19.6527957916	5.29871034622	35.6763123274
2	9.63562965393	0.90920650959	6.52556800842	2.55474281311	19.6251469851
3	6.95868968964	1.42499411106	11.804643631	5.23322820663	25.4215556383
4	7.3607840538	0.545476138592	6.82496213913	1.97862267494	16.7098450065
5	10.045586586	1.09755170345	5.487408638	3.34807562828	19.9786225557
6	1.93174004555	0.598107457161	5.75304794312	5.83460283279	14.1174982786
7	7.58034944534	0.672174096107	3.61913132668	7.12990188599	19.0015567541
8	10.1616106033	0.742486596107	1.05201208591	12.6282539368	24.5843632221
9	5.58162260056	0.520260632038	4.93288564682	3.24565339088	14.2804222703
10	5.73899126053	1.15896689892	4.37699794769	7.83190250397	19.1068586111
11	3.02588367462	0.677036702633	3.77662682533	1.51018798351	8.9897351861
12	10.3808097839	0.730695009232	6.80448389053	1.06954848766	18.9855371714
13	9.3824634552	0.417532444	26.6126174927	4.49890756607	40.9115209579
14	4.8500752449	1.18757665157	3.81243276596	17.4506950378	27.3007797003
15	6.8124370575	1.86716878414	4.45329046249	4.11599063873	17.2488869429
16	11.0491323471	0.686296343803	13.8456125259	5.81766891479	31.3987101316
17	6.69067430496	0.60262042284	2.7519338131	8.31369686127	18.3589254022
18	10.0256290436	1.61810386181	1.30548155308	0.548959255219	13.4981737137
19	4.00505304337	2.94864320755	4.8764257431	4.29804992676	16.1281719208
20	7.1207575798	0.566672444344	1.75462460518	6.19009542465	15.632150054
21	10.173122406	0.539880812168	6.69736194611	2.39030504227	19.8006702065
22	1.29662156105	1.21795856953	4.33400249481	2.87750196457	9.72608458996
23	5.61664676666	0.924206972122	3.19254732132	4.66183280945	14.3952338696
24	7.90370559692	0.210321336985	25.096578598	9.01231384277	42.2229193747
25	1.55820202827	0.244523867965	12.6718997955	2.08230566978	16.5569313616
26	1.82846605778	0.858703792095	3.84914016724	1.56576120853	8.10207122564
27	10.1007289886	0.967902064323	4.5346493721	1.41323590279	17.0165163279
28	11.3995447159	2.09655404091	4.45740127563	7.45681476593	25.4103147984
29	7.06721353531	0.501807749271	5.37927341461	6.10076284409	19.0490575433
30	5.46649694443	2.43378853798	9.62407112122	3.74750947952	21.2718660831
31	1.84097921848	1.42557299137	2.61807227135	4.54666471481	10.431289196
32	1.15236544609	0.431966215372	4.1535153389	0.719414234161	6.45726123452
33	7.037586689	0.736533164978	6.83439207077	5.48521995544	20.0937318802
34	10.0896091461	1.11966085434	5.36300468445	1.21149897575	17.7837736607
35	1.16219556332	1.61077439785	4.95377635956	8.17110919952	15.8978555202
36	5.65386676788	0.27796626091	2.31303119659	8.00980854034	16.2546727657
37	10.1257638931	0.630290985107	4.63465976715	7.50040531158	22.891119957
38	7.11653184891	0.559719920158	17.114944458	6.65879440308	31.4499906301
39	5.03109645844	2.48618459702	5.89921092987	2.45361304283	15.8701050282
40	3.92205190659	1.00453662872	1.09936761856	6.63079452515	12.656750679
41	5.94177436829	0.739830672741	5.98539590836	24.3135757446	36.980576694
42	10.3188638687	0.872153520584	6.55288267136	1.98886072636	19.732760787
43	6.21843910217	0.683314800262	6.2495303154	0.689108788967	13.8403930068
44	3.46734189987	1.12433445454	12.9101829529	4.76238584518	22.2642451525
45	9.0588350296	0.52515912056	3.91918802261	2.63196063042	16.1351428032
46	6.49089479446	0.68694549799	6.58229827881	5.84614038467	19.6062789559
47	1.86415433884	0.506205320358	5.6764831543	1.56766140461	9.6145042181
48	8.06334114075	1.16628205776	18.6608486176	6.34769678116	34.2381685972
49	10.9325094223	0.611809015274	7.42066907883	7.77641820908	26.7414057255
50	8.75354671478	0.680736899376	6.29850673676	14.3694257736	30.1022161245

Table B.3: Full results for DARRTH in Plate Domain World 0

World 0 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	0.77250289917	0.211775064468	3.29035043716	1.00748324394	5.28211164474
2	2.65905427933	0.297981888056	11.6811294556	2.1125805378	16.7507461607
3	0.784318447113	0.337315499783	6.98726272583	1.22238337994	9.33128005266
4	1.37117481232	0.739108324051	1.7576379776	1.57804536819	5.44596648216
5	11.5213308334	0.206532031298	10.8310050964	6.5007815361	29.0596494973
6	3.53317689896	0.41279605031	1.04726982117	0.800378441811	5.79362121224
7	1.1070587635	0.494071781635	24.9291381836	1.14325928688	27.6735280156
8	1.09528696537	0.717492818832	24.4413509369	6.49080514908	32.7449358702
9	1.14446783066	2.83719134331	4.16814994812	2.44115066528	10.5909597874
10	1.41628861427	0.422903388739	6.59776353836	3.20862460136	11.6455801427
11	1.12380313873	0.312393128872	7.29653024673	4.01561641693	12.7483429313
12	1.56983590126	0.287453144789	10.1881437302	3.27692461014	15.3223573864
13	3.61839532852	0.787944734097	39.0582313538	1.49801337719	44.9625847936
14	1.33846330643	0.599414646626	6.6506357193	5.70009279251	14.2886064649
15	1.04371261597	0.257680356503	3.91077232361	1.17258238792	6.384747684
16	6.37299776077	0.171077787876	7.3791103363	6.71247911453	20.6356649995
17	0.988083004951	0.395926207304	2.18427276611	3.79027462006	7.35855659842
18	1.21800017357	0.326641589403	2.42505216599	3.0164744854	6.98616841435
19	6.08577013016	0.174622729421	5.20431375504	2.28391242027	13.7486190349
20	2.98166775703	0.609936118126	2.15495610237	3.35265374184	9.09921371937
21	1.13866317272	0.69756269455	23.443775177	2.54325461388	27.8232556581
22	2.88499116898	0.291399985552	20.2630634308	4.81859922409	28.2580538094
23	1.42451429367	0.244793385267	4.41784906387	2.44327378273	8.53043052554
24	2.28193116188	0.163116902113	51.520236969	1.63318967819	55.5984747112
25	3.22830533981	0.183196872473	3.47740912437	0.778833210468	7.66774454713
26	4.34945678711	0.2389652282	1.32717502117	1.00935506821	6.92495210469
27	1.00291192532	2.08697223663	2.42127156258	1.96620392799	7.47735965252
28	0.720756173134	0.457884252071	0.694322049618	1.02067804337	2.89364051819
29	0.958373010159	0.621207356453	7.11837863922	1.23769068718	9.93564969301
30	1.23613238335	0.251918822527	4.47292709351	3.10256028175	9.06353858113
31	1.10012340546	0.376365751028	25.2006950378	1.43509197235	28.1122761667
32	8.97057056427	0.270092070103	7.11261701584	10.3076887131	26.6609683633
33	1.63150405884	0.398513972759	68.3379745483	2.34519815445	72.7131907344
34	1.11100959778	0.36914741993	27.3090896606	3.39584326744	32.1850899458
35	1.84883570671	0.650287330151	38.2629928589	2.89310765266	43.6552235484
36	2.64778804779	0.245967850089	4.39626932144	4.9522690773	12.2422942966
37	1.01673328876	4.65399885178	6.99197244644	2.95672130585	15.6194258928
38	2.1427628994	0.5213804245	14.3709201813	3.729814291	20.7648777962
39	1.22023761272	0.283925831318	6.39967727661	0.796385109425	8.70022583008
40	0.727526664734	0.737427473068	1.85557830334	2.49234819412	5.81288063526
41	8.32415485382	0.385351628065	15.721507907	1.91111648083	26.3421308696
42	1.05238735676	0.305933386087	35.4985198975	20.750164032	57.6070046723
43	0.894856333733	0.251848012209	10.941447258	3.57214331627	15.6602949202
44	1.01995790005	0.296290397644	7.01096343994	2.38149809837	10.708709836
45	1.09640491009	0.353909611702	1.63499701023	2.91849303246	6.00380456448
46	1.24851119518	1.0046633482	6.16528940201	2.74615740776	11.1646213531
47	0.994697093964	0.279809862375	8.43884563446	0.614100456238	10.327453047
48	3.37482333183	4.64140748978	13.776389122	1.78101301193	23.5736329556
49	12.953420639	0.709335803986	5.83460283279	7.18181848526	26.6791777611
50	1.03780460358	0.721106410027	1.77448940277	1.06585085392	4.59925127029

Table B.4: Full results for DARRTHCONNECT in Plate Domain World 0

World 1 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	4.78764104843	0.589584708214	15.5603494644	15.0096683502	35.9472435713
2	2.82160806656	0.570531487465	3.30339431763	15.4126996994	22.1082335711
3	2.70826983452	0.441555947065	7.33828926086	25.2867889404	35.7749039829
4	5.07540559769	0.372867047787	1.52293682098	8.76914978027	15.7403592467
5	4.25285339355	2.4445836544	0.973838150501	7.06217718124	14.7334523797
6	8.75323009491	1.35064053535	7.18405199051	26.0156860352	43.3036086559
7	9.29387760162	1.4322963953	2.39652991295	24.0366592407	37.1593631506
8	3.61681127548	1.17284822464	5.6984667778	8.44541168213	18.9335379601
9	3.33641672134	0.468669861555	3.08033370972	35.1659240723	42.0513443649
10	8.18855667114	0.918941915035	8.08543205261	39.7921600342	56.985090673
11	6.12625455856	0.743388652802	3.86690807343	3.07705187798	13.8136031628
12	2.70412921906	0.518957018852	8.09117889404	14.7440042496	26.0582693815
13	5.25935840607	0.501424670219	2.41217899323	22.7931022644	30.9660643339
14	4.61229896545	1.30797874928	4.6596326828	10.8840827942	21.4639931917
15	10.3516693115	0.500996112823	7.90878915787	22.3207702637	41.0822248459
16	8.87123012543	0.302240490913	4.07279539108	40.1945419312	53.4408079386
17	3.09781312943	0.453988105059	3.71165728569	20.8898849487	28.1533434689
18	2.57711029053	0.706234514713	3.60648202896	5.2700047493	12.1598315835
19	9.2982378006	2.10246539116	3.59152436256	14.1933116913	29.1855392456
20	3.20240259171	0.593235433102	23.072807312	106.288330078	133.156775415
21	3.64880657196	1.20758855343	1.10806262493	153.517166138	159.481623888
22	4.90557527542	1.28544425964	25.585269928	14.680809021	46.457098484
23	9.8557472229	0.599272370338	2.58410286903	7.39678478241	20.4359072447
24	5.64633369446	0.493772298098	6.86467504501	7.82480573654	20.8295867741
25	2.60432291031	1.2976680994	5.92829036713	26.688369751	36.5186511278
26	7.70213031769	0.782212197781	4.90955018997	104.463027954	117.85692066
27	3.18292665482	0.700020849705	3.76881957054	24.6571846008	32.3089516759
28	3.89699149132	0.389903306961	4.59352731705	11.5032138824	20.3836359978
29	5.85106039047	0.17964720726	29.5932235718	13.9356060028	49.5595371723
30	7.57531738281	1.22707307339	5.81648254395	11.7694530487	26.3883260489
31	10.3673830032	0.84098726511	7.08808517456	29.1398391724	47.4362946153
32	10.3187980652	0.330533832312	12.9720058441	40.1392593384	63.76059708
33	10.1879711151	0.751051306725	8.20904159546	27.7105560303	46.8586200476
34	4.2794303894	0.906408667564	6.93849420547	25.8805160522	38.0048493147
35	11.1946077347	0.57596629858	22.0611419678	10.0315322876	43.8632482886
36	6.60582351685	0.859153807163	4.92367124557	9.956615448	22.3452640176
37	10.7474994659	0.424735605717	5.2021522522	8.66482639313	25.039213717
38	10.8031225204	0.49695700407	2.94008660316	10.8116436005	25.0518097281
39	1.89200341702	2.99555969238	0.935333848	26.9795227051	32.8024196625
40	7.3402466774	1.34759819508	5.73640871048	12.1646223068	26.5888758898
41	7.63870334625	1.33436799049	2.16381287575	6.12968015671	17.2665643692
42	4.28087472916	3.64208078384	11.3608522415	11.2052297592	30.4890375137
43	1.89094138145	0.441799223423	18.029384613	22.8456287384	43.2077539563
44	7.48298692703	0.529619574547	8.33093738556	5.01833057404	21.3618744612
45	6.88019609451	1.01289987564	6.70697689056	22.2716522217	36.8717250824
46	7.02991294861	0.429287642241	9.79586982727	53.0355758667	70.2906462848
47	7.30086660385	0.759940266609	2.42251324654	37.5440292358	48.0273493528
48	6.97370481491	0.361464142799	21.0186691284	13.0515708923	41.4054089785
49	4.0736413002	0.37124273181	2.62673592567	53.7460250854	60.8176450431
50	6.37729215622	0.351568162441	3.41607022285	21.4727783203	31.6177088618

Table B.5: Full results for DARRTH in Plate Domain World 1

World 1 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	1.94421684742	3.77603292465	14.0128555298	11.6170558929	31.3501611948
2	5.44019556046	0.350419014692	3.04821610451	2.11226940155	10.9511000812
3	4.37092351913	0.869492828846	4.67573547363	4.89420843124	14.8103602529
4	1.80377137661	2.12239575386	1.21403455734	20.1420345306	25.2822362185
5	2.2434732914	0.431173235178	28.6069107056	12.6894006729	43.9709579051
6	2.10675191879	0.438759565353	5.48283910751	10.9249153137	18.9532659054
7	1.5807518959	0.725228130817	21.9539871216	8.20439624786	32.4643633962
8	3.0912771225	0.245635479689	18.5514125824	2.7112057209	24.5995309055
9	1.72975969315	0.32561892271	20.2341442108	13.9499969482	36.2395197749
10	1.87656617165	8.60628318787	17.6757354736	8.19093513489	36.349519968
11	7.07995223999	2.53013586998	0.988551139832	13.1831684113	23.7818076611
12	3.64275979996	0.296676278114	31.6832313538	4.32394313812	39.94661057
13	1.71306347847	0.81223076582	23.4672050476	28.0456943512	54.0381936431
14	1.6740885973	0.83359849453	14.4837779999	25.3413848877	42.3328499794
15	7.62434387207	1.1021105051	6.890846848	12.1569261551	27.7742273808
16	1.66048014164	0.532732903957	9.10782051086	3.95196390152	15.252997458
17	5.11535596848	1.4532558918	12.4902133942	28.3635349274	47.4223601818
18	1.64408016205	0.481240779161	14.3930273056	3.46645855904	19.9848068058
19	8.40407657623	0.280127972364	7.257784367	5.39804553986	21.3400344551
20	1.92864573002	0.389704942703	26.2264213562	3.26612949371	31.8109015226
21	2.5990626812	1.13657963276	12.0322732925	11.2690572739	27.0369728804
22	2.63362669945	0.221067011356	0.940578520298	9.63028907776	13.4255613089
23	3.02254104614	0.319279432297	9.70112800598	4.16278457642	17.2057330608
24	4.3726978302	0.304238468409	25.5336685181	7.13268661499	37.3432914317
25	2.02778625488	0.490239948034	6.05942058563	7.25595617294	15.8334029615
26	1.69765222073	0.404244989157	4.27059555054	9.47785949707	15.8503522575
27	2.80549764633	0.236935153604	3.47545862198	18.9721641541	25.490055576
28	2.21663117409	0.17310141027	2.7622859478	9.28174591064	14.4337644428
29	1.92056524754	0.240494117141	8.86918735504	2.73924970627	13.769496426
30	2.30223345757	0.431531727314	8.61051845551	8.16934776306	19.5136314034
31	1.72706925869	0.646061241627	4.89734649658	8.77074050903	16.0412175059
32	1.84914815426	0.50656235218	13.5989198685	13.1125926971	29.0672230721
33	1.83560967445	0.910371243954	7.3425359726	22.1429443359	32.2314612269
34	2.10268521309	0.392475008965	0.955797553062	2.26816606522	5.71912384033
35	1.86880648136	0.218884766102	4.50629091263	12.0078048706	18.6017870307
36	2.13764882088	0.343217939138	27.0759525299	14.6480827332	44.2049020231
37	2.06946706772	0.309035569429	5.97068977356	23.2849292755	31.6341216862
38	1.87508022785	0.623503684998	3.26139307022	24.1428775787	29.9028545618
39	6.58712244034	0.725449442863	21.6045875549	36.6339874268	65.5511468649
40	4.85581016541	0.264449447393	1.43847048283	3.08997511864	9.64870521426
41	2.30187797546	0.616336882114	21.9828529358	21.5434265137	46.444494307
42	2.42123174667	0.263936668634	40.9948692322	3.86170125008	47.5417388976
43	2.37802052498	3.61549496651	4.81535387039	9.16097450256	19.9698438644
44	1.95571422577	0.206673726439	6.81780290604	7.34565830231	16.3258491606
45	6.02977848053	0.449420630932	2.94046497345	5.07032632828	14.4899904132
46	4.31649017334	0.203835397959	18.0245418549	12.6641073227	35.2089747488
47	7.85996723175	0.230847686529	1.40056276321	34.9594497681	44.4508274496
48	3.0130007267	0.207042768598	38.2702560425	12.0421533585	53.5324528962
49	7.43368291855	0.220334917307	1.13951516151	9.49820423126	18.2917372286
50	2.13546180725	0.364076167345	17.03565979	24.0337791443	43.5689769089

Table B.6: Full results for DARRTHConnect in Plate Domain World 1

World 2 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	2.49189805984	1.26490223408	27.2822151184	2.13565468788	33.1746701002
2	4.51882648468	4.17787647247	17.2005615234	4.08259344101	29.9798579216
3	2.79570436478	61.430519104	1.5619045496	2.94548082352	68.7336088419
4	7.05079030991	1.10197138786	1.30066680908	10.1092996597	19.5627281666
5	10.8418712616	20.9510784149	4.64358282089	10.1970443726	46.63357687
6	3.80433154106	18.2057056427	2.13426375389	14.7150497437	38.8593506813
7	9.01696968079	5.713367939	13.7708654404	27.7856407166	56.2868437767
8	8.26310348511	2.62377238274	2.89901041985	24.8862991333	38.672185421
9	10.1786146164	3.0310280323	2.80115914345	7.00165462494	23.0124564171
10	5.52416563034	17.1764011383	6.3165397644	14.7211141586	43.7382206917
11	8.38958930969	10.2050600052	3.61300492287	9.74182128906	31.9494755268
12	6.11663007736	12.8634605408	16.3455162048	8.5377779007	43.8633847237
13	10.4378404617	24.7069740295	2.97226524353	4.35484790802	42.4719276428
14	2.85339021683	21.2144565582	1.59957408905	15.816860199	41.4842810631
15	2.88555598259	1.26814711094	33.2152557373	4.47737598419	41.846334815
16	2.76135373116	2.6442861557	4.90672969818	6.6007270813	16.9130966663
17	10.0554389954	66.7451019287	2.47155547142	15.2965812683	94.5686776638
18	3.52063918114	5.55808734894	7.83217906952	7.22442674637	24.135332346
19	3.7051076889	2.52277398109	1.79204618931	26.4379673004	34.4578951597
20	6.98112773895	18.2537956238	2.00211286545	12.4302053452	39.6672415733
21	2.45875382423	10.9334459305	3.96483874321	10.9029893875	28.2600278854
22	1.99806904793	2.24364995956	3.98862433434	38.5663414001	46.796684742
23	3.55255556107	31.1992607117	5.86001634598	23.1163711548	63.7282037735
24	9.40247631073	20.3865737915	5.72411346436	4.90935659409	40.4225201607
25	8.02344608307	1.22642028332	2.04571056366	100.181144714	111.476721644
26	5.78351545334	5.79784822464	2.19298791885	35.2073669434	48.9817185402
27	4.71351289749	6.60465478897	3.90840554237	5.8861913681	21.1127645969
28	7.59382677078	16.7812099457	4.45917034149	12.8993415833	41.7335486412
29	10.1957654953	6.4538640976	2.00347518921	41.5674591064	60.2205638885
30	4.94217586517	1.51500868797	1.34912395477	11.54155159	19.3478600979
31	2.61847639084	7.24054670334	9.28338050842	22.1961078644	41.338511467
32	5.34582042694	1.38750970364	1.57677662373	41.083114624	49.3932213783
33	4.75865697861	1.83455371857	1.96269762516	29.2322406769	37.7881489992
34	7.65729475021	19.5283870697	12.5748376846	49.6720657349	89.4325852394
35	5.42952346802	13.729970932	3.25226902962	55.4852294922	77.8969929218
36	6.35753679276	5.70624256134	9.47665405273	38.410785675	59.9512190819
37	11.0065479279	38.0491676331	5.15044879913	21.7169380188	75.9231023788
38	8.02982330322	49.0617599487	4.73487234116	12.6998577118	74.5263133049
39	10.8436203003	2.2630906105	4.21925830841	42.4823226929	59.8082919121
40	10.5298376083	17.0605373383	1.89064383507	14.9415464401	44.4225652218
41	3.59767079353	2.1459479332	1.2221852541	14.3932151794	21.3590191603
42	4.56245613098	4.06388807297	1.93878114223	10.1730947495	20.7382200956
43	4.93484258652	15.7446289062	10.247549057	29.214012146	60.1410326958
44	2.65580058098	11.7198867798	4.35440349579	6.94386053085	25.6739513874
45	3.8861811161	5.49581432343	9.55946731567	56.7083969116	75.6498596668
46	9.64785385132	1.48798286915	4.58122491837	18.7998981476	34.5169597864
47	3.24654698372	5.16654157639	24.3847408295	23.4526443481	56.2504737377
48	11.1818084717	8.91835021973	10.5822114944	14.1563501358	44.8387203217
49	4.44073629379	1.86481809616	116.019058228	7.83159637451	130.156208992
50	10.6401920319	17.5552406311	1.6051337719	3.86796617508	33.6685326099

Table B.7: Full results for DARRTH in Plate Domain World 2

World 2 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	2.62128138542	12.2281827927	52.3579902649	5.83093881607	73.038393259
2	5.83433294296	1.75788235664	7.21719408035	16.0399055481	30.8493149281
3	1.88467860222	3.27892971039	11.6603183746	6.25694608688	23.0808727741
4	1.78420984745	6.74601364136	10.2235164642	7.83218717575	26.5859271288
5	5.75171995163	0.61735868454	4.27274179459	6.81009578705	17.4519162178
6	1.86637341976	12.6878967285	4.68699979782	6.75252532959	25.9937952757
7	2.45440864563	3.93575119972	13.301692009	4.37132310867	24.063174963
8	1.75813043118	2.80891942978	25.3752365112	5.32546520233	35.2677515745
9	2.13940811157	2.3551299572	17.1120376587	4.98281669617	26.5893924236
10	2.34984493256	1.85613107681	2.92092871666	9.68410205841	16.8110067844
11	4.46075487137	1.88597142696	66.4610443115	8.74144649506	81.5492171049
12	2.10559463501	30.7205600739	54.2392959595	5.66878461838	92.7342352867
13	3.68032217026	6.36663007736	17.6463851929	7.54360675812	35.2369441986
14	2.13699769974	6.57594060898	0.878984749317	9.90987586975	19.5017989278
15	2.29558205605	6.13386249542	5.26671361923	2.85487055779	16.5510287285
16	2.08367681503	3.09759497643	48.7572441101	3.23270344734	57.1712193489
17	2.17204356194	2.58675932884	12.0778112411	6.22782468796	23.0644388199
18	1.6760392189	32.1911277771	9.77958679199	3.1691942215	46.8159480095
19	2.2003531456	24.2599258423	5.23855400085	16.3572864532	48.056119442
20	1.89315116405	11.1098222733	8.24953269958	9.18325901031	30.4357651472
21	3.55238604546	0.921060562134	4.97614812851	6.64737892151	16.0969736576
22	1.92967438698	9.39532279968	12.6187314987	16.1030902863	40.0468189716
23	1.76127791405	14.4684753418	41.556137085	19.7132854462	77.499175787
24	1.93929505348	8.325715065	4.73013830185	5.3877620697	20.38291049
25	2.72310233116	12.4431858063	21.7291202545	5.62714290619	42.5225512981
26	2.72095680237	7.41942501068	15.1311330795	4.76391410828	30.0354290009
27	2.23695969582	2.83442831039	6.58376455307	9.99410915375	21.649261713
28	3.14891171455	0.969143509865	9.55314922333	29.1351852417	42.8063896894
29	4.92058801651	1.38534748554	6.22513866425	4.21352529526	16.7445994616
30	2.0279405117	6.72682476044	3.84867024422	5.49237060547	18.0958061218
31	1.76199758053	4.84352207184	28.9494895935	4.092648983	39.6476582289
32	2.47430205345	13.8257627487	1.30226385593	8.44856071472	26.0508893728
33	2.61036491394	6.24213552475	16.8004417419	9.83232879639	35.485270977
34	4.88580799103	3.01181769371	7.94941282272	2.7129483223	18.5599868298
35	2.28476166725	7.57708215714	3.57374954224	3.17116570473	16.6067590714
36	2.24936795235	7.49249982834	4.76511001587	7.49430608749	22.001283884
37	2.16188526154	1.33727908134	45.1207427979	2.97182512283	51.5917322636
38	2.07175469398	22.8791179657	36.1560707092	7.74805545807	68.854998827
39	6.65632677078	2.6889064312	48.5637016296	5.82032060623	63.7292554379
40	2.24157261848	2.20181250572	22.9069728851	17.7443714142	45.0947294235
41	1.98871457577	4.96688556671	1.3398873806	2.44067907333	10.7361665964
42	1.59272611141	2.09875798225	11.6889076233	4.20969390869	19.5900856256
43	2.45858335495	14.5244731903	1.49714040756	21.124420166	39.6046171188
44	1.72076129913	1.561165452	13.2278652191	12.2560005188	28.7657924891
45	9.61679077148	1.51999092102	8.00356483459	21.773443222	40.9137897491
46	1.99442219734	1.01003170013	2.00633621216	23.0974712372	28.1082613468
47	5.46146583557	4.43232774734	6.22928524017	4.68294000626	20.8060188293
48	1.96150839329	14.6244220734	3.51145100594	4.21796321869	24.3153446913
49	2.5375187397	8.42338466644	48.2596588135	4.99154758453	64.2121098042
50	2.4963722229	8.37486553192	2.32063221931	2.63748240471	15.8293523788

Table B.8: Full results for DARRTHConnect in Plate Domain World 2

World 3 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	7.8907327652	6.60276079178	43.193523407	57.6638069153	115.350823879
2	3.55352878571	6.19611263275	2.41009688377	47.2013549805	59.3610932827
3	5.00985336304	32.6482276917	2.78916192055	17.7699699402	58.2172129154
4	2.49162578583	2.44809651375	3.90775299072	18.2983570099	27.1458323002
5	1.1993137598	31.4413566589	4.85798549652	57.8006744385	95.2993303537
6	4.76740932465	19.3047485352	12.7128744125	143.220062256	180.005094528
7	1.27584683895	3.93491983414	2.50017023087	20.0314598083	27.7423967123
8	6.59040260315	75.9561080933	12.3115558624	382.576965332	477.435031891
9	3.07402205467	6.13657331467	4.68238067627	13.3398141861	27.2327902317
10	5.48681497574	41.6020736694	2.71635603905	55.7078170776	105.513061762
11	11.8236656189	12.1665058136	9.93705558777	437.969390869	471.896617889
12	10.035779953	32.8629302979	3.99721431732	200.005966187	246.901890755
13	2.25229930878	5.09490728378	4.96930599213	57.0300064087	69.3465189934
14	13.9763689041	57.8907852173	7.2488117218	413.253936768	492.369902611
15	4.7372879982	33.2637634277	25.8378562927	76.0091018677	139.848009586
16	7.12491464615	8.42892169952	3.16275286674	111.642303467	130.358892679
17	5.20598173141	115.453186035	4.919069767	59.2300415039	184.808279037
18	6.63941240311	1.72712635994	69.08152771	115.255226135	192.703292608
19	5.52002286911	1.24439144135	3.90033149719	201.031646729	211.696392536
20	2.9967417717	59.2021026611	4.67078208923	85.1446990967	152.014325619
21	7.98266458511	30.4395027161	19.7087631226	538.503051758	596.633982182
22	4.34829378128	4.69843053818	1.96502888203	209.980438232	220.992191434
23	3.98487830162	7.90620851517	3.81024765968	75.9319458008	91.6332802773
24	2.55937838554	38.5395355225	1.339620471	173.484268188	215.922802567
25	6.12359142303	34.2564125061	6.19843244553	211.331893921	257.910330296
26	8.43472003937	5.70338916779	2.74023723602	266.712646484	283.590992928
27	8.25444316864	41.6228027344	2.14541864395	205.651138306	257.673802853
28	4.9013209343	10.7304191589	3.8191318512	81.6245803833	101.075452328
29	7.90921354294	11.0816774368	10.8994426727	424.859588623	454.749922276
30	3.17964172363	15.8693027496	12.7671060562	107.643341064	139.459391594
31	2.62261724472	4.42181015015	2.1988825798	57.7147865295	66.9580965042
32	13.456495285	58.014213562	10.7276449203	514.249633789	596.447987556
33	2.27827906609	7.44040489197	1.66595125198	19.7986755371	31.1833107471
34	18.3355674744	5.79215431213	17.1567153931	438.56842041	479.85285759
35	10.5635261536	1.80776309967	6.07386255264	91.1371536255	109.582305431
36	5.34636688232	44.8272857666	4.73830509186	28.319683075	83.2316408157
37	13.9721441269	87.3599090576	7.37683010101	383.802856445	492.511739731
38	5.65052032471	2.38554692268	6.68625354767	86.1936340332	100.915954828
39	6.32239484787	4.42726230621	2.95774078369	92.4370117188	106.144409657
40	4.07482051849	1.46703147888	13.4812545776	47.6858940125	66.7090005875
41	8.22327613831	38.4037895203	3.45763897896	414.715087891	464.799792528
42	7.07309103012	9.84250831604	5.59563159943	49.0313911438	71.5426220894
43	8.12314796448	38.4314460754	5.70619106293	447.332275391	499.593060493
44	3.51227283478	21.9594211578	1.33872008324	181.706954956	208.517369032
45	4.83108377457	6.11396074295	6.82408857346	137.24307251	155.012205601
46	10.3953409195	8.92914772034	10.1941022873	9.75995731354	39.2785482407
47	3.49341821671	12.2583341599	1.30236148834	168.356201172	185.410315037
48	10.2254362106	3.81149935722	4.03609085083	16.4452838898	34.5183103085
49	9.02659606934	6.9449133873	6.27011013031	61.1953277588	83.4369473457
50	1.59904158115	11.1236400604	2.88460612297	180.060394287	195.667682052

Table B.9: Full results for DARRTH in Plate Domain World 3

World 3 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	0.929114282131	4.74088811874	6.48641347885	12.6983690262	24.8547849059
2	1.22430670261	6.86392831802	2.53502845764	116.008399963	126.631663442
3	1.30420768261	9.11079978943	5.28566932678	49.6400375366	65.3407143354
4	0.942583978176	1.20010972023	13.2227430344	24.1811504364	39.5465871692
5	1.21112668514	7.00303506851	5.76414680481	26.6408596039	40.6191681623
6	6.84740924835	7.01873683929	12.4056348801	52.06016922	78.3319501877
7	2.46887254715	1.04820239544	12.9493494034	12.5287332535	28.9951575994
8	1.27395033836	5.1649222374	4.41120147705	79.3830566406	90.2331306934
9	3.09552168846	2.55534005165	20.8724536896	43.7585029602	70.2818183899
10	0.949756741524	30.1493415833	2.61545157433	5.1853222847	38.8998721838
11	1.79169344902	1.43177783489	10.2412624359	25.2130260468	38.6777597666
12	2.24330687523	4.15134429932	2.46278190613	66.5575637817	75.4149968624
13	3.17253565788	12.2899417877	7.10920953751	26.4891452789	49.060832262
14	2.09734249115	36.1228981018	2.43824791908	20.2479877472	60.9064762592
15	9.24330806732	10.2722387314	11.5309715271	52.8841285706	83.9306468964
16	1.89635980129	10.134303093	6.94453954697	22.2135334015	41.1887358427
17	1.85722208023	9.01490497589	3.70626044273	21.2865695953	35.8649570942
18	1.38661897182	6.14827537537	12.3438882828	31.5517654419	51.4305480719
19	1.88326323032	5.64798498154	7.34304666519	22.2931346893	37.1674295664
20	3.54475831985	5.37579059601	2.34941315651	24.385351181	35.6553132534
21	8.33368015289	6.17352962494	1.35775399208	46.6274108887	62.4923746586
22	2.18240690231	8.44568729401	14.3317203522	26.8276977539	51.7875123024
23	2.42037367821	5.25830554962	10.6564369202	13.5791044235	31.9142205715
24	1.93793201447	21.2429466248	1.35917365551	4.65693044662	29.1969827414
25	1.67767071724	7.31600999832	2.42593336105	136.478897095	147.898511171
26	6.61556768417	2.79746675491	3.47071456909	21.0511016846	33.9348506927
27	1.88492023945	4.10716199875	4.29547166824	19.5185909271	29.8061448336
28	1.44950056076	5.85103130341	8.36264896393	46.4041671753	62.0673480034
29	1.02242159843	20.0456485748	8.43298625946	38.8045806885	68.3056371212
30	1.56430494785	11.7178659439	2.57207274437	22.671667099	38.5259107351
31	1.32887220383	10.8763084412	6.09080648422	31.4054355621	49.7014226913
32	1.47244870663	5.86784696579	16.4855613708	15.7236289978	39.5494860411
33	1.1132311821	6.29814767838	9.32034873962	15.5731163025	32.3048439026
34	4.72129297256	7.49244976044	14.772605896	6.74910783768	33.7354564667
35	1.18035554886	35.2682533264	4.64648771286	106.938285828	148.033382416
36	9.21416568756	1.17567420006	2.79756379128	23.4559459686	36.6433496475
37	1.82555902004	4.12298965454	5.2324719429	53.7789382935	64.9599589109
38	1.27736711502	3.00823354721	15.6148452759	7.52915096283	27.4295969009
39	2.7960164547	10.6757974625	1.60404860973	113.809524536	128.885387063
40	4.74768447876	10.0868759155	8.46612453461	45.1031570435	68.4038419724
41	4.05335521698	2.64786434174	45.1536254883	58.9708251953	110.825670242
42	1.50410234928	7.50831317902	12.651925087	39.9401512146	61.6044918299
43	6.54908418655	16.7861366272	8.18709087372	26.9942512512	58.5165629387
44	1.59822165966	19.9258975983	8.37424373627	42.5979423523	72.4963053465
45	1.70748198032	16.3984012604	2.07405281067	54.9051933289	75.0851293802
46	1.97781300545	2.26871681213	11.4619693756	23.077703476	38.7862026691
47	1.44014799595	2.00521683693	1.24676132202	23.5902042389	28.2823303938
48	1.67306256294	2.12967061996	3.10993599892	199.058654785	205.971323967
49	1.90988004208	0.682794332504	19.3099956512	29.5457706451	51.448440671
50	1.215015769	5.99435758591	12.5405235291	47.2584457397	67.0083426237

Table B.10: Full results for DARRTHConnect in Plate Domain World 3

World 4 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	25.2843284607	78.2000045776	924.789123535	5.62371206284	1033.89716864
2	1.09546458721	17.568862915	28.0364379883	4.70013141632	51.4008969069
3	3.54665517807	8.89539432526	18.2001056671	1.18211770058	31.824272871
4	6.60044622421	10.962726593	194.37272644	2.46190190315	214.397801161
5	9.29562568665	11.674246788	187.929092407	2.28480601311	211.183770895
6	2.56353449821	62.0663223267	9.82335472107	1.67973387241	76.1329454184
7	10.3502922058	51.2245559692	14.8227319717	1.02563238144	77.4232125282
8	15.6908683777	34.611114502	196.279281616	0.824765622616	247.406030118
9	10.1499834061	69.604598999	9.35137462616	4.07552194595	93.1814789772
10	10.4952468872	40.3467216492	916.261901855	0.81055521965	967.914425611
11	6.123711586	4.88306665421	29.0124149323	11.8034439087	51.8226370811
12	19.7243232727	26.6162147522	183.82989502	2.90800189972	233.078434944
13	9.73361873627	8.16017723083	3.82040429115	16.5263767242	38.2405769825
14	16.6249313354	5.75381660461	213.280273438	3.90804219246	239.56706357
15	1.3722820282	6.77061700821	7.25317144394	5.66726970673	21.0633401871
16	5.51510286331	20.9680213928	11.1749649048	0.759667158127	38.417756319
17	5.09665870667	10.0243616104	52.9185791016	6.26924562454	74.3088450432
18	1.72661411762	10.6555185318	25.5595417023	1.15369236469	39.0953667164
19	14.007938385	13.2985801697	201.961685181	1.04935240746	230.317556143
20	5.20277738571	13.4249916077	129.23085022	24.6016254425	172.460244656
21	5.63352775574	98.6122512817	367.453704834	3.09864759445	474.798131466
22	15.1604003906	14.2281208038	198.168182373	12.3162193298	239.872922897
23	36.2528572083	96.0704956055	1089.17431641	0.447214096785	1221.94488332
24	9.97167873383	13.0851068497	6.16926670074	4.84184265137	34.0678949356
25	5.35928153992	9.09792709351	6.50700473785	3.5236518383	24.4878652096
26	11.0165157318	19.795085907	185.061935425	3.93378329277	219.807320356
27	3.06035971642	8.06404495239	4.91935920715	0.796135127544	16.8398990035
28	4.47540807724	21.533575058	3.90340805054	7.03294229507	36.9453334808
29	4.2007651329	12.7640924454	25.4298210144	10.2232923508	52.6179709435
30	7.79782152176	2.96528720856	13.9657382965	4.16791152954	28.8967585564
31	7.4377117157	10.2009534836	8.79982089996	15.1942987442	41.6327848434
32	3.12368702888	34.6879959106	29.9284191132	1.41312229633	69.153224349
33	24.0032539368	13.6701469421	545.774169922	2.66312432289	586.110695124
34	4.95538663864	3.73415923119	10.0610628128	7.49916601181	26.2497746944
35	3.60720348358	21.1913337708	6.55207777023	10.8914718628	42.2420868874
36	10.8424320221	6.43265485764	7.17212057114	1.06380951405	25.5110169649
37	2.05340147018	6.8101439476	10.1829395294	1.56842434406	20.6149092913
38	4.89312410355	11.8199510574	128.168457031	7.78244924545	152.663981438
39	8.77701282501	7.78746080399	2.10682916641	2.70132303238	21.3726258278
40	1.72717940807	5.84304904938	31.1204586029	0.72367978096	39.4143668413
41	5.19721603394	14.2569389343	2.47818493843	2.43276953697	24.3651094437
42	8.38301563263	21.5659866333	15.7159862518	1.41604673862	47.0810352564
43	10.0265522003	13.5906639099	188.193893433	5.71935796738	217.53046751
44	4.57757425308	16.9451618195	8.0528755188	7.7134809494	37.2890925407
45	25.4577865601	28.7496261597	554.601501465	7.55999469757	616.368908882
46	27.9480400085	40.0116271973	548.903137207	3.41113758087	620.273941994
47	11.6775617599	43.9597129822	562.872436523	3.60508608818	622.114797354
48	34.0974769592	88.4248123169	551.892822266	1.5200381279	675.93514967
49	20.6224784851	17.5649108887	194.774871826	9.96700191498	242.929263115
50	7.73839521408	10.123044014	34.5366668701	2.08737778664	54.4854838848

Table B.11: Full results for DARRTH in Plate Domain World 4

World 4 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Total
1	2.79370617867	29.1581344604	58.0486526489	5.22528648376	95.2257797718
2	1.14843916893	5.48514986038	48.0957679749	0.598185837269	55.3275428414
3	0.962378799915	23.7769775391	14.0270814896	2.39009284973	41.1565306783
4	0.981349408627	4.63312625885	0.918997943401	0.378318428993	6.91179203987
5	3.74295926094	13.5241346359	198.559249878	1.93018639088	217.756530166
6	1.79559743404	17.1353626251	2.77829766273	1.58441281319	23.2936705351
7	5.07720518112	8.0679807663	82.1609344482	1.0740994215	96.3802198172
8	0.960819244385	0.5955119133	64.3663330078	1.51015102863	67.4328151941
9	1.16293728352	5.52789735794	56.9191665649	2.92689418793	66.5368953943
10	2.37961411476	31.5399665833	216.428436279	1.54532384872	251.893340826
11	1.42528223991	75.5756225586	224.069442749	3.21074748039	304.281095028
12	4.84707403183	5.14457511902	84.8259811401	1.06264638901	95.88027668
13	5.91220664978	24.3212947845	363.502960205	0.959941208363	394.696402848
14	4.99659252167	58.45860672	316.313415527	1.2546659708	381.02328074
15	8.63805580139	95.0071105957	652.829711914	1.57089078426	758.045769095
16	4.15634012222	20.7541160583	186.611480713	2.28306889534	213.805005789
17	4.2003493309	18.1458034515	234.793136597	1.77411818504	258.913407564
18	4.46081972122	19.2995605469	344.230865479	2.85281038284	370.844056129
19	7.81307792664	107.34262085	167.266586661	0.859200775623	283.281486213
20	2.1418762207	84.4508285522	244.038482666	0.321936994791	330.953124434
21	1.39573478699	3.3867790699	15.2276859283	1.05718064308	21.0673804283
22	0.653962731361	5.9354929924	51.7370262146	1.3844575882	59.7109395266
23	6.59463024139	21.4406280518	27.3012962341	1.32996499538	56.6665195227
24	2.89526891708	27.4189224243	375.818847656	4.22311449051	410.356153488
25	0.608731508255	2.44846200943	1.35760986805	1.40406680107	5.81887018681
26	1.4781588316	8.98510742188	1.34413206577	65.2973098755	77.1047081947
27	2.48995280266	24.94231987	4.38202428818	5.51558971405	37.3298866749
28	1.52901232243	14.7935733795	54.8340072632	3.17704296112	74.3336359262
29	2.91171216965	19.7643718719	34.2938117981	0.803237617016	57.7731334567
30	6.43197917938	8.01305294037	236.898483276	3.78359699249	255.127112389
31	3.17009329796	8.13238811493	7.772260785	2.26597499847	21.3407171965
32	4.52246999741	5.5441365242	34.0548973083	1.6529392004	45.7744430304
33	1.00285565853	63.9151535034	22.4594459534	10.1562337875	97.5336889029
34	1.72088289261	6.27627563477	43.2441673279	2.47105240822	53.7123782635
35	18.7423229218	48.9606933594	1535.03686523	0.760684967041	1603.50056648
36	5.80212402344	35.7895431519	554.976928711	6.6349029541	603.20349884
37	2.52946019173	47.0342903137	117.270454407	0.997329831123	167.831534743
38	9.49632358551	142.585144043	544.953918457	0.591139853001	697.626525939
39	6.10720825195	31.0219154358	181.157989502	1.31677877903	219.603891969
40	4.13757896423	9.24217510223	193.976959229	0.851770579815	208.208483875
41	5.99018764496	78.4101104736	452.528900146	0.712118268013	537.641316533
42	1.37486314774	10.3320093155	33.191570282	3.02038121223	47.9188239574
43	2.16749739647	37.3561820984	2.18904995918	0.64849537611	42.3612248302
44	6.01708030701	83.0595169067	258.407867432	0.876994669437	348.361459315
45	3.9473323822	31.3764533997	385.919067383	2.47415947914	423.717012644
46	2.75125336647	5.19052124023	31.7510433197	1.90741562843	41.6002335548
47	1.91474056244	19.1081142426	336.497802734	3.89570593834	361.416363478
48	5.74626350403	54.1558761597	408.561798096	0.585334658623	469.049272418
49	2.7394156456	21.4528312683	182.124938965	2.06583213806	208.383018017
50	16.5667514801	52.9311523438	362.530792236	3.25390291214	435.282598972

Table B.12: Full results for DARRTHConnect in Plate Domain World 4

B.2 Tool Use Domain

Trial	World 0		World 1	
	DARRT	DARRTConnect	DARRT	DARRTConnect
1	128.744842529	29.4313602448	661.723937988	171.723510742
2	31.800201416	115.839874268	1275.97290039	91.6639022827
3	364.683929443	22.3649616241	1337.40576172	67.0450286865
4	71.3487091064	162.722122192	46.4863739014	21.8948993683
5	70.308380127	145.399353027	16.7018203735	40.7331924438
6	22.8180065155	111.968902588	907.831054688	221.375030518
7	23.0886058807	46.5296020508	541.991027832	219.729782104
8	80.6808547974	21.6622276306	386.573059082	120.034248352
9	87.5059051514	51.019947052	1003.07580566	155.548141479
10	175.582321167	21.8219909668	1232.90002441	299.995910645
11	403.665161133	18.489692688	194.733093262	18.1959896088
12	1320.82250977	18.7652721405	1399.0135498	159.163986206
13	158.306747437	51.4804954529	557.767578125	30.4029140472
14	181.117507935	16.2350616455	798.063354492	69.7020111084
15	359.198974609	17.634645462	1456.74169922	37.2375259399
16	76.4973602295	24.1885318756	1001.02648926	161.621765137
17	84.7786636353	20.3854198456	143.339065552	167.518753052
18	28.7031822205	80.7417755127	4526.77783203	248.263153076
19	59.3860321045	58.4194526672	297.774108887	193.262542725
20	494.49710083	25.7943515778	251.953552246	44.6461105347
21	215.654541016	26.2222290039	3003.85717773	37.2663574219
22	258.949432373	22.0622806549	135.784606934	174.794876099
23	407.551757812	25.621925354	711.129150391	21.7633304596
24	222.112350464	24.5006790161	756.480163574	109.629547119
25	229.083816528	23.7799224854	524.996459961	132.276702881
26	576.704162598	58.8769683838	333.102661133	30.2487640381
27	127.814376831	27.5070934296	3496.66674805	54.7259292603
28	101.957504272	20.1308364868	1113.41992188	162.47668457
29	273.080291748	49.6950759888	313.875823975	113.814575195
30	37.8101882935	40.1885681152	1723.27954102	69.2724990845
31	263.679595947	24.9812660217	826.719482422	70.1352386475
32	38.6792106628	27.3543128967	273.260375977	36.0677185059
33	23.6524772644	29.7633724213	89.999961853	24.2055168152
34	269.11428833	49.3139724731	473.385284424	33.0291442871
35	634.646240234	47.7568588257	452.701416016	164.471893311
36	106.406784058	54.3859558105	318.924255371	28.6685180664
37	34.6254844666	13.0172996521	857.779968262	135.334960938
38	74.9237518311	26.3803291321	1194.76147461	23.4791526794
39	181.337982178	15.0890932083	960.006652832	76.1202545166
40	15.428355217	25.2486572266	945.131591797	131.214233398
41	219.68522644	53.9824142456	263.172485352	106.167816162
42	165.917495728	8.6326084137	474.665008545	80.9223709106
43	159.517623901	17.1481361389	2047.05541992	71.1905212402
44	88.8485412598	20.873298645	260.990325928	72.5878295898
45	35.3368339539	22.9538059235	1995.91906738	37.6641426086
46	132.776687622	19.460559845	139.257141113	72.8894042969
47	211.490768433	58.17420578	4520.29003906	37.3032341003
48	260.379425049	20.0302429199	6011.09814453	221.221191406
49	439.284393311	26.6595020294	287.359191895	42.0540809631
50	260.78826904	25.8562698364	542.227416992	124.507781982

Table B.13: Full results for DARRT and DARRTConnect in Tool Use Domain Worlds 0-1.

Trial	World 2		World 3	
	DARRT	DARRTConnect	DARRT	DARRTConnect
1	1286.21057129	127.093536377	1257.86108398	3268.93261719
2	1914.1184082	1476.59960938	9295.03613281	10732.8027344
3	839.624084473	95.6041717529	1746.4987793	1480.79528809
4	598.991760254	685.620361328	873.557373047	3884.6003418
5	1597.32775879	317.159423828	8796.94628906	2606.15478516
6	346.323608398	777.775146484	2966.55541992	12256.6201172
7	692.842285156	170.088119507	3358.48681641	1403.51989746
8	144.832489014	96.7589569092	2333.7355957	187.341430664
9	692.469116211	583.025146484	1476.01611328	12599.8564453
10	92.7838058472	568.836975098	21444.9082031	4398.16162109
11	1558.94519043	796.438964844	8942.02050781	23343.5390625
12	97.1469268799	355.212371826	2201.55688477	3828.30395508
13	949.834594727	41.624168396	19800.8769531	4989.79736328
14	695.605285645	910.291503906	445.794403076	15206.8095703
15	572.305114746	551.908752441	1793.54980469	27451.8027344
16	481.745056152	61.3731880188	977.590148926	33851.3320312
17	524.195556641	1735.65612793	8334.00878906	8595.68847656
18	360.626464844	121.221733093	996.155639648	282.852416992
19	481.441680908	138.866149902	35246.1640625	3813.828125
20	650.980834961	371.059387207	13858.4414062	34220.03125
21	521.121887207	64.8695297241	7505.05273438	6818.56542969
22	1746.66003418	3111.66113281	10153.0273438	13799.5810547
23	587.365478516	363.02130127	885.080444336	8383.93652344
24	1940.9239502	43.4618682861	7376.87988281	5575.52685547
25	1175.59924316	265.211395264	12920.7626953	17344.0644531
26	1879.52929688	37.4944458008	49062.734375	13145.8691406
27	524.146728516	261.49822998	10474.1816406	552.820068359
28	1082.5390625	307.83694458	1406.28417969	143.325836182
29	296.878753662	926.013061523	3963.94116211	4694.44580078
30	109.998832703	68.0531539917	934.141357422	10949.9208984
31	276.393554688	119.012130737	7832.51708984	9568.05175781
32	76.2529067993	946.082214355	17434.0683594	217.215591431
33	44.0208320618	2691.68945312	985.057678223	28410.4121094
34	1822.47949219	275.866333008	4451.69287109	16469.5429688
35	899.790649414	866.778198242	10444.4931641	4151.42236328
36	68.7725372314	131.134643555	1721.62475586	2392.12158203
37	332.603515625	792.885742188	315.698455811	8304.9453125
38	456.446807861	115.37727356	1833.60009766	3576.45605469
39	163.573135376	752.110534668	2392.49902344	8375.609375
40	349.8984375	140.969802856	2213.51489258	584.387451172
41	1506.52209473	176.779373169	3402.27514648	39625.640625
42	999.343505859	443.202514648	22967.1074219	5579.88476562
43	76.633644104	277.701171875	21311.9316406	10660.5195312
44	546.519104004	144.627807617	4325.38232422	888.935424805
45	84.3584976196	986.503173828	2857.22729492	1664.37744141
46	326.308563232	43.9518051147	31373.4199219	34409.9453125
47	323.653778076	375.124206543	4344.77050781	20933.8242188
48	2736.99291992	314.051239014	21457.1601562	9795.25390625
49	1271.89575195	371.744445801	1249.69250488	5096.13671875
50	1852.5090332	278.355865479	373.957397461	7790.35546875

Table B.14: Full results for DARRT and DARRTConnect in Tool Use Domain Worlds 2-3.

World 0 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	6.78437328339	4.06131696701	1.82976329327	1.63004243374	16.7236289978	31.0291249752
2	21.8380374908	1.79594182968	8.63844299316	1.00646996498	13.5883073807	46.8671996593
3	9.96635532379	3.35620975494	10.3352365494	2.40451788902	5.99730968475	32.0596292019
4	4.57136392593	3.21692466736	0.950329899788	0.41481500864	17.5316085815	26.6850420833
5	4.3895907402	5.60350370407	16.5989074707	1.5273938179	5.79968118668	33.9190769196
6	14.3151254654	0.895852386951	1.74430060387	1.61194288731	24.2945632935	42.861784637
7	12.6303138733	4.40542316437	41.1590423584	0.413874387741	71.7841644287	130.392818213
8	4.76948165894	2.15333175659	13.7806358337	0.647675216198	20.0066604614	41.3577849269
9	22.0997505188	7.34216165543	10.5092496872	0.4402782619	5.93517541885	46.3266155422
10	5.52796173096	5.06755542755	5.1668138504	2.94174766541	14.4980487823	33.2021274567
11	3.06760811806	6.62841176987	1.08494222164	2.68555927277	11.3620290756	24.828550458
12	11.1467161179	2.9764380455	20.1047897339	0.41400167346	10.8457555771	45.4877011478
13	15.3209724426	11.1371469498	18.182849884	2.40094971657	17.1006412506	64.1425602436
14	12.6864156723	9.51661872864	14.4199266434	0.442200958729	62.3420028687	99.4071648717
15	23.7906303406	2.28429675102	19.1340255737	0.898890972137	44.6896324158	90.7974760532
16	29.255147934	1.16846334934	1.39222383499	1.380610466	6.99711275101	40.1935583353
17	10.8075170517	15.61008358	6.65048742294	1.78249228001	14.0293865204	48.879966855
18	8.63702297211	11.0148448944	20.7372436523	0.810380578041	51.224647522	92.4241396189
19	10.3610992432	2.68842887878	19.7739181519	1.30055034161	9.1958398819	43.3198364973
20	4.5086274147	13.8005952835	15.3114452362	3.00823569298	13.7215614319	50.3504650593
21	8.80606079102	21.0813789368	1.69549906254	2.5206155777	19.2752075195	53.3787618876
22	5.26158428192	2.4681854248	20.6689872742	3.1649081707	7.9676232338	39.5312883854
23	6.20169305801	12.7202749252	27.8016281128	1.37032151222	52.4909133911	100.584830999
24	12.8612775803	12.6446523666	12.1964483261	1.27085852623	50.3483886719	89.3216254711
25	5.39513587952	2.36585712433	19.0289592743	0.646850466728	10.3934249878	37.8302277327
26	11.5886926651	2.79902291298	1.69774901867	2.55228805542	2.45076084137	21.0885134935
27	8.48848438263	2.11969637871	10.0054368973	0.441958248615	40.2324867249	61.2880626321
28	8.99137020111	5.03320264816	12.0781202316	0.424325197935	9.17531776428	35.7023360431
29	4.95359802246	5.21673154831	1.93084192276	1.18683695793	18.2747516632	31.5627601147
30	3.49432373047	9.70508956909	4.13573789597	2.6260881424	26.8740539551	46.835293293
31	5.67399692535	10.2585868835	15.0814285278	0.411943376064	1.04953575134	32.4754914641
32	4.90596961975	11.1676082611	12.5838069916	1.51441192627	11.7203969955	41.8921937943
33	5.64227485657	5.05793952942	1.01824140549	2.71012449265	41.9103164673	56.3388967514
34	11.7801942825	0.684830486774	0.975771307945	2.0399954319	23.4721412659	38.952932775
35	6.95702648163	2.40660452843	14.1061782837	1.14499914646	8.62009525299	33.2349036932
36	4.411028862	2.7533056736	17.9721317291	2.02467465401	6.41853475571	33.5796756744
37	15.4027452469	3.23140120506	8.38610744476	11.8540143967	16.7464084625	55.6206767559
38	3.14269042015	17.708480835	35.8932723999	2.77523469925	28.8001251221	88.3198034763
39	11.5222787857	3.67302274704	11.1191396713	0.723409950733	1.22857773304	28.2664288878
40	8.21352481842	0.850754201412	45.0884895325	0.697282671928	14.0715026855	68.9215539098
41	8.27512168884	2.65619468689	1.93714666367	1.00416588783	7.16723155975	21.039860487
42	13.6340522766	7.56536197662	14.5198984146	2.31148672104	0.901193797588	38.9319931865
43	13.4506950378	0.946588277817	1.80851864815	0.988283395767	12.9937152863	30.1878006458
44	7.98202610016	6.27831554413	35.2421684265	1.21968400478	3.25239372253	53.9745877981
45	8.42096424103	6.23109006882	51.1792297363	0.502514362335	10.0218458176	76.3556442261
46	4.37130641937	8.07423686981	15.2429475784	1.28159964085	11.0239019394	39.9939924479
47	9.21610927582	2.37118959427	10.8822584152	0.893675863743	7.2244591713	30.5876923203
48	9.02298736572	3.40881443024	9.27548599243	2.49980211258	26.4727897644	50.6798796654
49	9.5178861618	3.66393494606	26.236946106	3.95653939247	13.2665700912	56.6418766975
50	7.54831218719	4.26824188232	21.4016590118	0.716384232044	59.5198783875	93.4544757009

Table B.15: Full results for DARRTH in Tool Use Domain World 0

World 0 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	4.07649278641	2.44679522514	8.87240314484	1.32009863853	3.1878426075	19.9036324024
2	2.01018595695	2.53020310402	4.3242764473	0.799663424492	15.3012952805	24.9656242132
3	5.396671772	1.15425539017	9.24297523499	0.477365195751	1.17736911774	17.4486367106
4	6.2740149498	5.55266666412	16.1950721741	0.444704264402	9.06650447845	37.5329625309
5	7.79952526093	0.742191195488	39.2886047363	0.615011632442	5.52198457718	53.9673174024
6	1.76420748234	1.85620367527	1.02896142006	0.511678874493	12.8815336227	18.0425850749
7	1.95402038097	7.68541288376	3.77127027512	0.51237142086	6.23980665207	20.1628816128
8	5.92714738846	5.80731630325	13.7771215439	2.81484007835	2.45827317238	30.7846984863
9	2.76693868637	5.56844139099	25.0282516479	1.44240832329	23.5099525452	58.3159925938
10	6.43841171265	3.90876293182	14.00826931	0.80803757906	15.9524688721	41.1159504056
11	6.20463943481	1.55366265774	15.2244358063	0.394286692142	15.5876588821	38.9646834731
12	8.08320045471	1.80180478096	5.35018253326	2.94497227669	3.0835711956	21.2637312412
13	2.0169980526	5.99294233322	11.0901031494	6.03276538849	11.5701198578	36.7029287815
14	2.56652736664	4.95675992966	9.89280033112	5.69849395752	21.0913887024	44.2059702873
15	2.05265855789	16.0697937012	42.07862854	1.54874920845	14.3354320526	76.0852620602
16	5.36157035828	2.33587932587	10.6040868759	0.55035418272	2.38090252876	21.2327932715
17	2.22143936157	1.54873299599	13.2000417709	0.478188127279	30.8493690491	48.2977713048
18	3.32318782806	1.88838350773	2.87834310532	0.52804595232	2.01975536346	10.6377157569
19	5.01601219177	4.7746052742	2.176633358	1.77528488636	29.6076755524	43.3502112627
20	3.48560643196	3.36172103882	21.2288360596	1.43838012218	11.4234209061	40.9379645586
21	3.30548977852	18.5593757629	6.01475477219	1.82556593418	4.5800037384	34.2851899862
22	7.42708683014	8.20492172241	5.46707630157	0.379502922297	8.9493227005	30.4279104769
23	3.63098454475	1.58574032784	0.86230802536	2.09887742996	9.51259326935	17.6905035973
24	3.60748457909	0.876603245735	0.999207496643	0.410989761353	2.00371718407	7.89800226688
25	8.66086196899	2.73296403885	17.535692215	1.25894773006	8.46644115448	38.6549071074
26	6.83641672134	9.99286460876	3.40455770493	0.495685696602	1.95255303383	22.6820777655
27	9.14774608612	4.87799596786	4.04834985733	0.963733077049	8.42443561554	27.4622606039
28	4.77172470093	1.57349693775	4.09251880646	0.935317754745	12.4711999893	23.8442581892
29	2.74964356422	2.4969189167	0.628461420536	0.660792589188	9.18777561188	15.7235921025
30	3.85335016251	1.45460271835	7.81269454956	0.88551145792	23.9120941162	37.9182530046
31	3.59524345398	5.13459014893	6.5724606514	0.924093484879	6.2843952179	22.5107829571
32	3.95110368729	3.4530518055	14.76512146	0.609861254692	7.68670463562	30.4658428431
33	2.42438721657	6.76706552505	12.8618602753	2.14635777473	3.37370562553	27.5733764172
34	4.43552589417	2.9659409523	8.28302669525	1.70787596703	3.88939881325	21.281768322
35	2.96393942833	1.56307601929	6.76755428314	1.19270730019	42.7229042053	55.2101812363
36	3.2970392704	1.4095133543	2.2867205143	1.5829668045	9.58322048187	18.1594604254
37	2.60427212715	5.19470405579	10.8216943741	0.729810893536	14.5147037506	33.8651852012
38	2.74149179459	3.55531787872	1.40292704105	1.28245890141	25.5621109009	34.5443065166
39	6.29598045349	3.33932805061	13.2711315155	0.884580373764	20.3823451996	44.173365593
40	5.93424940109	5.93538999557	4.72844600677	0.844536721706	14.8997325897	32.3423547149
41	3.60982298851	1.01246619225	3.60710906982	0.783709764481	1.71368932724	10.7267973423
42	4.14527511597	5.32298898697	2.03114128113	0.409245818853	0.344023674726	12.2526748776
43	4.79292917252	3.42419362068	19.9101600647	0.707327067852	10.5750656128	39.4096755385
44	4.64205551147	1.18400847912	6.7357211113	0.894956171513	11.4464092255	24.9031504989
45	2.41724729538	1.51256251335	5.96155881882	1.58798456192	2.34993696213	13.8292901516
46	4.4806022644	1.59254050255	11.1428813934	0.562449872494	5.79632043839	23.5747944713
47	4.52325725555	3.0407242775	1.09527909756	1.3598306179	6.35320091248	16.372292161
48	4.43269634247	4.96189117432	18.1482563019	1.40115392208	5.71113729477	34.6551350355
49	3.90623474121	5.09941625595	10.7043800354	0.823646783829	0.849557220936	21.3832350373
50	6.71340942383	2.32204294205	19.6139812469	0.602684020996	25.4893913269	54.7415089607

Table B.16: Full results for DARRTHConnect in Tool Use Domain World 0

World 1 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	6.74131059647	8.13917350769	71.7639694214	2.98054766655	9.12120723724	98.7462084293
2	10.9342842102	2.92612910271	75.8257522583	7.99224281311	48.3411407471	146.019549131
3	7.00240135193	5.46371459961	140.077804565	3.47368597984	36.293170929	192.310777426
4	14.500869751	8.35724639893	75.4184646606	14.8966035843	22.3814697266	135.554654121
5	19.1966075897	5.99921894073	249.200668335	9.90920639038	2.95590400696	287.261605263
6	7.214823246	14.2317390442	23.3613986969	0.930235385895	75.214630127	120.9528265
7	19.6058959961	8.66651439667	118.673439026	15.4647130966	10.296248436	172.706810951
8	10.02364254	4.65326118469	75.208366394	0.583560049534	33.3036994934	123.772529662
9	15.1257324219	3.89649009705	281.316040039	3.78442788124	57.3151550293	361.437845469
10	11.3736991882	6.18520832062	48.8665771484	0.699643611908	23.5360832214	90.6612114906
11	10.6657352448	5.61015415192	295.55947876	21.6621875763	7.26972675323	340.767282486
12	23.6726360321	1.21761357784	5.82931184769	2.14273834229	12.395236969	45.2575367689
13	13.7272548676	4.64020395279	54.4651794434	2.13267302513	43.6969032288	118.662214518
14	8.51401233673	10.5144367218	12.1544713974	11.0473022461	1.89832830429	44.1285510063
15	15.7803039551	1.01911604404	197.098876953	3.67468190193	2.3084936142	219.881472468
16	27.1665992737	3.30799388885	3.32925319672	0.49895876646	17.1720695496	51.4748746753
17	25.6818237305	14.7741527557	44.7299919128	0.796319782734	38.2708740234	124.253162205
18	4.93009757996	1.12652897835	5.55352306366	3.50047540665	7.36856412888	22.4791891575
19	18.1255626678	6.32430648804	129.350250244	8.50351905823	7.44248819351	169.746126652
20	10.3027591705	5.94087553024	73.6541748047	1.99937653542	10.3859558105	102.283141851
21	13.8517036438	2.26353430748	45.0016822815	1.86350739002	39.8515167236	102.831944346
22	8.42910003662	7.04644632339	21.1184635162	8.0453453064	16.6362609863	61.275616169
23	5.86315870285	3.46307659149	16.8364601135	18.2003192902	4.5906047821	48.9536194801
24	5.58257102966	11.4197225571	15.3126478195	7.30513858795	12.138710022	51.7587900162
25	17.0315151215	4.74046802521	44.3700866699	0.748897075653	5.35296964645	72.2439365387
26	8.20270061493	5.78128004074	23.3086338043	2.14705920219	8.21812057495	47.6577942371
27	7.55813550949	9.480260849	131.45161438	4.22207546234	29.2002391815	181.912325382
28	13.772939682	8.01076984406	2.12058210373	0.750220119953	23.8253154755	48.4798272252
29	14.1028223038	3.65502405167	55.6453704834	0.622055113316	2.29277348518	76.3180454373
30	11.1764554977	6.75480890274	135.964447021	8.8651676178	2.79396867752	165.554847717
31	7.98603343964	7.47775268555	344.997741699	2.37967944145	18.9458370209	381.787044287
32	9.60001754761	6.4331240654	106.407600403	21.2513847351	5.95987653732	149.652003288
33	37.8737602234	7.61984682083	112.790283203	0.711508989334	65.1992950439	224.194694281
34	11.4132490158	10.9793710709	64.1557312012	0.786167562008	10.1218976974	97.4564165473
35	11.3215093613	25.0726585388	33.6979522705	4.25920724869	20.4589538574	94.8102812767
36	15.4164953232	3.92275547981	17.5026130676	2.23229384422	12.3948459625	51.4690036774
37	28.0245819092	1.83434963226	51.0662956238	6.76596546173	47.359790802	135.050983429
38	7.18090963364	3.74167656898	85.2195358276	0.869406163692	39.3514785767	136.363006771
39	16.3231658936	3.0791079998	66.1882400513	1.66195178032	41.205581665	128.45804739
40	15.654296875	3.23327946663	140.578643799	0.936304330826	3.76143908501	164.163963556
41	12.3982362747	11.4811534882	1.59127759933	1.69704174995	19.2770061493	46.4447152615
42	7.50551462173	4.37127637863	88.2607879639	1.11559987068	86.9879302979	188.241109133
43	28.0996227264	2.81299138069	105.480377197	5.15861988068	6.77669668198	148.328307867
44	6.1600651741	5.67269420624	125.793739319	1.56150710583	34.132068634	173.320074439
45	19.1143627167	1.69050478935	93.4391326904	1.44948065281	7.21798801422	122.911468863
46	10.6479034424	8.92382717133	40.9990921021	7.46129846573	34.7816886902	102.813809872
47	5.0767159462	4.34487104416	10.8027687073	1.25498080254	57.7839050293	79.2632415295
48	7.65678501129	3.35771155357	41.0858726501	6.49284124374	6.60407018661	65.1972806454
49	4.04553985596	8.40874671936	23.993768692	27.813375473	12.7657661438	77.0271968842
50	13.8317623138	7.86381578445	64.7761993408	15.1421079636	11.0299396515	112.643825054

Table B.17: Full results for DARRTH in Tool Use Domain World 1

World 1 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	6.25988912582	2.31360411644	71.7348175049	2.74802160263	2.94223022461	85.9985625744
2	3.61150717735	1.78345453739	46.1049346924	2.36589503288	18.1150512695	71.9808427095
3	4.31432437897	1.51669430733	45.9887237549	10.4794483185	12.8000421524	75.0992329121
4	3.82531380653	7.47447729111	42.9529533386	44.0012168884	6.69388008118	104.947841406
5	4.3966012001	3.65601754189	57.8081359863	1.289021492	10.742937088	77.8927133083
6	5.36013412476	2.22994351387	15.6243114471	1.37482535839	1.0786960125	25.6679104567
7	5.26048517227	2.6421277523	14.8345899582	15.3623418808	2.17320775986	40.2727525234
8	8.53739070892	2.33862352371	30.4078102112	0.548611879349	13.805355072	55.6377913952
9	4.10798978806	16.8954277039	73.6943969727	28.9065895081	3.55881524086	127.163219213
10	3.14038443565	1.17222058773	51.9582977295	3.12878537178	4.98530626297	64.3849943876
11	6.33594942093	12.3931455612	8.48969268799	0.686252653599	29.5118598938	57.4169002175
12	3.14682579041	10.090883255	21.6081218719	8.91034317017	22.1308746338	65.8870487213
13	2.63869595528	10.0282812119	129.603088379	20.4043216705	20.9347057343	183.609092951
14	3.0864803791	4.94226694107	41.611579895	3.08415961266	8.98728370667	61.7117705345
15	3.58322072029	5.59177494049	37.0179862976	2.87563252449	6.64939975739	55.7180142403
16	2.07165765762	3.50676250458	26.5666122437	2.91795229912	10.9395132065	46.0024979115
17	5.59675884247	1.94475102425	18.855381012	6.01239061356	11.5203075409	43.9295890331
18	4.83612060547	1.26781094074	39.081867218	7.22489452362	12.2447137833	64.6554070711
19	4.09472990036	1.3621045351	13.0987529755	16.3793373108	0.659521102905	35.5944458246
20	3.17092514038	1.69712674618	52.2713241577	8.7153673172	15.7281827927	81.5829261541
21	2.27222251892	20.1981182098	16.5047664642	30.8853340149	3.04340600967	72.9038472176
22	2.18958067894	0.801259696484	22.0976047516	0.79681456089	3.34836411476	29.2336238027
23	4.93903827667	1.06524813175	53.169631958	0.728730499744	1.75899219513	61.6616410613
24	2.55355596542	1.50251889229	36.0093460083	0.616511940956	2.74138879776	43.4233216047
25	5.68502759933	1.14037930965	79.397064209	17.6188259125	4.16830062866	108.009597659
26	4.41038274765	3.75361990929	54.3105545044	1.59899938107	8.49850082397	72.5720573664
27	4.24995660782	4.43832969666	6.02330875397	4.74558162689	14.0004577637	33.457634449
28	3.17618703842	4.16104841232	57.0786705017	9.72562503815	9.94784927368	84.0893802643
29	3.45861458778	1.02128517628	7.64691257477	17.3705043793	16.8484916687	46.3458083868
30	2.04714155197	3.22906112671	9.62075614929	1.84438741207	2.876039505	19.617385745
31	9.65002155304	4.36654043198	66.2931060791	17.3164196014	7.42528104782	105.051368713
32	2.73476338387	5.51476621628	11.1375112534	14.6852264404	14.1375465393	48.2098138332
33	4.01289844513	2.97220873833	3.35415959358	3.22999930382	12.350520134	25.9197862148
34	2.9750187397	2.69251084328	20.2245979309	0.892518460751	31.6972064972	58.4818524718
35	6.46788835526	16.8513393402	4.43122959137	8.20016098022	4.23327970505	40.1838979721
36	6.08333826065	0.682803273201	16.065867126	25.4700508118	6.59485578537	54.896915257
37	5.19116449356	1.87408876419	1.97022485733	1.14149522781	25.5580425262	35.7350158691
38	2.97337245941	4.48852539062	39.3931350708	2.29744505882	17.4494285583	66.601906538
39	5.12740802765	3.73846840858	111.00920105	9.18102455139	22.432849884	151.488951921
40	6.02633619308	1.1249755621	27.9093284607	1.05400729179	11.1125173569	47.2271648645
41	11.0806016922	10.7737131119	8.15114116669	13.1567020416	4.17172384262	47.333881855
42	1.97211790085	14.6202001572	11.2586765289	11.3007144928	3.42062282562	42.5723319054
43	3.47670269012	0.752612352371	32.2658233643	26.8964557648	4.02715539932	67.4187495708
44	5.58481502533	4.49624633789	83.6779556274	22.7493476868	1.3535348177	117.861899495
45	4.85086584091	3.39047694206	40.0206108093	7.38193655014	3.30500841141	58.9488985538
46	3.38050913811	1.39118027687	2.53437590599	28.7219314575	23.2291927338	59.2571895123
47	3.506752491	2.28342270851	24.6477355957	18.2854347229	30.8268146515	79.5501601696
48	10.7287988663	4.18120098114	5.33402967453	0.535456240177	26.3128547668	47.092340529
49	4.90046215057	1.30501806736	22.346031189	1.49613618851	7.72135162354	37.7689992189
50	5.4775428772	6.23041296005	17.7667102814	0.850612580776	8.9034986496	39.228777349

Table B.18: Full results for DARRTHConnect in Tool Use Domain World 1

World 2 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	15.0712633133	8.49743461609	82.337387085	1.91127955914	1.75825309753	109.575617671
2	16.106595993	5.14140176773	166.161865234	7.63293981552	0.240019217134	195.282822028
3	8.04125404358	4.26422452927	78.5476608276	2.87995958328	1.32173800468	95.0548369884
4	9.25408935547	2.30404019356	75.0247039795	1.62771737576	0.717810153961	88.9283610582
5	6.04989767075	11.4562883377	23.4119586945	6.11571884155	0.569907844067	47.6037713885
6	19.5659885406	3.43706989288	5.86837053299	21.2493019104	0.132169783115	50.25290066
7	8.72282028198	2.98958539963	108.791183472	7.89923477173	1.33913660049	129.741960526
8	10.8378019333	7.00386619568	73.7839202881	2.3487868309	0.507865309715	94.4822405577
9	5.31336069107	1.90802145004	16.8041229248	18.8584251404	0.319287568331	43.2032177746
10	16.8281269073	5.74888944626	27.2822532654	1.31487953663	3.76720428467	54.9413534403
11	4.31670379639	12.1718616486	32.8015975952	6.26282739639	1.23687160015	56.7898620367
12	8.90999984741	2.75441932678	99.1471557617	1.45010471344	0.225288659334	112.486968309
13	7.51049900055	1.09070670605	112.969741821	2.5811727047	5.35004472733	129.50216496
14	10.0463371277	4.68086671829	162.715270996	12.9998931885	0.370177865028	190.812545896
15	6.56373119354	8.22097492218	190.861679077	2.81146001816	0.191611230373	208.649456441
16	8.66177940369	10.8030939102	342.786834717	1.99998855591	0.83494591713	365.086642504
17	17.1734695435	2.14430618286	15.9691171646	4.13085699081	3.70780611038	43.1255559921
18	18.1896495819	2.36244940758	50.1503410339	0.752250432968	5.22183704376	76.6765275002
19	5.94596576691	5.88955068588	26.8326416016	3.95399260521	1.333101511	43.9552521706
20	20.6454658508	13.6427097321	31.8015136719	6.96356630325	2.37249350548	75.4257490635
21	8.35697078705	2.4159116745	220.833984375	1.57167804241	0.786989629269	233.965534508
22	9.27654361725	2.34545493126	85.0493164062	2.56751704216	2.25440692902	101.493238926
23	8.09968185425	4.55953168869	17.0564041138	3.3576271534	0.461191684008	33.5344364941
24	8.64199924469	8.29708003998	13.3690500259	1.24557578564	0.305305480957	31.8590105772
25	8.47316455841	2.68873143196	612.541015625	3.57484197617	6.1104221344	633.388175726
26	10.4198074341	4.44538116455	171.850692749	2.1906273365	2.41511321068	191.321621895
27	10.05402565	3.80821466446	126.274368286	6.8594326973	1.39950621128	148.395547509
28	23.3198680878	3.58433842659	119.38671875	11.851313591	0.539755225182	158.681994081
29	14.6783733368	2.86547017097	135.712554932	2.89054179192	0.433516800404	156.580457032
30	15.8137388229	3.43647742271	108.885353088	11.7215957642	1.16750884056	141.024673939
31	21.8811721802	1.3831974268	6.69278097153	8.39197540283	0.512256801128	38.8613827825
32	14.2202644348	4.92369937897	120.19367981	2.8338227272	0.48155105114	142.653017402
33	6.37805461884	2.08203887939	470.166534424	4.09275484085	1.60717129707	484.32655406
34	24.3271274567	2.56274294853	169.190063477	5.11345624924	3.36662077904	204.56001091
35	6.68665790558	4.04071187973	96.347442627	10.4881343842	0.602676093578	118.16562289
36	14.8788757324	7.67409658432	37.4011497498	1.85786402225	0.619324088097	62.4313101768
37	5.73770332336	3.28454566002	59.2764472961	0.882383882999	1.17542552948	70.356505692
38	9.41666984558	11.7773694992	30.7938289642	2.47470927238	0.333074212074	54.7956517935
39	12.2423725128	4.71194696426	36.1559715271	8.44540214539	0.264993876219	61.8206870258
40	20.3908863068	7.94864559174	91.5554428101	2.12176251411	0.408545911312	122.425283134
41	7.59320878983	4.0382027626	167.762969971	8.36852645874	0.914403259754	188.677311242
42	9.19406795502	7.50651788712	99.377746582	9.09192371368	0.573326289654	125.743582428
43	10.1298799515	1.07583975792	16.6296043396	6.14312696457	1.67611169815	35.6545627117
44	9.54915618896	5.20874595642	21.2422447205	5.67717218399	1.9720441103	43.6493631601
45	16.6057357788	10.3803672791	19.3710346222	2.48324370384	2.31695604324	51.1573374271
46	24.5293159485	4.5826420784	74.2897109985	3.36141610146	0.787619709969	107.550704837
47	15.500828743	6.32451438904	72.8318252563	3.32917714119	2.35594034195	100.342285872
48	10.8586816788	3.85756754875	69.8522033691	6.89896392822	0.734478294849	92.2018948197
49	11.1775989532	9.03844070435	105.98210144	7.69353199005	0.684578180313	134.576251268
50	6.87521219254	10.0513820648	26.4559726715	8.47065353394	0.745603501797	52.5988239646

Table B.19: Full results for DARRTH in Tool Use Domain World 2

World 2 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	2.43685460091	1.27639663219	11.87931633	4.73056268692	0.367667853832	20.6907981038
2	4.76190662384	1.1765730381	33.4952850342	6.09020662308	0.193191945553	45.7171632648
3	2.68863987923	2.73669695854	5.63141822815	13.7300338745	0.357801765203	25.1445907056
4	5.33990859985	12.001616478	10.5261087418	3.96822762489	0.47197791934	32.3078393638
5	5.14939022064	4.71309232712	64.8652191162	0.851758539677	0.394594311714	75.9740545154
6	6.17936134338	1.49659407139	58.6399917603	2.6172478199	0.33976405859	69.2729590535
7	2.68170881271	1.20517325401	6.26643276215	11.7392244339	1.54101538658	23.4335546494
8	5.75252723694	4.6505408287	72.4319152832	0.7777556777	0.221729233861	83.8344682604
9	9.94447135925	0.529910385609	10.6227684021	4.03640031815	0.138101235032	25.2716517001
10	2.88113927841	3.02086186409	8.36651134491	2.51830863953	0.276952207088	17.063773334
11	1.69183075428	3.97775125504	64.0194625854	3.76956415176	1.38432788849	74.842936635
12	5.02024745941	3.58547306061	98.6455307007	3.25984406471	0.441693812609	110.952789098
13	2.26957392693	7.39141082764	19.6954727173	4.29714822769	0.308256477118	33.9618621767
14	2.94373750687	4.14621591568	17.0907058716	1.45888435841	0.821606636047	26.4611502886
15	2.0860850811	5.73893404007	6.79432153702	1.31071996689	0.669729471207	16.5997900963
16	5.59946870804	7.74267911911	16.704208374	0.823318839073	0.783191025257	31.6528660655
17	5.37317085266	0.674676597118	23.3035106659	1.74498164654	0.189035743475	31.2853755057
18	5.38960981369	3.18443369865	16.9838466644	39.5874176025	0.26919952035	65.4145072997
19	6.24373817444	2.62390375137	3.4277973175	4.09250450134	0.698185265064	17.0861290097
20	9.510222435	6.35331344604	17.3933677673	9.34238910675	0.231868907809	42.8311616629
21	2.72842431068	2.23375272751	22.0462474823	4.56494379044	0.258704960346	31.8320732713
22	3.22023582458	12.0301485062	42.8279876709	0.551497936249	0.47698572278	59.1068556607
23	8.10684967041	1.06972754002	36.3658561707	12.3682689667	0.510928213596	58.4216305614
24	2.82507658005	2.75321054459	5.74545812607	1.1378518343	1.28774487972	13.7493419647
25	4.54391813278	0.974942982197	8.75590705872	3.43698978424	0.251670747995	17.9634287059
26	3.25465917587	8.70685005188	12.0253944397	4.92019224167	0.144467160106	29.0515630692
27	2.01884293556	0.871699869633	50.1918754578	3.79603981972	0.712153613567	57.5906116962
28	4.90106534958	6.68321561813	14.5646762848	3.04355621338	0.192241773009	29.3847552389
29	2.64110064507	2.31155872345	8.01370429993	2.51274585724	0.266343325377	15.7454528511
30	3.75056624413	3.64764499664	2.54826617241	5.9946436882	0.805759966373	16.7468810678
31	2.84352731705	4.08623552322	62.9956665039	4.68936395645	0.189830482006	74.8046237826
32	4.11888790131	6.57702827454	17.6632556915	2.33316326141	0.491930663586	31.1842657924
33	3.38081359863	8.99559783936	60.3507385254	1.27838003635	0.402624666691	74.4081546664
34	2.7343070507	2.02334499359	5.11948060989	1.26782178879	0.139822050929	11.2847764939
35	4.80195426941	1.98665893078	6.85260057449	7.49053764343	0.335084617138	21.4668360353
36	5.32885980606	4.83683633804	51.5598678589	16.4625396729	0.489582479	78.6776861548
37	7.08903169632	11.198797226	19.8558979034	9.53225708008	0.257266908884	47.9332508147
38	5.44460868835	3.15131473541	11.5013647079	2.71476912498	1.15783929825	23.9698965549
39	5.87576675415	3.91621899605	123.572380066	5.8648109436	0.392152607441	139.621329367
40	7.53019762039	0.865674972534	46.6289024353	8.68226051331	0.2887981534	63.9958336949
41	4.72920370102	2.14843845367	11.0889358521	0.784203588963	0.479307174683	19.2300887704
42	8.67640304565	1.51594388485	27.1710929871	5.39192724228	0.21511015296	42.9704773128
43	2.25151753426	4.59303665161	18.4426136017	1.76748812199	1.40956318378	28.4642190933
44	2.8755402565	4.68758392334	6.76784610748	4.0382733345	0.194589018822	18.5638326406
45	4.52245187759	2.60040235519	8.23013591766	3.68065261841	0.143365368247	19.1770081371
46	4.46262741089	1.08616805077	12.4022769928	1.91712939739	0.34875074029	20.2169525921
47	5.30988693237	3.99247670174	10.138092041	3.19696116447	0.242464587092	22.8798814267
48	4.18911933899	2.68851852417	35.577293396	1.50336945057	0.389340937138	44.3476416469
49	4.82964324951	1.08711051941	3.83707094193	1.04890632629	0.152316451073	10.9550474882
50	4.51693153381	4.17544221878	17.1994934082	11.4623908997	0.163787260652	37.5180453211

Table B.20: Full results for DARRTHConnect in Tool Use Domain World 2

World 3 DARRTH

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	10.7607030869	42.7542877197	203.124649048	92.0725708008	8.7206697464	357.432880402
2	19.0999565125	28.060836792	6.78773546219	2.08718895912	45.7033119202	101.739029646
3	20.2769527435	49.5246391296	448.639801025	13.3447904587	38.8151702881	570.601353645
4	14.6408185959	36.4726295471	293.362487793	16.10430336	20.8814601898	381.461699486
5	6.09371948242	21.4710960388	671.531494141	5.76589202881	3.12544941902	707.98765111
6	23.1173019409	35.4879341125	89.1042633057	29.8821926117	18.2198448181	195.811536789
7	14.0300655365	42.1122207642	28.6020908356	21.6693477631	4.74996948242	111.163694382
8	9.90807151794	18.4436168671	27.7076931	48.6489868164	51.5559082031	156.264276505
9	7.79870462418	31.7665691376	369.905975342	6.99106740952	76.4846343994	492.946950912
10	14.5926504135	149.622894287	29.3342227936	5.84766244888	41.2392578125	240.636687756
11	3.90508031845	53.3474845886	492.712860107	44.7238998413	6.54337787628	601.232702732
12	11.1596031189	35.9479942322	190.731750488	114.190216064	6.04286956787	358.072433472
13	10.7832450867	40.3736457825	317.65536499	77.234046936	0.746406137943	446.792708933
14	15.1528778076	73.4733734131	13.8244781494	32.664478302	21.2957687378	156.41097641
15	9.75694942474	206.283828735	82.5913085938	71.3162231445	11.2805089951	381.228818893
16	15.6637659073	54.7341690063	141.065063477	7.75407171249	70.4347915649	289.651861668
17	8.87819766998	22.3151817322	49.5642738342	13.7726011276	7.07289838791	101.603152752
18	6.79013204575	22.1916294098	165.788513184	3.53474259377	17.7124652863	216.017482519
19	28.5362434387	11.6231565475	25.6954841614	1.1361079216	20.9693374634	87.9603295326
20	7.7124786377	11.3134231567	103.776367188	8.99368095398	2.38947987556	134.185429811
21	13.1900539398	19.5979919434	115.13621521	15.9367580414	2.4215028286	166.282521963
22	11.343834877	205.648223877	20.8805789948	23.6328411102	9.80379199982	271.309270859
23	16.5068721771	10.4440870285	82.6850967407	4.13965797424	4.40268993378	118.178403854
24	18.038312912	97.7183609009	154.177871704	111.429519653	23.6210975647	404.985162735
25	9.76899337769	26.6111545563	159.635116577	3.69771718979	64.338104248	264.051085949
26	6.11855459213	26.4100112915	435.002929688	118.586990356	34.0428161621	620.16130209
27	11.0803489685	102.809967041	11.92146492	118.898345947	34.4017677307	279.111894608
28	9.10174560547	8.88621997833	15.1332511902	26.6688728333	9.92617893219	69.7162685394
29	5.66953802109	149.944030762	248.841583252	2.12759613991	29.7782917023	436.361039877
30	12.9052629471	23.8197631836	8.43099975586	3.45653796196	72.6406555176	121.253219366
31	10.4267206192	18.5411491394	159.173080444	34.2062911987	28.4734287262	250.820670128
32	24.0691585541	51.1746559143	79.0422515869	10.6822090149	62.2327575684	227.201032639
33	18.1967601776	57.9455375671	83.7688903809	55.1583900452	4.29209852219	219.361676693
34	9.29942893982	81.100479126	89.6875	47.5161819458	88.7559509277	316.359540939
35	10.8228826523	97.03175354	453.136688232	5.80380439758	10.84678936	577.641918182
36	21.4685726166	57.9309158325	298.201751709	5.48412179947	38.9037284851	421.989090443
37	14.9870595932	99.2542419434	146.750778198	9.2744178772	13.4508066177	283.71730423
38	6.97922086716	169.901397705	31.128370285	40.0161514282	2.59292888641	250.618069172
39	7.9559044838	59.7129364014	33.6995620728	18.7674884796	28.3913955688	148.527287006
40	10.0094060898	22.058429718	1142.97851562	9.96306419373	23.8140201569	1208.82343578
41	11.3974733353	29.4578304291	69.0092163086	38.2444839478	27.2562084198	175.36521244
42	15.0604343414	47.4411354065	144.190994263	9.46560001373	34.7198944092	250.878058434
43	11.5210008621	170.503570557	28.9172897339	9.80007648468	42.7904090881	263.532346725
44	8.40588569641	91.8651351929	374.312103271	16.6317806244	0.98995923996	492.204864025
45	16.5871639252	106.202507019	230.682647705	65.7913131714	5.69114303589	424.954774857
46	9.55940341949	111.350730896	9.04644680023	33.7193832397	9.74722862244	173.423192978
47	17.6375102997	119.207397461	13.6779069901	2.38249158859	10.1683921814	163.073698521
48	11.5754127502	46.5596046448	144.787902832	35.8114318848	41.5422515869	280.276603699
49	13.1706266403	13.4096231461	79.6026306152	36.1369628906	45.7147674561	188.034610748
50	11.2688312531	89.3865356445	149.700149536	111.329467773	58.590637207	420.275621414

Table B.21: Full results for DARRTH in Tool Use Domain World 3

World 3 DARRTHConnect

Trial	Object Time	Subgoal 1	Subgoal 2	Subgoal 3	Subgoal 4	Total
1	6.31830215454	29.9770584106	14.556801796	0.854779303074	0.641179084778	52.348120749
2	5.35546970367	12.8878440857	287.896820068	9.81659317017	18.2631626129	334.219889641
3	10.4704217911	11.6491670609	89.1675415039	4.91392946243	23.171453476	139.372513294
4	6.33593654633	16.4113330841	30.739118576	16.5742053986	13.6530666351	83.7136602402
5	5.74918794632	41.0780525208	13.8825073242	68.1302108765	11.3777227402	140.217681408
6	6.65925264359	19.8991661072	78.1554870605	63.9911231995	7.58030128479	176.285330296
7	3.61024641991	21.1646251678	153.414001465	36.9319953918	8.40286064148	223.523729086
8	4.75511169434	25.3264827728	114.386856079	57.0791244507	37.4338150024	238.981389999
9	5.5899643898	85.7066802979	237.201049805	44.8885879517	23.9535388947	397.339821339
10	6.88067674637	109.807266235	23.8739280701	26.3838539124	8.65630817413	175.602033138
11	5.68321275711	60.418182373	39.9975776672	32.7177696228	61.6395874023	200.456329823
12	12.6772756577	85.7739562988	35.7159538269	22.759847641	68.6863708496	225.613404274
13	4.81549406052	23.2929439545	66.6019897461	76.5305480957	5.24829816818	176.489274025
14	5.24766397476	5.696352005	19.7186584473	3.00083041191	27.7948436737	61.4583485126
15	4.37740135193	3.52992486954	17.1635990143	13.227560997	2.81110453606	41.1095907688
16	4.28545999527	51.6308517456	37.0356559753	16.0953273773	8.24178123474	117.289076328
17	8.07359695435	66.4815750122	9.04209804535	5.89723110199	1.44529426098	90.9397953749
18	4.16209220886	107.11681366	29.4289321899	71.1009902954	7.52958917618	219.33841753
19	13.9263801575	113.951469421	107.58089447	36.9551277161	28.7166404724	301.130512238
20	9.03417682648	34.2395553589	11.2814226151	22.6888027191	140.117492676	217.361450195
21	5.80626153946	3.74549794197	22.588684082	25.456413269	13.9245271683	71.5213840008
22	6.11937952042	38.6085624695	24.785987854	22.5180969238	30.847120285	122.879147053
23	9.78820419312	33.8861083984	28.2545986176	2.06852221489	13.1613454819	87.1587789059
24	4.89494943619	6.26033687592	6.05459737778	12.467206955	27.8530902863	57.5301809311
25	6.18716049194	51.0528373718	12.6916313171	1.11204767227	11.437253952	82.4809308052
26	6.60920095444	32.7861747742	105.034698486	64.41796875	18.391872406	227.239915371
27	5.1724524498	29.1951351166	69.6720428467	52.0367507935	10.5354204178	166.611801624
28	3.44226789474	12.8302679062	103.307411194	22.3866386414	22.4833335876	164.449919224
29	5.27383422852	84.7227172852	36.9780349731	10.5937643051	35.5571861267	173.125536919
30	5.06947135925	61.5303535461	45.0382804871	18.8410377502	27.0653896332	157.544532776
31	8.72538566589	38.215587616	34.8237609863	234.289916992	18.5719738007	334.626625061
32	5.03977489471	18.8146934509	39.5308227539	38.0473518372	44.2943687439	145.727011681
33	10.9161863327	16.0610160828	32.5157737732	6.96833086014	17.8064155579	84.2677226067
34	5.42342710495	38.8842658997	78.8359069824	11.0998334885	9.61551856995	143.858952045
35	8.57485866547	43.2159881592	11.2930660248	16.490487248	44.9990653992	124.573465496
36	7.17452955246	23.6220207214	31.5759429932	7.55734729767	0.762651503086	70.6924920678
37	5.62911748886	38.6545448303	5.19581413269	4.22777366638	7.64607286453	61.3533229828
38	4.33323335648	47.4028015137	96.1017456055	111.455482483	21.6749229431	280.968185902
39	11.6670627594	7.06906938553	7.58625364304	4.88362312317	62.5526199341	93.7586288452
40	6.55490112305	15.416686058	52.4597892761	5.96877717972	77.9572906494	158.357444286
41	3.68283772469	43.0821914673	73.326385498	5.94779825211	77.8433609009	203.882573843
42	9.44822597504	58.6459922791	35.304725647	5.39638710022	18.3001308441	127.095461845
43	10.614780426	20.5740203857	190.541381836	0.980843484402	21.1529140472	243.863940179
44	6.3016076088	14.8238039017	30.4470481873	243.43586731	6.64205598831	301.650382996
45	9.00475120544	29.8349914551	18.8119926453	2.17339920998	15.0161504745	74.8412849903
46	4.72807788849	39.0589027405	18.5354290009	9.88052845001	58.9352836609	131.138221741
47	6.52924251556	6.62394189835	40.0198135376	5.81892728806	21.0217647552	80.0136899948
48	9.97854042053	109.077095032	142.020294189	50.3177108765	51.2555961609	362.649236679
49	14.159362793	8.89634799957	7.41119003296	1.87581825256	3.21600580215	35.5587248802
50	5.53671360016	3.09226870537	59.0564193726	11.1506748199	22.3433208466	101.179397345

Table B.22: Full results for DARRTHConnect in Tool Use Domain World 3

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] R. Alterovitz, T. Siméon, and K. Goldberg. The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty. In *Robotics: Science and Systems III*, June 2007.

[2] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Manipulation with Multiple Action Types. In *International Symposium on Experimental Robotics*, 2012.

[3] J. Barry, L. P. Kaelbling, and Tomás Lozano-Pérez. A Hierarchical Approach to Manipulation with Diverse Actions. In *International Conference on Robotics and Automation*, 2013.

[4] D. Berenson. *Constrained Manipulation Planning.* PhD thesis, Carnegie Mellon University, 2011.

[5] D. Berenson and S. S. Srinivasa. Probabilistically Complete Planning with End-Effector Pose Constraints. In *International Conference on Robotics and Automation*, May 2010.

[6] V. Boor, M. H. Overmars, and A. F. van der Stappen. Gaussian Sampling for Probabilistic Roadmap Planners. In *International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[7] R. C. Brost. Automatic Grasp Planning in the Presence of Uncertainty. *International Journal of Robotics Research*, 7(1), 1988.

[8] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential Composition of Dynamically Dexterous Robot Behaviors. *International Journal of Robotics Research*, 18(6):534–555, June 1999.

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

[10] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *Robotics & Automation Magazine*, 19(4):72–82, December 2012.

[11] M. Dogar and S. Srinivasa. Push-Grasping with Dexterous Hands: Mechanics and a Method. In *International Conference on Intelligent Robots and Systems*, 2010.

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

[13] Willow Garage. PR2 Robot for Research and Innovation. `http://www.willowgarage.com/pages/pr2/overview`.

[14] K. Hauser. *Motion Planning for Legged and Humanoid Robots.* PhD thesis, Stanford University, 2008.

[15] K. Hauser and V. Ng-Throw-Hing. Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task. *International Journal of Robotics Research*, 30(6), 2011.

[16] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners. In *International Conference on Robotics and Automation*, pages 4420–4426, 2003.

[17] D. Hsu, J-C Latombe, and R. Motwani. Path Planning in Expansive Configuration Spaces. *International Journal of Computational Geometry & Applications*, 9(4-5):495–512, 1999.

[18] W. Huang, E. Krotkov, and M. T. Mason. Impulsive Manipulation. In *International Conference on Robotics and Automation*, 1995.

[19] W. Huang and M. T. Mason. Experiments in Impulsive Manipulation. In *International Conference on Robotics and Automation*, volume 2, 1998.

[20] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, 30:846–894, 2011.

[21] L. E. Kavraki, M. N. Kolountzakis, and J-C Latombe. Analysis of Probabilistic Roadmaps for Path Planning. In *International Conference on Robotics and Automation*, pages 3020–3026, 1996.

[22] L. E. Kavraki, P. Švestka, J-C Latombe, and M. H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[23] J. J. Kuffner and S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *International Conference on Robotics and Automation*, 2000.

[24] H. Kurniawati and D. Hsu. Workspace Importance Sampling for Probabilistic Roadmap Planning. In *International Conference on Intelligent Robots and Systems*, pages 1618–1623, 2004.

[25] H. Kurniawati and D. Hsu. Workspace-based Connectivity Oracle: An Adaptive Sampling Strategy for PRM Planning. In *International Workshop on the Algorithmic Foundations of Robotics*, 2006.

[26] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive Path Deformation for Nonholonomic Mobile Robots. *Transactions on Robotics*, 20(6):967–977, December 2004.

[27] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[28] S. M. LaValle and J. J. Kuffner Jr. Rapidly-Exploring Random Trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[29] T. Lozano-Pérez. Spatial Planning: A Configuration Space Approach. *Transactions on Computers*, 32(2), February 1983.

[30] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell. *Handey: A Robot Task Planner.* MIT Press, Cambridge, MA, 1992.

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

[32] M. T. Mason. *Manipulator Grasping and Pushing Operations.* PhD thesis, MIT, 1982.

[33] M. T. Mason. *Mechanics of Robotic Manipulation.* MIT Press, Cambridge, MA, August 2001.

[34] N. Melchior and R. Simmons. Particle RRT for Path Planning with Uncertainty. In *International Conference on Robotics and Automation*, pages 1617–1624, April 2007.

[35] P. E. Missiuro and N. Roy. Adapting Probabilistic Roadmaps to Handle Uncertain Maps. In *International Conference on Robotics and Automation*, pages 1261–1267, Orlando, FL, May 2006.

[36] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue. Environment Manipulation Planner for Humanoid Robots Using Task Graph That Generates Action Sequence. In *International Conference on Intelligent Robots and Systems*, 2004.

[37] J. Ota. . In *International Conference on Robotics and Automation*, volume 2, 2004.

[38] S. Prentice and N. Roy. The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *International Journal of Robotics Research*, 28(11-12), 2009.

[39] S. Schaal and C. G. Atkeson. Open Loop Stable Control Strategies for Robot Juggling. In *International Conference on Robotics and Automation*, volume 3, pages 913–918, Atlanta, GA, May 1993.

[40] S. Schaal and C. G. Atkeson. Robot Juggling: Implementation of Memory-Based Learning. *Control Systems*, 14(1):57–71, February 1994.

[41] J. Scholz, S. Chitta, B. Marthi, and M. Likhachev. Cart Pushing with a Mobile Manipulation System: Towards Navigation with Moveable Objects. In *International Conference on Robotics and Automation*, Shanghai, China, May 2011.

[42] T. Senoo, A. Namiki, and M. Ishikawa. High-Speed Throwing Motion Based on Kinetic Chain. In *International Conference on Intelligent Robots and Systems*, pages 3206–3211, Nice, France, September 2008.

[43] T. Siméon, J-P Laumond, J. Cortés, and A. Sahbani. Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research*, 23(7-8), 2004.

[44] S. S. Srinivasa, C. R. Baker, E. Sacks, G. B. Reshko, M. T. Mason, and M. A. Erdmann. Experiments with Non-Holonomic Manipulation. In *International Conference on Robotics and Automation*, 2002.

[45] M. Stilman. Task Constrained Motion Planning in Robot Joint Space. In *Intelligent Robots and Systems*, 2007.

[46] M. Stilman and J. Kuffner. Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments. In *HUMANOIDS*, 2004.

[47] M. Stilman and J. Kuffner. Planning Among Movable Obstacles with Artificial Constraints. *International Journal of Robotics Research*, 27(11-12), 2008.

[48] M. Stilman, J-U Schamburek, J. Kuffner, and T. Asfour. Manipulation Planning Among Movable Obstacles. In *International Conference on Robotics and Automation*, 2007.

[49] J. Tan and X. Ning. Unified Model Approach for Planning and Control of Mobile Manipulators. In *International Conference on Robotics and Automation*, volume 3, pages 3145–3152, 2001.

[50] J. Tan and X. Ning. Integrated Sensing and Control of Mobile Manipulators. In *International Conference on Intelligent Robots and Systems*, volume 2, pages 865–870, Maui, HI, October 2001.

[51] J. Tan, N. Xi, and Y. Wang. Integrated Task Planning and Control for Mobile Manipulators. *International Journal of Robotics Research*, 22(5):337–354, May 2003.

[52] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path Planning among Movable Obstacles: A Probabilistically Complete Approach. In *International Workshop on Algorithmic Foundations of Robotics*, 2008.

[53] S. Walker and J. K. Salisbury. Pushing Using Learned Manipulation Maps. In *International Conference on Robotics and Automation*, pages 3808–3813, Pasadena, CA, May 2008.

[54] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space. In *International Conference on Robotics and Automation*, pages 1024–1031, 1999.

[55] Y. Yang and O. Brock. Adapting the Sampling Distribution in PRM Planners Based on an Approximated Medial Axis. In *International Conference on Robotics and Automation*, volume 5, pages 4405–4410, May 2004.

[56] Z. Yao and K. Gupta. Path Planning with General End Effector Constraints: Using Task Space to Guide Configuration Space Search. In *International Conference on Intelligent Robots and Systems*, pages 1875–1880, 2005.