# ROBOTS THAT LEARN AND PLAN – UNIFYING ROBOT LEARNING AND MOTION PLANNING FOR GENERALIZED TASK EXECUTION

Chris Bowen

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the College of Arts and Sciences.

Chapel Hill
2018

Approved by:

Ron Alterovitz

Alex Berg

Kris Hauser

Vladimir Jojic

Dinesh Manocha

## ABSTRACT

Chris Bowen: Robots that Learn and Plan – Unifying Robot Learning and Motion Planning for Generalized Task Execution
(Under the direction of Ron Alterovitz)

Robots have the potential to assist people with a variety of everyday tasks, but to achieve that potential robots require software capable of planning and executing motions in cluttered environments. To address this, over the past few decades, roboticists have developed numerous methods for planning motions to avoid obstacles with increasingly stronger guarantees, from probabilistic completeness to asymptotic optimality. Some of these methods have even considered the types of constraints that must be satisfied to perform useful tasks, but these constraints must generally be manually specified. In recent years, there has been a resurgence of methods for automatic learning of tasks from human-provided demonstrations. Unfortunately, these two fields, task learning and motion planning, have evolved largely separate from one another, and the learned models are often not usable by motion planners.

In this thesis, we aim to bridge the gap between robot task learning and motion planning by employing a learned task model that can subsequently be leveraged by an asymptotically-optimal motion planner to autonomously execute the task. First, we show that application of a motion planner enables task performance while avoiding novel obstacles and extend this to dynamic environments by replanning at reactive rates. Second, we generalize the method to accommodate time-invariant model parameters, allowing more information to be gleaned from the demonstrations. Third, we describe a more principled approach to temporal registration for such learning methods that mirrors the ultimate integration with a motion planner and often reduces the number of demonstrations required. Finally, we extend this framework to the domain of mobile manipulation. We empirically evaluate each of these contributions on multiple household tasks using the Aldebaran Nao, Rethink Robotics Baxter, and Fetch mobile manipulator robots to show that these approaches improve task execution success rates and reduce the amount of human-provided information required.

To my parents for their steadfast support and boundless patience.

## ACKNOWLEDGEMENTS

I would like to thank Ron Alterovitz for encouraging me to persue a Ph.D. and providing me the opportunity to do so. In this, as in more technical matters, his advice was impeccable. Similarly, I would like to thank the other members of my committee for their valuable time and invaluable feedback.

Other students in the Computational Robotics Research Group have come and gone over the past few years, but the countless discussions held and whiteboards filled here remain with me. Specifically, I would like to thank Alan Kuntz for opportunities to explore new problems and my co-author Gu Ye for his early guidance.

This work would not have been possible without the support of the National Science Foundation under awards IIS-0905344, IIS-1117127, IIS-1149965, CNS-1305286, and CCF-1533844.

Last but not least, I would like to thank my family for their seemingly endless patience, unwavering support, and love.

## TABLE OF CONTENTS

**LIST OF FIGURES** . . . . . **x**

**LIST OF TABLES** . . . . . **xii**

**CHAPTER 1: INTRODUCTION** . . . . . **1**

1.1 Challenges . . . . . 2

1.2 Contributions . . . . . 4

1.2.1 Asymptotically Optimal Motion Planning for Learned Robotic Tasks . . . . . 4

1.2.2 Reactive Replanning for Learned Robotic Tasks . . . . . 5

1.2.3 Learning Virtual Landmarks . . . . . 6

1.2.4 Improving Temporal Registration . . . . . 6

1.2.5 Motion Planning for Learned Mobile Manipulation Tasks . . . . . 6

1.3 Thesis Statement . . . . . 7

1.4 Organization . . . . . 7

**CHAPTER 2: ASYMPTOTICALLY OPTIMAL MOTION PLANNING FOR LEARNED ROBOTIC TASKS** . . . . . **8**

2.1 Related Work . . . . . 9

2.2 Method Overview . . . . . 11

2.3 Background . . . . . 12

2.3.1 Hidden Markov Models . . . . . 12

2.3.2 Probabilistic Roadmaps . . . . . 13

2.3.3 Product Graphs . . . . . 13

2.4 Learning the Time-Dependent Cost Metric . . . . . 14

2.4.1 Inputs and Outputs of Cost Metric Learning . . . . . 15

2.4.2 Extracting Motion Features from Demonstrations . . . . . 16

2.4.3 Statistical Modeling of the Motion Features . . . . . 17

2.4.4 Cost Metric . . . 19
2.5 Motion Planning Using the Learned Cost Metric . . . 21
2.5.1 Inputs and Outputs of Motion Planning . . . 21
2.5.2 Sampling-Based Planning for the Learned Cost Metric . . . 22
2.5.3 Demonstration-Guided Speedups for Motion Planning . . . 25
2.5.4 Analysis . . . 26
2.6 Results . . . 30
2.6.1 Simulated 2D Navigation Task . . . 30
2.6.2 Physical Task 1: Left-to-Right Powder Transfer Task . . . 31
2.6.3 Physical Task 2: General Powder Transfer Task . . . 33
2.6.4 Physical Task 3: Push a Button . . . 35
2.7 Conclusion . . . 38

**CHAPTER 3: REACTIVE REPLANNING FOR LEARNED ROBOTIC TASKS 39**

3.1 Related Work . . . 40
3.2 Method Overview . . . 42
3.2.1 Problem Statement . . . 42
3.2.2 Learned Task Model . . . 43
3.3 Closed-loop Replanning with Learned Costs . . . 44
3.3.1 Spatiotemporal Roadmap . . . 45
3.3.2 Cost Function . . . 46
3.3.3 Biased Sampling . . . 47
3.3.4 Real-Time Execution . . . 49
3.4 Results . . . 50
3.4.1 Simulated Navigation Task . . . 52
3.4.2 Physical Powder Transfer Task . . . 53
3.4.3 Physical Liquid Pouring Task . . . 55
3.5 Conclusion . . . 56

**CHAPTER 4: LEARNING VIRTUAL LANDMARKS . . . 57**

4.1 Related Work . . . 58

4.2 Problem Definition . . . 60
4.2.1 Inputs and Outputs . . . 60
4.2.2 Probabilistic Task Model . . . 61
4.3 Learning . . . 62
4.3.1 Feature Space using Virtual Landmarks . . . 62
4.3.2 Maximum a Posteriori Estimation . . . 63
4.3.3 Estimation via Minimization . . . 65
4.4 Results . . . 66
4.4.1 Powder Transfer Task . . . 66
4.4.2 Liquid Pouring Task . . . 69
4.5 Conclusion . . . 70

**CHAPTER 5: IMPROVING TEMPORAL REGISTRATION . . . 71**

5.1 Related Work . . . 73
5.2 Problem Definition . . . 74
5.3 Method . . . 75
5.3.1 Dynamic Time Warping as a Graph Algorithm . . . 76
5.3.2 Temporal Registration Using the Viterbi Algorithm . . . 78
5.3.3 Probability-weighted Temporal Registration Using the Forward-Backward Algorithm . . . 80
5.3.4 Non-degenerate Temporal Registration . . . 81
5.4 Application to Learning from Demonstrations . . . 82
5.5 Results . . . 84
5.5.1 Applied to the van den Berg et al. Method . . . 85
5.5.2 Applied to the Bowen et al. Method . . . 89
5.6 Conclusion . . . 90

**CHAPTER 6: MOTION PLANNING FOR LEARNED MOBILE MANIPULATION TASKS . . . 91**

6.1 Related Work . . . 92
6.2 Problem Definition . . . 93
6.3 Motion Planning using the Learned Model . . . 94

6.3.1 Mobile Manipulation Hybrid Roadmap . . . . . 94
6.3.2 Task Roadmap . . . . . 97
6.3.3 Task-Guided Gibbs Sampling . . . . . 100
6.3.4 Guiding Manifolds . . . . . 103
6.4 Results . . . . . 103
6.4.1 Sweeping Task . . . . . 103
6.4.2 Liquid Pouring Task . . . . . 106
6.5 Conclusion . . . . . 108

**CHAPTER 7: CONCLUSION . . . . . 109**
7.1 Limitations and Future Work . . . . . 109

**REFERENCES . . . . . 112**

## LIST OF FIGURES


1.1 Robots performing households tasks. . . . . 1

1.2 Overview of task learning and reactive execution . . . . . 4

2.1 Nao robot performing powder transfer and button pushing tasks . . . . . 9

2.2 Bayesian network for a hidden Markov model. . . . . 12

2.3 DGMP learning process diagram . . . . . 15

2.4 DGMP planning process diagram . . . . . 21

2.5 An example roadmap for a 2D configuration space with 3 time partitions in yellow, red, and blue. A path is shown in green. . . . . 23

2.6 Simulated 2D navigation task . . . . . 30

2.7 Nao robot performing a powder transfer task . . . . . 31

2.8 Evaluation on a left-to-right powder transfer task with a Nao robot . . . . . 33

2.9 Evaluation on a general powder transfer task with a Nao robot . . . . . 34

2.10 Planner convergence for a general powder transfer task with a Nao robot . . . . . 35

2.11 Scenarios for a button pushing task with the nao robot . . . . . 36

2.12 Planner convergence for a button pushing task with a Nao robot . . . . . 37

2.13 Execution of a button pushing task with the Nao robot . . . . . 37

3.1 Reactive execution of a powder transfer task with the Baxter robot . . . . . 39

3.2 Hidden Markov model with 3 discrete states (corresponding to time steps of the task) and normally distributed observations. . . . . 43

3.3 Cartesian product of the configuration space roadmap and the graph of the learned task model. . . . . 44

3.4 Biased sampling for a powder transfer task with the Baxter robot . . . . . 48

3.5 Flow of computation during real-time execution. . . . . 49

3.6 Simulated navigation task . . . . . 51

3.7 Environment for a powder transfer task . . . . . 51

3.8 Execution of a powder transfer task . . . . . 52

3.9 Execution of a liquid pouring task . . . . . 55

4.1 The Baxter robot executes the liquid pouring task while avoiding obstacles, including the paper towel roll, vase, lamp shade, and books. . . . . 57

4.2 Bayesian network for the learned task model with time-invariant parameters . . . . . 61

4.3 Baxter robot performs a powder transfer task without being informed of which landmarks were relevant . . . . . 66

4.4 Comparison of reference points from the robot's gripper . . . . . 67

4.5 Baxter robot performs a liquid pouring task without being informed of which landmarks are relevant . . . . . 68

5.1 Baxter robot performing a knot-tying task . . . . . 71

5.2 Temporal registration with dynamic time warping vs. the forward-backward algorithm 73

5.3 Probabilistic model for time alignment . . . . . 77

5.4 Dynamic time warping using a tensor product graph . . . . . 78

5.5 The Viterbi algorithm using a tensor product graph . . . . . 79

5.6 The forward-backward algorithm using a tensor product graph . . . . . 81

5.7 The forward-backward algorithm with a non-degeneracy constraint . . . . . 82

5.8 Simulated drawing task with Gaussian noise using different temporal registration methods . . . . . 85

5.9 Evaluation of the simulated drawing task with Gaussian noise using different temporal registration methods . . . . . 86

5.10 Simulated drawing task with Brownian motion noise using different temporal registration methods . . . . . 86

5.11 Evaluation of the simulated drawing task with Brownian motion noise using different temporal registration methods . . . . . 87

5.12 Knot-tying task on the Baxter robot . . . . . 87

5.13 Transfering powder into a cup with the Baxter robot . . . . . 88

6.1 Various roadmaps for a liquid pouring task with the Fetch robot . . . . . 91

6.2 Hybrid roadmap formed by the Cartesian product of an arm PRM and a base RRG 94

6.3 Graphical task model . . . . . 97

6.4 Fetch robot planning a sweeping task . . . . . 100

6.5 Environment for mobile manipulation tasks with the Fetch robot . . . . . 104

6.6 Sweeping task execution. . . . . 104

6.7 Fetch robot executes a liquid pouring task. . . . . 106

6.8 Plan costs with various planning times and sampling approaches for the liquid pouring task using the Fetch robot . . . . . 107

## LIST OF TABLES

3.1 Evaluation of the powder transfer task with a stationary bowl . . . . . 53

3.2 Evaluation of the powder transfer task where the bowl was moved during execution . 54

3.3 Evaluation of the liquid pouring task . . . . . 56

4.1 Evaluation of the powder transfer task where the robot was not informed of which landmarks were relevant . . . . . 67

4.2 Evaluation of the liquid pouring task where the robot was not informed of which landmarks were relevant . . . . . 69

5.1 Evaluation of the knot-tying task using different temporal registration methods . . . 88

5.2 Evaluation of the powder-transfer task using different temporal registration methods 89

6.1 Evaluation of the sweeping task . . . . . 105

CHAPTER 1

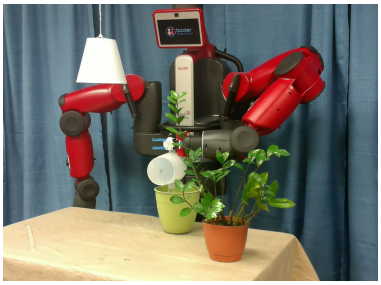
# Introduction

New robotic hardware has the potential to assist people with a variety of routine tasks in people's homes and workplaces like those shown in Figure 1.1 [1, 2]. However, to achieve this potential, robots require software. From assisting a person with a disability with an activity of daily living (such as cooking or cleaning) to performing small-scale manufacturing tasks, assistive robots need to be capable of planning and executing motions in cluttered environments that may contain unforeseen obstacles. Further complicating the software challenge, many assistive tasks involve important constraints on motion that humans are aware of from context and intuition. For example, when carrying a plate of food, a person knows that tilting the plate sideways, while feasible, results in task failure because it will spill the food. In order to autonomously and safely accomplish many assistive tasks, a robot must be aware of or approximate such task constraints and must plan and execute motions that consider these constraints while avoiding obstacles.

Identifying these constraints for each candidate task individually is time-consuming at best,

(a) Aldebaran Nao robot autonomously performing a powder transfer task.

(b) Fetch mobile manipulator robot autonomously performing a sweeping task.



(c) Rethink Robotics Baxter robot autonomously performing a liquid pouring task.

Figure 1.1: Robots performing households tasks.

infeasible at worst. To combat this bottleneck, much research has been devoted to automatic task learning. In this work, we focus specifically on learning from demonstrations, wherein a human kinesthetically demonstrates multiple successful executions of a task. Because these demonstrations were successful, similarity to them can be used an approximation for constraint satisfaction. This allows the robot to learn to execute the task autonomously with natural-looking motions.

From a user's perspective, after performing a few demonstrations in different scenarios, a learning algorithm is run which estimates an opaque model of the task. Then after sensing a new environment, the robot should be capable of using this learned model to plan motions to accomplish the task in that environment.

To be adopted and useful, this process of learning from demonstrations followed by execution cannot require a human to manually provide unrealistic amounts or types of information to achieve success. This information might take many forms; for example:

- Information about the intended execution environment, including obstacles.
- Task-specific specifications which a non-technical user is unlikely to be able to correctly provide.
- Tens or hundreds of kinesthetic demonstrations.

In this dissertation, we discuss and evaluate technological contributions that address each of these by rendering them unnecessary for a broad class of useful tasks, with the goal of learning a model of the task which can be generalized to enable execution in new environments by effectively bridging the gap between prior work in task learning and motion planning. Specifically we will adapt sampling-based motion planners [3] to execute task models learned through statistical methods [4, 5, 6, 7].

### 1.1 Challenges

**Unstructured Environments** The first key challenge is that homes and workspaces are unlike the manufacturing floors where robots have already been successfully applied. Those spaces were designed to accommodate robots. By contrast, homes are often cluttered with obstacles, including people, which must be avoided to safely perform a task, and the locations of objects in these environments are not fixed. While much prior work in robotics has focused on navigation in such environments and even while satisfying manual constraints, considerably less has addressed the

challenge of doing so for learned tasks. These problems are distinct because uncertainty is an inherent aspect of learning which the resulting task model should capture. At planning time, this uncertainty implies soft constraints rather than hard ones which usually arise from manual task specification.

**Changing Environments** A further challenge in environments where people live and work is that objects in the environment are often not static—objects relevant to a task may move independently or be moved by people. Again, prior work has considered this problem, but often by relying on heuristics like potential fields that sacrifice global guarantees [8, 9, 10]. Loss of such guarantees is not a purely academic issue; without some guarantee of global optimality (or near-optimality), solutions may be arbitrarily suboptimal, which is simply not good enough when operating around people or in other safety-critical scenarios.

**Mobility and Manipulation** Unlike manufacturing floors, household environments are generally designed with mobile occupants in mind. In fact, it is precisely occupants who have lost some portion of their mobility that may most need a robotic assistant to perform activities of daily living. So robots in these environments must be able to navigate in order to perform a variety of tasks. However, for a mobile manipulator robot performing a task, the problems of navigation and manipulation are inherently coupled [11]. To see this, observe that on the one hand, the manipulator may need to be repositioned to navigate through a narrow passage while adhering to task constraints (e.g., levelness of the grasped pitcher to avoid spilling). On the other hand, the robot may need to reposition itself to enable the manipulator to accomplish a given task (e.g., pouring liquid from the pitcher into a bowl). This coupling presents multiple challenges, mostly stemming from the associated increase in dimensionality. While a traditional robot arm might have only six degrees of freedom, a mobile manipulator like the Fetch robot [2] might have eleven or more. This increase in dimensionality can dramatically affect the performance of motion planners.

**Limited and Inconsistent Demonstrations** Human demonstrators are often inconsistent, and in fact, spatial inconsistency in demonstrations is vital when learning the full range of motions which result in task success. However, demonstrations are also inconsistent in time; some parts of a given task may be performed quickly in one demonstration and slowly in another, while a different

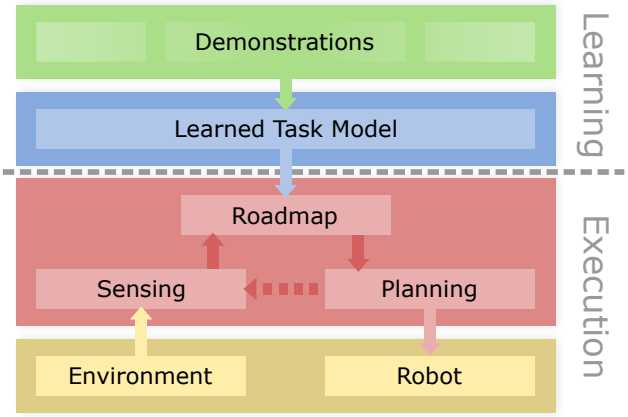



Figure 1.2: General overview of the components of the system, with human input shown in green, task learning shown in blue, execution execution shown in red, and environment shown in yellow. Note that the task model is not specific to an environment and so need only be learned once per task. It can then be used for execution across many environments. All arrows indicate execution flow and solid arrows indicate data flow.

portion of the task may exhibit the opposite effect. Factoring out this inconsistency is a common step across many robot task learning algorithms, including ours, but some of these methods may require tens or hundreds of demonstrations to adequately learn a task. These problems are not unrelated. As we will later show, improving how temporal inconsistency is handled can reduce the number of demonstrations the user must perform before the learner becomes effective.

## 1.2 Contributions

Roboticists have already developed robust methods for motion planning that focused largely on reachability. Simultaneously, numerous models of tasks and algorithms for estimating them from demonstrations have been developed, but without application of a motion planner in mind. So, to address all of the above challenges, we adapt and integrate these existing methods while introducing new methods specific to the problem of motion planning for learned tasks. We discuss our approach to each briefly here and in full detail in the corresponding chapter.

### 1.2.1 Asymptotically Optimal Motion Planning for Learned Robotic Tasks

To address the challenge of unstructured environments, our approach consists of two major phases: learning a task model from a set of demonstrations followed by task performance in the execution environment. Note that the task model is not tied to a specific environment, so it only needs to be learned once per task. It can then be used for execution across many environments, even if task-relevant objects are in different places and new obstacles are present. Figure 1.2 illustrates

an overview of the approach. For now, we focus on tasks in static environments that do not require dynamics considerations.

During the learning phase, the user provides demonstrations of a task. These demonstrations can be performed with no obstacles present. We use statistical methods to learn a task model, which can be mapped to new environments with previously unseen obstacles and in which task-relevant objects may be in different locations.

Methods based on learning from demonstrations have already proven to be highly effective at automatically learning task constraints and controllers from demonstrations by people who may not have programming expertise but typically do not handle obstacles at all or do so in a way that is not globally optimal, often relying on potential fields (e.g. [12, 13, 14]). Fortunately, sampling-based motion planners for robotic manipulators have become increasingly efficient at computing feasible plans that avoid obstacles [15]. However, these motion planners typically require that the task constraints (such as keeping a plate level) be manually programmed, which can be tedious and requires a programmer with domain knowledge. So we integrate ideas from demonstration-based learning into a sampling-based motion planner with asymptotic optimality, yielding a framework for robots to compute motion plans that (1) avoid obstacles in unstructured environments and (2) aim to satisfy learned features of the motion that are required for the task to be successfully accomplished.

This contribution is discussed in Chapter 2 and was also presented in [16].

### 1.2.2 Reactive Replanning for Learned Robotic Tasks

To address the unique challenge of environments which may change during execution, we continuously replan by rebuilding the roadmap and then searching for a new plan. This replanning approach computes plans that avoid obstacles while performing the task based on the current state of the environment. This approach closes the loop between planning and sensing as indicated by the dotted line in Figure 1.2. Ultimately, we enable the robot to replan in real-time, averaging more than 5 plans per second in our experiments, by leveraging information in the task model and using appropriate data structures and algorithms.

This contribution is discussed in Chapter 3 and includes work originally presented in [17] along with work which later appeared in journal form [18].

### 1.2.3 Learning Virtual Landmarks

We extend on the learning approach by allowing some additional time-invariant parameters of the task to be learned from the demonstrations while simultaneously learning the time-variant parameters considered previously. In that model, time-variant parameters are used to encode how the robot should move as the task progresses by identifying low variance features that are likely important for successful task performance due to their consistency in the demonstrations. In contrast, we use time-invariant parameters to define the space in which we learn by determining which features to consider. So rather than requiring manual specification of what features might potentially be important, we automatically learn them by estimating the most informative (i.e. consistent) value for the time-invariant parameter from the demonstrations. For example, these parameters are used to define which objects in a cluttered environment are actually relevant to the task that is being learned.

This contribution is discussed in Chapter 4 and was originally presented in [19].

### 1.2.4 Improving Temporal Registration

Many existing methods for robot learning from demonstrations require registering a time sequence of observations to a reference model, either for aligning demonstrations during preprocessing or as an integral part of task model estimation. Our learning phase is no exception. Often dynamic time-warping (DTW) is used for this purpose thanks in part to its performance and ease of implementation, but DTW aims to find only the single most likely temporal registration. We introduce probability-weighted temporal registration, a more general form of temporal registration that explicitly captures uncertainty in the registration. Instead of assuming each observation is registered to (at most) one time step of the reference model, we use the forward-backward algorithm to compute probability-weighted assignments and avoid degenerate registrations. We apply this approach to two learning methods from prior work on both simulated and physical tasks and show that incorporating uncertainty into robot learning algorithms can yield higher-quality task models that enable faster task executions and higher task success rates.

This contribution is discussed in Chapter 5.

### 1.2.5 Motion Planning for Learned Mobile Manipulation Tasks

We present an approach to accelerating motion planning for mobile manipulation tasks learned from demonstrations. This method guides sampling toward configurations most likely to be useful

for successful task execution while avoiding manual heuristics and preserving asymptotic optimality of the motion planner. We leverage the learned task model which is used by the motion planner to evaluate plan cost, to also guide sampling, yielding plans with high rates of success faster than unbiased or goal-biased sampling. This is accomplished by tightly integrating sampling with a hybrid motion planner that builds separate base and arm roadmaps using Gibbs sampling [20]. Such an approach allows the sampled arm configurations to depend on the reachable base configurations and vice-versa. We evaluate our method on two household tasks using the Fetch robot and greatly improve upon motion planners that rely on unbiased sampling or either of two goal-biased planners when using the same cost metric.

This contribution is discussed in Chapter 6 and was originally presented in [21].

### 1.3 Thesis Statement

The unifying theme throughout this work is that of better leveraging the information and methods available to us to avoid requiring more from the user than necessary. Concisely, we aim to show that:

*Robotic systems can learn, from demonstrations, to perform tasks in unstructured environments while avoiding obstacles with less prior knowledge by better extracting information from the demonstrations and leveraging an asymptotically optimal motion planning method during execution.*

Each chapter of this dissertation supports this thesis statement as outlined below.

### 1.4 Organization

In Chapter 2, we present a learned task model and show that it can be directly used by an asymptotically optimal motion planner to perform tasks in new environments with new obstacles. In Chapter 3, we further leverage the learned model to make this approach fast enough to be performed multiple times per second to react to sensed changes in the environment. In Chapter 4 we show that time-invariant parameters of the task can be learned simultaneously with the time-variant ones of the aforementioned model, effectively enabling automatic feature space selection in the form of virtual landmarks on grasped tools. In Chapter 5 we present an improved approach to time alignment for learning robotic tasks using the forward-backward algorithm, which can enable the robot to learn better models of a task from fewer demonstrations. Finally in Chapter 6, we extend this method to mobile manipulators and tasks which require mobility while retaining global optimality guarantees before concluding in Chapter 7.

CHAPTER 2

## Asymptotically Optimal Motion Planning for Learned Robotic Tasks

In this chapter, we present *demonstration-guided motion planning* (DGMP), a framework for robots to compute motion plans that (1) avoid obstacles in unstructured environments and (2) aim to satisfy learned features of the motion that are required for the task to be successfully accomplished. We focus on tasks in static environments that do not require dynamics and in which task success depends on the relative pose of the robot's end effector to objects in the environment.

At the core of DGMP is an asymptotically optimal sampling-based motion planner that computes motion plans that are both collision-free and globally minimize a cost metric that encodes learned features of the motion. The motivation for our cost metric is that if the robot is shown multiple demonstrations of a task in various settings, features of the demonstrations that are consistent across all the demonstrations are likely to be critical to task success, while features that vary substantially across the demonstrations are likely unimportant. For example, when transferring instant coffee powder from a container to a cup (see Figure 2.1), the feature of the levelness of the spoon will be consistent across the demonstrations (i.e., low variance) while the height of the spoon from the table may vary (i.e., high variance) due to the presence of other objects on the table. Leveraging this insight, our method takes as input a set of kinesthetic demonstrations in which a person holds the robot's limbs and guides the robot to perform the task while we record time-dependent motion features, including the robot's configurations and the pose of the end effector relative to task-relevant objects in the environment. The placement of these objects, such as the coffee container and cup in the example above, are randomized for each demonstration. We then estimate a learned task model that encodes the covariances of the motion features as a function of time.

Once the task model is learned, DGMP can be used to autonomously execute the learned task in a static environment in which task-relevant objects may be in different locations and new obstacles may be present. Using the learned task model, we define a time-dependent cost map specific to the current environment. The cost map, defined over the robot's configuration space, considers the

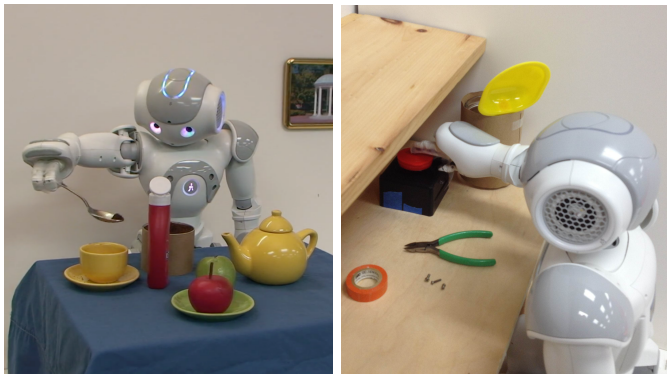


Figure 2.1: Tasks in many domains require both avoiding obstacles and also satisfying task constraints. Transferring powder (e.g., instant coffee, sugar, mixes) by spoon requires avoiding obstacles while keeping the spoon level to avoid spills (left). Pushing a (red) button requires that the robot avoid obstacles while ensuring the finger advances at an appropriate orientation when approaching the button (right).

covariances of the motion features across demonstrations using a Mahalanobis distance metric and is parameterized by the locations of the task-relevant objects. The cost map is defined such that a trajectory that minimizes cost has the highest probability density given the distribution of the successful demonstrations in motion feature space.

This chapter is based on work previously published in [16].

## 2.1 Related Work

Sampling-based methods have been highly successful for computing feasible and optimal motion plans for a wide variety of robots, including manipulators with many degrees of freedom [15, 3]. While most sampling-based motion planners aim to minimize metrics such as Euclidean distance in the workspace or configuration space, some methods have investigated incorporating task constraints. Several approaches are based on rapidly exploring random trees (RRTs) [15], a highly successful method for computing feasible, obstacle-avoiding trajectories but which does not guarantee plan optimality [3]. Transition-based RRT (T-RRT) [22] biases expansion of an RRT to low cost regions of the configuration space cost map, and Mainprice et al. used T-RRT to generate natural motions based on a predefined cost map for human robot interaction [23]. RRTs have also been used in conjunction with analytically-defined task constraints [24] and with symbolic representations of manipulation strategies [25]. Recent sampling-based motion planners have also investigated integrating motion constraints and properties learned from demonstrations. Algorithms include sampling only inside a user-specified number of standard deviations of a mean demonstrated trajectory [26], finding

low-cost paths over cost maps using local optimization [27], locally optimizing a specified objective function using gradient descent [28], and enforcing constraints using sampling strategies [29]. Prior sampling-based motion planning approaches, unlike our proposed method, do not simultaneously guarantee asymptotic optimality and allow for time-dependent task constraints.

At the heart of our method is an asymptotically optimal sampling-based motion planner, meaning the computed plan is guaranteed to approach a globally optimal plan (based on the given cost metric) as computation time is allowed to increase. Karaman and Frazzoli proposed motion planning algorithms such as RRG and PRM* that guarantee asymptotic optimality [3]. Asymptotically optimal motion planners avoid the suboptimal plans resulting from local minima that can occur when using potential field methods [15] or sampling-based planners not designed for asymptotic optimality like RRT [3]. Related work has investigated asymptotically optimal planners that balance exploration and refinement [30], asymptotic near-optimal planners using smaller roadmaps [31], and anytime solution optimization [32]. Our method integrates a learned cost metric with RRG or a PRM variant [3] to guarantee asymptotic optimality for our learned cost metric.

Our method combines a new sampling-based motion planner with ideas from demonstration-based learning, which has been highly successful in enabling robots to learn task constraints and imitate task motions [4, 33]. Our focus is not on learning control policies for dynamic systems (e.g., [34, 35, 36, 37]) but rather on computing robot trajectories that avoid obstacles while satisfying learned constraints. Our aim is globally optimal obstacle avoidance in which the robot considers plans in all homotopic classes and selects the best one. Prior work has investigated using search methods such as A* or D* where cost maps or movement costs are learned from demonstrations (e.g., [38, 39, 40, 41]), which are highly effective for 2D, discrete state spaces but do not scale well to higher degree of freedom systems like robotic arms.

An alternative approach is to locally avoid obstacles, which works well for some applications but does not guarantee global optimality. Potential field approaches have been applied to dynamic movement primitives [8] and a Gaussian mixture model (GMM) [9] to locally avoid obstacles, but potential fields require setting parameters for obstacle repulsion and can result in a robot being trapped in local minima, especially in obstacle concavities or narrow passages [15].

Another approach is to include the obstacles in the demonstrations. Calinon et al. introduced a GMM and Gaussian mixture regression (GMR) approach to learn motions relative to task-relevant

objects and obstacles that are present in both the demonstration and execution environments [7, 42]. This approach represents a task using a hidden Markov model (HMM); HMM's have been used for motion recognition (e.g., [43, 5, 6]) and generation (e.g., [7, 6]).

We also use an HMM; however, we use a restricted form which permits us to build on prior work on dynamic time-warping [34, 35] to create high quality alignments using an expectation-maximization approach and then directly compute means and covariances in the space of motion features. We use this model to construct a time-dependent cost map which we can integrate into an asymptotically optimal sampling-based motion planner for obstacle avoidance.

### 2.2 Method Overview

Let $\mathcal{Q} \subseteq \mathbb{R}^d$ be the $d$-dimensional configuration space of a holonomic robot and $\mathcal{Q}_{\text{free}} \subseteq \mathcal{Q}$ denote the set of configurations in which the robot is not in collision with an obstacle. We assume the robot has also sensed the poses of $L$ task-relevant objects (such as the cup and instant coffee container in Figure 2.1(left)), which are stored in a vector $\mathbf{a} \in \text{SE}(3)^L$, and that these objects remain stationary as the task is performed. We also assume the robot is holonomic with position-controlled joints, and we do not consider dynamics. Our objective is to compute a trajectory $\Phi \in [0, 1] \rightarrow \mathcal{Q}_{\text{free}}$ from the robot's initial configuration $\mathbf{q}_{\text{start}} \in \mathcal{Q}_{\text{free}}$ to a goal configuration $\mathbf{q}_{\text{goal}} \in \mathcal{Q}_{\text{free}}$ such that the robot successfully accomplishes the task.

To address this challenge, we develop an approach for demonstration-guided motion planning (DGMP) that consists of two major phases: learning and execution. The approach requires as input a set of user-provided demonstrations of the task. During the cost metric learning phase, the robot learns from the demonstrations a time-dependent cost metric for the task that considers the robot's configuration and its motion relative to task-relevant objects. The learning phase need only be performed once per task. When the robot is in a new environment, the robot enters the motion planning phase in which it computes a path that minimizes the learned cost metric, which captures aspects of the demonstrated motions that are required to perform the task.

During the DGMP learning phase, presented in Section 2.4, we first extract from each demonstration a set of *motion features* that quantify properties of the motion as a function of time, such as joint angles or the location of the end effector with respect to a task-relevant object. After time-aligning the demonstrations, we compute statistics on the motion features, including their means and variances, over time across the demonstrations. The lower the variance of a motion

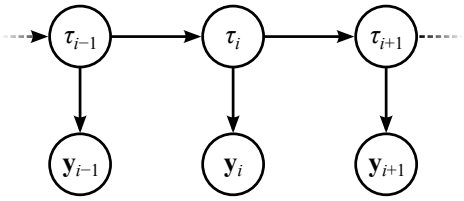



Figure 2.2: Bayesian network for a hidden Markov model.

feature across demonstrations at a given time, the higher the consistency of the demonstrations with respect to that feature, which implies the mean value of a motion feature should be followed more closely when performing the task. In contrast, high variance motion features likely do not need to be closely reproduced during execution.

To compute a cost metric, we will leverage the intuition above regarding the increased importance of motion features with low variances. Formally, we consider the demonstrations (as encoded in the space of motion features) to be samples from the distribution of trajectories that will succeed at the task. We then model the quality of a candidate motion plan in the execution environment as the likelihood that it, too, is a sample from this distribution of successful trajectories and define a cost metric such that better plans have lower costs.

In the DGMP execution phase, presented in Section 2.5, the robot first senses its environment to collect sufficient information to evaluate the learned cost metric and to perform collision detection. We then execute our new asymptotically optimal motion planning algorithm, DGPRM, to search for a feasible, collision-free motion plan that minimizes the learned cost metric, and hence reproduces the demonstrator's intent as closely as possible in the new environment.

## 2.3 Background

Because this dissertation builds on concepts from prior work, we very briefly describe some necessary background material here for the reader's convenience.

### 2.3.1 Hidden Markov Models

Discrete-time Markov processes (or Markov chains) are distributions over sequences of states $(\tau_0, \tau_1, \ldots)$ in some state space $\mathcal{T}$ such that $\tau_{i+1}$ is independent of $(\tau_0, \ldots, \tau_{i-1})$ given $\tau_i$. Intuitively, the process is forgetful; it exhibits no hysteresis. We say such a process is time-homogeneous if it is also independent of the index $i$ (given $\tau_{i-1}$). Such a process is fully described by the transition probabilities between states. In this dissertation, we will be concerned only with Markov processes

with finite state spaces, so a process can be parameterized by its transition probabilities.

A hidden Markov model (HMM) can be viewed as a probabilistic transformation of the sequence produced by a Markov process. It is a distribution over sequences of observations $(\mathbf{y}_0, \mathbf{y}_1, \ldots)$ in some observation space $\mathcal{Y}$, where $\mathbf{y}_i$ is independent of both $(\tau_0, \ldots, \tau_{i-1})$ and $(\mathbf{y}_0, \ldots, \mathbf{y}_{i-1})$ given $\tau_i$ (see Figure 2.2). The name derives from the fact that we are generally only given the resulting sequence of observations, and a common problem is to infer the (hidden) states which produced it. In addition to the parameters of the underlying Markov process, parameterization of an HMM necessitates choosing a parameterization of the mapping from states to observations.

### 2.3.2 Probabilistic Roadmaps

Let $\mathcal{Q}_{\text{free}}$ denote the free configuration space of a holonomic robot and $M \in \mathcal{Q}_{\text{free}} \times \mathcal{Q}_{\text{free}} \to \mathbb{R}$ be any metric on this space. Recall that a metric is non-negative, symmetric, and zero if and only if both arguments are equal. A probabilistic roadmap (PRM) is an undirected graph with vertices denoted $\mathcal{V} \subseteq \mathcal{Q}_{\text{free}}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Given a method for querying when vertices can be locally connected in a collision-free way, CanLink, and a fixed connection radius $r > 0$, the PRM is constructed using a relatively simple procedure.

**Algorithm 2.1** BuildPRM($n$)

**Input:** Numbers of samples $n$
  **while** $|\mathcal{V}| < n$ **do**
    Sample $\mathbf{q}$ uniformly at random from $\mathcal{Q}_{\text{free}}$ // via rejection sampling
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{q}\}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{q}, \mathbf{q}') \mid \mathbf{q}' \in \mathcal{V} \wedge M(\mathbf{q}, \mathbf{q}') < r \wedge \text{CanLink}(\mathbf{q}, \mathbf{q}')\}$
  **end while**

Paths in this graph thus represent collision-free paths in configuration space, so the problem of motion planning is reduced to that of graph search. There are many variants of probabilistic roadmaps. The description here matches what Karaman and Frazzoli call sPRM [3] where they show it is asymptotically optimal, meaning that as the order of the graph approaches infinity, the costs of shortest paths in the graph approach those of true minimum cost plans.

### 2.3.3 Product Graphs

For a directed or undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$, let $s(e) \in \mathcal{V}$ and $t(e) \in \mathcal{V}$ denote the source and target of an edge $e \in \mathcal{E}$ respectively (where in the undirected case we simply ignore the semantic distinction). Given two such graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$,

their product is another graph with order (cardinality of vertices) equal to the product of the orders of $\mathcal{G}_1$ and $\mathcal{G}_2$. The size (cardinality of edges) of the product is determined by the type of product graph under consideration, two of which are relevant here: the Cartesian product and the tensor product.

Let $\mathcal{G}_\times = (\mathcal{V}_\times, \mathcal{E}_\times)$ denote the tensor product of $\mathcal{G}_1$ and $\mathcal{G}_2$. The vertices of this graph are given by $\mathcal{V}_\times = \mathcal{V}_1 \times \mathcal{V}_2$ and the edges similarly by $\mathcal{E}_\times = \mathcal{E}_1 \times \mathcal{E}_2$. So the size of $\mathcal{G}_\times$ is the product of the sizes of $\mathcal{G}_1$ and $\mathcal{G}_2$. The source and target of an edge $e = (e_1, e_2) \in \mathcal{E}_\times$ are intuitively defined by $s(e) = (s(e_1), s(e_2))$ and $t(e) = (t(e_1), t(e_2))$.

Let $\mathcal{G}_\square = (\mathcal{V}_\square, \mathcal{E}_\square)$ denote the Cartesian product of $\mathcal{G}_1$ and $\mathcal{G}_2$. As with the Tensor product, $\mathcal{V}_\square = \mathcal{V}_1 \times \mathcal{V}_2$. However, $\mathcal{E}_\square = \mathcal{E}_1 \times \mathcal{V}_2 \cup \mathcal{V}_1 \times \mathcal{E}_2$. So the size of $\mathcal{G}_\square$ is $|\mathcal{E}_1| \cdot |\mathcal{V}_2| + |\mathcal{V}_1| \cdot |\mathcal{E}_2|$. The source and target of an edge $e \in \mathcal{E}_\square$ are given by

$$s(e) = \begin{cases} (s(e_1), v_2) & \text{if } e = (e_1, v_2) \in \mathcal{E}_1 \times \mathcal{V}_2 \\ (v_1, s(e_2)) & \text{if } e = (v_1, v_2) \in \mathcal{V}_1 \times \mathcal{E}_2 \end{cases}$$

and

$$t(e) = \begin{cases} (t(e_1), v_2) & \text{if } e = (e_1, v_2) \in \mathcal{E}_1 \times \mathcal{V}_2 \\ (v_1, t(e_2)) & \text{if } e = (v_1, v_2) \in \mathcal{V}_1 \times \mathcal{E}_2 \end{cases}.$$

### 2.4 Learning the Time-Dependent Cost Metric

In the first phase of DGMP, we learn a time-dependent cost metric for a task based on the robot's configurations and motions relative to task-relevant objects in a set of demonstrations. The cost metric encodes the spatial variations and temporal ordering of the task. The robot will later use this cost metric when planning its motions to complete the task in new environments where task-relevant objects may have moved and new obstacles may be present.

Rather than directly learning constraints, we learn a cost metric, an approach that offers two advantages. First, a cost metric better models how we observe humans perform a task. A human holding a spoon to transfer powder typically holds the spoon roughly level with no explicit, hard bounds on deviations from level. Second, the use of a cost function allows us to learn relatively

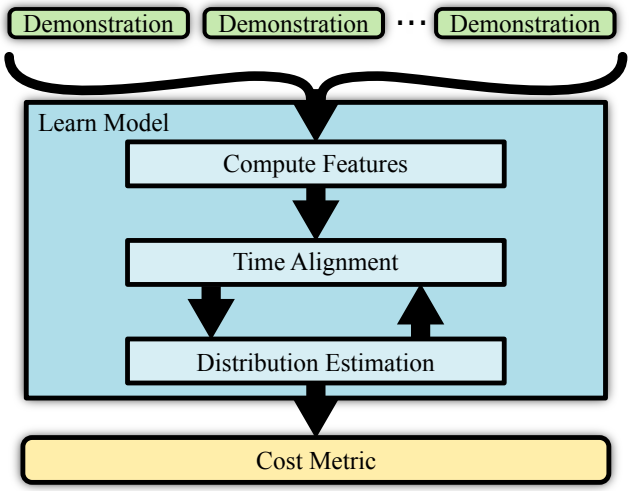



Figure 2.3: The DGMP learning phase lifts demonstrations into a feature space, iteratively time aligns the demonstrations, and learns a cost metric in the feature space.

complex tasks from a small number of demonstrations. While it is hard to differentiate the relevant similarities across demonstrations from the infinitely many irrelevant coincidences without the extensive semantic knowledge that a human would have, it is relatively easy to measure similarity between a candidate trajectory and the demonstrations (as is required for cost metric learning). We show an overview of the learning phase in Figure 2.3.

### 2.4.1 Inputs and Outputs of Cost Metric Learning

As input to the DGMP learning phase, a human controls the robot to perform $M$ demonstrations of the task. For each demonstration, the robot's joints are placed in a passive mode while a human manually moves the robot's limbs to perform the task and indicates where the task begins and ends. We assume the robot has encoders at every joint, allowing the robot to sense its own motion and record its configuration (e.g., a vector of its joint angles) as a function of time. During each demonstration $j \in \{1, \ldots, m\}$, we record a time sequence of the robot's configuration $\{\mathbf{q}_{j,i}\}_{i=1}^{n_j}$, where $n_j$ is the length of the demonstration and $\mathbf{q}_{j,i}$ is the configuration at time $i$. For each demonstration, we also require an annotation $\mathbf{a}_j \in \mathrm{SE}(3)^L$ identifying the poses of $L$ task-relevant objects in the environment (e.g., the coffee container and cup in the task shown in Figure 2.1), which could be identified either manually by the human or automatically using computer vision algorithms. We denote the poses in demonstration $j$ of the task-relevant objects $\{(\boldsymbol{R}_{j,l}, \mathbf{o}_{j,l}) \mid j = 1, \ldots, m; l = 1, \ldots, L\}$, where $\boldsymbol{R}_{j,l}$ is the rotation matrix and $\mathbf{o}_{j,l}$ is the translation vector of landmark $l$ with respect to a global frame.

Because the objects are task-relevant, they should be present in all demonstrations and necessarily must be present in the execution environment.

The output of the DGMP learning phase is a time dependent cost metric $c(\mathbf{q}, t, \mathbf{a})$ for robot configuration $\mathbf{q}$ at time $t$ for an execution environment with annotations $\mathbf{a}$, which may be different from any of the values of $\mathbf{a}$ in the demonstrations.

### 2.4.2 Extracting Motion Features from Demonstrations

Since each demonstration is a successful task execution, we expect that the task constraints for a problem are satisfied in each demonstration. To enable learning of these task constraints, we consider a set of motion features that are designed to help identify aspects of the robot motions that are consistent across demonstrations. These motion features may depend on both the configuration and annotation. We denote motion feature $k$ for time step $i$ of demonstration $j$ as $\mathbf{y}_{j,i}^{(k)}$. Inspired by results from Calinon et al. [7, 42], for our experiments we consider two classes of motion features:

- A *configuration motion feature* is the robot's configuration at a particular time. When there are redundant degrees of freedom, this data enables learning natural motions that are lost when only considering end-effector motions. We define this motion feature as

$$\mathbf{y}_{j,i}^{(0)} = \mathbf{q}_{j,i}.$$

- A *landmark-based motion feature* is a vector of the coordinate of a point on the robot $\mathbf{x}'$ (e.g., end-effector, grasped object) relative to a landmark on a task-relevant object in the environment (e.g., cup, button). This motion feature facilitates task execution for cases in which task-relevant objects may be located in different places across demonstrations and during execution. We define the motion feature relative to landmark $l$ as

$$\mathbf{y}_{j,i}^{(l)} = \boldsymbol{R}_{j,l}^{-1}(\mathbf{x}'_{j,i} - \mathbf{o}_{j,l}).$$

Because we compute these motion features from the same information, namely the configuration and annotation, it is convenient to consider both types of motion features in the context of a unifying joint motion feature space $\mathcal{Y}$ along with some general function $\phi \in (\mathcal{Q}, \mathrm{SE}(3)^L) \to \mathcal{Y}$ which lifts a configuration $\mathbf{q}$ into the motion feature space given some annotation $\mathbf{a}$. Such a function can

represent multiple motion features simply by computing each motion feature $\mathbf{y}^{(k)}$ individually and concatenating them into a single higher-dimensional motion feature vector $\mathbf{y} = \phi(\mathbf{q}, \mathbf{a})$.

Similarly, we may consider a trajectory of motion features for each demonstration which we denote $Y_j = \{\mathbf{y}_{j,1}, \mathbf{y}_{j,2}, \ldots, \mathbf{y}_{j,n_j}\}$ where $\mathbf{y}_{j,i} = \phi(\mathbf{q}_{j,i}, \mathbf{a}_j)$. Constructing these motion features is the sole purpose of the demonstrations. The remainder of the cost metric learning method operates exclusively in motion feature space.

### 2.4.3 Statistical Modeling of the Motion Features

Our objective in this space is to identify consistent aspects of the motion feature trajectories across demonstrations in order to create a cost metric, parameterized by the locations of the task-relevant objects, which will guide the motion planner.

To achieve this, we learn a statistical model consisting of $T$ multivariate Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ in motion feature space, each of which models the distribution of motion feature vectors of the demonstrations at some time step $t \in \{1, \ldots, T\}$ in the task. Hence, our model of the task is parameterized by the mean $\boldsymbol{\mu}_t$ and the covariance matrix $\boldsymbol{\Sigma}_t$ of motion feature vectors for each time step $t$. These may be considered as output distributions in a simple HMM with $T$ sequential states, wherein each state $t$ has nonzero and equal transition probabilities only to itself and the next state $t + 1$. This induces a linear order structure to the HMM corresponding to time. The problem then is to find the output distributions most likely to have generated the demonstrations.

To learn these distributions, we must find the correct monotonic mapping, or *alignment*, between each of the observed configurations in the demonstrations and the $T$ time steps. This corresponds to determining the walk in the HMM which generated the demonstration. This is necessary because the demonstrations are of different lengths and may perform parts of the task more or less quickly. For instance, the demonstrations of the powder transfer task mentioned previously varied from 21 to 32 seconds in length. Estimating to which state of the task a given observed configuration in a demonstration corresponds requires constructing a model of the task, which is exactly why we needed such an alignment in the first place. We resolve this cyclic dependence by applying an expectation-maximization (EM) method, a common approach to learning models of processes with latent variables. We note that the procedure described here is in actuality a maximization-maximization method while a true expectation-maximization procedure is considered in Chapter 5.

First we choose a random initial alignment for each demonstration to the time steps in the task. Next we estimate each distribution $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ using the sample mean and covariance of the motion feature vectors which are aligned to time step $t$ (weighted inversely proportional to the number of observed configurations aligned to time step $t$ from the same demonstration). This is the M step of the EM algorithm because we use maximum likelihood estimates. Next we find, for each demonstration, the walk in the estimated HMM most likely to have generated the demonstration using the Viterbi algorithm [44]. This is the E step of the EM algorithm. Finally, if the algorithm has not yet converged, we go back to the M step. Because EM algorithms can become caught in local minima, we repeat the entire method a number of times with different randomized initial alignments and use the resulting alignment with maximum likelihood.

More formally, let $Y_{j,t}$ denote the set of motion feature vectors in demonstration $m$ which are aligned to time step $t$. The M step computes the weighted sample mean and covariance (maximum likelihood estimates) at each time step from $Y_{j,t}$ as follows:

$$
\begin{aligned}
w_j^{(t)} &\leftarrow \frac{1}{|Y_{j,t}|} \\
\boldsymbol{\mu}_t &\leftarrow \frac{1}{M} \sum_{j=1}^{m} \left( w_j^{(t)} \sum_{\mathbf{y} \in Y_{j,t}} \mathbf{y} \right) \\
\boldsymbol{\Sigma}_t &\leftarrow \frac{M}{M^2 - \sum w_j^{(t)}} \sum_{j=1}^{m} \left( w_j^{(t)} \sum_{\mathbf{y} \in Y_{j,t}} (\mathbf{y} - \boldsymbol{\mu}_t)(\mathbf{y} - \boldsymbol{\mu}_t)^T \right) .
\end{aligned}
$$

The E step uses the Viterbi algorithm to align each of the demonstrations to the distributions learned in the M step, using a cost function which maximizes the likelihood that the motion feature vectors aligned to time step $t$ in the demonstration came from the distribution $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. The formulation of this cost is discussed in more detail in Section 2.4.4. Our results show that using an EM approach, rather than only a dynamic-time warping [45] preprocessing step as is commonly done in prior work, is crucial for effectively learning the task.

Accurately estimating the covariance matrices for the Gaussian distribution at each time step requires that we have a sufficient number of demonstrations. The number of demonstrations should exceed the dimension of the motion feature space. Intuitively, if the number of the demonstrations is smaller, then one or more time steps could have too few motion feature vectors aligned to it,

resulting in a singular matrix. In this case, we are only learning in a subspace of the motion feature space. We note that this lower bound is empirically tight for some problems, as shown in the results section.

It is possible to reduce the number of required demonstrations by approximating the motion features as independent. This corresponds to computing the covariance matrix of each configuration or landmark-based motion feature independently, resulting in $\mathbf{\Sigma}_t$ being a block diagonal matrix. This reduces the number of required demonstrations to be one plus the dimension of the largest motion feature vector. We used this approach in our experiments with the NAO robot in Section 2.6 and were able to effectively capture relevant task constraints.

### 2.4.4 Cost Metric

To formally define the cost metric, we consider the demonstrations to be samples from the distribution of trajectories that will succeed at the task. Given an annotation $\hat{\mathbf{a}}$ for the environment, we define the cost of a candidate trajectory in the environment based on how likely it is a sample from the distribution of successful trajectories.

With this approach, we wish the probability density of the trajectory being generated by the task model to be maximized when the cost metric is minimized. At a given time step $t$, with a configuration $\mathbf{q}$ this probability density is given by

$$p\left(\mathbf{q}, t \mid \boldsymbol{\mu}_t, \mathbf{\Sigma}_t, \hat{\mathbf{a}}\right) = N_{\mathbf{\Sigma}_t} e^{-\frac{1}{2}(\phi(\mathbf{q},\hat{\mathbf{a}})-\boldsymbol{\mu}_t)^T \mathbf{\Sigma}_t^{-1}(\phi(\mathbf{q},\hat{\mathbf{a}})-\boldsymbol{\mu}_t)}$$

where $N_{\mathbf{\Sigma}_t}$ is a normalization factor. However, operating in the space of probabilities is numerically inconvenient, so we instead consider the log probability density, yielding

$$\begin{aligned}&\log p\left(\mathbf{q}, t \mid \boldsymbol{\mu}_t, \mathbf{\Sigma}_t, \hat{\mathbf{a}}\right) = \\ &\log(N_{\mathbf{\Sigma}_t}) - \frac{1}{2}(\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t)^T \mathbf{\Sigma}_t^{-1}(\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t).\end{aligned}$$

When performing time alignment, it is this value which we maximize in the M step, but in the case of the cost metric, it can be simplified further. First by observing that the log normalization term is constant for a given time step and thus has constant contribution to the total cost of a trajectory, we can safely ignore it. Finally, we drop the $-\frac{1}{2}$ constant factor. This changes the sign, which is desirable because we wish to formulate the problem as a minimization rather than a

maximization. The final cost map is thus

$$c(\mathbf{q}, t, \hat{\mathbf{a}}) = (\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}_t^{-1} (\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t). \tag{2.1}$$

We note that this is simply the squared Mahalanobis distance [46] in feature space from $\mathbf{q}$ to the configurations observed in the demonstrations at time $t$. The cost metric we will minimize is the integral over this cost map.

In the following discussion we will drop the dependence on $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and $\hat{\mathbf{a}}$ from the notation for the sake of brevity.

This log probability density formulation has desirable properties, the most notable of which is composition. The sum of the log densities is the log of the product of the densities, $\log p(\mathbf{q}, t) + \log p(\mathbf{q}', t') = \log(p(\mathbf{q}, t)\, p(\mathbf{q}', t'))$. We first assume that $\mathbf{q}$ and $\mathbf{q}'$ are independent as will be the case when they are drawn independently from a sampling distribution by a sampling-based motion planner. We then assume independence between time steps, which while not generally true, is a convenient simplifying assumption. Under these independence assumptions, $p(\mathbf{q}, t)\, p(\mathbf{q}', t')$ is the joint probability density given the task model, so $\log p(\mathbf{q}, t) + \log p(\mathbf{q}', t') = \log p(\mathbf{q}, t, \mathbf{q}', t')$. This is important because our motion planner will find a trajectory which minimizes the integral of this cost map, which under these assumptions is equivalent to maximizing the probability density of the entire trajectory in our learned model.

We note that computing the likelihood given the model requires that we compute $\boldsymbol{\Sigma}_t^{-1}$, which exists only if $\boldsymbol{\Sigma}_t$ is non-singular. Cases where $\boldsymbol{\Sigma}_t$ is singular will arise, e.g., when multiple landmarks are fixed relative to each other. To overcome this in our implementation we employ a pseudoinverse, specifically the Moore-Penrose pseudoinverse. This approach has the desirable property of effectively collapsing multiple landmarks fixed relative to each other into a single landmark.

The learned cost map depends on knowing $\hat{\mathbf{a}}$, which contains the locations of the task-relevant objects. Hence, the cost metric is used after the robot senses the locations of the task-relevant objects in the execution environment. We describe in Section 2.5 how the learned cost metric is used in motion planning.

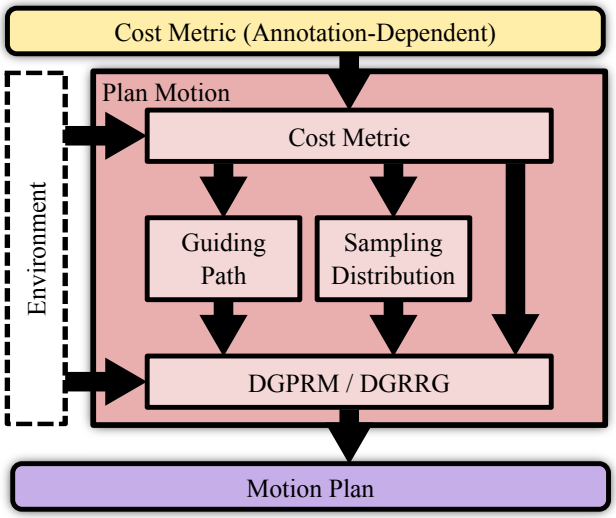



Figure 2.4: The DGMP execution phase uses output from the learning phase to construct a cost metric for the current execution evironment and then computes a motion plan that minimizes the cost metric.

## 2.5 Motion Planning Using the Learned Cost Metric

In the DGMP execution phase, the robot computes a feasible, collision-free motion plan in configuration space that minimizes the learned cost metric. We show an overview of the execution phase in Figure 2.4.

### 2.5.1 Inputs and Outputs of Motion Planning

The DGMP execution phase requires as input an annotation $\hat{\mathbf{a}}$ describing the new environment and a model of the obstacles that must be avoided. In our experiments, we used color and depth data from a Microsoft Kinect to automatically create models of obstacles as described in Section 2.6. The method also requires as input the robot's start and goal configurations $\mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}} \in \mathcal{Q}_{\text{free}}$ and the time-dependent cost map $c \in (\mathcal{Q}, [0,1]) \to \mathbb{R}^+$ as given by Eq. 2.1. For notational convenience, we scale time to be between 0 and 1 and drop the annotation parameter from $c$ since for motion planning this is always the observed execution environment $\hat{\mathbf{a}}$.

**Definition.** *We say* $\Phi \in [0,1] \to \mathcal{Q}$ *is a* trajectory *if and only if it is Lipschitz continuous. That is* $\exists K_\Phi \in \mathbb{R}^+, \quad \forall t_1, t_2 \in [0,1], \quad |\Phi(t_2) - \Phi(t_1)| < K_\Phi(|t_2 - t_1|)$.

**Definition.** *We say a trajectory* $\Phi$ *is* feasible *if and only if* $\forall t \in [0,1], \quad \Phi(t) \in \mathcal{Q}_{free}$, $\Phi(0) = \mathbf{q}_{\text{start}}$, *and* $\Phi(1) = \mathbf{q}_{\text{goal}}$.

**Definition.** *Let* $C(\Phi) = \int_0^1 c(\Phi(t), t)\,\mathrm{d}t$ *denote the cost of a trajectory* $\Phi$.

As discussed previously, our choice of cost metric has the advantageous property that the sum of the costs is the log likelihood in the joint distribution under the assumption of independent time steps. This is the discrete analogue of the integral formulation of $C$ used by the motion planner.

Our objective is to compute a feasible trajectory $\Phi^*$ that minimizes cost $C(\Phi^*)$.

### 2.5.2 Sampling-Based Planning for the Learned Cost Metric

We introduce DGPRM, a new sampling-based motion planner for computing plans that minimize the DGMP time-dependent cost metric. We employ a variation of a probabilistic roadmap (PRM) [47], because of its asymptotic optimality (using the sPRM variant) [3] and ease of parallelization, which we leverage. We also integrate DGMP with an RRG-based roadmap [3], which performs roughly equivalently when used with our DGMP-based extensions as discussed in the results. Our sampling-based motion planner guarantees that, as computation time is allowed to increase, all homotopic classes of plans will be considered and an optimal plan approached.

PRM methods construct a graph (called a roadmap) where each vertex (called a waypoint) corresponds to a configuration of the robot and each edge corresponds to a local plan for navigating from the configuration of one waypoint to another. This graph is constructed by repeatedly sampling a configuration from $\mathcal{Q}$ and adding a new waypoint to the roadmap corresponding to this configuration if it is collision-free. When a new waypoint is added to the roadmap, edges are constructed between it and other waypoints which are nearby in configuration space and connectable by collision-free paths. As the number of waypoints currently in the roadmap increases, the roadmap becomes a denser approximation of the collision-free configuration space.

To accommodate the time-dependency in the cost metric, we associate a time value with each waypoint and use a directed graph for the roadmap to forbid traversing edges backwards in time. We choose the time value associated with a given waypoint by maintaining a partitioning of the time span $[0, 1]$ which is initially just a single partition consisting of the entire time span $T = \{[0, 1]\}$. These partitions can be thought of as *layers* within the roadmap. We also choose some initial value $\Delta\mathbf{q}_{\max}$. We then alternate between two phases, *expansion* and *splitting*. Intuitively, these increase sampling density in configuration space and in time respectively. Throughout the process, we track the size of the largest partition, which we denote $\Delta t_{\max}$.

This iterative refinement of time partitions provides multiple benefits. The first and most notable of which is the capability to handle time-dependent cost metrics as present in DGMP. The second

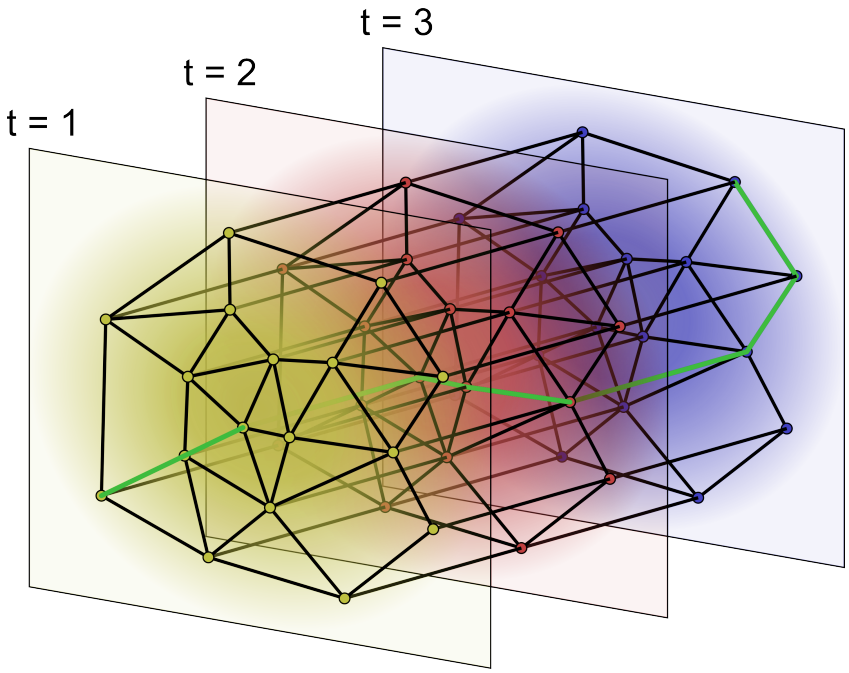



Figure 2.5: An example roadmap for a 2D configuration space with 3 time partitions in yellow, red, and blue. A path is shown in green.

benefit of this approach is that we do not require numeric integration of cost along roadmap edges, which typically requires an integration step size parameter. This approach can result in "missing" a small, high-cost region if this parameter is not properly tuned. In contrast, as DGPRM progresses, the maximum step size in both configuration space and time automatically decrease in such a way that the error in computing the cost metric for a trajectory approaches 0 under certain reasonable assumptions (see Section 2.5.4).

DGPRM begins by adding the start and goal configurations to the roadmap. Due to the layers, multiple waypoints may correspond to the same configuration. We say that the start waypoint is the waypoint corresponding to the start configuration in the first time partition, and the goal waypoint is the waypoint corresponding to the goal configuration in the last time partition. We let $n$ denote the number of configurations currently in the roadmap, so initially $n = 2$.

At any time, we may search the roadmap for the shortest path from the start waypoint to the goal waypoint. Interpolating linearly along this path yields a feasible trajectory, which is taken as an approximation of $\Phi^*$. An illustrative roadmap is shown in Figure 2.5.

In the algorithm below, $B \in \mathbb{R}^+$ is a constant which serves as a bias towards sampling more densely in configuration space or time which affects performance but not correctness.

**Algorithm 2.2** DGPRM($\mathcal{V}_{\text{initial}}$, $\mathcal{E}_{initial}$)

$\mathcal{V} \leftarrow \{ (\hat{\mathbf{q}}_t, t) \mid \hat{\mathbf{q}}_t \in \mathcal{F} \}$
$\mathcal{E} \leftarrow \left\{ (\hat{\mathbf{q}}_t, t, \hat{\mathbf{q}}_{t+1}, t+1) \mid \overline{\hat{\mathbf{q}}_t \hat{\mathbf{q}}_{t+1}} \subset \mathcal{F} \right\}$
**loop**
  $n_{\text{new}} \leftarrow B|T|^{d+2} \log(|T|)$
  ExpandRoadmap($n_{\text{new}} - n$, $\mathcal{V}$, $\mathcal{E}$)
  $n \leftarrow n_{\text{new}}$
  $t_{\text{split}} \leftarrow$ DetermineSplit()
  SplitRoadmap($t_{\text{split}}$, $\mathcal{V}$, $\mathcal{E}$)
  $\pi \leftarrow$ ShortestPath()
  **yield** $\pi$
**end loop**

**Expansion** In the expansion phase, $\Delta n$ additional configurations are sampled from $\mathcal{Q}$. For each configuration, if the configuration is collision free we add a waypoint to the roadmap in each time partition for the configuration. As in other PRM methods, edges lying entirely in $\mathcal{Q}_{\text{free}}$ are then added between nearby waypoints. In this case, nearby means nearby in both space and time. Specifically, edges are only added if the waypoint configurations are within $\Delta \mathbf{q}_{\max}$ and the time values are within adjacent partitions. The edges are also directional, and oriented forward in time.

**Algorithm 2.3** ExpandRoadmap($\Delta n$, $\mathcal{V}$, $\mathcal{E}$)

$\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{\Delta n}\} \leftarrow \Delta n$ samples from $\mathcal{Q}_{\text{free}}$
**for all** $\mathbf{q}_v \in \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{\Delta n}\}$ **do**
  **for all** $[t_i, t_{i+1}] \in T$ **do**
    $t_v \leftarrow$ arbitrary value $\in [t_i, t_{i+1}]$
    $v \leftarrow (\mathbf{q}_v, t_v)$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v\}$
    **for all** $v' \in \mathcal{V} \mid t_{i-1} \leq t_{v'} \leq t_{i+2}$ **do**
      **if** $|\mathbf{q}_v - \mathbf{q}_{v'}| < \Delta \mathbf{q}_{\max} \wedge |t_v - t_{v'}| < \Delta t_{\max}$ **then**
        **if** $\forall s \in [0,1]$, $\mathbf{q}_v + s(\mathbf{q}'_v - \mathbf{q}_v) \in \mathcal{Q}_{\text{free}}$ **then**
          $w \leftarrow |t_v - t_{v'}|\, c(\mathbf{q}_v, t_v)$
          $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v, v', w)\}$
        **end if**
      **end if**
    **end for**
  **end for**
**end for**

**Splitting** In the splitting phase, we first choose a time value $t$ at which to split. The only restriction on this choice is that to approach optimality in the limit, the size of every partition

must approach 0. That is, $\Delta t_{\max} \to 0$. To perform the split, all the vertices in the time partition containing $t$ are duplicated, including their incoming and outgoing edges. All of these vertices are then assigned new time values in one of the two new partitions. Next, $\Delta \mathbf{q}_{\max}$ is updated based on the new value of $\Delta t_{\max}$, and edges longer than $\Delta \mathbf{q}_{\max}$ or that span multiple time partitions are pruned from the roadmap. Edges which violate the temporal ordering property are reoriented. Finally, all affected edge costs are recomputed.

**Algorithm 2.4** SplitRoadmap($t$, $\mathcal{V}$, $\mathcal{E}$)

Find the partition $[t_i, t_{i+1}]$ containing $t$
Insert $t$ after $t_i$ producing partitions $[t_i, t]$ and $[t, t_{i+1}]$
$\Delta t_{\max} \leftarrow \max_j (t_{j+1} - t_j)$
$\Delta \mathbf{q}_{\max} \leftarrow (\Delta t_{\max})^{1-\epsilon}$
$\{v_1, v_2, \ldots\} \leftarrow \{v \in \mathcal{V} \mid t_i \leq t_v \leq t_{i+1}\}$
$\{v'_1, v'_2, \ldots\} \leftarrow$ duplicate subgraph $\{v_1, v_2, \ldots\}$
**for** $v \in \{v_1, v_2, \ldots\}$ **do**
  $t_v \leftarrow$ arbitrary value $\in [t_i, t]$
  $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(v, v') \mid t_{v'} \geq t_{i+1} \vee |\mathbf{q}_v - \mathbf{q}_{v'}| \geq \Delta \mathbf{q}_{\max}\}$
**end for**
**for** $v \in \{v'_1, v'_2, \ldots\}$ **do**
  $t_v \leftarrow$ arbitrary value $\in [t, t_{i+1}]$
  $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(v, v') \mid t_v \leq t_i \vee |\mathbf{q}_v - \mathbf{q}_{v'}| \geq \Delta \mathbf{q}_{\max}\}$
**end for**
**for** $v \in \{v_1, v_2, \ldots\} \cup \{v'_1, v'_2, \ldots\}$ **do**
  Reorient edges to and from $v$ as required
  Recompute edge costs as when expanding
**end for**

This method frequently requires that we measure distance in configuration space, both for connecting nearby waypoints and for pruning long edges. While any distance metric could be used here, a metric which better approximates the actual cost of traversing an edge improves performance. In DGMP we use the Mahalanobis distance in motion feature space with the covariance matrix of all the motion feature vectors from all the demonstrations.

### 2.5.3 Demonstration-Guided Speedups for Motion Planning

While not strictly necessary for motion planning, we compute a *guiding path*, the trajectory which minimizes $C$ in the absence of obstacles. The configuration at time $t$ along the guiding path is given by

$$\hat{\mathbf{q}}_t = \operatorname*{argmin}_{\mathbf{q} \in \mathcal{Q}} (\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}_t^{-1} (\phi(\mathbf{q}, \hat{\mathbf{a}}) - \boldsymbol{\mu}_t). \tag{2.2}$$

Efficient methods exist for locally approximating this computation [48]. Computing the guiding path is fast and can be used to speed up computation time in several ways.

First, we implemented *seeding* which adds configurations along the guiding path to the initial roadmap. These waypoints serve as local minima in regions of configuration space that are collision-free.

Second, we use the guiding path to *bias* configuration space sampling to reduce computation time in environments with obstacles. In the expansion step of DGPRM, rather than using uniform sampling from $\mathcal{Q}$, we sample configurations based on the learned model, sampling more densely in regions of the configuration space that are more likely to result in high quality plans. Specifically, we sample $\mathbf{q}$ from a Gaussian in configuration space with mean at the guiding path configuration at that time step $\hat{\mathbf{q}}_t$. The covariance matrix is chosen to be the sample covariance of all the configurations across all the demonstrations.

In addition to the speedups based on the demonstrations, we note that our use of layers provides a speedup for problems in static environments with time dependence. When a configuration is added to the roadmap, we add corresponding waypoints and edges to all layers but only need to check for collisions once, which reduces computation time compared to prior approaches that sample in the product of configuration space and time and require collision checking for each sample.

### 2.5.4 Analysis

In this section, we will provide an outline of a proof showing that, under certain assumptions and for suitable choices of parameters, the method is guaranteed to almost surely (with probability one) converge to an optimal trajectory as computation time increases.

To simplify the analysis, we consider a modified version of the method in which a new roadmap is constructed at each iteration and time partitions are evenly distributed. In this section we assume that a minimal feasible trajectory $\Phi^*$ exists.

**Assumption 1.** *The cost metric c is Lipschitz continuous:*

$$\exists K_c \in \mathbb{R}^+, \quad \forall \mathbf{q}_1, \mathbf{q}_2 \in \mathcal{Q}, \quad \forall t_1, t_2 \in [0,1], \quad |c(\mathbf{q}_2, t_2) - c(\mathbf{q}_1, t_1)| < K_c \max(|\mathbf{q}_1 - \mathbf{q}_2|, |t_2 - t_1|).$$

This assumption holds for the DGMP cost metric if $\phi$ is Lipschitz continuous with constant $K_\phi$ and the smallest singular value among the learned covariances matrices $\sigma_{\min} > 0$. Although a tighter bound can be given, $K_c = (K_\phi^2 + 2)\sigma_{\min}^{-1}$ suffices. We note that the restriction that $\sigma_{\min} > 0$

is equivalent to assuming all of the learned covariance matrices are non-singular. Intuitively we expect this to be the case almost surely when the number of demonstrations exceeds the dimension of the feature space. However, for pedantic choices of $\phi$, this will not necessarily be the case [49].

**Definition.** *Where $\pi = (\mathbf{q}_1, t_1, \ldots, \mathbf{q}_k, t_k)$ is a path in the roadmap defined by the sequence of configurations $(\mathbf{q}_i)$ at times $(t_i)$, let $\Phi_\pi$ be the trajectory formed by linearly interpolating between configurations in $\pi$. Specifically, let $\Phi_\pi(t) = \mathbf{q}_i + \frac{t - t_i}{t_{i+1} - t_i}(\mathbf{q}_{i+1} - \mathbf{q}_i)$ where $t_i \leq t \leq t_{i+1}$. By construction of the roadmap, all $t_i$ are distinct and $\forall i < k, \quad |\mathbf{q}_{i+1} - \mathbf{q}_i| < \Delta\mathbf{q}_{\max}$ so $\Phi_\pi$ is Lipschitz continuous with $K_\Phi = \frac{\Delta\mathbf{q}_{\max}}{\min\limits_{1 \leq i < k} t_{i+1} - t_i}$. Furthermore, edges are only added to the roadmap if the line segment between the waypoints is contained in $\mathcal{Q}_{free}$, so $\Phi_\pi$ is feasible.*

**Definition.** *For a given path $\pi = (\mathbf{q}_1, t_1, \ldots, \mathbf{q}_k, t_k)$, the* weight of a path, *denoted $W(\pi)$, is given by $\sum_{i=1}^{k}(t_{i+1} - t_i)c(\mathbf{q}_i, t_i)$.*

**Lemma 1.** *For every path $\pi$ in the roadmap, $W(\pi)$ approaches $C(\Phi_\pi)$ as $\Delta t_{\max}$ and $\Delta\mathbf{q}_{\max}$ approach 0 where $\Delta t_{\max}$ denotes the length of the longest time partition and $\Delta\mathbf{q}_{\max}$ denotes the longest distance between adjacent waypoints in the roadmap.*

The proof of this lemma follows fairly simply from the observation that the Lipschitz continuity of $c$ implies that as both $\Delta\mathbf{q}_{\max}$ and $\Delta t_{\max}$ approach 0, the rectangle rule becomes arbitrarily accurate.

*Proof.* For a given path $\pi = (\mathbf{q}_1, t_1, \ldots, \mathbf{q}_k, t_k)$, the error is given by

$$\begin{aligned}
\left| C(\Phi_\pi) - \sum_{i=1}^{k}(t_{i+1} - t_i)c(\mathbf{q}_i, t_i) \right| = \\
\sum_{i=1}^{k} \left| \int_{t_i}^{t_{i+1}} c(\mathbf{q}_i + \frac{t - t_i}{t_{i+1} - t_i}(\mathbf{q}_{i+1} - \mathbf{q}_i), t) - c(\mathbf{q}_i, t_i)\,\mathrm{d}t \right| < \\
\sum_{i=1}^{k} \int_{t_i}^{t_{i+1}} K_c \max(\Delta\mathbf{q}_{\max}, \Delta t_{\max})\,\mathrm{d}t = \\
\int_{t_1}^{t_k} K_c \max(\Delta\mathbf{q}_{\max}, \Delta t_{\max})\,\mathrm{d}t
\end{aligned}$$

Thus,

$$\lim_{\Delta\mathbf{q}_{\max}, \Delta t_{\max} \to 0} \max_{\pi} C(\Phi_\pi) - C(\pi) = 0. \tag{2.3}$$

□

**Assumption 2.** *The distribution from which samples are drawn has probability density function $D \in \mathcal{Q} \to \mathbb{R}^+$ which is everywhere nonzero. Furthermore, $D$ is Lipschitz continuous with constant $K_D$.*

This assumption holds for the sampling distribution considered in Section 2.5.2 if the none of the learned covariance matrices are singular, as was required for Assumption 1 to hold for the cost metric.

**Lemma 2.** *For any error bound $\epsilon > 0$, there exist a choices of $n$ and $\Delta\mathbf{q}_{\max}$ as functions of $\epsilon$ and $\Delta t_{\max}$ such that the probability that there exists a path in the roadmap constructed from $n$ waypoints with weight less than $C(\Phi^*) + \epsilon$ approaches 1 as $\Delta t_{\max}$ and $\Delta\mathbf{q}_{\max}$ approach 0.*

The proof of this result revolves around a few key observations. First, the non-zero density of the sampling distribution implies that the probability of sampling arbitrarily close to any configuration along $\Phi^*$ approaches 1. Second, the Lipschitz continuity of $c$ implies that as samples approach the configurations along $\Phi^*$, their costs approach the costs of these configurations. Third, the Lipschitz continuity of $\Phi^*$ implies that as a path with sufficiently small time steps approaches $\Phi^*$, the cost of this path approaches the cost of $\Phi^*$. Finally, as shown in [3] the expansiveness of $\mathcal{Q}_{\text{free}}$ implies that the probability that there exists a path arbitrarily close to $\Phi^*$ in a sufficiently-connected roadmap approaches 1 as the number of samples approaches infinity.

*Proof.* For each of $k$ time partitions $[t_i, t_{i+1}]$ consider a configuration with minimal cost in the optimal trajectory within that time span. That is

$$\mathbf{q}_i^* = \min_{t_i^* \in [t_i, t_{i+1}]} c(\Phi^*(t_i^*), t_i^*).$$

The path $\pi^* = (\mathbf{q}_1^*, t_1^*, \ldots, \mathbf{q}_k^*, t_k^*)$ then satisfies $W(\pi^*) \leq C(\Phi^*)$, but this path is not necessarily in the roadmap or even feasible. However, consider $d$-balls $\beta_i$ of radius $\delta = \epsilon K_c / k$ around each configuration $\mathbf{q}_i^*$. Observe that $\forall i < k \quad \forall \mathbf{q}_i \in \beta_i \quad |c(\mathbf{q}_i^*, t_i^*) - c(\mathbf{q}_i, t_i)| < \max(\epsilon/k, K_c \Delta t_{\max})$, and for sufficiently small $\Delta t_{\max}$, $\epsilon/k$ dominates. For $\pi = (\mathbf{q}_1, t_1, \ldots, \mathbf{q}_k, t_k)$ we note that $W(\pi) < W(\pi^*) + \epsilon \leq C(\Phi^*)$.

Let $p_i$ be the probability of drawing a free sample in $\beta_i$ from the distribution, and let $D_{\min}^*$ be

the minimum probability density across $\Phi^*$, that is $D^*_{\min} = \min_{t\in[0,1]} D(\Phi(t))$. By Lipschitz continuity of $D$ and expansiveness of $\mathcal{Q}_{\text{free}}$,

$$\forall i, \quad p_i \geq F \int_0^{\min(\delta, D^*_{\min}/K_D)} S_{d-1} r^{d-1} (D^*_{\min} - K_D r)\, \mathrm{d}r.$$

Where $S_{d-1}$ denotes the surface area of the unit $(d-1)$-sphere and $F$ is a constant which depends only on the expansiveness of $\mathcal{Q}_{\text{free}}$. So for sufficiently small $\delta$ we have,

$$\forall i, \quad p_i \geq F S_{d-1} \left( \frac{\delta^d}{d} - \frac{K_D \delta^{d+1}}{d+1} \right) = \mathrm{O}(\delta^d)\ .$$

The probability of drawing at least one sample from each $\{\beta_1, \ldots, \beta_k\}$ when drawing $n$ samples is at least $P_{\min} = \prod_{i=1}^{k} (1 - (1-p_i)^{\lfloor n/k \rfloor})$. If $n$ is of order $\Omega(k^2/\delta^d)$ then $(1-p_i)^{\lfloor n/k \rfloor}$ approaches 0 with sufficient convergence rate for $P_{\min}$ to approach 1. Because time steps are evenly distributed, $k = \left\lceil \frac{1}{\Delta t_{\max}} \right\rceil$, so if $n \in \Omega(1/(\Delta t^2_{\max} \delta^d))$ then the probability of sampling waypoints which form a path with weight within $\epsilon$ of the cost of the optimal trajectory approaches 1. Expressed in terms of $\epsilon$ and $\Delta t_{\max}$ we have,

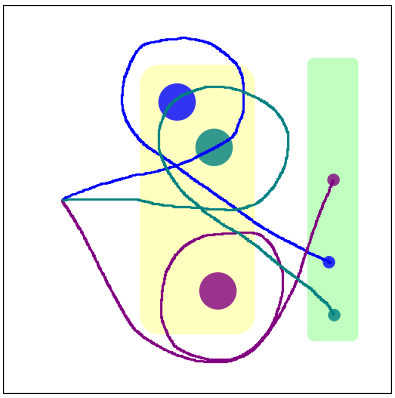
$$n = \Omega\left(\frac{k^2}{\delta^d}\right) = \Omega\left(\frac{k^{d+2}}{(K_c \epsilon)^d}\right) = \Omega\left((\Delta t^{d+2}_{\max} \epsilon^d)^{-1}\right)\ .$$

Finally, if two waypoints $\mathbf{q}_i$ and $\mathbf{q}_j$ lie within $\Delta\mathbf{q}_{\max}$ of one another, we say that $(\mathbf{q}_i, \mathbf{q}_j)$ is a *candidate edge*. By Lipschitz continuity of trajectories, notably $\Phi^*$, this will always be the case between waypoints in $\beta_i$ and $\beta_j$ if $\Delta\mathbf{q}_{\max} > K_\Phi \Delta t_{\max} + 2\delta$. Thus it suffices for $\Delta\mathbf{q}_{\max}$ to shrink more slowly than $\Delta t_{\max}$ for sufficiently small $\epsilon$ as shown by

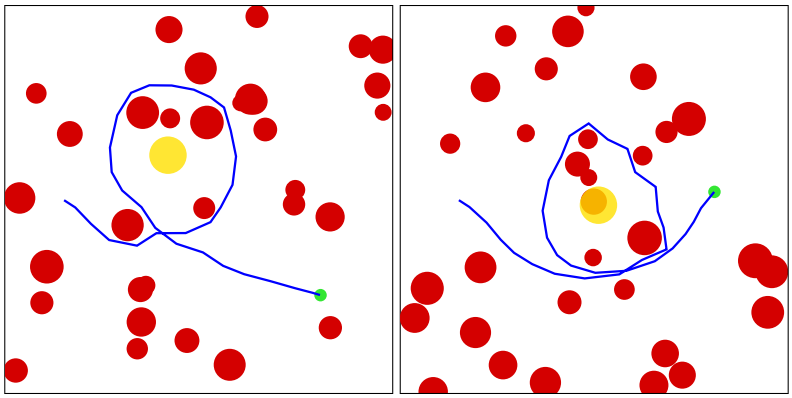
$$\Delta\mathbf{q}_{\max} > K_\Phi \Delta t_{\max} + 2\delta = K_\Phi \Delta t_{\max} + 2\frac{\epsilon K_c}{k} = K_\Phi \Delta t_{\max} + 2\epsilon \Delta t_{\max} = o\left(\Delta t_{\max}\right)\ .$$

However, a *candidate edge* will only be added to the roadmap if the line segment between the waypoints lies entirely in $\mathcal{Q}_{\text{free}}$. As was shown in length in [3], these collisions do not affect asymptotic optimality of relevant variants of PRM (e.g., PRM* and sPRM) in expansive free spaces. While these results are not reproduced here, they transfer to DGPRM. □

**Theorem 1.** *As $\Delta\mathbf{q}_{\max} \to 0$ there will almost surely exist a path $\pi$ in the roadmap, the piecewise*



(a) Three example input demonstrations (blue, violet, teal) and the sampling regions for the beacon (yellow) and goal (green).



(b) The trajectory (blue) computed by our method for two environments with obstacles (red), a beacon (yellow), and a goal (green).

Figure 2.6: Simulated 2D navigation task

*linear interpolation of which has cost* $C(\Phi_\pi)$ *arbitrarily close to that of the optimal trajectory when* $n$ *is of order* $\Omega(k^{d+2})$ *and* $\Delta t_{\max} \to 0$ *asymptotically faster than* $\Delta \mathbf{q}_{\max}$.

*Proof.* Lemma 2 shows that under these conditions, the weight of the minimum weight path $\pi$ in the roadmap will approach a cost no greater than that of optimal trajectory with probability one and Lemma 1 shows that this weight becomes an arbitrarily good approximation of $C(\Phi_\pi)$. Therefore, with probability one, $C(\Phi_\pi)$ approaches $C(\Phi^*)$. □

## 2.6 Results

We applied DGMP to a simulated 2D point robot and to the Aldebaran NAO small humanoid robot [50]. In the physical experiments, we used 6 joints of the NAO robot: 5 in the right arm and 1 at the hip. Collision detection for motion planning was done using Bullet [51] to detect intersections between a cylindrical approximation of the NAO robot's links and point cloud data obtained from a Microsoft Kinect sensor mounted next to the robot. All computation was performed on a PC with two 6-core 2.0 GHz Intel Xeon E5-2620 processors.

### 2.6.1 Simulated 2D Navigation Task

We consider a point robot that is to navigate on a 2D plane by starting at a fixed location, moving counter-clockwise completely around a beacon without intersecting it, and then stopping at a specified goal. In the learning phase, we used one configuration feature and two landmark features corresponding to the beacon and intended goal position specified by the annotations, each of which

Figure 2.7: Execution of DGMP for the powder transfer task. The robot successfully keeps the spoon level while avoiding obstacles not seen in the demonstrations.

had dimension 2. For 6 linearly independent features, the method requires at least 7 demonstrations, which we performed by manually drawing successful trajectories. For the demonstrations, we randomly sampled the beacon location from the yellow region and the goal location from the green region in Fig. 2.6(left).

In the execution phase, the test environment included 32 circular obstacles which were not present in the demonstrations. We randomly generated 20 test cases with different obstacle locations and with randomly chosen beacon and goal locations (sampled independently from the locations chosen for demonstrations but using the same regions). As can be seen in Fig. 2.6, the trajectories computed using our method consistently navigate clockwise around the beacon and reach their intended goal. DGMP was successful in all 20 test cases while the method was never successful when using Euclidean time alignment, indicating that our EM-based approach is important for learning non-trivial tasks.

### 2.6.2 Physical Task 1: Left-to-Right Powder Transfer Task

In the first physical task, the NAO robot used a spoon to transfer a powder from one container to another in the presence of obstacles as shown in Figure 2.7. In our test environment, we placed on a table an instant coffee canister, a cup, and, in some cases, other objects to serve as obstacles. The task was to scoop instant coffee using a spoon and transfer it to the cup without spilling coffee or displacing any objects on the table.

We evaluated DGMP for scenarios in which the coffee canister was always on the right side of the table and the cup was always on the left. This is a simplified version of the general task in which the coffee canister and cup can be anywhere on the table, which will be discussed in Section 2.6.3. We conducted 7 kinesthetic demonstrations for this task, and the locations of the objects on the table were randomized for each demonstration. We drew the positions of the coffee canister and cup

from uniform distributions based on 4 inch line segments on the left and right sides, respectively, of the reachable surface of the table.

In the learning phase, we used the configuration motion feature defined by the robot's 6 DOF as well as landmark-based motion features based on the sensed locations of the two task-relevant objects, the canister and cup. For the landmark-based motion features, we considered two points on the robot's end effector (the spoon): the top and bottom surface of the tip of the spoon. (We note that at least two points are required to learn end effector orientations.) We enforced independence between the configuration and each of the landmark-based motion features, as described in Section 2.4.3, so that the largest dependent block in $\mathbf{\Sigma}$ was $6 \times 6$. Other than the demonstrations, we did not provide the method any input regarding task constraints; e.g., we never explicitly expressed the constraint that the spoon must be level. The learning phase took 2.5 seconds of computation time.

We then created 20 test cases in which the locations of the coffee canister and cup were drawn randomly from the same distribution as the demonstrations. We also placed a bottle on the table as an obstacle at a position drawn uniformly from the reachable surface of the table. The bottle was sufficiently tall that it created a narrow passageway in the NAO's feasible configuration space when the NAO attempted to carry a level spoon over it. A test case was considered *successful* if the robot (1) scooped coffee from the canister and transferred it to the cup without spilling, and (2) did not displace the obstacle, canister, or cup. We considered a test case to be *feasible* if it was possible for the robot to successfully accomplish the task given its kinematic limitations. Of the 20 test cases, 4 were not feasible due to the obstacles being too close to the coffee canister or cup and the robot not having sufficient range of motion. We report statistics for the 16 feasible test cases.

As shown in Figure 2.8, the robot running DGMP successfully accomplished the task in 14 of the 16 feasible test cases. The two failures were both due to the Kinect sensor failing to properly sense the extent of the bottle. We also evaluated DGMP using the demonstrations aligned using a simple Euclidean cost metric in configuration space which only considers similarity in joint angles when aligning demonstrations (as in most prior work). Because this Euclidean cost metric does not depend upon the task model, there is no need for EM. Our results show that this approach is ineffective for this task and succeeded in only 9 test cases, indicating that time-alignment that explicitly considers the task model as in the full DGMP approach is beneficial to task success. We also executed the guiding path without motion planning, which resulted in only 8 successful runs

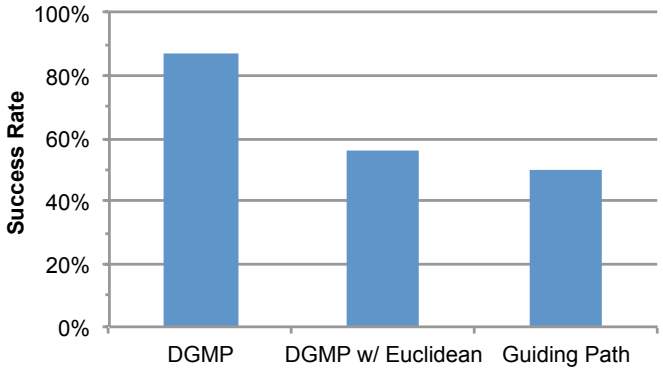



Figure 2.8: The performance of DGMP on the left-to-right powder transfer task. We also evaluate the performance of DGMP using Euclidean time alignment (rather than maximizing learned likelihood) and also executing the guiding path (without sampling-based motion planning).

due to collision with obstacles.

### 2.6.3 Physical Task 2: General Powder Transfer Task

We also evaluated DGMP on a more difficult variant of the powder transfer task in which the coffee canister and cup were each permitted to be anywhere on the reachable surface of the table, approximated by a rectangular region spanning 7 inches left to right and 4 inches front to back. This meant that the robot's motions were no longer strictly following a left-to-right trajectory, and thus were more difficult to align. We performed 20 new kinesthetic demonstrations with the coffee canister and cup positions drawn uniformly from the reachable table surface. After completing the demonstrations, the learning phase took 2.6 seconds of computation time. We then created 20 test cases, drawing coffee canister, cup, and bottle obstacle positions randomly from the reachable table surface. Of the 20 test cases, 17 were feasible.

When employing all 20 demonstrations, DGMP succeeded in 16 of the test cases, resulting in a success rate of 94% of the 17 feasible test cases (see Figure 2.9). In the one failure case, the obstacle was very close to both the coffee canister and cup, which resulted in a narrow passage in the robot's configuration space that was too narrow for the planner to find a feasible plan in the maximum time allotted (20 seconds). We also evaluated the performance of DGMP for different numbers of demonstrations. When fewer demonstrations are used, the performance of the method degrades gracefully, with a greater than 80% success rate even for just 7 demonstrations.

To illustrate the need for motion planning for this scenario, we also executed the guiding path with no motion planning, resulting in a success rate of under 60%. We also evaluated DGMP using

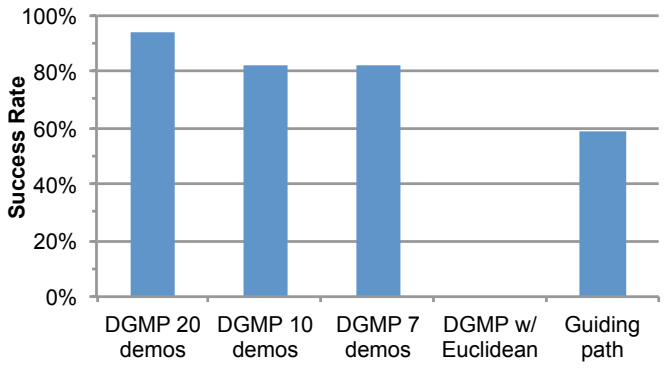



Figure 2.9: The performance of DGMP on the general powder transfer task when 20, 10, and 7 demonstrations are provided to the learning phase. We also evaluated the performance of DGMP using Euclidean time alignment (rather than maximizing learned likelihood) and the guiding path (without sampling-based motion planning). DGMP performed better with more demonstrations, but still exceeded an 80% success rate when only 7 demonstrations were provided.

the demonstrations aligned using the simplified Euclidean distance metric and achieved a success rate of 0%. This highlights the benefit of aligning demonstrations by maximizing log-likelihood using our EM-based approach rather than by using the more traditional Euclidean metric, which fails to properly align demonstrations in which the direction of end effector motion varies substantially across demonstrations. A video of a NAO robot performing this task using DGMP is available at: `http://robotics.cs.unc.edu/DGMP2`.

To illustrate the impact of each component of the DGMP framework, we executed different variants of the motion planner and plot in Figure 2.10 the cost of the computed plan based on the learned DGMP metric. Each curve is the average of 6 runs for the same randomly selected test case. As expected, allowing more computation time results in lower cost plans. As described in Section 2.5, the DGMP cost metric can be used with either PRM or RRG against which we will compare our proposed extensions that accelerate performance by biasing sampling based on the learned metric, seeding along the guiding path, and incorporating layers. DGPRM and DGRRG, which both include all the speedups, are roughly equivalent for this application. We also show DGPRM with some of its components removed (i.e., removing speedups gained by layers and/or seeding). The results show that the biggest speedup in DGPRM comes from biasing configuration samples based on the learned cost metric during roadmap expansion. DGPRM and DGRRG are both over 20 times faster than the traditional PRM algorithm for plans of equivalent cost.

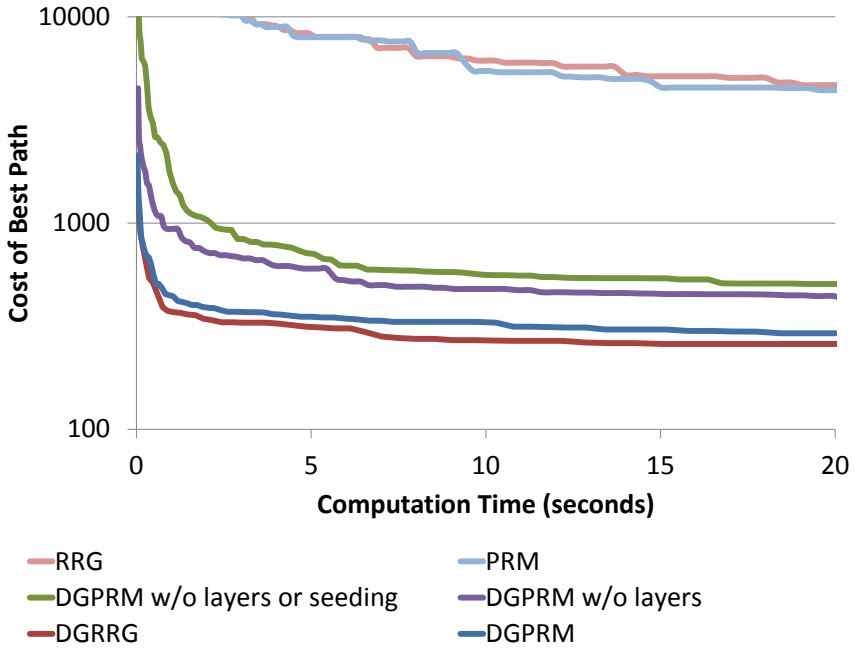



Figure 2.10: The cost of the best solution found as a function of roadmap planning time for different variants of the method applied to the general powder transfer task. Note the logarithmic scale on the vertical axis.

### 2.6.4 Physical Task 3: Push a Button

We also considered the task of pressing a button, where the button may be positioned on a table, a slanted surface, or even a vertical wall. To correctly perform the task, the robot needed to learn how to push a button in any of these orientations from the same set of demonstrations. Furthermore, additional obstacles were introduced into the execution environment.

To train the method, we performed 9 demonstrations of pressing a 3 cm diameter button. In 3 of these demonstrations, the button was placed randomly on the reachable surface of a table; in another 3, the button was randomly placed on the reachable surface of a plane inclined 40 degrees; and in the final 3, the button was randomly affixed to the reachable surface of a vertical wall in front of the robot.

The motion features we used were the 6 joint angles of the robot's right arm and hip and the 3-dimensional positions of both the hand and finger relative to the pose of the button. As before, we enforced independence in the covariance matrix between the configuration motion feature and each of the two landmark-based motion features. The learning phase took 0.9 seconds of computation

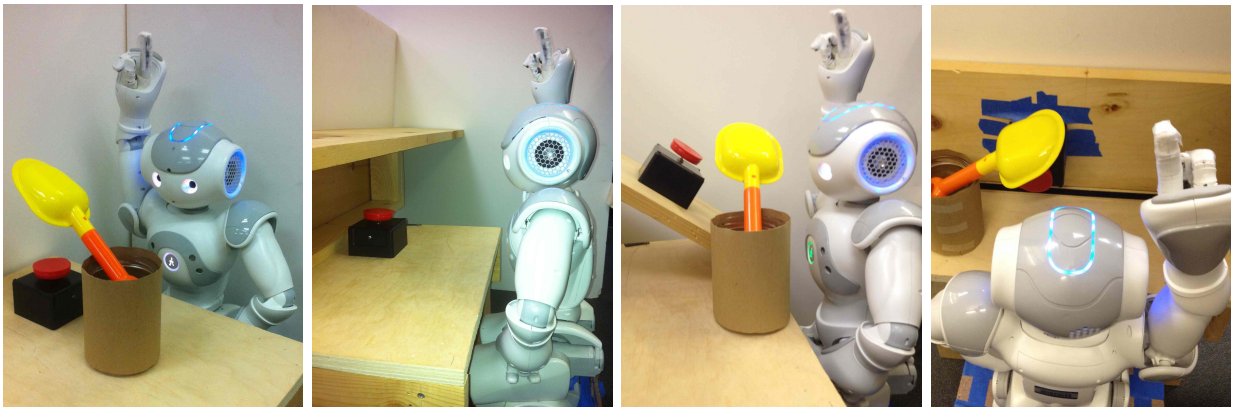


Figure 2.11: Scenarios for the button pushing task.

time.

To test the method, we considered 4 scenarios, and performed 3 random tests on each for a total of 12 test cases. The scenarios, shown in Figure 2.11, included (1) placing the button and a non-convex obstacle randomly on the reachable surface of the table, (2) placing a shelf over the table and placing the button randomly on the table under the shelf, (3) placing the button randomly on a plane inclined at 40 degrees and placing a non-convex obstacle randomly beside the inclined plane such that it hung over the inclined plane, and (4) placing the button on a vertical wall and placing a tall obstacle randomly on the surface of the table.

In Figure 2.12 we show the rapid convergence of DGPRM compared to regular RRG and PRM without layers or accelerations based on the learned task model. While DGRRG did find lower cost paths for the same number of samples, we see in the figure that DGPRM performed better because each sample could be generated more quickly due to greater opportunities for parallel execution. We believe this is because the biased sampling distribution derived from the demonstrations largely subsumes the role of RRG's roadmap expansion approach in effectively biasing samples towards the relevant portions of the configuration space.

Figure 2.13 shows the execution of a DGMP plan. We considered an execution successful if it avoided obstacles and depressed the button. DGMP succeeded in 11 of the 12 test cases, a greater than 90% success rate. The sampling-based motion planner was crucial to success in this task as the guiding path was successful in only 1 of the 12 tests cases.

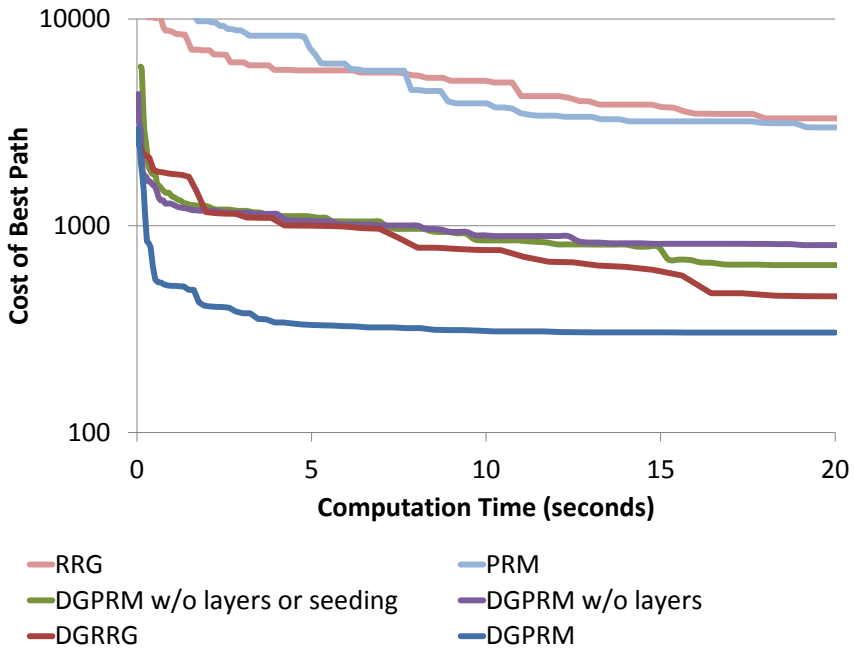



Figure 2.12: The cost of the best solution found as a function of roadmap planning time for different variants of the method applied to the button pushing task.

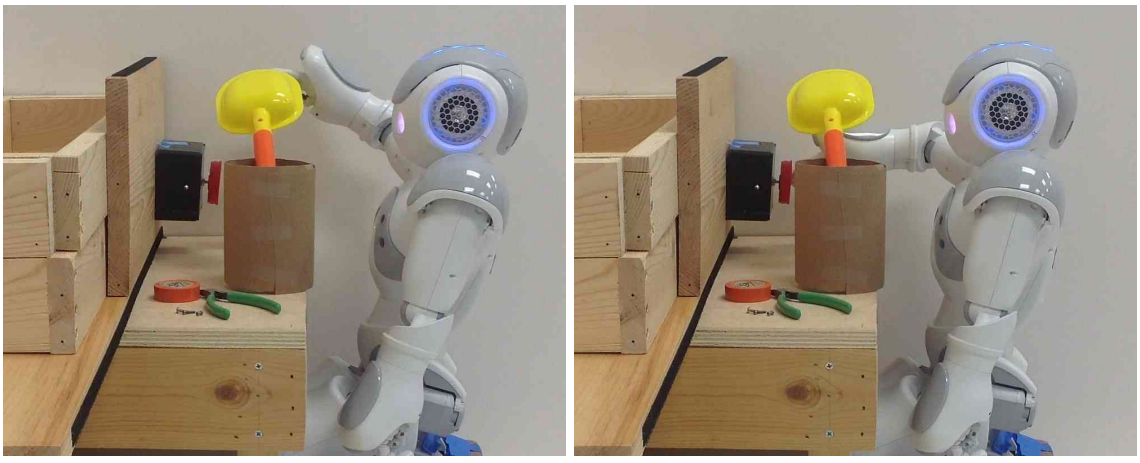


Figure 2.13: Execution of a DGMP plan for one of the button pushing scenarios.

## 2.7 Conclusion

This chapter presented demonstration-guided motion planning (DGMP), a framework for planning motions for assistive robots to perform tasks in unstructured environments such as homes or offices. DGMP combines the strengths of demonstration-based learning and sampling-based motion planning to generate motion plans that (1) aim to satisfy learned features of the motion that are required for the task to be successfully accomplished and (2) avoid obstacles in unstructured environments. We used kinesthetic demonstrations and statistical modeling methods to learn a time-dependent cost metric that encodes features of a task's motion that are consistent across the demonstrations and, hence, are likely required to successfully execute the task. We formalized the cost metric as a Mahalanobis distance between a planned trajectory and the distribution of demonstrations in a feature space parameterized by the locations of task-relevant objects. Our asymptotically-optimal sampling-based motion planner computed plans that simultaneously avoid obstacles and asymptotically globally minimize the learned time-dependent cost metric. The planner also leveraged the demonstrations to significantly reduce motion plan computation time. We showed the effectiveness of combining learning with sampling-based motion planning on the NAO robot performing assistive tasks. In the following chapter, we will extend this approach to dynamic execution environments and the Baxter robot, which has a higher-dimensional configuration space.

CHAPTER 3

## Reactive Replanning for Learned Robotic Tasks

In this section, we adapt the motion planning approach of the previous chapter to execute in a closed-loop manner, enabling the robot to be responsive to the motion of task-relevant objects and obstacles which should be avoided. We model the task in the same manner described in Chapter 2 to learn consistent features of the demonstrations, like the orientation of the spoon in Figure 3.1 while it contained powder. During execution, our motion planner uses this learned model to define a cost function which is minimized by an interactive-rate motion planner to reproduce motions that are consistent with the demonstrations. We leverage a sampling-based motion planner which uses the model to guide sampling toward low-cost regions (in terms of the cost metric from Chapter 2) by incorporating the positions of task-relevant obstacles. The solutions found by the motion planner asymptotically approach global optimality, in contrast to approaches that only guarantee local optimality and thus may become caught in the basin of attraction of a bad solution. We search for plans in a roadmap defined by the Cartesian product of the learned task model, represented as a hidden Markov model, and a probabilistic roadmap over the robot's configuration space.

To consider the movement of task-relevant objects, we continuously replan by rebuilding the roadmap and then searching for a new plan. This replanning approach computes plans that avoid obstacles while explicitly considering task-relevant objects based on the current state of

Figure 3.1: As the Baxter robot performs the learned task of transferring powder from the yellow bucket to the green bowl using the blue spoon, a person moves the green bowl. Our method automatically replans in a closed-loop manner, enabling the robot to avoid obstacles and perform the learned task even when task-relevant objects or obstacles are moved mid-task.

the environment. The robot replans in real-time, averaging more than 5 plans per second in our experiments, by leveraging information in the task model and using appropriate data structures and algorithms. First, we employ a sampling distribution biased toward low-cost regions of configuration space to produce a small, high-quality roadmap. Second, we use a parallel bidirectional search over an implicit graph combined with lazy computation to quickly and globally search for optimal plans.

This chapter is based on work previously published in [17] and [18].

### 3.1 Related Work

Sampling-based methods have been highly successful for computing feasible (and optimal) motion plans for a wide variety of robots, including manipulators with many degrees of freedom [52, 15, 3]. While most sampling-based motion planners that consider optimality aim to minimize metrics such as Euclidean distance in the workspace or configuration space, some methods have investigated incorporating more general task-based cost functions. Several approaches are based on rapidly exploring random trees (RRTs) [52], a highly successful sampling-based method for computing feasible, obstacle-avoiding trajectories. Transition-based RRT (T-RRT) [22] biases expansion of an RRT to low cost regions of the configuration space cost map, and Mainprice et al. used T-RRT to generate natural motions based on a predefined cost map for human robot interaction [23]. RRTs have also been used in conjunction with analytically-defined task constraints [24] and with symbolic representations of manipulation strategies [25]. Sampling-based motion planners have also been extended to integrate motion constraints and properties learned from demonstrations. Claasens extended RRT to sample only inside a user-specified number of standard deviations of a mean demonstrated trajectory [26] and later within learned affordances [53], Berenson et al. integrated local optimization with an RRT to find low-cost paths over cost maps [27], Scholz et al. incorporated gradient descent into an RRT to locally optimize a specified objective function [28], and Şucan and Chita investigated sampling strategies that enforce constraints [29]. Finally, RRTs have been used for replanning when the environment changes [54, 55, 56]. However, plans produced by RRT methods like those above are almost surely suboptimal, even as computation time approaches infinity. This limitation also applies to some roadmap methods [47], like elastic roadmaps [57], which have been applied to the problem of motion planning in the presence of soft constraints.

We employ an asymptotically optimal sampling-based motion planner, meaning the computed

plan is guaranteed to approach a globally optimal plan (based on the given cost metric) as the number of iterations is allowed to increase. This property is important even when only considering finite planning times because as computational power grows, the probability of finding a near-optimal plan within that deadline approaches one. Towards this end, Karaman and Frazzoli proposed asymptotically optimal motion planning algorithms such as RRT*, RRG*, and PRM* that guarantee asymptotic optimality [3]. Asymptotically optimal motion planners avoid the suboptimal or even infeasible plans resulting from local minima that can occur when using potential field methods [15] or local trajectory optimizers [58]. Trajectory sampling has been used to address these shortcomings [59], including with a focus on dynamic environments [60], but does not guarantee asymptotic optimality without also incorporating roadmap- or tree-based methods [61]. Related work has investigated asymptotically optimal planners that balance exploration and refinement [30], asymptotic near optimal planners using smaller roadmaps [31], the near-optimality of solutions in finite time [62], and anytime solution optimization [32].

Our method integrates a learned hidden Markov model (HMM) representing a task with a sampling-based motion planner to guarantee asymptotic optimality. HMMs have previously been applied to motion recognition (e.g., [43, 5, 6]) and generation (e.g., [7, 6]). However, prior approaches for motion generation, unlike our proposed method, do not simultaneously guarantee global optimality while enabling fast replanning. Motion planners that operate entirely on constraint manifolds have also been developed [63, 64], even while retaining asymptotic optimality [65, 66]. However, statistical task models learned from demonstrations like the HMM we consider naturally encode uncertainty and formulating hard constraints discards this information. Model predictive control approaches have also considered time-varying cost functions and may use sampling to avoid obstacles [67]. A number of methods learn high level tasks from demonstrations, e.g. [68, 69]. For execution, these methods may use visual servoing in conjunction with subtask-specific motion planners [70] or more general constraints [71]. Rather than competing with these methods, our approach is more appropriate for learning and executing the subtasks on which such methods rely. These subtasks could then be combined to solve more complex tasks using general frameworks based on generic task planners [72], SMT solvers [73], or geometric backtracking [74]. Dynamical systems have also been used to learn such motions from demonstrations, e.g. [75], [76], and [77]. These systems have further been extended to avoid obstacles in [10], but this approach retains no notion

of global optimality.

Prior work has also considered pouring tasks similar to one we consider here. An SVM classifier learned from 170 demonstrations has been shown to enable execution in dynamic environments [78], albeit without obstacles. When the task is known rather than learned, the physical constraints of the task can often be exploited, and specialized methods have been developed for liquid pouring. These methods have used physical reasoning [79] or fluid simulation [80] to model the problem, even combined with trajectory optimization to enable obstacle avoidance [81]. By incorporating more prior knowledge rather than learning the task from demonstrations, these approaches have the potential to be more robust, but at the cost of generality.

## 3.2 Method Overview

### 3.2.1 Problem Statement

Let $\mathcal{Q} \subset \mathbb{R}^d$ be the $d$-dimensional configuration space of a holonomic robot. Let $\mathcal{Q}_{\text{free}} \subseteq \mathcal{Q}$ denote the subset of the configuration space for which the robot is not in collision with an obstacle in the current execution environment. We again assume the robot is capable of sensing the positions of $K$ task-relevant objects, called *landmarks* (such as the green bowl in Figure 3.1) and a model of the environment against which we can test for collisions with obstacles. In our physical experiments, both landmark and obstacle sensing rely on a Kinect RGBD sensor.

As in the previous chapter, during execution (described in Section 3.3), our objective is to compute a trajectory $\Phi$ in the robot's configuration space from a start configuration $\mathbf{q}_{\text{start}} \in \mathcal{Q}_{\text{free}}$ to a goal configuration $\mathbf{q}_{\text{goal}} \in \mathcal{Q}_{\text{free}}$ such that the trajectory (1) avoids obstacles in the robot's current environment and (2) successfully accomplishes the task, which may depend on task-relevant objects in the environment. The robot continually senses its environment to collect sufficient information to perform collision detection and to compute costs in the current environment based on the learned task model: all the information necessary to compute collision-free motion plans based on the learned costs. In contrast to the prior chapter, we then enter a closed loop; the robot executes the current plan while, repeatedly replanning based on the learned costs as task-relevant objects and obstacles might move in the environment.

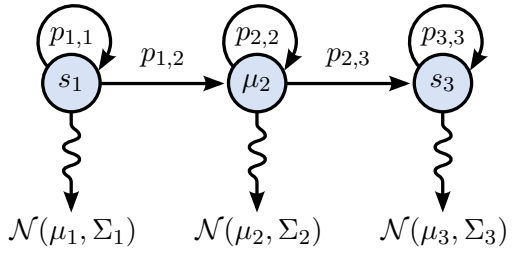



Figure 3.2: Hidden Markov model with 3 discrete states (corresponding to time steps of the task) and normally distributed observations.

### 3.2.2 Learned Task Model

To learn a task, we consider a reformulation of the method originally presented in [16] (see Chapter 2), wherein a task was learned from a set of demonstrations of a human kinesthetically guiding the robot through the task. This is in contrast to methods that attempt to model the effect on the environment of the robot's actions (e.g. [81]), effectively augmenting the state space of the robot. Rather, we try only to mimic the motion of the human demonstrator while accounting for the current state of the environment, which while less accurate is also less task-specific.

For each demonstration, we record a sequence of configurations evenly-spaced in time as well as an annotation $\mathbf{a}$ that encodes the poses of task-relevant objects during the demonstration. The recorded robot configurations are then lifted via a function $\phi_{\mathbf{a}}(\mathbf{q})$ from configuration space into a motion feature space $\mathcal{Y} \subseteq \mathbb{R}^f$ comprising the robot's end-effector pose relative to task-relevant objects in the environment. We then time-align the demonstrations (using the Viterbi algorithm) and use these aligned demonstrations to estimate a sequence of multivariate normal distributions $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ in the motion feature space. For full details on this process and variations on it, see Chapter 5.

In the remainder of this dissertation, for ease of integration with a constantly updating motion planning roadmap, we reformulate this approach as learning a time-homogeneous hidden Markov model (HMM) with a restricted structure (see Figure 3.2). We assume the model has $T$ discrete, sequential states, wherein each state $t \in \{1, \ldots, T\}$ has nonzero transition probabilities only to itself and the next state $t+1$. This imparts a total ordering on the model corresponding to time. As such, we refer to these states as *time steps*. For our experiments, we let the observed outputs in each state $t$ be distributed according to a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ in motion feature space. We learned the parameters of this model using the Viterbi path-counting algorithm [82].

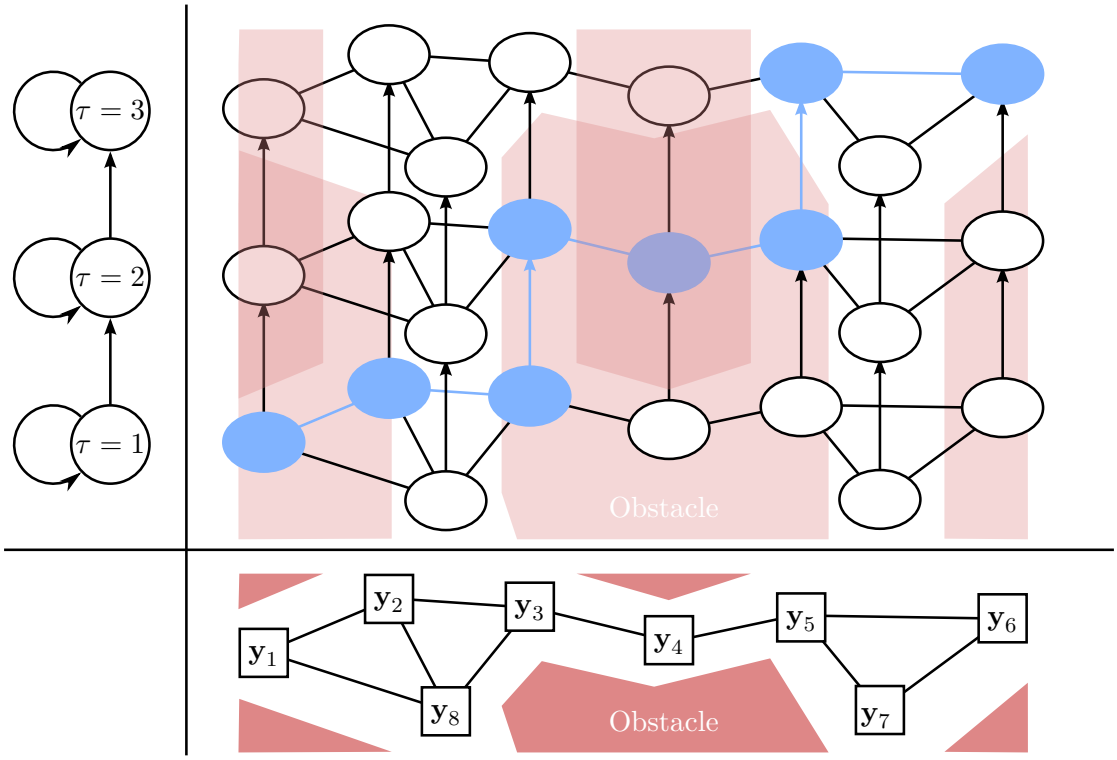



Figure 3.3: Cartesian product of the configuration space roadmap and the graph of the learned task model.

This learned model will be used to compute a cost function for motion planning during task execution. The cost function is parameterized by the annotation $\mathbf{a}$, which will vary across execution environments as well as during execution when a task-relevant object moves. The cost function [83] is defined such that a trajectory which minimizes it, maximizes the likelihood in the learned model, and thereby should successfully perform the task in the execution environment with high probability.

### 3.3 Closed-loop Replanning with Learned Costs

In each replanning step, our method builds a spatiotemporal roadmap in which the edge costs are set based on the learned task model and execution environment. We perform a shortest path search in this roadmap to update the execution paths. We now describe each of the components of the closed-loop motion planning approach.

### 3.3.1 Spatiotemporal Roadmap

We employ an asymptotically optimal variation of a probabilistic roadmap with a fixed connection radius (sPRM) [47], which we chose to adapt because it produces globally optimal plans to within the roadmap resolution. A roadmap is a graph in which vertices represent the states of the robot and edges represent feasible local plans between these states. In the simplest case these local plans are just straight line trajectories in configuration space.

A traditional roadmap is undirected and is constructed as follows. First, $n$ configurations $\{\mathbf{q}_0, \ldots, \mathbf{q}_n\}$ are randomly sampled from $\mathcal{Q}_{\text{free}}$ via rejection sampling. Then, edges are constructed between all configurations $\mathbf{q}_i$ and $\mathbf{q}_j$ for which $\|\mathbf{q}_i - \mathbf{q}_j\|_D < \epsilon$ if a feasible local plan can be found. We construct exactly such a roadmap, which we will call the *spatial roadmap*.

To accommodate dependence on the time step in the learned task, we also define a *temporal roadmap*. This is not a roadmap in the traditional sense, but rather the graph representation of the Markov chain implied by the task model, where time steps correspond to vertices in the graph. Because the transitions in the Markov chain are directed, the temporal roadmap is a directed graph.

Finally, we define a *spatiotemporal roadmap*, a directed graph that combines the information in the spatial and temporal roadmaps. The vertices of the spatiotemporal roadmap are each defined by a pair composed of a vertex from the spatial roadmap and a vertex from the temporal roadmap. The set of edges are given by the vertex-wise union of edges in the spatial and temporal roadmaps (see Fig. 3.3). Put another way, the spatiotemporal roadmap is the Cartesian product of the spatial and temporal roadmaps. Such a roadmap is necessary because the state of the robot needs to incorporate the task progress (see Sec. 3.4.1).

This full roadmap can be quite large, but because of its regular structure, it need not be explicitly constructed. Rather, we can implicitly traverse it by keeping track of vertices in the constituent roadmaps. This approach has multiple advantages over directly sampling in the product of configuration space and time. First, because the full roadmap does not need to be explicitly constructed, it uses less memory and exhibits better locality of reference. Second, because every vertex and edge in the spatial roadmap is effectively duplicated across all time steps, fewer collision queries are required. Finally, small additions to either constituent roadmap are immediately reflected as larger additions to the full roadmap without the need to perform an explicit construction, making updates fast. As an aside, we also experimented with the tensor product of the roadmaps, which

while more natural, had many more edges and adversely impacted search performance.

### 3.3.2 Cost Function

In a traditional roadmap, the goal is to find shortest paths, so edges in the roadmap are assigned costs based on the lengths of the local plans they represent. We wish to find paths which are most likely to successfully perform the task, so choose edge costs based on the learned task model. Specifically, we define a notion of cost which, when minimized, maximizes probability of successfully executing the task as defined by the learned model.

By the construction of the Cartesian product, each edge is from a configuration $\mathbf{q}$ and time step $t$ to another configuration $\mathbf{q}'$ and time step $t'$. Our definition of edge cost will be based on the negative log probability of entering time step $t'$ and observing configuration $\mathbf{q}'$ after being in time step $t$ and observing configuration $\mathbf{q}$. The Markov assumption inherent in the task model implies that the configuration and subsequent time step depend on the current time step. This allows the following simplification in the edge cost computation:

$$\begin{aligned} -\log(p(\mathbf{q}', t' \mid \mathbf{q}, t)) &= \\ -\log(p(\mathbf{q}' \mid t') \cdot \mathrm{P}(t' \mid t)) &= \qquad\qquad \text{(Markov Property)} \\ -\log p(\mathbf{q}' \mid t') - \log \mathrm{P}(t' \mid t) & \end{aligned}$$

where the probability $\mathrm{P}(t' \mid t)$ is given by $p_{t,t'}$ and the probability density $p(\mathbf{q}' \mid t')$ by

$$|2\pi\Sigma_{t'}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\phi_{\mathbf{a}}(\mathbf{q}')-\mu_{t'})^{\mathrm{T}}\Sigma_{t'}^{-1}(\phi_{\mathbf{a}}(\mathbf{q}')-\mu_{t'})}.$$

We consider the log probability because the shortest path algorithm will minimize the *sum* of the edge costs while joint probabilities (under independence assumptions) are multiplicative, as seen above. We then negate the result to formulate the problem as one of minimization. However, recall that edges in the spatial roadmap represent local plans. So for such edges, we use the line integral of the negative log probability along the local plan $\mathbf{q}(s)$, yielding:

$$\mathrm{cost}(\mathbf{q}(\cdot), t, t') = -\log \mathrm{P}(t' \mid t) - \int_0^1 \log p(\mathbf{q}(s) \mid t')\mathrm{d}s =$$

$$-\log p_{t,t'} - \frac{1}{2}\int_0^1 \log m \log(2\pi) + \log\det\Sigma_{t'} + (\phi_{\mathbf{a}}(\mathbf{q}') - \mu_{t'})^{\mathrm{T}}\Sigma_{t'}^{-1}(\phi_{\mathbf{a}}(\mathbf{q}') - \mu_{t'})\mathrm{d}s$$

where the local plan is parameterized in the interval $s \in [0, 1]$.

The following assumes the robot has velocity upper bounded by $\upsilon$, in that $|\nabla_s \mathbf{q}(s)| \leq \upsilon$. When the integrand has a strictly negative upper bound $\epsilon$, which can be ensured by appropriate choice of model like the one we use, asymptotic optimality is guaranteed by the sPRM method [3] because a fixed connection radius $r$ using any $p$-norm is a superset of some other fixed radius using this cost metric. This can be seen by observing that a d-ball $\beta_r$ centered at $\mathbf{q}_0$ includes those configurations with costs from $\mathbf{q}_0$ not greater than $\frac{r\epsilon}{\upsilon} - \log p_{t,t'}$ for all $t, t'$. But a d-ball corresponds to a fixed radius under a 2-norm, and all $p$-norms are equivalent in finite-dimensional spaces [84].

Given this formula for edge costs, the full motion planning problem can be formulated as a constrained optimization problem as follows:

$$\begin{aligned}
\min_{S_0 \leq S_1 \leq \ldots \leq S_T,\, \mathbf{q}(\cdot) \in [0, S_T] \to \mathcal{Q}} & \sum_{t=1}^{T} \left( (\log p_{t,t} - \log p_{t-1,t}) - \int_{S_{t-1}}^{S_t} \log p_{t,t} + \log p(\mathbf{q}(s) \mid t)\mathrm{d}s \right) \\
\text{s.t. } & S_0 = 0 \\
& \mathbf{q}(0) = \mathbf{q}_0 \\
& \forall s \leq S_T.\ \mathbf{q}(s) \in \mathcal{Q}_{\text{free}}
\end{aligned}$$

where the path $\mathbf{q}(\cdot)$ is subject to the kinematic and dynamics constraints of the robot.

Thus, under the independence assumptions of our model, we are minimizing negative log probability over the space of time-step-augmented trajectories. Formally, the learned task model implies a distribution over trajectories (and latent variables $S_0, S_1, \ldots$). By defining edge costs as we do, we are able to use a motion planner to (asymptotically) find the *mode* of this distribution (the trajectory with maximum probability density) restricted to feasible motions.

### 3.3.3 Biased Sampling

Although asymptotic optimality ensures that the method converges to the optimal solution, this convergence may be too slow to achieve reactive execution. To accelerate this convergence, when sampling configurations to add to the roadmap we use a biased sampling distribution rather than a uniform one. We construct the distribution in such a way that asymptotic optimality is retained while producing samples which produce lower cost edges and are thus more likely to be useful. This

Figure 3.4: End-effector positions of sampled configurations (shown in blue) from the biased distribution for the task of scooping powder from the yellow bucket into the green bowl. The distribution is dependent on the landmark positions. Sampling configurations in useful places enables us to build higher quality roadmaps with fewer samples, which facilitates faster motion planning.

takes the place of the guiding path considered in [83].

The intuition behind the distribution we construct is that when lifted into motion feature space, it should approximate the distribution implied by the task model, conditional on the poses of landmarks. Formally, we wish to sample a sequence $\{\mathbf{q}_0, \ldots, \mathbf{q}_n\}$ such that $\phi_{\mathbf{a}}(\mathbf{q}_i) \sim \mathcal{N}(\boldsymbol{\mu}_{t_i}, \boldsymbol{\Sigma}_{t_i})$ where $t_i \sim \mathcal{U}(0, T)$. Recall that $\phi_{\mathbf{a}}$ lifts a configuration into motion feature space given landmark poses $\mathbf{a}$.

To approximate this distribution, we work backwards. First, we sample $t_i \sim \mathcal{U}(0, T)$, followed by $\mathbf{y}_i \sim \mathcal{N}(\boldsymbol{\mu}_{t_i}, \boldsymbol{\Sigma}_{t_i})$, then solve the following non-linear least squares problem using a Levenberg-Marquardt method:

$$\mathbf{q}_i = \arg\min_{\mathbf{q}} (\phi_{\mathbf{a}}(\mathbf{q}) - \mathbf{y}_i)^{\mathrm{T}} \boldsymbol{\Sigma}_{t_i}^{-1} (\phi_{\mathbf{a}}(\mathbf{q}) - \mathbf{y}_i).$$

This yields a configuration which when lifted into motion feature space is close to $\mathbf{y}_i$ in feature space under the metric implied by $\boldsymbol{\Sigma}_{t_i}$. An example of the resulting distribution for one of the experimental scenarios is shown in Fig. 3.4. Note that the distribution appears to follow the path of a successful execution and has greater variance where deviation would likely preserve plan success. For non-singular $\phi_{\mathbf{a}}$ and $\boldsymbol{\Sigma}_t$, the support of this distribution is all of $\mathcal{Q}$, so asymptotic optimality

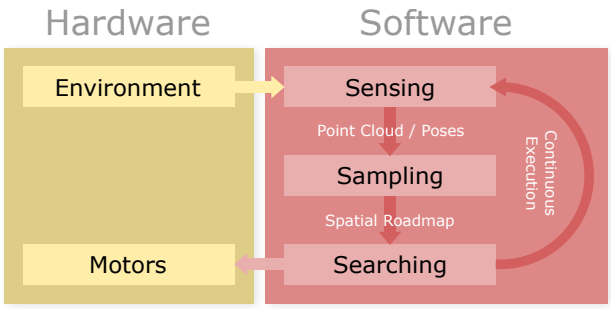



Figure 3.5: Flow of computation during real-time execution.

holds.

In practice, the initial iterate can be sampled uniformly at random, and the least squares problem need not be solved to great precision. We found that even very few iterations of the Levenberg-Marquardt method result in distributions that sufficiently approximate the desired one.

### 3.3.4 Real-Time Execution

During execution, we repeatedly replan using the latest sensed information about the environment in a closed-loop manner, updating the path being executed with high frequency. In Fig. 3.5 we show a schematic of the computation during real-time execution. Replanning comprises three logical steps: *sensing*, *sampling*, and *searching*.

**Sensing** We refer to sensing as the general process of acquiring information about the environment used by the planning procedure. In our physical experiments, sensing consists of processing the raw point cloud from an RGBD sensor to locate task-relevant objects and build a collision model of the environment. These computations are performed asynchronously, ensuring that after each planning iteration, new environment information is available to immediately begin replanning.

**Sampling** Sampling refers to the first step of building the spatial roadmap (as described in Section 3.3.1), which is randomly sampling configurations. Like prior work [16, 17], we use a fixed connection radius (sPRM) rather than a shrinking one (PRM*) to retain asymptotic optimality [3] under the cost metrics we consider (by metric equivalence). However, in contrast to prior work, our method builds a roadmap from scratch each cycle and does not reuse roadmaps from previous cycles. The rationale for this is twofold. First, in our problem obstacles may move, so the validity of vertices and edges in the roadmap need to re-evaluated during each replanning step. This greatly

reduces the benefit of retaining the previous roadmap. Second, the sampling distribution depends on the environment, yielding samples that are appropriate for a specific set of landmark poses. This not only means that the samples based on current landmark poses are more useful than those from prior work, but that samples based on older landmark poses are *less* useful. Consequently, the benefit of building off the previous roadmap is outweighed by the cost incurred by the corresponding increase in roadmap size.

**Searching** After the spatial roadmap is constructed, we perform a parallel bidirectional shortest-path search [85] on the spatio-temporal roadmap. Many calculations are performed lazily as edges of the graph are explored, including collision detection and edge cost evaluation. This is beneficial because fewer than 40% of the edges needed to be evaluated in order to identify the shortest path in our experiments, a positive consequence of the cost-space chasm induced by the task model.

Unlike many methods for real-time planning or control, our approach is globally optimal in that it selects the true minimal cost path present in the roadmap. Consequently, our method may consider multiple homotopic classes of paths and can avoid being trapped in a local minimum. Because we use an asymptotically optimal PRM variant, our approach is also asymptotically optimal in the sense that the plans produced approach optimality as the size of the roadmap used increases. While we only build relatively small roadmaps, asymptotic optimality of the planner allows the method to find better solutions, approaching the global optimum, as computational power increases. Our execution is real-time in the sense that we impose a firm deadline on the planner, from sensing to execution, to ensure the robot is never acting on sensor information that is excessively out of date. A missed deadline manifests as a pause while the robot waits for a new plan to be sent to the motors for actuation. If the replanning cycle finishes before the deadline, then the next replanning cycle begins immediately. In our implementation, we used a deadline of 250 milliseconds, although in our experiments we were on average able to achieve replanning cycles substantially faster than the deadline.

### 3.4 Results

To demonstrate the applicability of our method, we considered a simulated 2D navigation task and two physical tasks using the Baxter robot [1]. Planning was performed using a C++ implementation on a 3.4 GHz Intel Xeon E5-1680 processor.

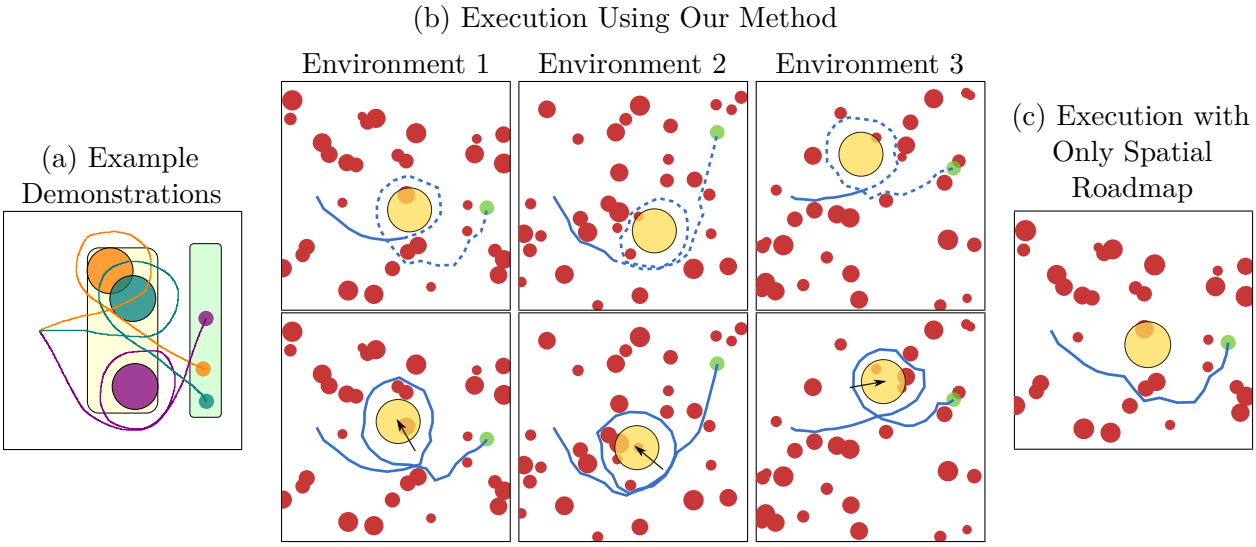



Figure 3.6: (a) Three of the 7 demonstrations for the simulated navigation task as well as the beacon and goal sampling regions. The demonstrated paths start at a fixed location, move counter-clockwise around a specified beacon, and then move to a specified goal. (b) Three executed trajectories (solid blue) and planned trajectories (dashed blue) around obstacles (red) to a goal (green) as computed by our method, before (top) and after (bottom) the beacon (yellow) was moved. (c) Execution without the temporal roadmap and using a temporal alignment heuristic [83].

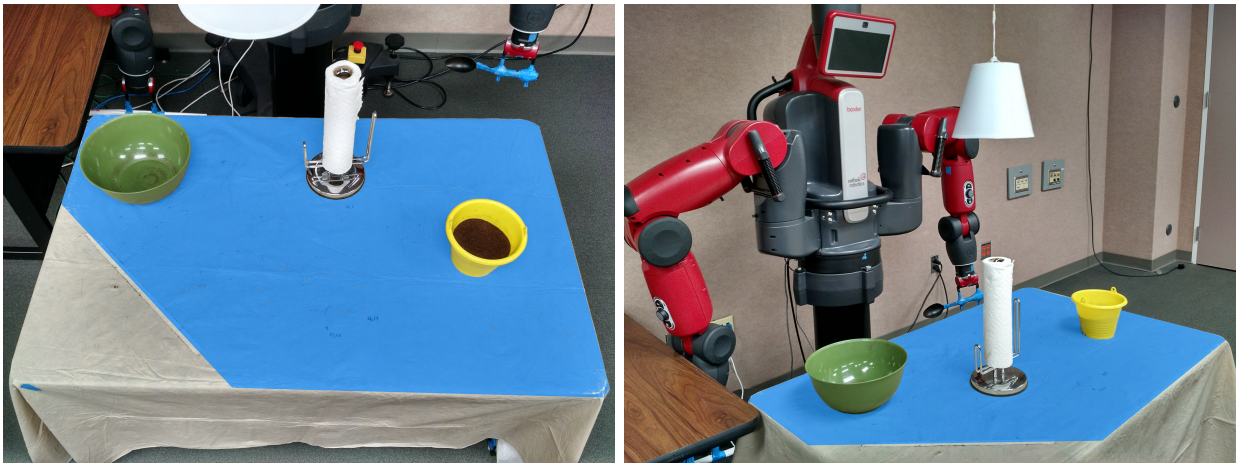


Figure 3.7: Environment for the powder transfer task with the permissible region for both the green bowl and paper towel roll highlighted in blue. The permissible region is based on the reachable workspace of the Baxter's arm.

Figure 3.8: Example execution for the powder transfer task, from a bucket (yellow) to a bowl (green).

### 3.4.1 Simulated Navigation Task

We first consider a point robot that is to navigate on a 2D plane by starting at a fixed location, moving counter-clockwise completely around a beacon without intersecting it, and then stopping at a specified goal. In the learning phase, the feature space consisted of the absolute position of the robot and its position relative to two landmarks, corresponding to the beacon and intended goal position specified by the annotations. For the demonstrations, we manually performed 7 successful trajectories with beacon locations randomly sampled from the yellow region and goal locations randomly sampled from the green region in Figure 3.6(a).

In the execution phase, the test environment included 32 circular obstacles which were not present in the demonstrations. We randomly generated 15 test cases with different obstacle locations and with randomly chosen beacon and goal locations (sampled independently from the locations chosen for demonstrations but using the same regions). A quarter of the way through the trajectory (approximately when the robot begins encircling the beacon) we moved the beacon 10% of the width of the environment in a random direction.

The method successfully accomplished the task in all 15 test cases. Several representative executions are shown in Figure 3.6(b). Due to the density of the obstacles, in most of the test cases it was necessary for the method to change homotopic classes when replanning, illustrating the need for global planning rather than local refinement of plans.

This task also illustrates the need to incorporate task progress into the planning state. Planning

| | Success Rate | Mean Pauses | Mean Update Time |
|---|---|---|---|
| Full Method | 100% | 0 | 133 ms |
| No Bidirectional Search | 50% | 23 | 170 ms |
| No Obstacle Avoidance | 10% | 0 | 44 ms |
| No Biased Sampling | 0% | – | – |
| No Replanning | 0% | – | – |

Table 3.1: Empirical results for the physical powder transfer task with moving bowl and stationary yellow bucket and obstacle during execution, including success rates, average number of pauses, and average update periods for variants of the method. The additional failures without bidirectional search were due to insufficient reaction time when the bucket or bowl were moved. Without biased sampling, the method was unable to find successful plans even after 5 minutes of roadmap computation time. Similarly, replanning was required for success in every scenario due to the motion of the bowl.

in configuration space alone is not sufficient for correct execution because a successful path crosses itself by design. Using the temporal alignment heuristic from [83] skips an important portion of the task as shown in Figure 3.6(c).

### 3.4.2 Physical Powder Transfer Task

In the second task, the goal was to scoop powder from a small bucket and transfer it into a bowl using a spoon using the Baxter robot. To successfully perform the task, the robot was required to transfer the powder without spilling while avoiding obstacles in the environment. The feature space consisted of the position and orientation of the spoon relative to the bucket and bowl, for a total feature dimension of 12. We provided the method with 13 successful kinesthetic demonstrations from which to learn.

At the beginning of execution the green bowl was randomly placed on the reachable surface of the table (see Figure 3.7) and tracked using a Kinect RGBD sensor. For obstacles, we affixed a hanging lamp shade above the table and randomly placed a vertical roll of paper towels. During task execution, when the robot positioned the spoon above the bowl but before it began dumping the contents, we quickly moved the bowl to a different location randomly sampled uniformly from the reachable surface of the table (see Figure 3.8).

We consider two variants of this task, as discussed below.

| | Success Rate | Mean Pauses | Mean Update Time |
|---|---|---|---|
| Full Method | 100% | 0 | 111 ms |
| No Bidirectional Search | 30% | 27 | 148 ms |
| No Obstacle Avoidance | 10% | 0 | 45 ms |

Table 3.2: Empirical results for the physical powder transfer task when moving the bowl, bucket, and obstacle during execution, including success rates, average number of pauses, and average update periods for variants of the method.

**Moving Bowl During Execution** We first evaluated the method on the same set of 10 scenarios used in [17] for comparison. In these scenarios, the bowl was moved during execution but the yellow bucket was always placed in a single, fixed location and the obstacles were stationary during execution.

We show empirical results for different variants of the method in Table 3.1. For each method variant, we report the success rate, the average update period (defined as the total time required to perform sensing and replan), and the average number of pauses due to missing a deadline (i.e., planning took longer than our deadline of 250 milliseconds). In some scenarios, the proximity of the paper towel roll and lamp shade created a narrow passage in configuration space through which the robot arm needed to navigate to accomplish the task. Collision detection against a point cloud was the primary bottleneck as reflected by the latency reduction when obstacle avoidance was disabled. For this reason, bidirectional search was very profitable because of the decrease in the number of edges for which collision checking was required.

**Moving Bowl, Bucket, and Obstacle During Execution** We next generalized the task by moving the bucket (before the powder was scooped), the roll of paper towels (at the same time as the bowl), and the bowl (as in the prior scenario) to random locations during execution. We also randomized the initial placement of the yellow bucket in these 10 scenarios.

Empirical results for these scenarios are shown in Table 3.2. We first note that the average latencies where shorter than their counterparts in the stationary scenarios. This is because moving the objects often required the planner to spend more time in later parts of the task, where the number of remaining time steps was fewer and thus the portion of the graph which needed to be searched was smaller. We note that this effect did not appear when obstacle avoidance was disabled,

Figure 3.9: Example execution for the liquid pouring task, from a pitcher into a pot (green).

likely because traversing edges occupies a smaller fraction of the computation time. However, there was also more variance in the latencies, as reflected in the increase in missed deadlines when bidirectional search was disabled. The success rate without bidirectional search was also lower than for the stationary scenarios because the robot failed to react in time to the motion of the bucket to satisfy the relatively narrow constraints required to successfully scoop powder from it.

### 3.4.3 Physical Liquid Pouring Task

We next evaluated the method on a liquid pouring task in which the Baxter robot pours liquid from a grasped pitcher into a pot without spilling. Here, the feature space consisted of the position and orientation of the pitcher relative to the bowl. We performed 11 successful demonstrations from which the task was learned. For the execution environment, we introduced as an obstacle a potted plant, which has complex geometry not amenable to modeling via geometric primitives but that can be sensed using a Kinect (see Figure 3.9). As before, in each of the 10 scenarios the pot and an obstacle were initially placed uniformly at random on the reachable surface of the table and then moved to different random locations, necessitating replanning.

Results for this task are shown in Table 3.3. The size of the pitcher and plant obstacle produced multiple challenging scenarios which required denser roadmaps to navigate than in the powder transfer task. This in turn resulted in marginally longer planning times and a few missed deadlines. Without bidirectional search, planning was substantially slower but the success rate was comparable because this task was less time-sensitive than the powder transfer task. Obstacle avoidance again proved crucial to successful execution in 60% of the scenarios.

| | Success Rate | Mean Pauses | Mean Update Time |
|---|---|---|---|
| Full Method | 100% | 1 | 138 ms |
| No Bidirectional Search | 90% | 26 | 212 ms |
| No Obstacle Avoidance | 40% | 0 | 61 ms |

Table 3.3: Empirical results for liquid pouring task, including success rates, average number of pauses, and average update periods for variants of the method. The additional failure without bidirectional search was due to a failure to find a difficult narrow passage and likely only a result of the randomized nature of method.

### 3.5 Conclusion

This chapter presented a closed-loop motion planning approach that is applicable to execution of learned tasks in which the environment changes during execution. Using this method, the Baxter robot was able to successfully perform several learned tasks while avoiding moving obstacles and reacting to the motion of task-relevant objects whose locations were relevant to task success. In the following chapter, we will extend the learned model to encompass time-invariant parameters to reduce the task information the human demonstrator must provide.

CHAPTER 4

# Learning Virtual Landmarks

In this chapter, we extend the learning phase introduced in Chapter 2 by allowing some additional time-invariant parameters of the task to be learned from the demonstrations while simultaneously learning the time-variant parameters considered previously. Consider the task of moving a pitcher across a table and pouring liquid into a bowl as shown in Figure 4.1. Successfully performing this task requires properly positioning and orienting the pitcher relative the bowl, and this relative position and orientation changes over time during the the task (i.e., the pitcher's orientation is initially level and then changes so that the liquid pours out). Implicit in performing this example task is that the robot must be aware of certain task-relevant landmarks, including (1) awareness that the bowl (rather than other landmarks in the scene, e.g., the paper towel roll) is important to the task, and (2) awareness that the position of the spout of the pitcher (as apposed to other landmarks on the pitcher) is most important to successfully pouring the liquid into the bowl. Utilizing appropriate task-relevant landmarks is often critical to successfully performing a task.

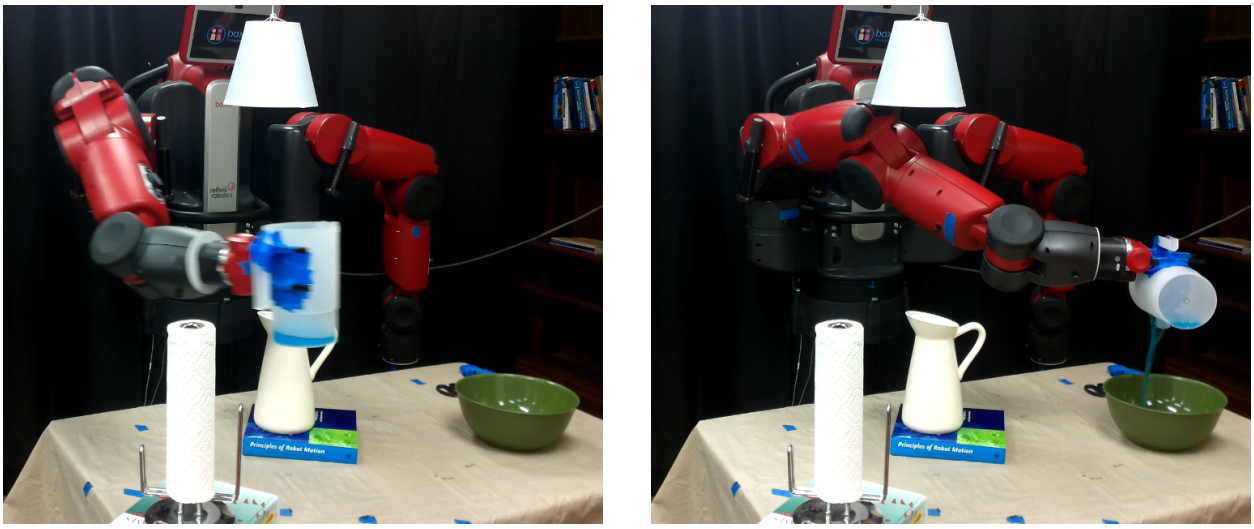


Figure 4.1: Our method automatically learns task-relevant virtual landmarks, such as the relevance of the center of the bowl and the spout of the pitcher. The method also learns a time-dependent task model parameterized by the poses of the virtual landmarks.

Prior approaches for learning manipulation tasks often assume that task-relevant landmarks are manually specified by a user (e.g., [16, 17, 86, 87, 9]). In this chapter, we propose a method to automatically learn time-invariant parameters which specify which landmarks on the robot and in the environment should be considered during execution. We assume the robot, using its kinematic model and vision system, can sense landmarks on the robot and in the environment, such as the poses of the robot's end-effector and significant objects in the scene, both during the demonstrations and during motion planning in a new environment. During the learning phase, our method learns virtual landmarks that are based on linear combinations or projections of sensed landmarks.

We demonstrate the efficacy of our approach on two tasks with the Baxter robot [1] in an environment with obstacles, a powder transfer task and a liquid pouring task. Our method improved the success rate compared to arbitrary or hand-selected landmarks by automatically selecting virtual landmarks.

This chapter is based on work previously published in [19].

### 4.1 Related Work

The problem of learning a task from human demonstrations has been studied extensively, and many different approaches have been considered [4, 33].

In [16], we presented an approach based on a user-specified feature space which incorporated the position of a landmark on the robot's end-effector (or grasped object) relative to landmarks in the environment. We extend that approach with the addition of learned time-invariant parameters on which the feature space is parameterized to reduce the amount of human-provided information required. Specifically, these parameters encode the definitions of landmark which were manually-specified previously. The feature space then incorporates the relative positions of these virtual landmarks.

One class of approaches for learning from demonstration are regression methods which directly learn a reference trajectory. This is the approach taken in [7, 42], and [86] using Gaussian Mixture Regression in a feature space. The feature space used in these works inspired the one used in [16, 17], and this paper. The learning approach we present in this paper could be adapted to learn virtual landmarks to establish coordinate systems in the context of Gaussian Mixture Regression methods as well. In [88], this was extended with a feature space selection step from a finite pool of predefined

features. Instead, we effective incorporate feature space selection in the learning as part of the optimization problem permitting more general feature spaces to be learned. Additionally, these approaches generally assume that the robot can always traverse the reference trajectory and are thus not directly applicable when new obstacles not present in the demonstrations are introduced.

A second class of approaches learn a control policy, mapping robot states to control inputs. Because of the dimensionality of this space, the class of policies must be restricted. In [89, 10], and [87] for example, the policies are restricted to nonlinear equations of a specific form the authors call Dynamic Movement Primitives, which are adapted to avoid obstacles locally, but not globally (i.e., by considering multiple homotopic classes of paths).

Finally, our approach can be contextualized in the class of approaches which learn a mapping from robot state to a cost (or reward). This can be thought of as a refinement of the second class of approaches, wherein the control policy learned is defined by the cost it optimizes. This is the approach taken in inverse reinforcement learning (e.g., [34, 35, 36, 37]), wherein this cost is assumed to be a parameterized function of some features of the robot state. In [9], a time-dependent multivariate Gaussian distribution is learned, and the Mahalanobis distance is taken as the cost. This approach was extended in [90] to incorporate feature space selection from a finite pool by using all the possible features and enforcing sparsity during the learning phase.

In this paper, we learn a probabilistic model based on a Hidden Markov Model. HMMs have previously been applied to motion recognition (e.g., [43, 5, 6]) and generation (e.g., [7, 6]). By framing the learning method as an optimization problem, we can simultaneously learn time-variant and time-invariant parameters of the task, including what virtual landmarks are most relevant to the task and produce the most consistent model. We then derive a cost which, when minimized, maximizes the probability that the path was generated by the learned model.

Cost-oriented approaches, like ours, are more amenable to global obstacle avoidance because asymptotically-optimal sampling-based planners like PRM*, RRG, and RRT* [3] are readily available, including variants designed to effectively explore low-cost regions (e.g., [27, 91]). Asymptotically optimal sampling-based planners avoid the local minima inherent in potential field methods [15], and can avoid the suboptimal plans resulting from sampling-based planners which are merely probabilistically *complete* (e.g., RRT [3]). Because we rapidly replan [54, 55] when landmarks move, it is also useful to be able to access the best known plan at any given time [32]. For these reasons,

we use a variation on a probabilistic roadmap (PRM), but do not allow it to grow arbitrarily large. Nevertheless, we could still guarantee near-optimality in finite time [31], [62].

## 4.2 Problem Definition

### 4.2.1 Inputs and Outputs

As in previous chapters, we consider a robot with a $d$-dimensional configuration space $\mathcal{Q} \subseteq \mathbb{R}^d$. In order to teach the robot a task, we provide $M$ demonstrations of the task in which the pose of task-relevant objects vary across the demonstrations. Each demonstration $m \in \{1 \dots M\}$ is a sequence of $S_m$ observed configurations of the robot at fixed time intervals. Let $\mathbf{q}_m^s \in \mathcal{Q}$ denote the $s$'th observed configuration in demonstration $m$. Additionally, we assume we have for each demonstration $m$, a description of the environment $\mathbf{a}_m$ which lists the poses in $SE(3)$ of Z sensed landmarks in the demonstration environment. We assume that the landmarks may be distinguished from each other (e.g., via visual feature matching). The resulting task model will ultimately specify which subset of these landmarks is relevant to the task and must be present in the execution environment.

Although we assume that obstacles in the execution environments do not move during execution, we do not require that they be present in the demonstrations or known during the learning phase. To enable the robot to successfully perform the task in environments with never-before-seen obstacles, the problem we consider is that of estimating the parameters in a parametric probabilistic task model amenable to use by a sampling-based motion planner given these demonstrations.

We consider the problem of estimating parameters defining the task model of two distinct types:

- $\zeta$-parameters, which will encode the position of a virtual landmark on a grasped object (e.g., a tool) relative to the robot's end-effector and what linear combination of the $Z$ sensed landmarks define a virtual landmark in the environment. Such parameters are time-invariant; they do not vary with time or between demonstrations.

- $\eta$-parameters, which represent the dependence on task progress, like the tendency for a landmark on a grasped object to be at a specific position relative to a landmark in the environment at some time point in the task. Specifically, these parameters consist of means and covariances in a $\zeta$-parameterized feature space incorporating the positions of points on a grasped object relative to landmarks (discussed in detail in Section 4.3.3). Such parameters

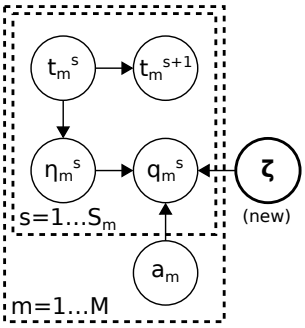



Figure 4.2: Bayesian network describing independence assumptions for the learned task model. Note that for a specific demonstration, this generalizes a Hidden Markov Model used previously with the addition of $\zeta$-parameters.

are time-variant; they vary during the task (although not between demonstrations).

We note that the challenge of learning the $\eta$-parameters was addressed in prior work [16], but that work assumed the $\zeta$-parameters were manually provided by a human user. To extend that work to consider time-invariant $\zeta$-parameters, which results in non-local interactions between time steps, we explicitly re-frame the problem as an optimization.

Once the task has been learned, we then consider the problem of executing the task in new environments with new obstacles. This requires computing an obstacle-free path in the robot's configuration space from its start configuration $\mathbf{q}_{\text{start}} \in \mathcal{Q}$ to a goal configuration $\mathbf{q}_{\text{goal}} \in \mathcal{Q}$ which incorporates learned information from the task model and considers the sensed locations of the landmarks that were found to be relevant to the task during learning. We compute paths by constructing a notion of cost for which minimum cost paths correspond to maximum probability paths given the model, and applying this cost metric to an asymptotically optimal motion planner.

### 4.2.2 Probabilistic Task Model

The $\zeta$- and $\eta$-parameters form the basis for a task model similar to a Hidden Markov Model (Figure 4.2) with discrete states, which we call *time steps*, $\{1 \dots T\}$, where the probability of observing configuration $\mathbf{q}$ at time step $t$ during demonstration $m$ is given by $p\left(() \, \mathbf{q} \mid \boldsymbol{\zeta}, \boldsymbol{\eta}_t\right)$ and the probability of transitioning to time step $t'$ from $t$ is given by $p\left(() \, t' \mid t\right)$. We also consider priors on each latent parameter, denoted $p\left(() \, \boldsymbol{\zeta}\right)$ and $p\left(() \, \boldsymbol{\eta}\right)$.

Let $\boldsymbol{Q} = \{\mathbf{q}_{1\dots M}^{1\dots S_m}\}$ denote the observed configurations from each demonstration, and let $\boldsymbol{A} = \{\mathbf{a}_{1\dots M}\}$ denote the environment descriptions from each demonstration that list the sensed landmark

poses. Let $\boldsymbol{H} = \{\boldsymbol{\eta}_{1\ldots T}\}$ denote the set of $\eta$-parameters from each time step, and let $p\left(() \boldsymbol{H}\right)$ be given by $p\left(() \boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_T\right) = \prod_{t=1}^{T} p\left(() \boldsymbol{\eta}_t\right)$ by independence. For convenience of notation, we will let $\boldsymbol{\eta}_m^s = \boldsymbol{\eta}_{t_m^s}$ denote the $\eta$-parameter corresponding to the time step associated with the $s^{\text{th}}$observation in demonstration $m$. There are still only $T$ such parameters; this notation merely serves as a convenient view of them. By the conditional independence properties of the given model, we have the following:

$$p\left(() \boldsymbol{Q}, \boldsymbol{\zeta}, \boldsymbol{H} \mid \boldsymbol{A}\right) = \tag{4.1}$$
$$p\left(() \boldsymbol{\zeta}\right) \cdot \prod_{t=1}^{T} p\left(() \boldsymbol{\eta}_t\right) \cdot \prod_{m=1}^{M} \prod_{s=1}^{S_m} (p\left(() t_m^{s+1} \mid t_m^s\right) p\left(() \mathbf{q}_m^s \mid \boldsymbol{\zeta}, \boldsymbol{\eta}_m^s, \mathbf{a}_m\right)).$$

Putting the model in this form separates the prior, transition, and observation distributions to facilitate parameter estimation in the following section.

### 4.3 Learning

We first define virtual landmarks as time-invariant parameters, and then describe our approach for simultaneously estimating the time-variant and time-invariant parameters of the task model. Although much of the problem is similar to our prior work [16], the introduction of time-invariant parameters necessitates a different, global, learning approach due to their non-local nature.

#### 4.3.1 Feature Space using Virtual Landmarks

We begin by defining the virtual landmarks that will be learned as the $\zeta$-parameters. These virtual landmarks are based on linear combinations or projections of sensed landmarks $\mathbf{a}$ whose pose is identified using the robot's kinematic model and vision sensors.

We first consider an *environmental virtual landmark*, which is based on a linear combination of sensed landmarks in the environment. $\zeta$ includes the coefficients of this linear combination. We denote this portion of $\zeta$ as $\zeta^{\text{env}}$ and require the sum of these coefficients be 1 by locally parameterizing the tangent space of this constraint. In addition to being an intuitive way to combine sensed landmarks, constraining the $L^1$-norm encourages sparsity. After learning, to further enforce sparsity, we discard sensed landmarks with coefficients less than 5% to form the set of sensed landmarks required during task execution. Specifically, we compute the pose of the environmental

virtual landmark as follows:

$$K_{\text{virt}}(\boldsymbol{\zeta}, \mathbf{a})^{-1} = \sum_{i=1}^{Z} \boldsymbol{\zeta}_i^{\text{env}} K_{\text{sens}}^i(\mathbf{a})^{-1}$$

where $K_{\text{sens}}^i(\mathbf{a}) \in SE(3)$ denotes the pose of the $i^{\text{th}}$sensed landmark in the environment described by the input $\mathbf{a}$ from 4.2.1.

The $\zeta$-parameters may also encode the position of a point relative to the robot's end effector. We will denote this portion of $\zeta$ as $\zeta^{\text{tool}}$. This point defines a *tool virtual landmark*, i.e., a point on a grasped object (e.g., a tool).

Together, these virtual landmarks, combined with the robot's configuration $\mathbf{q}$, serve to define a *feature space* that augments that used in [17] by including the position of a tool virtual landmark relative to an environmental virtual landmark. This feature space incorporates both configuration and task spaces, enabling the method to learn tasks which require both. Specifically, we define a function $f$ to lift configurations into the feature space for learning (see Section 4.3.3):

$$f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) = \begin{bmatrix} \mathbf{q} \\ K_{\text{virt}}(\boldsymbol{\zeta}, \mathbf{a})^{-1} K_{\text{end}}(\mathbf{q}) \boldsymbol{\zeta}^{\text{tool}} \end{bmatrix}, \tag{4.2}$$

where $K_{\text{end}}(\mathbf{q})$ denotes the pose of the end effector when the robot is in configuration $\mathbf{q}$, and $K_{\text{virt}}(\mathbf{a})$ denotes the pose of the environmental virtual landmark.

### 4.3.2 Maximum a Posteriori Estimation

Given a set of demonstrations, our goal is to find the maximum *a posteriori* probability (MAP) estimates for $\boldsymbol{\zeta}$ and $\boldsymbol{H}$. To accomplish this, we combine dynamic time warping (DTW) [45] and local optimization using an expectation-maximization (EM) approach.

For numerical convenience, we first take the negative logarithm of (4.1), a common loss function in the machine learning literature. By monotonicity, minimizing this quantity is equivalent to maximizing the original function. To facilitate this transformation, let $L(\cdot)$ denote $-\log p\left(\left(\right)\cdot\right)$,

yielding the following:

$$L(\boldsymbol{Q}, \boldsymbol{\zeta}, \boldsymbol{H} \mid \boldsymbol{A}) =$$
$$L(\boldsymbol{\zeta}) + \sum_{t=1}^{T} L(\boldsymbol{\eta}_t) +$$
$$\sum_{m=1}^{M} \sum_{s=1}^{S_m} \Big( L(t_m^{s+1} \mid t_m^s) + L(\mathbf{q}_m^s \mid \boldsymbol{\zeta}, \boldsymbol{\eta}_m^s, \mathbf{a}_m) \Big).$$

In this form, it is perhaps apparent that if we fix the time step $t_m^s$ corresponding to each observation $\mathbf{q}_m^s$, the problem of finding the most likely $\zeta$- and $\eta$-parameters becomes one of performing a simple, if high-dimensional, nonlinear optimization. In our implementation, this is accomplished using the Ceres nonlinear least-squares solver [92]. The transformation from this form to a nonlinear least squares problem is discussed in Section 4.3.3.

Dynamic time warping is used to find the most likely sequence of time steps $t_m^{1 \ldots S_m}$ corresponding to the observations $\mathbf{q}_m^{1 \ldots S_m}$ in each demonstration $m$ using the current best estimates for the latent parameters. This is analogous to the time-alignment steps used in prior methods [9, 83]. The value update equations for the dynamic time warping which maximize likelihood are as follows:

$$l_m^0[t'] = 0$$
$$l_m^s[t'] = \min_{t \in [1,T]} \Big( l_m^{s-1}[t] + L(t' \mid t) \Big) + L(\mathbf{q}_m^s \mid \boldsymbol{\zeta}, \boldsymbol{\eta}_{t'}, \mathbf{a}_m),$$

where $l_m^s[u]$ denotes the value of the best assignment of time steps $t_m^{1 \ldots s}$ to observed configurations $\mathbf{q}_m^{1 \ldots s}$ from demonstration $m$ with $t_m^s = u$.

Following the EM approach, this most likely sequence of time steps is then fixed and used to estimate new $\zeta$- and $\eta$-parameters as discussed above. This process is repeated until convergence. Because EM approaches may become caught in local optima, we employ random restarts with randomly-chosen initial alignments.

The transition probabilities $p\left(() \, t' \mid t\right)$ may either be fixed (the approach taken in [16]), estimated from the alignments, or a combination of the two approaches with a fixed set of permitted transitions with estimated probabilities (the approach taken in [17]). In the experiments, we use the last of these approaches.

### 4.3.3 Estimation via Minimization

We consider a specific class of distribution based on the assumption that observations have multivariate Gaussian distributions in a feature space at each time step. We cannot estimate time steps independently as in prior work because we simultaneously estimate time-invariant parameters.

Given the differentiable function $f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a})$ defined in Section 4.3.1 which lifts a configuration into feature space $\mathbb{R}^F$, we consider $\boldsymbol{\eta}$ which captures the mean $\boldsymbol{\mu} \in \mathbb{R}^F$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{F \times F}$ in feature space. Specifically, $\boldsymbol{\eta} = [\boldsymbol{\mu}^{\mathrm{T}}, \mathrm{vec}[\boldsymbol{\sigma}^{-1}]^{\mathrm{T}}]^{\mathrm{T}}$ where $\mathrm{vec}[\boldsymbol{\sigma}^{-1}] \in \mathbb{R}^{F(F+1)/2}$ denotes a vector containing the components of the inverse of the lower-triangular Cholesky factorization of $\boldsymbol{\Sigma} = \boldsymbol{\sigma}\boldsymbol{\sigma}^{\mathrm{T}}$. In this feature space model, we consider $p(\mathbf{q} \mid \boldsymbol{\zeta}, \boldsymbol{\eta}, \mathbf{a})$ defined by $f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We then have the following negative log conditional probability:

$$
\begin{aligned}
&L(\mathbf{q} \mid \boldsymbol{\zeta}, \boldsymbol{\eta}, \mathbf{a}) = \\
&L(f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim \\
&\log\det\boldsymbol{\Sigma} + (f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) - \boldsymbol{\mu}) = \\
&\left|\sqrt{\log\det\boldsymbol{\Sigma}}\right|^2 + \left|\boldsymbol{\sigma}^{-1}(f(\mathbf{q}, \boldsymbol{\zeta}, \mathbf{a}) - \boldsymbol{\mu})\right|^2.
\end{aligned} \tag{4.3}
$$

Minimizing this is explicitly a nonlinear least-squares problem.

The choice of representation in terms of $\boldsymbol{\sigma}^{-1}$ is particularly convenient because it admits fast computation of the residual of the transformed problem. Specifically, to compute the first term, we use the fact that $\log\det\boldsymbol{\Sigma} = -2\sum_{i=1}^{F}\log|\boldsymbol{\sigma}_{i,i}^{-1}|$ by the multiplicative property of the determinant and triangularity of $\boldsymbol{\sigma}$ (and thus of $\boldsymbol{\sigma}^{-1}$). In the second term, the residual is linear in both $\boldsymbol{\sigma}^{-1}$ and $\boldsymbol{\mu}$ and thus multilinear in $\boldsymbol{\eta}$. This not only decreases the computation time needed to evaluate the residual, but greatly improves the convergence rate and reduces local minima.

We note that the first term depends only on $\boldsymbol{\sigma}^{-1}$ and so need not be recomputed for each observation. For computational efficiency, it can even be incorporated into the prior (effectively treating the conditional as an unnormalized distribution). In fact, this term is proportional to the logarithms of multiple classical priors [93], so such a prior can be incorporated simply by changing the scale on this term.

The only restriction we place on the priors is that they be log differentiable and note that they may be transformed similarly to the way the normalization term was above, although they may

Figure 4.3: The Baxter robot performing the powder transfer task with the spoon being used as a tool in blue, the source container in yellow, and the destination container in green. Both the lamp and paper towel obstacles are white.

also admit more elegant forms (e.g., exponential distributions). In the experiments conducted in Section 4.4, we used (unnormalized) uniform priors for all parameters.

## 4.4 Results

We evaluated our method on two simplified food preparation tasks on the Baxter research robot [1]. Once learned, these tasks were performed autonomously using the method described in [17], which is very similar to that from Chapter 3. All computation was performed on two 2.0 GHz 6-core Intel Xeon E5-2620 processors. Landmark poses were continuously tracked using a Kinect sensor.

### 4.4.1 Powder Transfer Task

We first tested our method on the same task as in Chapter 3, wherein the Baxter robot learned to transfer powder from one container to another (see Figure 4.3) while the bowl moved. However, unlike prior work, a human did not specify a landmark on the spoon, which previously was manually specified as the tip. Instead, we automatically learned a tool virtual landmark via $\zeta^{\text{tool}}$ specifying a position relative to the pose of the robot's gripper. Additionally, we synthetically introduced 4 sensed landmarks into each demonstration in addition to the bowl, randomly sampled in the reachable space of the robot, bringing $Z$ to 5 and the dimensionality of the $\zeta$-parameter to 8. The feature space for learning included the robot's configuration specified by its 7 joint angles as well as the position of the tool virtual landmark relative to a learned environmental virtual landmark based on (4.2).

As mentioned in Section 4.3.3, the $\eta$-parameters used were means and covariance matrices in this feature space. For this task, we used $T = 24$ time steps. The learning algorithm was then used to estimate these parameters from the same 11 demonstrations used to evaluate the previous method [17], but with the 4 synthetic new sensed landmarks added to the set $\boldsymbol{A} = \{\mathbf{a}_{1\ldots M}\}$ for each

Figure 4.4: A spoon being used as a tool in the powder transfer task with an arbitrary landmark at the robot's gripper marked in blue, the hand-picked landmark at the tip marked in red, and the tool virtual landmark learned by our method marked in green.

| Tool Landmark (on spoon) | Environment Landmark (bowl) | Success Rate | Learning Time | Planning Latency |
|---|---|---|---|---|
| Arbitrary | Manual | 60% | 26s | 201ms |
| Manual | Manual | 80% | 26s | 226ms |
| **Learned** | Manual | **100%** | **1,658s** | **182ms** |
| **Learned** | **Learned** | **100%** | **1,686s** | **182ms** |
| *(without obstacle avoidance)* | | 10% | 1,686s | 108ms |

Table 4.1: Powder transfer task results averaged across 10 scenarios.

Figure 4.5: Baxter robot performing the liquid pouring task with the pitcher (being used as a tool) with blue liquid and the green bowl. The lamp, vase, and paper towel obstacles are white.

demonstration.

The actual virtual tool landmark learned for the spoon differed notably from the hand-picked point at the tip used previously (see Figure 4.4). The point estimated by the learning method corresponded roughly to the average point of rotation on the spoon during the dumping motion. The method also correctly learned a virtual environmental landmark which consisted only of the sensed landmark corresponding to the bowl, with effectively no contribution from the other sensed landmarks. This was because the standard deviation of the tool landmark relative to the bowl in the facing direction of the robot in one time step was only 7 cm, while for the other landmarks, it was as high as 78 cm.

We tested the new learned model using the same planner on the same 10 scenarios as in the previous method's evaluation [17]. The scenarios included uniformly random paper towel obstacle and bowl locations as well as a hanging lamp obstacle. The bowl was moved to another random location midway through the task, requiring the closed-loop motion planner to react quickly. An execution was considered successful if the robot avoided obstacles in the environment and transferred the powder without spilling. A video of the robot executing this task is available at `https://youtu.be/QaiRBRwE3Lo` and quantitative results are provided in Table 4.1.

Surprisingly, the new learned model resulted in a higher success rate than the previous method even though less information was manually provided, likely because the relevant covariances were better captured with the new virtual landmarks in $\zeta$. This also slightly improved planning latency because lower variances imply narrower low-cost regions, which allow the path search to explore and consequently lazily evaluate fewer edges.

| Tool Landmark (on pitcher) | Environment Landmark (bowl) | Success Rate | Learning Time | Planning Latency |
|---|---|---|---|---|
| Arbitrary | Manual | 40% | 216s | 225ms |
| **Learned** | **Learned** | **90%** | **71,462s** | **200ms** |
| *(without obstacle avoidance)* | | 30% | 71,462s | 132ms |

Table 4.2: Liquid pouring task results averaged across 10 scenarios.

To further demonstrate the importance of appropriately choosing the tool virtual landmark, we also ran the learner arbitrarily fixing $\zeta^{\text{tool}} = \mathbf{0}$ corresponding to the robot's gripper. This was only successful near the center of the table and not at the extremes where the robot completely missed the bowl.

Finally, we evaluated planning using only local optimization of the learned task model without any obstacle avoidance. As expected, this resulted in many failures due to collisions with obstacles. The results of each approach are shown in Table 4.1.

### 4.4.2 Liquid Pouring Task

We next tested our method on the task of pouring liquid from a grasped pitcher into a bowl (see Figure 4.5). The pose of the bowl and 4 other sensed landmarks (a pair of scissors, a spoon, a vase, and a roll of paper towels) varied between executions, bringing $Z$ to 5. The feature space used to construct the probabilistic model was the same as for the powder transfer task, again with $T = 24$ time steps and the dimensionality of the $\zeta$-parameter was again 8.

We performed 11 demonstrations of the task which we then supplied to the learning method with the pose of each sensed landmark, including the bowl, selected uniformly at random from a 30-inch square on the surface of the table. Additionally, the extra sensed landmarks were lifted up to 10 inches off the table surface. The learning phase took significantly longer than for the transfer task (see Table 4.2) because the demonstrations were sampled at 50Hz rather than 10Hz, resulting in significantly more observations per demonstration. This could have been mitigated simply by subsampling during learning. However, this does show that the learning method is robust to different sampling rates. We note that learning only has to be done once after the demonstrations are provided, and does not need to be performed again when the robot executes the task in a new environment using the motion planner.

The method found a tool virtual landmark corresponding to a point just below the spout of the pitcher (distinct from that learned in Section 4.4.1). The only sensed landmark with a learned contribution to the environmental virtual landmark was the bowl. So the objects corresponding to the other sensed landmarks served only as obstacles during execution.

We tested the motion planner on 10 randomly selected scenarios which included the same objects as the demonstrations as well as a hanging lamp obstacle. An execution was considered successful if the robot avoided obstacles in the environment and poured the liquid into the bowl without spilling. During execution, the large size of the pitcher resulted in more constrained motion planning problems in some of the scenarios than in the powder transfer task, which caused one trial to result in failure to pour the liquid. A video of the robot successfully executing this task is available at `https://youtu.be/QaiRBRwE3Lo` and quantitative results are provided in Table 4.2.

## 4.5 Conclusion

This chapter presented a method for performing tasks relative to initially unknown landmarks using a task model which encodes both the task motion and the landmarks required by it. We showed that such a model can be learned from human-guided demonstrations by simultaneously estimating time-variant and invariant parameters. Furthermore, this model is amenable to fast, global sampling-based motion planning.

Our method requires less user-provided information compared to prior work by enabling the robot to learn, from demonstrations, relevant virtual landmarks both on tools and in the environment. These virtual landmarks help define the feature space of the learned task model. We demonstrated the efficacy of our approach by learning and executing two manipulation tasks on the Baxter robot, including a powder transfer task and a liquid pouring task. The following chapter will target the other half of the learning process, namely temporal registration, to reduce the number of demonstrations required for learning.

CHAPTER 5

# Improving Temporal Registration

Registering a time sequence of observations to a reference model is a common subproblem in many robotics algorithms, including algorithms for both motion recognition [94] and robot task learning [95, 96] like that described in Chapter 2. This subproblem may be referred to as time alignment, time warping, or *temporal registration* (which is the term we use in this chapter). Formally, temporal registration is an assignment of individual observations (from a sequence of observations) to the ordered steps in some underlying reference model. In the domain of robot learning from demonstrations, the observations correspond to the time-ordered data collected during each demonstration, where each demonstration must be temporally aligned to an underlying task model (e.g., a reference demonstration or task representation) to facilitate learning features of the motion over time from a set of demonstrations. The process of temporal registration abstracts away differences in execution speed both between and within demonstrated trajectories and is often a

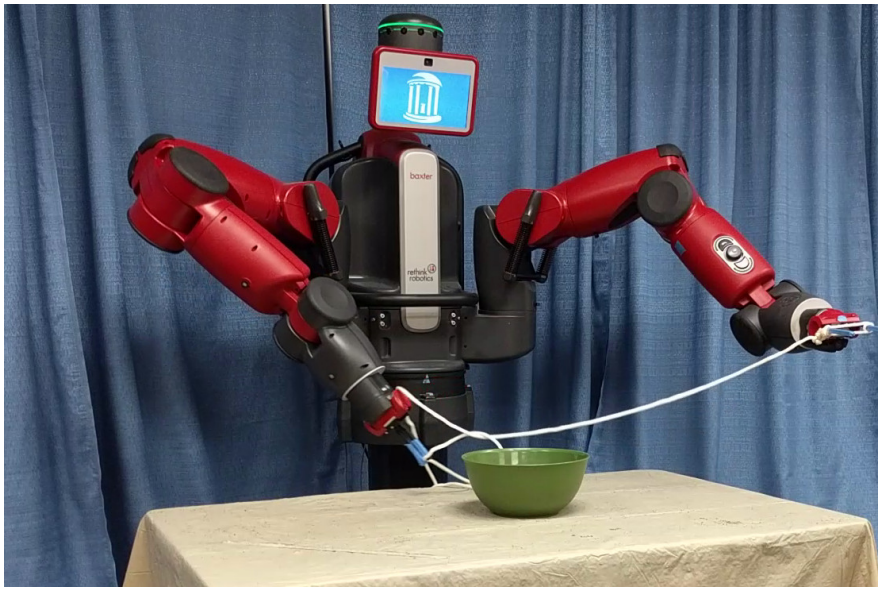



Figure 5.1: The Baxter robot performing a knot-tying task learned from demonstrations. The temporal registration of demonstrations can have a significant impact on the quality of the learned task model.

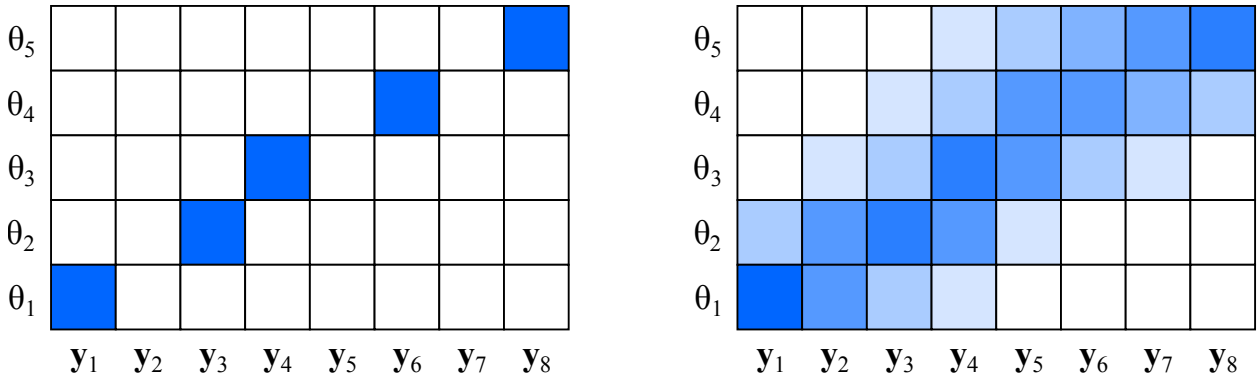
critical step to learning effective task models.

Many prior methods for learning synthesizable models rely on dynamic time-warping (DTW) [97] for temporal registration, either as a preprocessing step [95, 98, 99] or interleaved with model estimation [35, 16]. DTW can be viewed as a maximum-likelihood approach to temporal registration. However, maximum-likelihood approaches are inherently prone to becoming caught in local minima, yielding suboptimal registrations when a different initialization would produce better results. Furthermore, DTW may drop some or even most observations from the temporal registration which can produce degenerate temporal registrations in which a time step of the reference model has too few observations aligned to it. Mostly importantly, DTW does not consider uncertainty in the temporal registration. We explore whether or not these limitations negatively impact robot learning algorithms that rely on temporal registration, noting that prior work both within and outside of robot task learning has aimed to address similar shortcomings [100, 101].

With these alternative approaches in mind, we compare maximum-likelihood temporal registration approaches against a more general form of temporal registration that explicitly captures uncertainty in the temporal registration in the form of *probability-weighted temporal registrations* (PTRs). Instead of assuming each observation is registered to (at most) one time step of the reference model, we instead compute probability-weighted assignments (as shown in Figure 5.2). These weighted assignments can be leveraged in robot learning algorithms to yield more robust task models.

Specifically, we propose to use the classical forward-backward algorithm [102] to compute probability-weighted temporal registration. This approach is not new, but we present a framework based on a tensor product graph in which DTW and the forward-backward algorithm are in fact remarkably similar in terms of implementation. Using this graphical approach, we describe a novel modification for avoiding degenerate registrations that improves registrations across methods in practical applications.

We demonstrate that it is often easy to adapt existing methods to utilize the probability-weighted temporal registrations produced by the forward-backward algorithm by doing so for two models from prior work [35] including the one presented earlier in this document (Chapter 2). We also show empirically that using the forward-backward algorithm (and the modification mentioned above) produces better models with smoother motions and higher success rates on challenging tasks both




(a) Traditional temporal registration using DTW

(b) Probability-wighted temporal registration

Figure 5.2: Comparison between traditional temporal registration (e.g., using dynamic time warping (DTW)) and probability-weighted temporal registration (using our approach) to register a sequence of observations $\{\mathbf{y}_1, \ldots, \mathbf{y}_8\}$ to a reference model $\{\tau_1, \ldots, \tau_5\}$. In our approach, we register all samples to the task model, avoid degenerate registrations, and assign probabilities to the alignment of each observation, which can be utilized by a learning algorithm to improve the quality of a learned task model.

in simulation and on the Baxter robot. And while the forward-backward algorithm used by PTR is more computationally expensive than DTW, it is not asymptotically slower in the general case, so it still scales well with the problem size. And in an expectation-maximization framework as used by some robot learning algorithms, the improved temporal registration of PTR may even reduce the number of iterations required for convergence of the robot learning algorithm, yielding overall similar running times for some problems.

### 5.1 Related Work

Temporal registration is necessary for solving many estimation, classificiation, and clustering problems in robotics, signal processing, and other fields. By far the most common method for temporal registration is dynamic time warping (DTW), which has been successfully applied to problems such as gesture recognition [94] and robot task learning using Guassian mixture models [95, 98] as a preprocessing step. Similar approaches have been applied to the problem of learning to manipulate deformable objects [96]. Other methods instead integrate DTW or similar methods into an iterative estimation process [35, 16].

DTW has also been used to learn and subsequently execute tasks in the presence of external perturbations [103]. Collaborative tasks add an additional cause for variable demonstration speed, which has been addressed using related modifications to DTW [104] or gradient descent to overcome

the non-smooth nature of DTW [105]. That weakness of DTW, combined with the desire for a global solution rather than one which relies on local optimization, is what motivates this work.

When viewed as the problem of registering demonstrations to an underlying hidden Markov model (HMM) [99] or semi-Markov model [106], DTW is quite similar to the well-known Viterbi algorithm [44] for inference of the hidden model variables. Work has been done to produce variants of the Viterbi algorithm for finding registrations with specific properties [107], lower risk [108], or approximations for switching linear dynamic systems (SLDSs). For SLDSs in particular, alternative approaches to estimation that do not rely on the Viterbi algorithm have been explored [100], and these systems have been combined with reinforcement learning to good effect [109].

Estimation of an HMM using the Viterbi algorithm in a simple expectation-maximization framework yields the Viterbi path-counting algorithm [82]. However, the classical forward-backward algorithm has nicer properties. So in this work, we explore how use of the forward-backward algorithm versus DTW or the Viterbi algorithm impacts learning and subsequent execution for HMM-based task models. We note that in an expectation-maximization framework, this approach produces the well-known Baum-Welch algorithm.

We do so by comparing these approaches applied to two methods, that presented in Chapter 2 and one presented by van den Berg et al. [35]. We find measurable improvements by using a temporal registrations approach that captures uncertainty. While adapting these methods to use such registrations, we also propose modifications to any of these approaches that enforce non-degeneracy constraints, resulting in improved model estimation in practice. Given these results, we suggest that the forward-backward algorithm be used more broadly for learning task models from demonstrations in robotics.

## 5.2 Problem Definition

Consider a user-provided demonstration given by a sequence $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ where each $\mathbf{y}_i$ is an observation which might be in the state space of the robot or some arbitrary feature space. It is often necessary for learning or recognition algorithms to register such a demonstration to a different sequence $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}$ of *time steps* (often in a learned task model) while abstracting away differences in demonstration speed. We are generally interested in a registration which minimizes some measure of loss $L(\mathbf{y}_i, \boldsymbol{\theta}_t)$ between matched elements of each sequence.

The most common approach to this problem in the domain of robotics is DTW, which produces

for each $\boldsymbol{\theta}_t$ an assignment $i = \iota_t$ (iota of $t$) corresponding to $\mathbf{y}_i$ (see Figure 5.2a). In particular, this assignment minimizes $\sum_t^T L(\mathbf{y}_{\iota_t}, \boldsymbol{\theta}_t)$ subject to a strict monotonicity constraint $\iota_t < \iota_{t+1}$. Stronger variants of this constraint may also be enforced as shown in Section 5.3.4.

However, this approach discards much of the demonstration and fails to encode uncertainty in the temporal registration. To address these weaknesses, we consider probability-weighted temporal registrations which assign to each $\mathbf{y}_i$ a weight $\omega_{i,t}$ for each $\boldsymbol{\theta}_t$ (see Figure 5.2b). These weights must sum to one and so form a distribution. Here, we minimize $\sum_{i,t}^{n,T} \omega_{i,t} L(\mathbf{y}_{\iota_t}, \boldsymbol{\theta}_t)$ subject to a monotonicity constraint similar to that above generalized to distributions (discussed in Section 5.3.1).

To make these temporal registrations truly probability-weighted, we impose a restriction on the loss $L$, namely that it be defined as the negative log likelihood for some statistical model $\mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_i)$: $L(\mathbf{y}_i, \boldsymbol{\theta}_t) \equiv -\log \mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_i)$ . This model could of course be one described in Chapters 2, 3, or 4.

More generally, of course, these models are estimated from multiple demonstrations. In such cases, each demonstration can be temporally registered to the model independently, and given the resulting registrations, the model can be re-estimated. The process repeats in an expectation-maximization loop until convergence. This is the approach taken by both the learning methods of both models we evaluate in Section 5.5.

## 5.3 Method

In this section, we describe a graphical approach to probability-weighted temporal registration using the forward-backward algorithm along with a practical improvement to any temporal registration method that fits into this graphical framework. In Section 5.3.1 and Section 5.3.2, we discuss two maximum-likelihood temporal registration methods: DTW and the Viterbi algorithm. We recast both approaches as shortest paths on a tensor product graph. In Section 5.3.3, we next discuss a true expectation-maximization algorithm that produces a temporal registration with probability-weighted assignments for the observations. Again, we reformulate this approach using the tensor product graph and show that it is in fact nearly the same algorithm as the Viterbi algorithm. Finally, in Section 5.3.4 we show that any of these methods can be improved upon by modifying the tensor product graph to enforce a non-degeneracy constraint, which is particularly valuable when integrated with robot learning and using very few demonstrations as input, as shown in Section 5.5. To our knowledge, the reformulation of these algorithms in terms of a tensor product graph is novel, as is the modification to enforce the non-degeneracy constraint. The combination of

using the forward-backward algorithm and enforcing the non-degeneracy constraint gives us our complete method for probability-weighted temporal registration (PTR).

### 5.3.1 Dynamic Time Warping as a Graph Algorithm

The structure of the DTW algorithm follows the familiar dynamic programming approach of iteratively solving each subproblem in a table. In particular, let $C[t, i]$ denote the cost of the best registration of the first $t$ time steps $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t\}$ to the first $i$ observations $\{\mathbf{y}_1, \ldots, \mathbf{y}_i\}$. We then have for all $t \leq T$ and $i \leq n$:

$$\begin{aligned} C[0,0] &= 0 \\ C[t,0] &= \infty \\ C[0,i] &= \infty \\ C[t,i] &= \min(C[t,i-1] + c_{\text{insert}}(t), \\ &\qquad C[t-1,i] + c_{\text{delete}}(i), \\ &\qquad C[t-1,i-1] + c_{\text{match}}(t,i)) \end{aligned} \tag{5.1}$$

where $c_{\text{insert}}(t)$ denotes the cost of not matching $\boldsymbol{\theta}_t$ to any observation, $c_{\text{delete}}(i)$ the cost of not matching observation $\mathbf{y}_i$ to any time step, and $c_{\text{match}}(t, i)$ the cost of matching $\boldsymbol{\theta}_t$ with $\mathbf{y}_i$. The actual registration $\iota$ may then be constructed by traversing this table.

The best temporal registration is the one which maximizes $\mathcal{L}(\iota \mid Y, \Theta)$, that is the likelihood of the entire registration given both sequences. Because the loss function we assumed depends only $\mathbf{y}_i$ and $\tau_t$ where $\iota_t = i$, it satisfies the Markov property, and thus the joint likelihood is simply $\prod_{t=1}^{T} \mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_{\iota_t})$, and maximizing this is equivalent to minimizing $\sum -\log \mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_{\iota_t})$. Substituting

$$\begin{aligned} c_{\text{insert}}(t) &= \infty \\ c_{\text{match}}(t,i) &= -\log \mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_i) \end{aligned} \tag{5.2}$$

in the recurrence above yields $C[t, i] = -\log \mathcal{L}(\Theta \mid Y, \iota_{1\ldots t})$ where $\iota_{t'} \leq i$. By monotonicity of log, minimizing $C[T, n]$ maximizes $\mathcal{L}(\Theta \mid Y, \iota)$.

Letting $c_{\text{insert}}(t) = \infty$ ensures that every time step is registered to an observation, but much of the prior work in robotics further assumes $c_{\text{delete}}(i) = 0$ [97]. That is, not every observation needs to


(a) Bayesian network

(b) Restricted transitions (time graph)

Figure 5.3: Discrete-time hidden Markov model in which time steps comprise the discrete states and observations are those from the demonstrations.

be considered during registration, and only the best $T$ observations contribute to model estimation. Not only does this not match the underlying model, it discards information which could otherwise be useful for robust parameter estimation. In the next section, we will consider an alternative, the Viterbi algorithm. But first, it will be convenient to recast the DTW algorithm as a graph algorithm. This view will become an important step in unifying and extending all the temporal registration algorithms we consider.

To view DTW as a graph algorithm, we first construct graphs representing both the time steps and the demonstration. The demonstration simply becomes a sequential graph of its observations (the *demonstration graph*). Time steps encode a more complex relation, but one which can be described by a hidden Markov model (HMM). To see this, consider the time steps to be the discrete states, denoted $\tau_i$, of a time-homogeneous hidden Markov model (HMM) with observation distributions parameterized by $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m\}$ and transition probabilities given by $p_{t,t'}$. To enforce a monotonicity constraint, we restrict the state transitions as shown in Figure 5.3b, and this forms the *time graph*. Finally, we then consider the tensor product [110] of these two graphs (see Figure 5.4).

Note that unlike the time graph, we omit self-edges in the demonstration graph, which similar to setting $c_{\text{insert}}(t) = \infty$ above, ensures that every time step is matched with an observation. We will later generalize this constraint in Section 5.3.4.

We next assign to each edge from $(t, i)$ to $(t', i')$ a cost as follows:

$$c_{\text{DTW}}((t,i) \to (t',i')) = \begin{cases} -\log \mathcal{L}\left(\boldsymbol{\theta}_{t'} \mid \mathbf{y}_{i'}\right) & t \neq t' \\ 0 & t = t' \end{cases}. \tag{5.3}$$

Under these edge costs, the DTW registration is simply the shortest path from $(1, 1)$ to $(T, n)$, where a match occurs whenever the time step changes along the path.

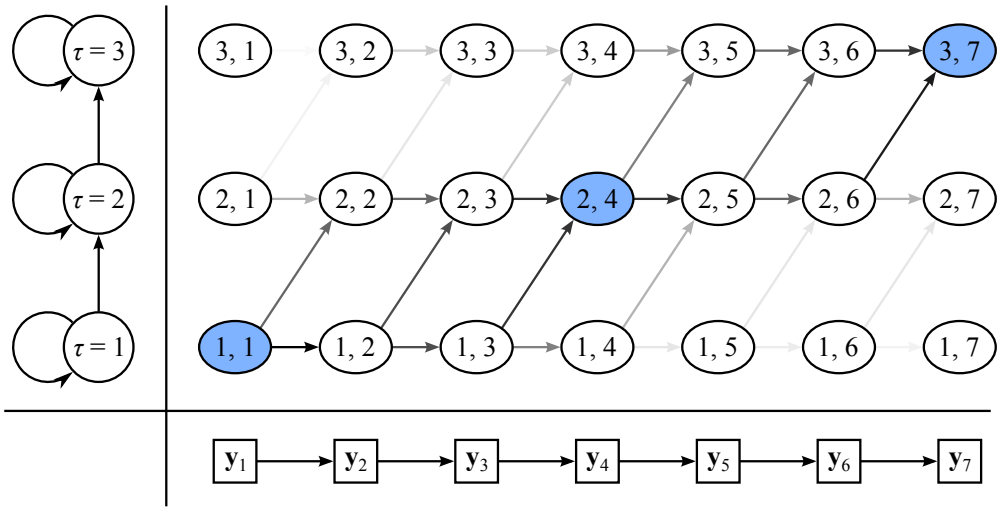



Figure 5.4: Tensor product of time graph (left) and demonstration graph (bottom). Darker edges have lower costs and blue vertices show the DTW registration.

### 5.3.2 Temporal Registration Using the Viterbi Algorithm

As we alluded to previously, this variant of DTW does not accurately reflect the underlying HMM because the result may not include some of the observations. However, there is a similar algorithm which does find the true maximum-likelihood registration of a sequence of observations to an HMM, namely the Viterbi algorithm [44], which has previously been used in task learning and recognition [111, 112]. This algorithm can be viewed as a specific instance of DTW with an appropriate choice of costs, but it is more illuminating to describe using the graph we introduced in the prior section.

To do so, we first need to reparameterize the registration, so that instead of mapping time steps to observations via $\iota_t$, we map observations to time steps via $\tau_i$. Then we need only change the edge costs as follows:

$$\begin{aligned} c((t,i) \to (t',i')) &= -\log \mathcal{L}\left(\tau_{i'} = t' \mid \tau_i = t, Y, \Theta\right) \\ &= -\log p_{t,t'} - \log \mathcal{L}\left(\boldsymbol{\theta}_{t'} \mid \mathbf{y}_{i'}\right) \ . \end{aligned} \tag{5.4}$$

Recall that $p_{t,t'}$ is the probability of transitioning from time step $t$ to $t'$, so this is not only arguably simpler than the DTW edge costs (5.3), but correctly considers transition probabilities. Under these edge costs, the Viterbi registration is again the shortest path from $(1,1)$ to $(T,n)$, but we consider $\boldsymbol{\theta}_t$ to be matched with $\mathbf{y}_i$ whenever $(t,i)$ occurs in this path. In particular, a time step may be

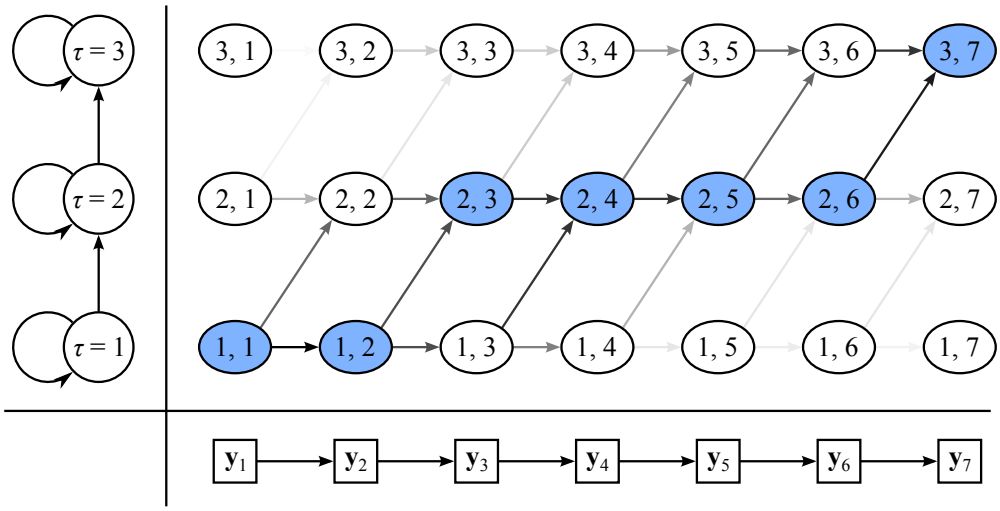



Figure 5.5: Tensor product of time graph (left) and demonstration graph (bottom). Darker edges are more probable and blue vertices show the maximum likelihood temporal registration. This corresponds to the shortest path in the graph.

matched with multiple observations, while every observation will be matched with exactly one time step (see Figure 5.5). This coincides with the underlying HMM model.

To see that the shortest path indeed corresponds to the maximum-likelihood registration, we again rely on monotonicity and the Markov property as follows:

$$\begin{aligned} \arg\min_{\tau} \sum_{i=1}^{n-1} -\log \mathcal{L}\left(\tau_{i+1} \mid \tau_i, Y, \Theta\right) = \\ \arg\max_{\tau} \prod_{i=1}^{n-1} \mathcal{L}\left(\tau_{i+1} \mid \tau_i, Y, \Theta\right) = \\ \arg\max_{\tau} \mathcal{L}\left(\tau \mid Y, \Theta\right) . \end{aligned} \tag{5.5}$$

Using either of the aforementioned model estimation methods, we can then estimate a distribution for a given time step from only the set of observations most likely to have been drawn from it. However, this approach still suffers from the issues associated with maximum likelihood approaches, namely local minima and sensitivity to noise.

One question that arises is what shortest path algorithm to use. Because the edge costs may be negative, Bellman-Ford [113] is a reasonable choice. However, we can relate this better to DTW and the forward-backward algorithm by noting that because the demonstration graph is a directed acyclic graph (DAG), so must be the tensor product graph. Because the graph is acyclic, we need

not perform the multiple passes usually required by the Bellmann-Ford algorithm thanks to the availability of a topological ordering. With this simplification, the Bellmann-Ford algorithm may be described by a recurrence:

$$C(v) = \bigoplus_{e}^{u \to v} C(u) \otimes c(e) \tag{5.6}$$

where

$$\begin{aligned} a \oplus_{\mathrm{ML}} b &= \min(a, b) \\ a \otimes_{\mathrm{ML}} b &= a + b \end{aligned} \tag{5.7}$$

with initial condition on the source vertex $s \in V$ set to the identity of $\otimes$, that is $\mathrm{R}(s) = 1_{\otimes}$. Expanding this out again yields the general DTW algorithm. The ML subscripts on these operators indicate that this choice produces maximum-likelihood estimates, and in the next section, we will show that simply changing our choice of semiring [114] (associative $\oplus$ and $\otimes$ with the distributive property) effectively yields the forward-backward algorithm.

### 5.3.3 Probability-weighted Temporal Registration Using the Forward-Backward Algorithm

In contrast to the Viterbi algorithm, the forward-backward algorithm [102] computes not only the most likely temporal registrations, but the posterior probabilities of all possible temporal registrations. Rather than presenting this approach in its original terms, however, we present an equivalent formulation using a graph algorithm that is more amenable to further modifications. The graph is the same as the one used in the previous section Section 5.3.2 and shown in Figure 5.5. However, to implement the forward-backward algorithm, we instead use:

$$\begin{aligned} a \oplus_{\mathrm{FB}} b &= -\log(e^{-a} + e^{-b}) \\ a \otimes_{\mathrm{FB}} b &= a + b \end{aligned} \tag{5.8}$$

which are simply the sum and product of the probabilities in negative log-space, producing (unnormalized) distributions over registrations given prior observations. Specifically,

$$\mathcal{L}\left(\tau_i = t \mid \Theta, \mathbf{y}_1, \ldots, \mathbf{y}_{i-1}\right) \propto e^{-\mathrm{C}(t,i)} \,. \tag{5.9}$$

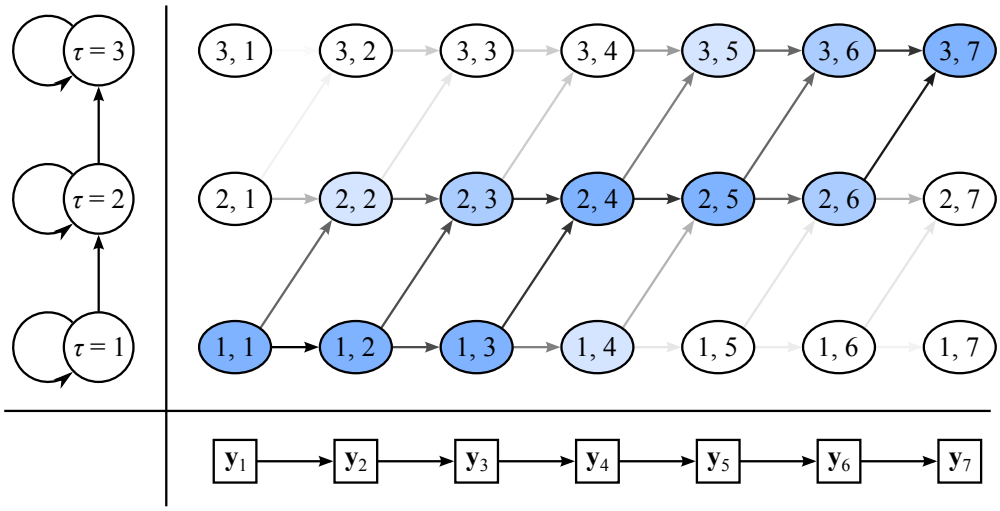



Figure 5.6: Tensor product of time graph (left) and demonstration graph (bottom). Darker edges are more probable and shaded vertices show the distribution of likely registrations in PTR.

To extend this to distributions over registrations given the entire sequence of observations, one need only compute the value $\mathrm{C}'(v)$ of each vertex in the reverse graph. The likelihood of an observation $\mathbf{y}_i$ registering to time step $t$ is then:

$$\mathcal{L}\left(\tau_i = t \mid \Theta, Y\right) \propto e^{-(\mathrm{C}(t,i)+\mathrm{C}'(t,i))} \,. \tag{5.10}$$

This view of these algorithms as recurrences over a tensor product graph enables us to further improve registrations by modifying the graph in Section 5.3.4.

### 5.3.4 Non-degenerate Temporal Registration

For many practical use cases, it is desirable to ensure that a sufficient number $K_\Delta$ of observations are registered to each time step. We say that such registrations are non-degenerate because if they are subsequently used to estimate covariance matrices as in prior learning methods [35, 16], this gaurantee is neccessary for the problem to be well-posed. Even with other choices of parameters or estimators, it may be desirable to enforce such a constraint because human demonstrators naturally perform precise parts of a task more slowly. This can be thought of as a stronger version of the Itakura parallelogram constraint used for Dynamic Time Warping [115].

This constraint can be enforced by modifying the structure of the graph used in Figure 5.6 to ensure that all paths satisfy the $K_\Delta$ constraint. An example of this modified graph structure is shown in Figure 5.7. The edge costs must then be modified similarly for the Viterbi and forward-backward

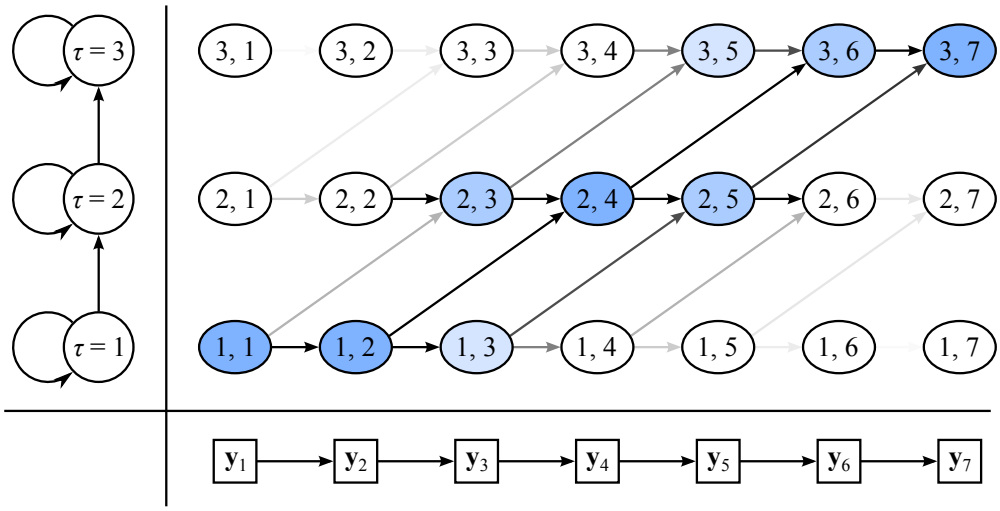



Figure 5.7: Tensor product of time graph (left) and demonstration graph (bottom) modified to exclude degenerate registrations ($K_\Delta = 2$). Darker edges are more likely and shaded vertices show the distribution of likely registrations.

algorithms:

$$c((t,i) \to (t',i')) = -(i'-i-1) \cdot \log p_{t,t} - \log p_{t,t'} - \sum_{k=i}^{i'-1} \log \mathcal{L}(\boldsymbol{\theta}_t \mid \mathbf{y}_k) \ . \tag{5.11}$$

Note that this cost is identical to that given in (5.4) when $i' = i+1$, that is, when the time step does not change.

The observation weights may then be computed from the vertex values using a similar approach to that used previously for the forward-backward algorithm:

$$\mathcal{L}(\tau_i = t \mid \boldsymbol{\theta}, Y) = e^{-\mathrm{C}(t,i)-\mathrm{C}'(t,i)} + \sum_{i'=i+1-K_\Delta}^{i-1} e^{-\mathrm{C}(t,i')-c((t,i')\to(t+1,i'+K_\Delta))-\mathrm{C}'(t+1,i'+K_\Delta)} \tag{5.12}$$

The combination of using the forward-backward algorithm and enforcing the non-degeneracy constraint gives us our complete method for probability-weighted temporal registration (PTR).

### 5.4 Application to Learning from Demonstrations

PTR has the potential to improve the the performance of many robotics algorithms that require temporal registration as a subroutine. For purposes of exposition and evaluation, we consider the application of PTR to the estimation of two specific task models from prior work on robot learning from demonstrations [35, 16]. These models were selected because they both explicitly require

temporal registration and each readily enable execution on a robot, either through application of a motion planner or definition of an explicit control policy.

First, consider a set of user-provided demonstrations $\{Y^{(1)}, \ldots, Y^{(m)}\}$ where each $Y^{(j)}$ is a sequence $\{\mathbf{y}_1^{(j)}, \ldots, \mathbf{y}_{n_j}^{(j)}\}$, and in general we will use a parenthesized superscript to indicate per-demonstration parameters. We reproduce the core learning algorithm of van den Berg et al. [35] in Algorithm 5.1 with small notational changes for parity and to highlight the underlying HMM model.

**Algorithm 5.1** EstimateVanDenBerg2010($T$, $Y^{(1)}, \ldots, Y^{(m)}$)

Initialize $R^{(j)} = \mathcal{I}$ and $\iota_t^{(j)} = \left\lfloor \frac{|Y^{(j)}| \cdot t}{T} \right\rfloor$.
**repeat**
  **for** $t = 1$ **to** $T$ **do**
    $Z_t = \left\{ (\mathbf{y}_i^{(j)}, R^{(j)}) \mid i = \iota_t^{(j)} \right\}$
  **end for**
  $\hat{\Theta} \leftarrow$ KalmanSmoother$(Z_1, \ldots, Z_n)$
  **for** $j = 1$ **to** $m$ **do**
    $R^{(j)} \leftarrow \arg\max_R \mathcal{L}\left(R \mid Y^{(j)}, \hat{\Theta}\right)$
    $\hat{\iota}^{(j)} \leftarrow \arg\max_\iota \mathcal{L}\left(\iota \mid Y^{(j)}, \hat{\Theta}\right)$ // Dynamic time-warping
  **end for**
**until** converged

Compare with this moderately abridged implementation of the method from Chapter 2 shown in Algorithm 5.2.

**Algorithm 5.2** EstimateBowen2015($T$, $Y^{(1)}, \ldots, Y^{(m)}$)

Initialize $\hat{\tau}_i^{(j)} = \left\lfloor \frac{T \cdot i}{|Y^{(j)}|} \right\rfloor$.
**repeat**
  **for** $t = 1$ **to** $T$ **do**
    $Z_t = \left\{ \mathbf{y}_i^{(j)} \mid \hat{\tau}_i^{(j)} = t \right\}$
    $\hat{\boldsymbol{\theta}}_t \leftarrow$ MaximumLikelihoodEstimator$(Z_t)$
  **end for**
  **for** $j = 1$ **to** $m$ **do**
    $\hat{\tau}^{(j)} \leftarrow \arg\max_\tau \mathcal{L}\left(\tau \mid Y^{(j)}, \hat{\Theta}\right)$ // Viterbi algorithm
  **end for**
**until** converged

Although each of these algorithms operates in a different space, we note that both fit the general mold of expectation-maximization (EM) methods, interleaving estimation of model parameters $\boldsymbol{\theta}$ and latent parameters ($\tau$ or $\iota$ and $R^{(j)}$). More accurately, however, these are maximization-

maximization methods, because their respective registration steps compute the best (in some sense) single registration given $\boldsymbol{\theta}$ rather than a distribution over all possible registrations.

To apply PTR to our method and that of van den Berg, we need only replace DTW and extend the model parameter expectation steps of these methods to handle weighted observations to accommodate probability-weighted temporal registrations. More specifically, when estimating $\boldsymbol{\theta}_t$, we use all $\mathbf{y}_i^{(j)}$, but weight each by its posterior likelihood of being registered to time step $t$, $\omega_{t,i}^{(j)} = \mathcal{L}\left(\tau_i^{(j)} = t \mid Y^{(j)}\right)$, which is given by (5.9). This produces a true expectation-maximization method, and in the method of Bowen et al., yields the Baum-Welch algorithm, which has the general effect of smoothing the estimation compared to the maximum likelihood approach, reducing (but not eliminating) local minima and improving robustness (see results in Section 5.5).

The transition probabilities in the HMM (Figure 5.3b) may be estimated by applying Bayes' rule, yielding:

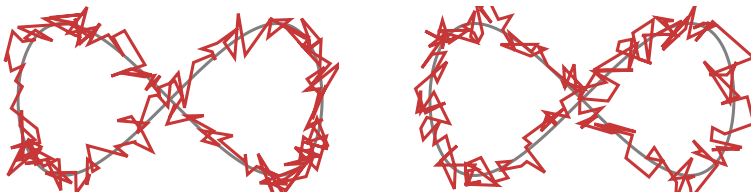
$$p_{t,t'} = \mathcal{L}\left(\tau_{i+1} = t' \mid \tau_i = t, Y\right) = \frac{\sum_{i,j} \omega_{t,i}^{(j)} \omega_{t',i+1}^{(j)}}{\sum_{i,j} \omega_{t,i}^{(j)}}. \tag{5.13}$$

In the method of van den Berg et al., at each time step $t$ the Kalman update step can be performed once for each observation $\mathbf{y}_i^{(j)}$, with weight $\omega_{t,i}^{(j)}$ applied by scaling the covariance matrix $R^{(j)}$ of the observation by $1/\omega_{t,i}^{(j)}$. Equivalently, and more efficiently, a single Kalman update step can be done at each time step $t$ by combining the weighted observations as follows:

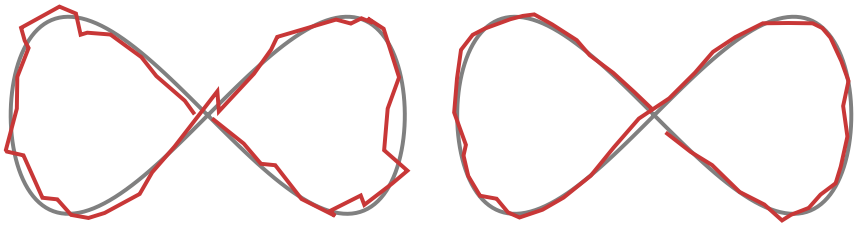
$$\hat{\Sigma}_t = \left(\sum_{i,j} \omega_{t,i}^{(j)} {R^{(j)}}^{-1}\right)^{-1}$$
$$\hat{\mu}_t = \hat{\Sigma}_t \left(\sum_{i,j} \omega_{t,i}^{(j)} {R^{(j)}}^{-1} \mathbf{y}_i^{(j)}\right)$$

### 5.5 Results

We evaluate the impact of our probability-weighted temporal registration (PTR) approach by applying it to several previously-developed robot learning methods that require temporal registration of demonstrations. In particular, we apply our temporal registration to the robot learning methods of van den Berg et al. [35] and Bowen et al. [16]. Throughout this section, we will use *Name*-$K_\Delta$ to indicate the various temporal registration methods (e.g. DTW-1 or PTR-12). Note that when $K_\Delta = 1$, as in DTW-1 or PTR-1, these are equivalent to their unconstrained variants, DTW and



(a) Demonstrations with medium Gaussian observation noise



(b) Trajectory learned with DTW-1 (c) Trajectory learned with PTR-3

Figure 5.8: Learning a simulated drawing task with reference trajectory shown in gray.

the forward-backward algorithm respectively. All computation was performed on an Intel Xeon E5-1680 CPU with 8 cores running at 3.40 GHz.

### 5.5.1 Applied to the van den Berg et al. Method

We first consider the method of van den Berg et al. [35] and the tasks of superhuman performance of a drawing task and a knot tying task.

**Simulated Drawing Task** For the simulated drawing task, instead of using human demonstrations, we perturbed a canonical figure eight using one of two noise models. The goal then, was to recover this canonical motion, allowing us to empirically evaluate different temporal registration approaches both in terms of learning time and error. The first noise model is that assumed by the underlying model (Eq. (4) in [35]), where observations within a demonstration are corrupted by independent and identically distributed Gaussian noise as seen in Figure 5.8. Results for low, medium, and high noise amplitudes with various numbers of demonstrations are shown in Figure 5.9.

We performed the same experiments with Brownian motion noise, effectively adding Gaussian velocity noise, as shown in Figure 5.10. While this does not match the assumptions of the underlying model, it produces demonstrations much more similar to what a human might. Results are shown in Figure 5.11.

Under both noise models, our method of PTR exhibited lower error relative to DTW, particularly with high noise and when only a small number of demonstrations are available. The improvements

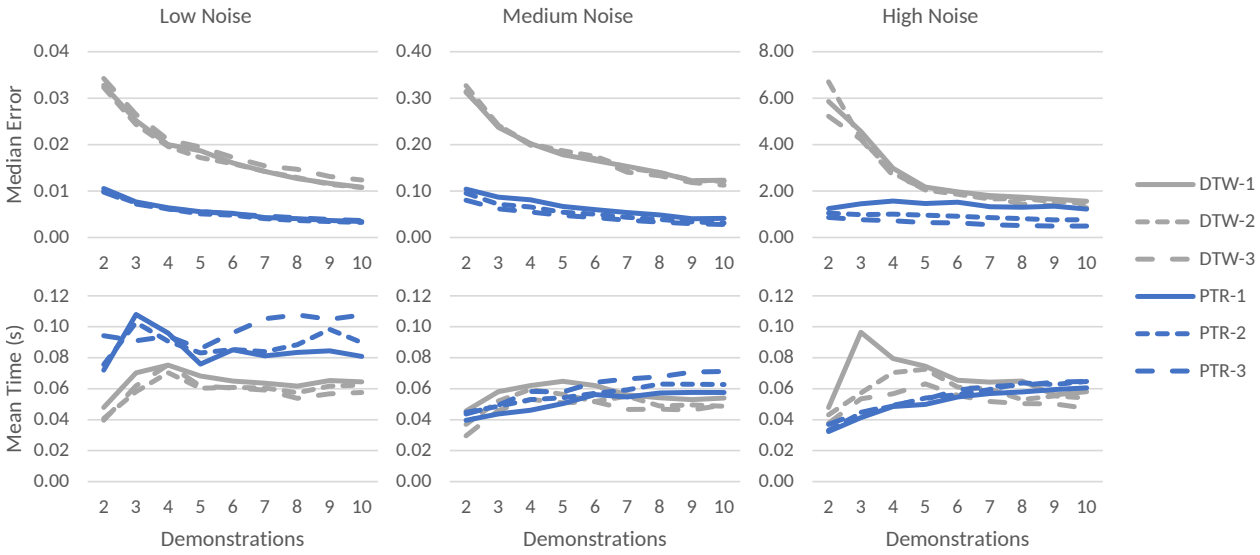
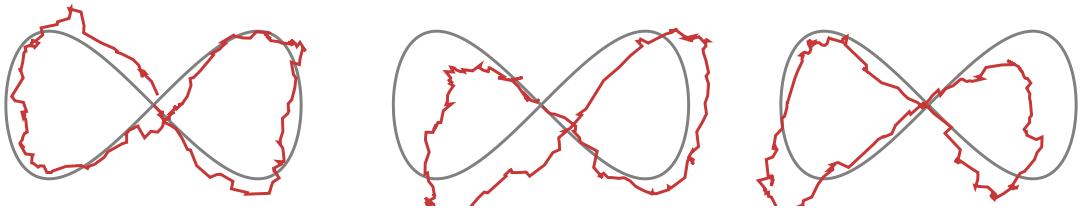
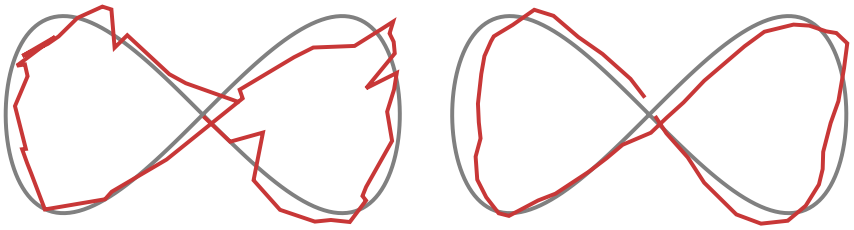



Figure 5.9: Results for the simulated drawing task with Gaussian observation noise. The standard deviations of the low, medium, and high noise variants were 0.05, 0.15, and 0.45 where the reference trajectory had unit height. PTR exhibited lower error relative to DTW, particularly when only a small number of demonstrations are available.



(a) Demonstrations with Brownian motion noise



(b) Trajectory learned with DTW-1 (c) Trajectory learned with PTR-3

Figure 5.10: Learning a simulated drawing task with reference trajectory shown in gray.

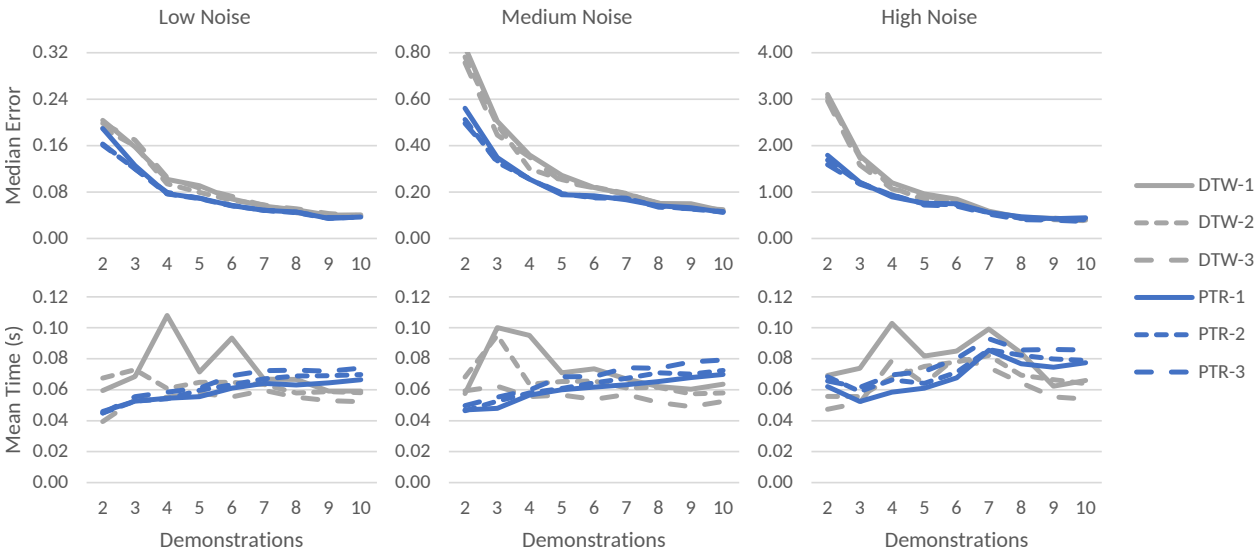



Figure 5.11: Results for the simulated drawing task with Brownian motion noise. The infinitesimal variances of the low, medium, and high noise variants were 0.05, 0.15, and 0.45 where the reference trajectory had unit height. PTR exhibited lower error relative to DTW, particularly when only a small number of demonstrations are available.

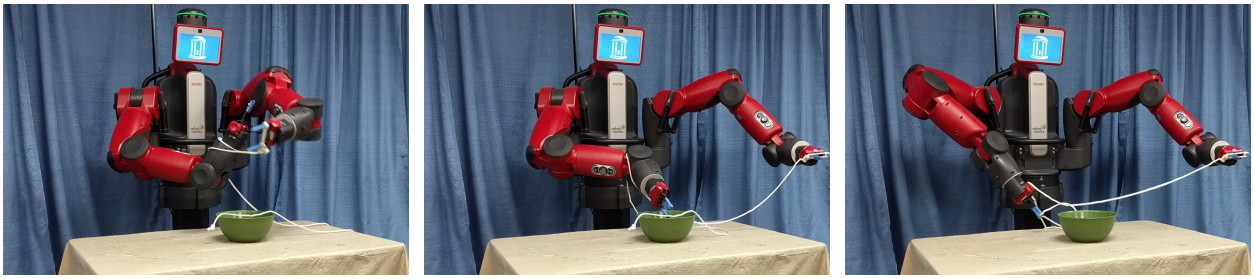


Figure 5.12: Using the method of van den Berg et al. [35] we learned a task wherein the Baxter robot tied a knot significantly faster than demonstrated.

were most dramatic when the noise model matched the underlying method assumptions because noise which violates the independence assumptions introduces bias that temporal registration alone cannot correct. Both approaches required comparable learning time.

**Physical Knot Tying Task** Our physical knot tying task was similar to that described by van den Berg et al. However, it was demonstrated and executed on the Baxter robot, which has more restrictive dynamics limitations than the Berkeley Surgical Robot on which the original experiments were performed. As in the original paper, we divided the task into three phases, an initial loop, a grasp, and an extraction (see Figure 5.12), but unlike in that paper, we learned all three phases rather than only the first and third. We performed five demonstrations at 20 Hz of the first two phases

| Phase | Registration | Max Motion Speedup | Learning Time (s) |
|---|---|---|---|
| 1 | DTW-1 | 2.5 | 0.40 |
| | PTR-1 | 2.3 | 0.64 |
| | DTW-3 | 2.4 | 0.33 |
| | PTR-3 | **3.3** | 0.52 |
| 2 | DTW-1 | 2.2 | 0.05 |
| | PTR-1 | 3.0 | 0.35 |
| | DTW-3 | 2.7 | 0.04 |
| | PTR-3 | **3.1** | 0.19 |
| 3 | DTW-1 | 1.1 | 0.85 |
| | PTR-1 | - | 0.57 |
| | DTW-3 | 2.0 | 0.35 |
| | PTR-3 | **3.2** | 1.20 |

Table 5.1: Results for the knot-tying task using the method of van den Berg et al. with different temporal registration approaches. Motion speedup indicates the maximum multiple of the average demonstration speed at which the task was still performed successfully.

and three of the third. For an execution to be considered successful, we required the robot to tie a slip knot without exceeding its kinematic or dynamics limitation, including avoiding self-collisions. To evaluate methods in the context of van den Berg et al.'s superhuman performance, we executed the algorithms at progressively greater speeds until the method failed and recorded the maximum multiple of the average demonstration speed at which the task was still performed successfully. Results separated by task phase are shown in Table 5.1.

A minimum of three samples was used for the non-degenerate variant, which corresponds to a 0.15 second window in each demonstration. In the first phase of the task, DTW-1 failed first

Figure 5.13: Using the method of Bowen et al. [16] we learned a task wherein the Baxter robot scooped powder from the yellow container and transferred it into the magenta one without spilling while avoiding obstacles in the environment.

| Demonstrations | Registration | Success | Learning Time (s) |
|---|---|---|---|
| 10 | Viterbi-1 | 80% | 2.86 |
| | PTR-1 | 90% | 78.64 |
| | PTR-12 | **100%** | 87.31 |
| 5 | Viterbi-1 | 60% | 3.16 |
| | PTR-1 | 80% | 36.37 |
| | PTR-12 | **90%** | 36.12 |

Table 5.2: Results for the powder transfer task using the method of Bowen et al. [16] with different demonstration counts and temporal registration approaches. PTR-12 had the highest success rate, regardless of the number of demonstrations.

due to incorrect placement of the rope, while PTR-1 and DTW-3 encountered self-collisions. Only PTR-3 reached the velocity limits of the robot. In the second and easiest phase, the cause of failure for every temporal registration method was reaching the velocity limits of the robot. However, some methods resulted in smoother registrations and subsequent motions, permitting overall faster execution. In the third and most difficult phase, DTW-1 exceeded the velocity limits of the robot while the other three methods failed to extract the arm through the newly-formed loop in the rope. In the case of PTR-1, even at 1x demonstration speed, this was the cause of failure because the registration failed to isolate a crucial part of the task. PTR-3 successfully achieved the highest speedup for all phases of the knot-tying task, showing the benefits of probability-weighted temporal registration with the non-degenerate registration feature.

### 5.5.2 Applied to the Bowen et al. Method

We next consider the method of Bowen et al. [16], which used the Viterbi algorithm for temporal registration, so it is this approach we compare against.

We performed a powder transfer task on the Baxter robot shown in Figure 5.13, as specified in [16]. In this task, the robot is to scoop powder onto a spoon from a source container (the yellow bucket) and transfer it to a destination container (the magenta thermos) while avoiding obstacles (e.g., the plant on the table and the white hanging lamp shade). The robot learned the task using the method of Bowen et al. [16] from 10 kinesthetic demonstrations (in which no obstacles were present). Task models were learned using three different temporal registration approaches. To evaluate each model, we introduced the obstacles and randomly sampled scenarios with container

positions such that the transfer would always cross the centerline of the table to ensure each scenario was challenging. An execution was considered successful if it transferred powder from one container to the other without spilling. Results are shown in Table 5.2.

As with the van den Berg method, the use of PTR yielded a measurably better task model in terms of success rate relative to the Viterbi algorithm for temporal registration. Learning times for PTR were substantially longer, but still very reasonable for an off-line process. The primary cause of the failures that occurred during task execution was slightly missing the cup during the dumping motion, resulting in spilled powder. We observe that the number of demonstrations had a marginally greater impact when using the Viterbi algorithm for registration than when using PTR. As with the method of van den Berg et al., PTR-12 performed best.

## 5.6 Conclusion

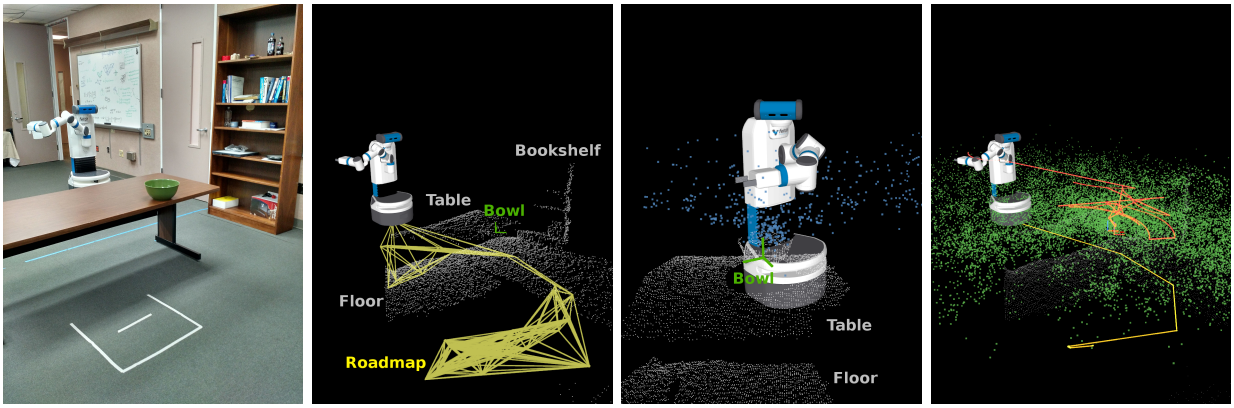
Many existing methods for robot learning from demonstrations, including those presented in Chapters 2, 3, and 4, require registering a time sequence of observations to a reference model, either for aligning demonstrations during preprocessing or as an integral part of task model estimation. We introduced probability-weighted temporal registration (PTR), a more general form of temporal registration that explicitly captures uncertainty in the registration. Instead of assuming each observation is registered to (at most) one time step of the reference model like DTW, we use the forward-backward algorithm to compute probability-weighted assignments and avoid degenerate registrations. We applied PTR to two learning methods from prior work on both simulated and physical tasks and showed that incorporating PTR into robot learning algorithms can yield higher-quality task models that enable faster task executions and higher task success rates. In the following chapter, we return to the problem of planning using the learned model, but in the domain of mobile manipulation.

CHAPTER 6

## Motion Planning for Learned Mobile Manipulation Tasks

In this chapter, we introduce Task-Guided Gibbs Sampling (TGGS) to accelerate motion planning in the context of learned mobile manipulation tasks by utilizing the same task model described in Chapter 2. TGGS enables the motion planner to more quickly sample configurations that are likely to be important for task success without relying on manual heuristics, thereby producing successful plans faster than sampling methods that do not explicitly consider the task and environment together, like goal-biasing or obstacle-biasing.

Our method depends on the task model which is used by the motion planner to generate plans that successfully perform the task to also inform the sampling strategy. Specifically, the method we propose combines local optimization with a Markov Chain Monte Carlo (MCMC) approach to effectively project the task space distribution learned from the demonstrations into the robot's configuration space. This sampling strategy is applicable to both the bounded subspace of the




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

Figure 6.1: **(a)** The Fetch robot's task is to pour liquid from the grasped pitcher into the green bowl on the table. **(b)** Roadmap of base motions biased towards those most likely to be useful (yellow) while avoiding the obstacle point cloud (white). **(c)** End-effector positions for sampled arm configurations biased towards those most likely to be useful (blue). **(d)** End-effector positions for configurations in the hybrid roadmap (green) with the base (yellow) and end-effector (red) motion of the final plan.

manipulator as well as the unbounded subspace of the mobile base. While we apply this approach to our task model and the Fetch robot is used in the experiments, we believe TGGS could also be adapted to a number of other models and robots.

Like many low-cost mobile manipulators, the Fetch robot's mobile base uses differential drive, which presents the challenge of nonholonomy. Fast motion planning for nonholonomic robots with 10 or more degrees of freedom remains computationally difficult, especially when asymptotic optimality is desirable, as when soft task constraints are present. Fortunately, the configuration spaces of many mobile manipulators, including the Fetch robot, have a large subspace that *is* holonomic: that of the manipulator arm. TGGS leverages this property by partitioning the sampling across holonomic and nonholonomic degrees of freedom, which accelerates sampling-based motion planners for learned mobile manipulation tasks while preserving their asymptotic optimality guarantees.

We validate our new approach analytically and then empirically using the Fetch robot to perform two household tasks in randomly-generated scenarios.

This chapter is based on work previously published in [21].

### 6.1 Related Work

Motion planning for mobile manipulation is challenging because of the high number of degrees of freedom. Various methods that lack completeness address this problem, including potential fields [12, 14] and local optimization [59]. However, completeness guarantees are important in many applications involving complex environments and higher-dimensional spaces. Probabilistically complete sampling-based motion planners that build roadmaps or trees in the robot's configuration space are effective in such spaces [116, 117] and can incorporate a class of task constraints [29, 53], even at reactive rates using elastic roadmaps [118].

Only some sampling-based motion planners consider plan cost, and asymptotic optimality guarantees become important when planning for tasks where feasibility is no longer sufficient for success [106]. But in high-dimensional spaces, the convergence of asymptotically optimal motion planners may be impractically slow. Even when only completeness is required, biased sampling may improve performance [119], particularly for challenging problems with narrow passages [120, 121, 122, 123]. Similarly, cost-based task constraints [124] may induce narrow passages in cost space that make planning more challenging, and biased sampling is again effective [27, 125, 126, 17].

In this chapter, we introduce a new sampling strategy that can be used for efficiently constructing motion planning roadmaps for learned mobile manipulation tasks while retaining asymptotic optimality of the motion planner. We apply this approach to one task model here (from Chapter 2), although it could be adapted to motion planning for tasks represented by other models that may be learned from demonstrations [127, 9].

Additionally, our approach relies on decomposing the configuration space of the robot. Related approaches have been considered in the contexts of hierarchical planning [128, 129] and hybrid planning [130], but without asymptotic optimality guarantees. We also note that high-level methods have considered this problem in a more traditional task planning framework [131, 132, 133], but these approaches depend on methods that can compute motion plans for low-level actions, which is precisely the domain of our method.

### 6.2 Problem Definition

We consider a mobile manipulator that consists of a holonomic arm with $d_{\text{arm}}$ degrees of freedom and a mobile base (which may be holonomic or nonholonomic) with $d_{\text{base}}$ degrees of freedom. Given a set of arm configurations $\mathcal{Q}_{\text{arm}} \subset \mathbb{R}^d_{\text{arm}}$ and a set of base configurations $\mathcal{Q}_{\text{base}} \subseteq \mathbb{R}^d_{\text{base}}$, let $\mathcal{F}_{\text{arm}} \subseteq \mathcal{Q}_{\text{arm}}$ denote those arm configurations which are potentially free for some base configuration (i.e., those that are not in self-collision). Similarly, let $\mathcal{F}_{\text{base}} \subseteq \mathcal{Q}_{\text{base}}$ denote those base configurations which are potentially free for some arm configuration (i.e., those for which the base itself does not collide with the environment). Finally, let $\mathcal{Q} = \mathcal{Q}_{\text{arm}} \times \mathcal{Q}_{\text{base}} \subset \mathbb{R}^d_{\text{arm}} + d_{\text{base}}$ denote the combined configuration space and $\mathcal{F} \subseteq \mathcal{F}_{\text{arm}} \times \mathcal{F}_{\text{base}}$ denote the free configuration space.

In addition to the above sets, we assume as input to the method an initial configuration $\mathbf{q}_0 \in \mathcal{F}$ and a task model. The task model depends on a feature function $\phi_{\mathbf{a}}(\mathbf{q}) \in Y \subseteq \mathbb{R}^b$ where $\mathbf{a}$ is an environment description that encodes the positions of salient objects (e.g. the bowl in Figure 6.1). We call $Y$ the feature space, and the task model consists of a finite sequence of feature space multivariate Gaussian distributions $\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2, \ldots, \boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T\}$. One can think of these feature space distributions as defining distributions over configurations when pulled back along $\phi$, conditional on a given environment described by $\mathbf{a}$. Task models of this form can be estimated from expert demonstrations for a variety of useful tasks [16, 127, 9].

The output of the motion planner is a feasible configuration space path $\pi$ comprising a sequence of local plans which maximizes similarity to this task model. In this chapter, we use the specific

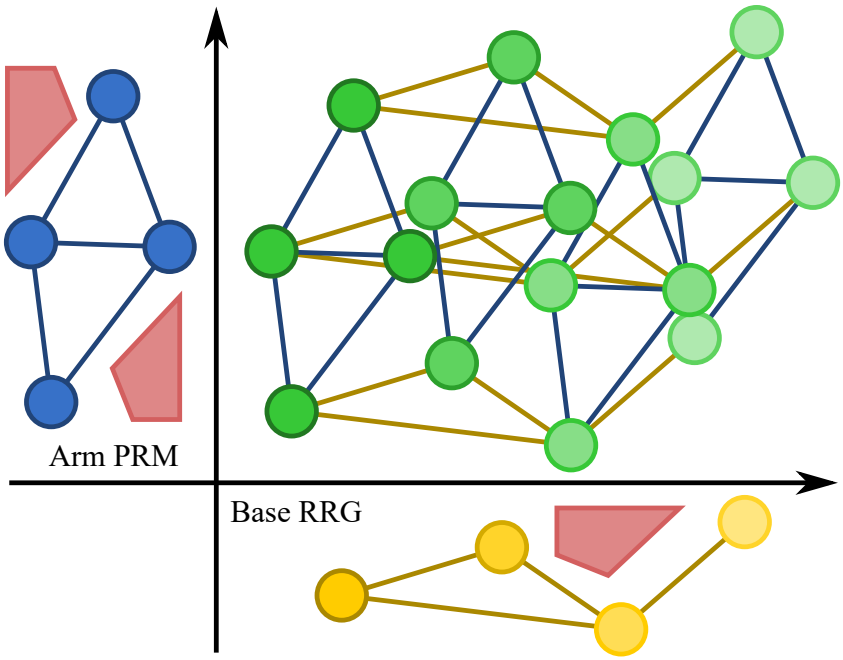



Figure 6.2: Hybrid roadmap (green) formed by the Cartesian product of an arm PRM (blue) and a base RRG (yellow) while avoiding obstacles (red).

definition of similarity described in Chapter 2 which we summarize in Section 6.3.2.

## 6.3 Motion Planning using the Learned Model

The problem of planning motions for mobile manipulation tasks presents a number of challenges but also a degree of structure. Notably, any mobile base that uses differential drive (e.g. the Fetch robot) is nonholonomic and thus so is the robot as a whole. This precludes a large class of motion planners [134], including probabilistic roadmaps with linear edges. However, the subspace of configuration space that represents only the arm *is* holonomic.

In Section 6.3.1, we first describe and analyze a hybrid roadmap that leverages this inherent structure by using different roadmap constructions for different subspaces of the problem as appropriate. Next, in Section 6.3.2, we apply a learned task model to this hybrid roadmap. Then, in Section 6.3.3, we describe how the nature of this construction lends itself to efficient sampling of high-quality configurations in both roadmaps using Gibbs sampling.

### 6.3.1 Mobile Manipulation Hybrid Roadmap

Mobile manipulators are generally characterized by low-dimensional base motion, usually SE(2), combined with a high-dimensional manipulator, $\mathbb{R}^5(S^1)^3$ in the case of the Fetch robot. The combined configuration space is often 11-dimensional or greater. Additionally, we are not interested

in merely feasible plans but ones which also have low costs, ideally approaching the optimal solution in the limit.

Asymptotically-optimal variants of rapidly-exploring random graphs, e.g RRG* [3], are applicable to such motion planning problems. However, in high-dimensional spaces, these planners can exhibit very slow convergence to optimality. On the other hand, less general methods like probabilistic roadmaps (e.g. PRM*) may exhibit faster convergence for multiple queries but often rely on additional structure, like an optimal local planner. Furthermore, applying PRM* to unbounded spaces, like SE(2), necessitates additional heuristics because new configurations will be added from the entire space unlike RRG* which only extends to configurations near those already explored.

We show that these planners can be combined to plan for mobile manipulation tasks in such a way that their relative strengths are exploited in the subspaces where they are most applicable. Specifically, we propose to build an RRG* of pure base motions and a separate PRM* of pure manipulator motions. We then search in the Cartesian product of the resulting graphs to find asymptotically-optimal motion plans. Additionally, this approach allows for eager collision checking on the base during roadmap construction but lazily checking for arm collisions during the shortest path search.

Let $C_{\text{base}}(e_{\text{base}}, \mathbf{q}_{\text{arm}})$ denote the cost of traversing edge $e_{\text{base}}$ in the base roadmap with arm configuration $\mathbf{q}_{\text{arm}}$ and $C_{\text{arm}}(\mathbf{q}_{\text{base}}, e_{\text{arm}})$ denote the cost of traversing edge $e_{\text{arm}}$ in the arm roadmap at base configuration $\mathbf{q}_{\text{base}}$. Every edge in the Cartesian product graph is composed of a vertex from one graph and an edge from the other. This implies that the arm and base are never actuated simultaneously. Let $C(e)$ be either $C_{\text{base}}$ or $C_{\text{arm}}$ as applicable. We assume $C_{\text{base}}$ and $C_{\text{arm}}$ are non-negative and Lipschitz continuous. We extend this cost to a path $\pi$ in the obvious way:

$$C(\pi) = \sum_{e \in \pi} C(e).$$

**Algorithm 6.1** BUILDHYBRIDROADMAP($\mathbf{q}_0$, $n_{\text{base}}$, $n_{\text{arm}}$)

**Input:** Initial configuration $\mathbf{q}_0$, number of samples $n$
$\mathcal{G}_{\text{base}} \leftarrow$ BUILDBASEROADMAP($\mathcal{F}_{\text{base}}$, $\mathbf{q}_0$, $n_{\text{base}}$)
$\mathcal{G}_{\text{arm}} \leftarrow$ BUILDARMROADMAP($\mathcal{F}_{\text{arm}}$, $\mathbf{q}_0$, $n_{\text{arm}}$)
**return** $\mathcal{G}_{\text{base}} \times \mathcal{G}_{\text{arm}}$ // graph Cartesian product

**Theorem 2.** *If both* BUILDBASEROADMAP *and* BUILDARMROADMAP *are asymptotically-optimal with respect to $C_{base}$ and $C_{arm}$ for all $\mathbf{q}_{arm}$ and $\mathbf{q}_{base}$ respectively, then* BUILDHYBRIDROADMAP *(see Algorithm 6.1) is asymptotically-optimal with respect to $C$ with the restriction that only the base or arm are actuated at any given time.*

*Proof.* For any $\epsilon > 0$, consider a feasible path $\pi$ such that $C(\pi) < C^* + \frac{\epsilon}{2}$ where $C^*$ is the infimum of costs among feasible paths. If no such path exists, then the theorem holds vacuously. Let $\pi_{\text{base}}$ denote the set of edges in $\pi$ which correspond to base motions and $\pi_{\text{arm}}$ those which correspond to arm motions. Observe that, by associativity, $C(\pi) = C(\pi_{\text{base}}) + C(\pi_{\text{arm}})$.

We next note that because the base and arm do not actuate simultaneously, $\pi_{\text{base}}$ can be divided up into no more than $|\pi|$ sub-paths, each with $\mathbf{q}_{\text{arm}}$ constant in each sub-path. Let $K_{\text{base}}$ denote the Lipschitz constant of $C_{\text{base}}$. By assumption, for all $\delta > 0$ there exists $m_{\text{arm}}$ such that BUILDARMROADMAP($\mathcal{F}_{\text{arm}}$, $\mathbf{q}_0$, $m_{\text{arm}}$) includes a reachable configuration $\mathbf{q}'_{i,\text{arm}}$ that is $\frac{\epsilon}{8K_{\text{base}}|\pi|}$-close to a given $\mathbf{q}_{i,\text{arm}}$ with probability $1 - \frac{\delta}{4|\pi|}$. Thus, with probability at least $(1 - \frac{\delta}{4|\pi|})^{|\pi|} \geq 1 - \frac{\delta}{4}$ this holds for each such $\mathbf{q}_{\text{arm}}$.

Now, consider one of these sub-paths $\pi_{i,\text{base}}$ with shared $\mathbf{q}_{i,\text{arm}}$. Again by assumption, there exists $n_{\text{base}}$ such that with probability at least $1-\frac{\delta}{4|\pi|}$ there is a path $\pi'_{i,\text{base}}$ in BUILDBASEROADMAP($\mathcal{F}_{\text{base}}$, $\mathbf{q}_0$, $n_{\text{base}}$) with $C(\pi'_{i,\text{base}}, \mathbf{q}_{i,\text{arm}}) < C(\pi_{i,\text{base}}, \mathbf{q}_{i,\text{arm}}) + \frac{\epsilon}{8|\pi|}$. By Lipschitz continuity of $C_{\text{base}}$, we have $C(\pi'_{i,\text{base}}, \mathbf{q}'_{i,\text{arm}}) < C(\pi'_{i,\text{base}}, \mathbf{q}_{i,\text{arm}}) + K_{\text{base}}\frac{\epsilon}{8K_{\text{base}}|\pi|} = C(\pi'_{i,\text{base}}, \mathbf{q}_{i,\text{arm}}) + \frac{\epsilon}{8|\pi|} < C(\pi'_{i,\text{base}}, \mathbf{q}_{i,\text{arm}}) + \frac{\epsilon}{4|\pi|}$. Together, these sub-paths form $\pi'_{\text{base}}$ with $C(\pi'_{\text{base}}) < C(\pi_{\text{base}}) + \frac{\epsilon}{4}$ with probability at least $(1 - \frac{\delta}{4})^2 \geq 1 - \frac{\delta}{2}$.

The above argument applies similarly to $\pi_{\text{arm}}$, with $\pi'_{\text{arm}}$, $m_{\text{base}}$, and $n_{\text{arm}}$. So, for $n = \max(m_{\text{base}}, n_{\text{base}}, m_{\text{arm}}, n_{\text{arm}})$, with probability at least $(1 - \frac{\delta}{2})^2 \geq 1 - \delta$ there exists a path $\pi'$ in BUILDHYBRIDROADMAP($\mathbf{q}_0, n, n$) such that $C(\pi') = C(\pi'_{\text{base}}) + C(\pi'_{\text{arm}}) < C(\pi) + \frac{\epsilon}{2} < C^* + \epsilon$.

■ □

The condition that BUILDBASEROADMAP be asymptotically optimal with respect to $C_{\text{base}}$ for all $\mathbf{q}_{\text{arm}}$ and the corresponding condition for BUILDARMROADMAP may initially seem too strong. However, these conditions are satisfied with many useful metrics for mobile manipulators. In the experiments we performed, we let $C_{\text{base}}(\pi_{\text{base}})$ be a non-negative scalar multiple of the time required to execute $\pi_{\text{base}}$, which is independent of $\mathbf{q}_{\text{arm}}$ and thus the assumption holds using an RRG* with


Figure 6.3: **(a)** Hidden Markov model defining a distribution of sequences of feature space vectors with the hidden state comprising discrete time steps. **(b)** Restricted structure of the hidden Markov model.

a closed-form local planner to optimally satisfy the non-holonomic constraints [135]. Because the manipulator arm is holonomic, asymptotic optimality of PRM* holds for all non-singular $C_{\text{arm}}$ by local metric equivalence, so the dependence on $\mathbf{q}_{\text{base}}$ is similarly unproblematic. More generally, however, asymptotic optimality of the method relies on that of the underlying methods, which may be chosen based on the requirements imposed based on the cost metrics.

The Cartesian product graph of these roadmaps need not be explicitly constructed. Rather an implicit representation can be lazily traversed by simply traversing the constituent graphs. Note also that because $\mathcal{F}$ may be a subset of $\mathcal{F}_{\text{base}} \times \mathcal{F}_{\text{arm}}$, collision checking must still be performed on the edges of the hybrid roadmap. In our implementation, this is done lazily during the graph search described in the following subsection.

### 6.3.2 Task Roadmap

To extend the hybrid roadmap described above to motion planning for a task, we use another Cartesian product to construct a spatio-temporal roadmap as described in Chapter 3 and Algorithm 6.2. This approach extends an asymptotically-optimal planner to maximize similarity to a learned task model comprising a finite sequence of *time steps* by permitting the cost metric to depend on task progress in the form of the current time step.

**Algorithm 6.2** MotionPlanForTask($\mathbf{q}_0$, $n_{\text{base}}$, $n_{\text{arm}}$, $T$)

**Input:** Initial configuration $\mathbf{q}_0$, numbers of samples $n_{\text{base}}$ and $n_{\text{arm}}$, number of time steps $T$
$\mathcal{G}_{\text{s}} \leftarrow$ BuildHybridRoadmap($\mathbf{q}_0$, $n_{\text{base}}$, $n_{\text{arm}}$)
$\mathcal{G}_{\text{t}} \leftarrow$ TaskStructure($T$) // Graph of time steps: linear graph with T vertices
$v_0 \leftarrow$ NearestVertex($\mathcal{G}_{\text{s}}$, $\mathbf{q}_0$)
**return** ShortestPath($\mathcal{G}_{\text{s}} \times \mathcal{G}_{\text{t}}$, $(v_0, 1)$, $\mathcal{V}(\mathcal{G}_{\text{s}}) \times \{T\}$)

The construction of the spatio-temporal roadmap $\mathcal{G}_{\text{st}} = \mathcal{G}_{\text{s}} \times \mathcal{G}_{\text{t}}$ is necessitated by our choice for $C_{\text{arm}}$, which depends on the current time step $t$. The specific choice of $C_{\text{arm}}$ depends on the learned

task model. In this chapter, we consider the model defined in Chapter 3 by a time-homogeneous hidden Markov model (HMM) with a restricted structure and discrete state space of time steps that capture task progress [16]. We denote the time step by $t$ and number them sequentially $1, 2, \ldots, T$ (see Figure 6.3).

Under this model, an observation is a feature space vector $\mathbf{y} \in Y$. Recall that $\mathbf{y} = \phi_{\mathbf{a}}(\mathbf{q})$ lifts a configuration $\mathbf{q}$ into this feature space conditional on an environ description $\mathbf{a}$ that encodes the positions of salient objects in the environment. This transformation enables the model to capture relationships between the environment and configurations, e.g. that the pitcher must be placed over the bowl before pouring in Figure 6.1. The specific choices of $\phi_{\mathbf{a}}$ used for the experiments are discussed in their respective subsections of Section 6.4, but the method requires only that it be differentiable. We model the observation distribution by a multivariate Gaussian distribution with a per-time step mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$.

$$\phi_{\mathbf{a}_i}(\mathbf{q}_i) = \mathbf{y}_i \sim \mathcal{N}(\boldsymbol{\mu}_{t_i}, \boldsymbol{\Sigma}_{t_i})$$

In general, the pullback of such a distribution in feature space along $\phi_{\mathbf{a}}$ is improper, which is nevertheless sufficient for planning. However, we note that if $\phi_{\mathbf{a}}$ is a submersion, then the pullback is a proper distribution in configuration space.

Because the model was estimated from successful demonstrations, we expect a plan which is similar to the task model to be likely to accomplish the task. Following this intuition, we define edge costs $C_{\text{arm}}$ that when minimized, maximizes the probability density function of the learned model. To do so, we first assign a cost to individual configurations for each time step as follows:

$$\begin{aligned} c(\mathbf{q}, t) &= -\log p(\phi(\mathbf{q}) \mid \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \\ &= \frac{1}{2}\left((\phi(\mathbf{q}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \boldsymbol{\Sigma}_t^{-1} (\phi(\mathbf{q}) - \boldsymbol{\mu}_t) + \log\det 2\pi\boldsymbol{\Sigma}_t\right). \end{aligned}$$

Then, we let $C_{\text{arm}}$ be simply the path integral over this cost $\int c(\mathbf{q}(s), t)\,\mathrm{d}s$. This is the continuous analog of the summation that arises from the Markov assumption made by the task model.

Specifically, rather than the following distribution over discrete paths:

$$p(\mathbf{q}_0, \ldots, \mathbf{q}_n \mid t) \propto \prod_{i=0}^{n} p(\mathbf{q}_i \mid t) = e^{-\sum_{i=0}^{n} c(\mathbf{q}_i, t)},$$

we instead define a distribution using a functional over configuration space curves:

$$p(\mathbf{q} \mid t) \propto e^{-\int c(\mathbf{q}(s), t)\, \mathrm{d}s}.$$

Due to the structure of the spatio-temporal roadmap $G_{\mathrm{st}}$, the cost of the same local plan associated with an edge in the hyrbid roadmap is often needed under multiple metrics, one for each time step. This structure allows the computation to be accelerated by a pre-computation as follows:

$$\begin{aligned}
& (\phi(\mathbf{q}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \boldsymbol{\Sigma}_t^{-1} (\phi(\mathbf{q}) - \boldsymbol{\mu}_t) \\
= \;& \mathrm{Tr}\left( (\phi(\mathbf{q}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \boldsymbol{L}_t \boldsymbol{L}_t^{\mathrm{T}} (\phi(\mathbf{q}) - \boldsymbol{\mu}_t) \right) \\
= \;& \mathrm{Tr}\left( \boldsymbol{L}_t^{\mathrm{T}} (\phi(\mathbf{q}) - \boldsymbol{\mu}_t)(\phi(\mathbf{q}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \boldsymbol{L}_t \right) \\
= \;& \mathrm{Tr}\left( \boldsymbol{L}_t^{\mathrm{T}} \left( \phi(\mathbf{q})\phi(\mathbf{q})^{\mathrm{T}} - 2\boldsymbol{\mu}_t \phi(\mathbf{q})^{\mathrm{T}} + \boldsymbol{\mu}_t \boldsymbol{\mu}_t^{\mathrm{T}} \right) \boldsymbol{L}_t \right)
\end{aligned}$$

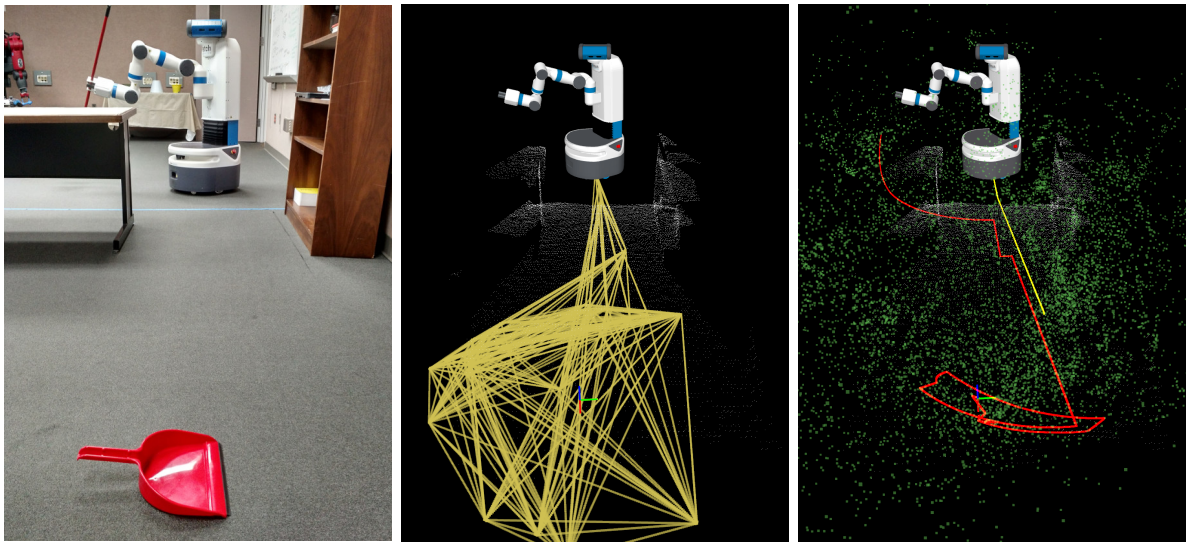
where $\boldsymbol{L}_t \boldsymbol{L}_t^{\mathrm{T}} = \boldsymbol{\Sigma}_t^{-1}$ is the Cholesky decomposition of $\boldsymbol{\Sigma}_t^{-1}$. So,

$$\begin{gathered}
\textstyle\int_0^S c(\mathbf{q}(s))\, \mathrm{d}s = \\
S \log\det 2\pi\boldsymbol{\Sigma}_t + \textstyle\int_0^S (\phi(\mathbf{q}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \boldsymbol{\Sigma}_t^{-1} (\phi(\mathbf{q}) - \boldsymbol{\mu}_t)\, \mathrm{d}s = \\
S \log\det 2\pi\boldsymbol{\Sigma}_t + \mathrm{Tr}\left( \boldsymbol{L}_t^{\mathrm{T}} \left( \boldsymbol{A} - 2\mathbf{b}\boldsymbol{\mu}_t^{\mathrm{T}} + S\boldsymbol{\mu}_t \boldsymbol{\mu}_t^{\mathrm{T}} \right) \boldsymbol{L}_t \right) \\
\text{where } \boldsymbol{A} = \textstyle\int_0^S \phi(\mathbf{q})\phi(\mathbf{q})^{\mathrm{T}}\, \mathrm{d}s \text{ and } \mathbf{b} = \textstyle\int_0^S \phi(\mathbf{q})\, \mathrm{d}s\,.
\end{gathered} \tag{6.1}$$

Note that $\boldsymbol{A}$ and $\mathbf{b}$ are independent of $t$ and can thus be precomputed numerically for a local path. Once this is done, edge costs may be computed using only simple matrix operations, making them very efficient to compute.

Under these definitions of $C_{\mathrm{arm}}$ and $C_{\mathrm{base}}$, the cost of a plan can be rewritten as

$$C(\pi) = \int_0^S g(\pi(s), t)\, \mathrm{d}s \tag{6.2}$$



(a) (b) (c)

Figure 6.4: **(a)** Sweeping task using the Fetch robot. **(b)** Biased RRG* of base motions (yellow) around the obstacle point cloud (white). **(c)** Hybrid roadmap (green) with the base (yellow) and end-effector (red) motion of the final plan.

where $g(\mathbf{q}, t) = \kappa$ if the base is moving at time $s$ and $g(\mathbf{q}, t) = c(\mathbf{q}, t)$ if the arm is moving. $\kappa$ is the weight with which base movement is penalized.

### 6.3.3 Task-Guided Gibbs Sampling

While the method described in the previous section is asymptotically optimal, it converges very slowly in high-dimensional configuration spaces. To improve convergence, we employ biased sampling. Motivated by importance sampling, our goal is to effectively project the task model from feature space into configuration space. To do so explicitly would require strong assumptions about the form of $\phi$. Rather, we observe that we only need to *sample* from the projected distribution. Furthermore, these samples do not need to be independent of each other, only approximate the desired distribution in the limit. Following this insight, we propose to use a Markov Chain Monte Carlo sampling strategy. Specifically, we employ a variant of Gibbs sampling tightly integrated with the hybrid nature of the roadmap.

Gibbs sampling is simply an approach to sampling from the joint distribution of two dependent variables by fixing one and sampling the other conditional on that value, then repeating that process with the variables' roles reversed. Following this approach, we sample individual base and arm configurations separately while the distributions as a whole are interdependent. To sample an arm

configuration, we first select a base configuration $\hat{\mathbf{q}}_{\text{base}}$ at random from the base roadmap $G_{\text{base}}$. We then sample a feature vector $\hat{\mathbf{y}}$ at random from the task model. Finally, we numerically solve for the arm configuration that results in a feature vector most similar to $\hat{\mathbf{y}}$ given $\hat{\mathbf{q}}_{\text{base}}$ by solving the following nonlinear least-squares minimization problem:

$$\arg\min_{\mathbf{q}_{\text{arm}}} \|\phi_{\hat{\mathbf{a}}}(\hat{\mathbf{q}}_{\text{base}}, \mathbf{q}_{\text{arm}}) - \hat{\mathbf{y}}\|_{\Sigma^{-1}}$$

Sampling a base configuration proceeds similarly and different numbers of base and arm samples can be accomodated by simply skipping some extension steps (see Algorithm 6.3). This procedure replaces Algorithm 6.1. Because the configurations in $G_{\text{base}}$ are also in $\mathcal{F}_{\text{base}}$, we are effectively biasing the distribution of arm configurations towards those which achieve the task with the base in feasible locations. Similarly, we are sampling base configurations which are useful when the arm is not in a self-colliding state.

The sampling approach described above can be readily adapted to sampling from a sequence of multivariate normal distributions as found in the task model. We first select a number of samples $n$ to compute in a batch to ensure the samples are spread evenly across the distributions. Rather than sampling a fixed number from each of the discrete distributions, we gradually alter the distribution from which $\hat{\mathbf{y}}$ is sampled at each iteration. Specifically, we linearly interpolate between adjacent distributions in the model, which can be computed by a simple procedure. For each $i < n$, we consider continuous $t = i \cdot (T-1)/n$. Because normal distributions are additive, we can interpolate between them in multiple equivalent ways. Perhaps the most intuitive way is by the explicit parameters:

$$\begin{aligned}
\boldsymbol{\mu}_{(i)} &= (t - \lfloor t \rfloor) \cdot \boldsymbol{\mu}_{\lfloor t \rfloor} + (\lceil t \rceil - t) \cdot \boldsymbol{\mu}_{\lceil t \rceil} \\
\boldsymbol{\Sigma}_{(i)} &= (t - \lfloor t \rfloor) \cdot \boldsymbol{\Sigma}_{\lfloor t \rfloor} + (\lceil t \rceil - t) \cdot \boldsymbol{\Sigma}_{\lceil t \rceil}
\end{aligned}$$

However, sampling using $\boldsymbol{\Sigma}$ directly requires that we compute the Cholesky decomposition which takes $\Theta(b^3)$ time. This is reasonable to perform for each distribution in the model, but unnecessary for each sample. Instead, we observe that, similar to the previous section, we are only interested in sampling from the distribution, not explicitly constructing it. So it suffices to sample

$\mathbf{y}_- \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\lfloor t \rfloor})$ and $\mathbf{y}_+ \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\lceil t \rceil})$, and appropriately combine these samples, yielding:

$$\tilde{\mathbf{y}}_i = (t - \lfloor t \rfloor) \cdot \boldsymbol{\mu}_{\lfloor t \rfloor} + (\lceil t \rceil - t) \cdot \boldsymbol{\mu}_{\lceil t \rceil} + \sqrt{t - \lfloor t \rfloor} \cdot \mathbf{y}_- + \sqrt{\lceil t \rceil - t} \cdot \mathbf{y}_+ \,.$$

This follows from the additive property of normally distributed random variables.

**Algorithm 6.3** BUILDHYBRIDROADMAPGIBBS($\mathbf{q}_0$, $n_{\text{base}}$, $n_{\text{arm}}$)

**Input:** Initial configuration $\mathbf{q}_0$, number of samples $n$

$\mathcal{G}_{\text{base}} \leftarrow \{\mathbf{q}_0\}$

$\mathcal{G}_{\text{arm}} \leftarrow \{\mathbf{q}_0\}$

$n \leftarrow \max(n_{\text{base}}, n_{\text{arm}})$

**for** $i \leftarrow 1$ **to** $n$ **do**

$\hat{\mathbf{q}}_{\text{base}} \leftarrow$ randomly from $\mathcal{V}(\mathcal{G}_{\text{base}})$ // sample arm using fixed base

$\hat{\mathbf{y}}_1 \leftarrow$ SAMPLE($\mathcal{N}(\boldsymbol{\mu}_{(i)}, \mathbf{\Sigma}_{(i)})$)

$\mathbf{q}_{\text{arm}} \leftarrow \arg\min_{\mathbf{q}_{\text{arm}}} (\phi_{\hat{\mathbf{a}}}(\hat{\mathbf{q}}_{\text{base}}, \mathbf{q}_{\text{arm}}) - \hat{\mathbf{y}}_1)^{\mathrm{T}} \Sigma_i^{-1} (\phi_{\hat{\mathbf{a}}}(\hat{\mathbf{q}}_{\text{base}}, \mathbf{q}_{\text{arm}}) - \hat{\mathbf{y}}_1)$

**if** $(i \cdot n_{\text{arm}}$ **rem** $n) < n_{\text{arm}}$ **then**

EXTEND($\mathcal{G}_{\text{arm}}$, $\mathbf{q}_{\text{arm}}$)

**end if**

$\hat{\mathbf{q}}_{\text{arm}} \leftarrow$ randomly from $\mathcal{V}(\mathcal{G}_{\text{arm}})$ // sample base using fixed arm

$\hat{\mathbf{y}}_2 \leftarrow$ SAMPLE($\mathcal{N}(\boldsymbol{\mu}_{(i)}, \mathbf{\Sigma}_{(i)})$)

$\mathbf{q}_{\text{base}} \leftarrow \arg\min_{\mathbf{q}_{\text{base}}} (\phi_{\hat{\mathbf{a}}}(\mathbf{q}_{\text{base}}, \hat{\mathbf{q}}_{\text{arm}}) - \hat{\mathbf{y}}_2)^{\mathrm{T}} \Sigma_i^{-1} (\phi_{\hat{\mathbf{a}}}(\mathbf{q}_{\text{base}}, \hat{\mathbf{q}}_{\text{arm}}) - \hat{\mathbf{y}}_2)$

**if** $(i \cdot n_{\text{base}}$ **rem** $n) < n_{\text{base}}$ **then**

EXTEND($\mathcal{G}_{\text{base}}$, $\mathbf{q}_{\text{base}}$)

**end if**

**end for**

**return** $G_{\text{base}} \times G_{\text{arm}}$

Examples of the resulting base roadmap and hybrid roadmaps are shown for a sweeping task in Figure 6.4 and for a liquid pouring task in Figure 6.1d.

### 6.3.4 Guiding Manifolds

In prior work [17], numeric optimization was used to seed the roadmap with local minima of each distribution to find high quality paths quickly in unconstrained space. There were generally finitely many such minima because the feature space was well-behaved and higher-dimensional than the configuration space. This is not always the case for mobile manipulators, where the configuration space may be larger than the feature space or the Jacobian of $\phi$ may be singular. One effect of this is that the local minima no longer form a so-called *guiding path* but a *guiding manifold* for each distribution. Generally, this manifold is where the configuration exactly satisfies the learned means, that is $\phi_{\hat{\mathbf{a}}}(\mathbf{q}) = \boldsymbol{\mu}$.

We find it both intuitive and empirically effective to sample on these guiding manifolds to find good paths quickly in addition to the random sampling that ensures asymptotic optimality. Seeding in this way also improves the initial distribution of Gibbs samples, replacing the usual burn-in step for MCMC samplers. To accomplish this, we locally-optimize randomly sampled configurations similarly to Section 6.3.3.

$$\arg\min_{\mathbf{q}_{\text{arm}}} (\phi_{\hat{\mathbf{a}}}(\hat{\mathbf{q}}_{\text{base}}, \mathbf{q}_{\text{arm}}) - \boldsymbol{\mu}_t)^{\mathrm{T}} \Sigma_t^{-1} (\phi_{\hat{\mathbf{a}}}(\hat{\mathbf{q}}_{\text{base}}, \mathbf{q}_{\text{arm}}) - \boldsymbol{\mu}_t)$$

While this approach does not provide guarantees about the distribution of the resulting samples, no such guarantee is required because these are only used to seed the roadmap. We note that prior work has considered similar problems in greater depth [65, 136].

## 6.4 Results

We evaluated the method on two household tasks: a liquid pouring task and a sweeping task. In both experiments, the salient objects and obstacles were sensed with the integrated Primesense RGBD. To demonstrate the real-world applicability of the method, we perform collision detection directly against the point cloud. The learned models for both tasks each had 12 time steps. All timings were performed on an Intel Xeon E5-1680 CPU with 8 cores running at 3.40 GHz and 64 GB of RAM.

### 6.4.1 Sweeping Task

In this task, we required the robot to navigate while holding a broom which it then used to sweep the floor toward a dustpan (see Figure 6.6). In each of the demonstrations and the subsequent

Figure 6.5: Household environment used for both tasks, with the floor highlighted in yellow. The table surface for the liquid pouring task is shown in green.

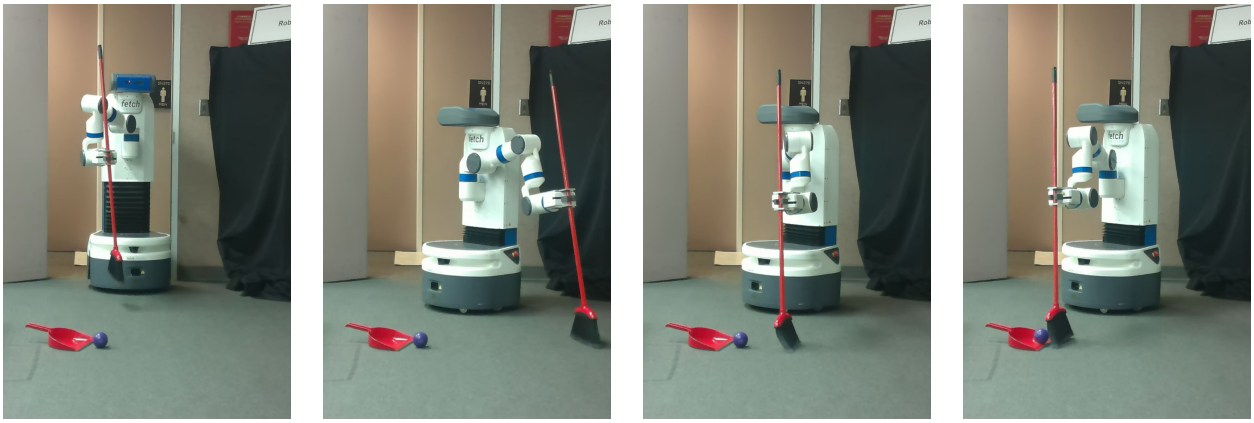


Figure 6.6: Sweeping task execution.

| Roadmap | Sampling | Success | Planning Time |
|---|---|---|---|
| **Hybrid** | **TGGS** | **90%** | **158s** |
| | *TGGS w/o Cost Optimization* | | 174s |
| | Goal Bias | 50% | 288s |
| | Uniform | 0% | 261s |
| RRG* | Goal Bias | 20% | 281s |

Table 6.1: Results for the sweeping task across 10 random scenarios with our full method highlighted in bold. The cost optimization is that described in (6.1), the goal bias preferred sampling base positions near the bowl, and uniform sampling was unbiased. Instead of using our hybrid roadmap approach, the RRG* roadmap was built in the full configuration space.

evaluations, we independently sampled the initial position of the robot and the dustpan uniformly at random from a 5.5m by 4.8m rectangle on the floor, rejecting those positions in collision with the environment (see Figure 6.5).

For the task, the feature function was given by:

$$\phi_{\mathbf{a}}(\mathbf{q}) = \begin{bmatrix} \mathbf{q}_{5\ldots11} \\ v(\boldsymbol{K}_{\text{end}}(\mathbf{q})^{-1}\boldsymbol{K}_{\text{dustpan}}(\mathbf{a})) \\ v(\boldsymbol{K}_{\text{dustpan}}(\mathbf{a})^{-1}\boldsymbol{K}_{\text{gripper}}(\mathbf{q})) \end{bmatrix}$$

where $v(\boldsymbol{K})$ denotes the translational part of affine transformation $\boldsymbol{K}$, $\boldsymbol{K}_{link}(\mathbf{q})$ denotes the forward kinematics of *link* in configuration $\mathbf{q}$, and $\boldsymbol{K}_{\text{dustpan}}(\mathbf{a})$ denotes the pose of the dustpan in the environment described by $\mathbf{a}$.

The resulting feature space was 13-dimensional. We performed 14 kinesthetic demonstrations of the task via teleoperation. The number of demonstrations was one greater than the dimensionality of the feature space to avoid singular covariance matrices.

We compared our TGGS approach to uniform and goal-biased sampling strategies. Goal-biasing was performed by sampling base positions in a Gaussian distribution around the dustpan and sampling arm configurations from a joint-space Gaussian distribution estimated from the demonstrations. For all sampling strategies, we sampled 250 arm configurations and 50 base configurations in the hybrid roadmap. The full configuration space RRG* used 7500 samples

Figure 6.7: Fetch robot executes a liquid pouring task.

because this produced comparable planning times. In all cases, a spatio-temporal roadmap was constructed from the resulting spatial roadmap and used for planning using the same learned cost metric. Furthermore, because all of the methods considered were asymptotically optimal, these results indicate how quickly each method converges to successful plans while in the limit, they will all converge to equally good solutions. The cost optimization is that described in (6.1), which only affects planning time, not the resulting path, and thus not the success rate.

Because the number of samples was fixed for each roadmap type, the differences in timings were largely caused by slower collision checks against the longer edges in the non-task-guided strategies despite the slower sample computation when using TGGS. The uniform distribution was slightly faster than goal-biasing because less of the roadmap needed to be explored to guarantee the shortest path. The most common cause of failure for the goal bias and uniform methods was a lack of samples low enough to sweep but high enough to avoid collision between the block of the broom and the floor, which together form a long narrow passage. In contrast, TGGS sampled densely in this region to produce high-quality sweeping motions.

### 6.4.2 Liquid Pouring Task

In this task, we required the robot to navigate while holding a pitcher of water and pour the water into a bowl on a table without spilling (see Figure 6.7). In each of the demonstrations and the subsequent evaluations, we independently sampled the initial position of the robot uniformly at random from a 5.5m by 4.8m rectangle, rejecting those positions in collision with the environment (see Figure 6.5). Similarly, we sampled the bowl's position uniformly at random from the 1.8m by 0.75m surface of the table.

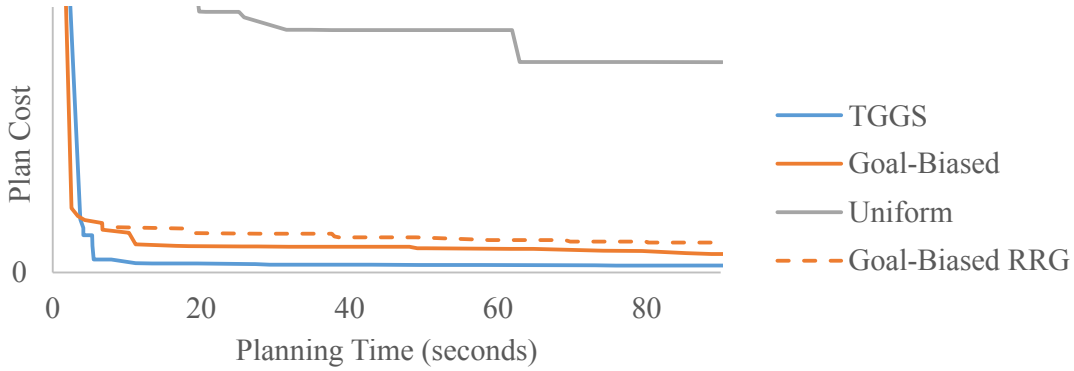



Figure 6.8: Plan cost with varying planning times using different sampling approaches on the liquid pouring task. Solid lines indicate variants using hybrid roadmaps, and the dotted line indicates an RRG* was used.

For this task, the feature function was given by:

$$\phi_{\mathbf{a}}(\mathbf{q}) = \begin{bmatrix} \mathbf{q}_{6\ldots11} \\ v(\boldsymbol{K}_{\text{end}}(\mathbf{q})^{-1}\boldsymbol{K}_{\text{bowl}}(\mathbf{a})) \\ v(\boldsymbol{K}_{\text{bowl}}(\mathbf{a})^{-1}\boldsymbol{K}_{\text{gripper}}(\mathbf{q})) \end{bmatrix}$$

where $v(\boldsymbol{K})$ denotes the translational part of affine transformation $\boldsymbol{K}$, $\boldsymbol{K}_{link}(\mathbf{q})$ denotes the forward kinematics of *link* in configuration $\mathbf{q}$, and $\boldsymbol{K}_{\text{bowl}}(\mathbf{a})$ denotes the pose of the bowl in the environment described by $\mathbf{a}$. We note that this differs from the feature function used for the sweeping task only in the omission of one joint, which was found to be unnecessary for capturing this task. The resulting feature space was 12-dimensional, and we performed 13 kinesthetic demonstrations using the same method as for the sweeping task.

The resulting plans were successful in 90% of the evaluations we performed. The single failure was due to the liquid missing the bowl during the pouring motion, indicating that the method failed to converge to a sufficiently good plan in the alloted time.

Additionally, we measured plan cost (as defined by (6.2)) with varying planning times using different sampling approaches (see Figure 6.8) on a representative scenario from the liquid pouring task. While goal-biasing converged much more quickly than uniform sampling, TGGS was faster still. Furthermore, using a hybrid roadmap improves convergence even when using goal-biasing.

## 6.5 Conclusion

In this chapter we described and analyzed TGGS, an approach to sampling configurations that incorporates information from a learned model of a mobile manipulation task. This sampling strategy was tightly incorporated into a hybrid roadmap construction scheme that decomposes the planning space into that of the manipulator arm and mobile base and uses sampling-based planners most appropriate to each. We demonstrated the efficacy of this approach on two household tasks with the Fetch robot.

CHAPTER 7

# Conclusion

To assist an aging population with activities of daily living, robots must be able to perform numerous tasks in highly unstructured environments. Manually modeling all the potential tasks in a way that facilitates planning and execution in these environments would be tedious for a technical user and infeasible for a non-technical one. Instead, we focused on learning a task model from kinesthetic demonstrations that anyone can perform and using this model to plan motions in novel environments.

In this dissertation, we addressed the following thesis statement:

*Robotic systems can learn, from demonstrations, to perform tasks in unstructured environments while avoiding obstacles with less prior knowledge by better extracting information from the demonstrations and leveraging an asymptotically optimal motion planning method during execution.*

To support this thesis, we described a model for robotic tasks based on a hidden Markov model which can be learned from relatively few demonstrations by improving temporal registration. This model was augmented with time-invariant parameters which were learned by recasting model estimation as a nonlinear least-squares minimization problem. These learned tasks were subsequently estimated by adapting a probabilistic roadmap using a Cartesian product graph, an approach which was made fast enough to be performed reactively by informing configuration sampling using the task model. Another Cartesian product graph was used to extend this approach to the domain of mobile manipulators where the previous biased sampling approach was naturally extended to a Gibbs sampling procedure.

## 7.1 Limitations and Future Work

There is much ongoing work in automatic task learning and performance. Given the scope of tasks to which we have applied our method, it seems ideally suited to integration with work on task-level planning frameworks [137, 138] to perform more complex tasks that combine primitives learned with our method.

Other future directions could relax some of the assumptions we made which limit the domains in which this method could be beneficially applied. These limitations can roughly be categorized under modeling, planning, and sensing.

**Modeling** We model a task as a distribution over plans conditional on the environment, however, by instead modeling some notion of success as a function of the plan, negative (failed) demonstrations could be incorporated into the learning process to help identify problematic behaviors including in a reinforcement or active learning paradigm. This would additionally allow the planning phase to determine when it cannot successfully perform task.

We have also only considered task models with Gaussian observation distributions. While learning in feature space means this is not overly restrictive, other distributions (e.g. Cauchy) might be more appropriate for some tasks in some domains. Similarly, relaxing the sequential transition structure of the hidden Markov model could allow tasks with multiple routes to completion to be learned. To expand the types of demonstrations from which we can learn, we could consider per-demonstration latent parameters in addition to the time-variant model parameters and invariant ones in Chapter 4 and the interpretation of these parameters during the planning phase.

Finally, planning in a state space which incorporates salient aspects of the environment affected by the robot's actions could allow the method to learn tasks requring more complex reasoning [81]. A model of this interaction with the environment might also be learned from the demonstrations.

**Planning** The planning approach is mostly restricted to holonomic robots where we do not need to consider constraints imposed by the dynamics. However, some tasks require robots with dynamics considerations (e.g. quadcopters) or even include dynamics as a fundamental part of the task (e.g. applying constant contact force). More general motion planners would need to be adapted to perform such tasks.

The planning approaches we considered might also by greatly accelerated by applying additional heuristics as has been done for traditional motion planners. In particularly, a task-specific admissible heuristic would allow bidirectional A* to be applied, and by relaxing this heuristic, near-optimality could still be preserved while further improving performance.

Recently, trajectory optimization method have been developed that locally optimize plans very

effectively. These approaches could be applied to the cost functions we presented and even combined with our sampling-based methods (e.g. in a framework like that presented by Kuntz et al. [61]).

**Sensing** Both the learning and planning phases assume perfect sensing, but in practical scenarios, occlusions and noise are abundant. For the motion planner in particular, this assumption is both strong and challenging to relax because the approach we take converges to the *mode* of the distribution over trajectories implied by the learned task mode with no consideration for uncertainty. However, addressing partial and noisy sensing in a principled way would likely be highly beneficial. With such a framework in place we could also consider predictive models of the environment to enable faster, safer reactivity, particularly when paired with real-time perception [139].

## REFERENCES


[1] Rethink Robotics, "Baxter Research Robot." www.rethinkrobotics.com/baxter-research-robot/, 2013.

[2] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots*, 2016.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robotics Research*, vol. 30, pp. 846–894, June 2011.

[4] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), ch. 59, pp. 1371–1394, Springer, 2008.

[5] N. T. Nguyen, D. Q. Phung, and S. Venkatesh, "Learning and detecting activities from movement trajectories using the hierarchical hidden Markov model," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 955–960, 2005.

[6] D. Kulic, W. Takano, and Y. Nakamura, "Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden Markov chains," *Int. J. Robotics Research*, vol. 27, pp. 761–784, July 2008.

[7] S. Calinon, F. D'halluin, D. G. Caldwell, and A. G. Billard, "Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, pp. 582–588, 2009.

[8] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 763–768, May 2009.

[9] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard, "Imitation learning with generalized task descriptions," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 3968–3974, May 2009.

[10] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, pp. 433–454, May 2012.

[11] D. Berenson, J. Kuffner, and H. Choset, "An optimization approach to planning for mobile manipulation," in *Int. Conf. Robotics and Automation (ICRA)*, pp. 1187–1192, IEEE, 2008.

[12] H. G. Tanner, S. G. Loizou, and K. J. Kyriakopoulos, "Nonholonomic navigation and control of cooperating mobile manipulators," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 53–64, 2003.

[13] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *8th IEEE-RAS International Conference on Humanoids*, pp. 91–98, 2008.

[14] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 20–33, 2012.

[15] H. Choset, K. M. Lynch, S. A. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[16] C. Bowen, G. Ye, and R. Alterovitz, "Asymptotically-optimal motion planning for learned tasks using time-dependent cost maps," *IEEE Trans. Automation Science and Engineering*, vol. 12, pp. 171–182, Jan. 2015.

[17] C. Bowen and R. Alterovitz, "Closed-loop global motion planning for reactive execution of learned tasks," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 1754–1760, Sept. 2014.

[18] C. Bowen and R. Alterovitz, "Closed-loop global motion planning for reactive, collision-free execution of learned tasks," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 7, no. 1, p. 10, 2018.

[19] C. Bowen and R. Alterovitz, "Asymptotically optimal motion planning for tasks using learned virtual landmarks," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1036–1043, 2016.

[20] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions and the bayesian restoration of images," *Journal of Applied Statistics*, vol. 20, no. 5-6, pp. 25–62, 1993.

[21] C. Bowen and R. Alterovitz, "Accelerating Motion Planning for Learned Mobile Manipulation Tasks using Task-Guided Gibbs Sampling," in *Proc. Int. Symp. Robotics Research (ISRR)*, 2017.

[22] L. Jaillet, J. Cortes, and T. Simeon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. Robotics*, vol. 26, pp. 635–646, Aug. 2010.

[23] J. Mainprice, E. A. Sisbot, L. Jaillet, J. Cortés, R. Alami, and T. Siméon, "Planning human-aware motions using a sampling-based costmap planner," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 5012–5017, May 2011.

[24] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Trans. Robotics*, vol. 26, pp. 576–584, June 2010.

[25] R. Jakel, S. R. Schmidt-Rohr, M. Losch, and R. Dillmann, "Representation and constrained planning of manipulation strategies in the context of programming by demonstration," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 162–169, May 2010.

[26] J. Claassens, "An RRT-based path planner for use in trajectory imitation," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 3090–3095, May 2010.

[27] D. Berenson, T. Simeon, and S. S. Srinivasa, "Addressing cost-space chasms in manipulation planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 4561–4568, May 2011.

[28] J. Scholz and M. Stilman, "Combining motion planning and optimization for flexible robot manipulation," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, pp. 80–85, Dec. 2010.

[29] I. A. Şucan and S. Chitta, "Motion planning with constraints using configuration space approximations," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 1904–1910, Oct. 2012.

[30] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 3706–3712, May 2011.

[31] J. D. Marble and K. E. Bekris, "Asymptotically near optimal planning with probabilistic roadmap spanners," *IEEE Trans. Robotics*, vol. 29, pp. 432–444, Apr. 2013.

[32] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 5068 – 5074, May 2013.

[33] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, pp. 469–483, 2009.

[34] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. Int. Conf. Machine Learning (ICML)*, (New York, New York, USA), ACM Press, 2004.

[35] J. van den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 2074–2081, May 2010.

[36] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Int. Conf. Artificial Intelligence and Statistics*, vol. 15, pp. 182–189, 2011.

[37] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, no. 3, pp. 1561–1566, 2011.

[38] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun, "Apprenticeship learning for motion planning with application to parking lot navigation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 1083–1090, Sept. 2008.

[39] D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *Int. J. Robotics Research*, vol. 29, pp. 1565–1592, Oct. 2010.

[40] S. J. Lee and Z. Popović, "Learning behavior styles with inverse reinforcement learning," in *ACM Trans. Graphics (Proc. SIGGRAPH)*, vol. 29, pp. 122:1–122:7, July 2010.

[41] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, pp. 25–53, June 2009.

[42] S. Calinon, *Robot Programming by Demonstration*. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2009.

[43] K. Bernardin, K. Ogawara, K. Ikeuchi, and R. Dillmann, "A sensor fusion approach for recognizing continuous human grasping sequences," *IEEE Trans. Robotics*, vol. 21, no. 1, pp. 47–57, 2005.

[44] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[45] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 26, pp. 43–49, Feb. 1978.

[46] P. C. Mahalanobis, "On the generalised distance in statistics," *National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.

[47] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Trans. Robotics and Automation*, vol. 12, pp. 566–580, Aug. 1996.

[48] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 431-441, 1963.

[49] M. Rudelson and R. Vershynin, "Non-asymptotic theory of random matrices: extreme singular values," in *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 1576–1602, World Scientific, 2010.

[50] Aldebaran Robotics, "Aldebaran Robotics NAO for education." http://www.aldebaran-robotics.com/en/naoeducation, 2010.

[51] "Bullet Physics Library," 2012.

[52] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[53] J. Claassens and Y. Demiris, "Exploiting affordance symmetries for task reproduction planning," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, pp. 653–659, IEEE, 2012.

[54] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 1243–1248, May 2006.

[55] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 1603–1609, Apr. 2007.

[56] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, pp. 35–48, Sept. 2012.

[57] Y. Yang and O. Brock, "Elastic roadmaps—motion generation for autonomous mobile manipulation," *Autonomous Robots*, vol. 28, no. 1, p. 113, 2009.

[58] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant Hamiltonian optimization for motion planning," *Int. J. Robotics Research (IJRR)*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[59] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 4569–4574, May 2011.

[60] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2012.

[61] A. Kuntz, C. Bowen, and R. Alterovitz, "Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization," in *International Symposium on Robotics Research*, 2017.

[62] A. Dobson and K. E. Bekris, "A study on the finite-time near-optimality properties of sampling-based motion planners," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 1236 – 1241, Nov. 2013.

[63] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Int. Conf. Robotics and Automation (ICRA)*, pp. 625–632, IEEE, 2009.

[64] B. Kim, T. T. Um, C. Suh, and F. C. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, no. 1, pp. 202–225, 2016.

[65] L. Jaillet and J. M. Porta, "Asymptotically-optimal path planning on manifolds," *Robotics: Science and Systems VIII*, p. 145, 2013.

[66] F. Burget, M. Bennewitz, and W. Burgard, "BI 2 RRT*: An efficient sampling-based path planning framework for task-constrained mobile manipulation," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 3714–3721, IEEE, 2016.

[67] P. Hämäläinen, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 81, 2015.

[68] S. Benson, "Inductive learning of reactive action models," in *Proc. Int. Conf. Machine Learning (ICML)*, pp. 47–54, 1995.

[69] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proc. ACM/IEEE Int. Conf. Human-Robot Interaction*, pp. 205–212, ACM, 2015.

[70] S. Ekvall and D. Kragic, "Robot learning from demonstration: A task-level planning approach," *Int. J. Advanced Robotic Systems*, vol. 5, no. 7, pp. 223–234, 2008.

[71] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *Int. J. Robotics Research (IJRR)*, vol. 33, no. 14, pp. 1726–1747, 2014.

[72] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 639–646, IEEE, 2014.

[73] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems*, pp. 1–6, 2016.

[74] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, 2017.

[75] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Trans. Robotics*, vol. 24, pp. 1463–1467, Dec. 2008.

[76] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 365–371, sep 2011.

[77] A. Shukla and A. Billard, "Coupled dynamical system based arm-hand grasping model for learning fast adaptation strategies," *Robotics and Autonomous Systems*, vol. 60, pp. 424–440, Mar. 2012.

[78] J. D. Langsfeld, K. N. Kaipa, R. J. Gentili, J. A. Reggia, and S. K. Gupta, "Incorporating failure-to-success transitions in imitation learning for a dynamic pouring task," in *Workshop on Compliant Manipulation: Challenges and Control*, 2014.

[79] E. Davis, "Pouring liquids: A study in commonsense physical reasoning," *Artificial Intelligence*, vol. 172, no. 12-13, pp. 1540–1578, 2008.

[80] Y. Kuriyama, K. Yano, and M. Hamaguchi, "Trajectory planning for meal assist robot considering spilling avoidance," in *IEEE Int Conf. Control Applications*, pp. 1220–1225, IEEE, 2008.

[81] Z. Pan, C. Park, and D. Manocha, "Robot motion planning for pouring liquids.," in *ICAPS*, pp. 518–526, 2016.

[82] R. I. Davis and B. C. Lovell, "Comparing and evaluating HMM ensemble training algorithms using train and test and condition number criteria," *Formal Pattern Analysis & Applications*, vol. 6, no. 4, pp. 327–335, 2004.

[83] G. Ye and R. Alterovitz, "Demonstration-guided motion planning," in *Proc. Int. Symp. Robotics Research (ISRR)*, Aug. 2011.

[84] E. A. Ok, *Real Analysis with Economic Applications*. Princeton University Press, 2007.

[85] D. de Champeaux, "Bidirectional heuristic search again," *J. ACM*, vol. 30, pp. 22–32, Jan. 1983.

[86] S. Calinon, D. Bruno, and D. Caldwell, "A task-parameterized probabilistic model with minimal intervention control," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 3339–3344, May 2014.

[87] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, pp. 328–373, Feb. 2013.

[88] M. Mühlig, M. Gienger, and J. J. Steil, "Interactive imitation learning of object movement skills," *Autonomous Robots*, vol. 32, pp. 97–114, 2012.

[89] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *IEEE-RAS Int. Conf. Humanoid Robots*, pp. 91–98, 2008.

[90] N. Jetchev and M. Toussaint, "Task space retrieval using inverse feedback control," in *Proc. Int. Conf. Machine Learning (ICML)*, 2011.

[91] M. Stollenga, L. Pape, M. Frank, J. Leitner, A. Forster, and J. Schmidhuber, "Task-relevant roadmaps: A framework for humanoid motion planning," in *IEEE Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 5772–5778, 2013.

[92] S. Agarwal, K. Mierle, and Others, "Ceres Solver." `http://ceres-solver.org`.

[93] D. Sun and J. Berger, "Objective Bayesian analysis for the multivariate normal model," *Bayesian Statistics*, vol. 8, pp. 525–547, 2007.

[94] S. Bodiroža, G. Doisy, and V. V. Hafner, "Position-invariant, real-time gesture recognition based on dynamic time warping," in *ACM/IEEE Int. Conf. Human-Robot Interaction (HRI)*, pp. 87–88, 2013.

[95] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[96] A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning force-based manipulation of deformable objects from multiple demonstrations," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 177–184, 2015.

[97] M. Müller, "Dynamic time warping," *Information Retrieval for Music and Motion*, pp. 69–84, 2007.

[98] N. Vuković, M. Mitić, and Z. Miljković, "Trajectory learning and reproduction for differential drive mobile robots based on GMM/HMM and dynamic time warping using learning from demonstration framework," *Engineering Applications of Artificial Intelligence*, vol. 45, pp. 388–404, 2015.

[99] A. Vakanski, I. Mantegh, A. Irish, and F. Janabi-Sharifi, "Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping," *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 4, pp. 1039–1052, 2012.

[100] V. Pavlovic, J. M. Rehg, and J. MacCormick, "Learning switching linear models of human motion," in *Advances in neural information processing systems*, pp. 981–987, 2001.

[101] S.-Z. Yu and H. Kobayashi, "An efficient forward-backward algorithm for an explicit-duration hidden Markov model," *IEEE signal processing letters*, vol. 10, no. 1, pp. 11–14, 2003.

[102] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[103] E. Berger, M. Sastuba, D. Vogt, B. Jung, and H. Ben Amor, "Estimation of perturbations in robotic behavior using dynamic mode decomposition," *Advanced Robotics*, vol. 29, no. 5, pp. 331–343, 2015.

[104] H. B. Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters, "Interaction primitives for human-robot cooperation tasks," in *IEEE Int Conf. Robotics and Automation (ICRA)*, pp. 2831–2837, 2014.

[105] G. Maeda, M. Ewerton, R. Lioutikov, H. B. Amor, J. Peters, and G. Neumann, "Learning interaction for collaborative tasks with probabilistic movement primitives," in *IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, pp. 527–534, 2014.

[106] A. K. Tanwani and S. Calinon, "Learning robot manipulation tasks with task-parameterized semitied hidden semi-Markov model," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 235–242, 2016.

[107] M. K. Titsias, C. C. Holmes, and C. Yau, "Statistical inference in hidden Markov models using $k$-segment constraints," *Journal of the American Statistical Association*, vol. 111, no. 513, pp. 200–215, 2016.

[108] J. Lember and A. A. Koloydenko, "Bridging Viterbi and posterior decoding: a generalized risk approach to hidden path inference based on hidden Markov models," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1–58, 2014.

[109] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, "SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards," in *WAFR*, 2016.

[110] P. M. Weichsel, "The Kronecker product of graphs," *Proc. American Mathematical Society*, vol. 13, no. 1, pp. 47–52, 1962.

[111] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Trans. Systems, Man and Cybernetics–Part B*, vol. 37, pp. 286–298, Apr. 2007.

[112] F. Lv and R. Nevatia, "Single view human action recognition using key pose matching and Viterbi path searching," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1–8, June 2007.

[113] R. Bellman, "On a routing problem," tech. rep., DTIC Document, 1956.

[114] I. N. Herstein, *Topics in algebra*. John Wiley & Sons, 2006.

[115] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[116] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh International Symposium*, pp. 36–54, 2005.

[117] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 2640–2645, 2011.

[118] Y. Yang and O. Brock, "Elastic roadmaps—motion generation for autonomous mobile manipulation," *Autonomous Robots*, vol. 28, no. 1, pp. 113–130, 2010.

[119] N. M. Amato, K. A. Dill, and G. Song, "Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures," *Journal of Computational Biology*, vol. 10, no. 3-4, pp. 239–255, 2003.

[120] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1618–1623, IEEE, 2004.

[121] B. Burns and O. Brock, "Single-Query Entropy-Guided Path Planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 2124–2129, Apr. 2005.

[122] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif, "Narrow passage sampling for probabilistic roadmap planning," *IEEE Transactions on Robotics (TRO)*, vol. 21, no. 6, pp. 1105–1115, 2005.

[123] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Int. Conf. Robotics and Automation (ICRA)*, pp. 3067–3074, 2015.

[124] T. Kunz and M. Stilman, "Manipulation planning with soft task constraints," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 1937–1942, 2012.

[125] R. Iehl, J. Cortés, and T. Siméon, "Costmap planning in high dimensional configuration spaces," in *IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics (AIM)*, pp. 166–172, 2012.

[126] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 4120–4125, 2013.

[127] S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell, "Statistical dynamical systems for skills acquisition in humanoids," in *12th IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, pp. 323–329, IEEE, 2012.

[128] S. Candido, Y.-T. Kim, and S. Hutchinson, "An improved hierarchical motion planner for humanoid robots," in *8th IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, pp. 654–661, IEEE, 2008.

[129] V. Pilania and K. Gupta, "A hierarchical and adaptive mobile manipulator planner with base pose uncertainty," *Autonomous Robots*, vol. 39, pp. 65–85, Jun 2015.

[130] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: from verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, vol. 34, no. 2, pp. 157–182, 2009.

[131] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. Journal of Robotics Research (IJRR)*, vol. 28, no. 1, pp. 104–126, 2009.

[132] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation.," in *Int. Conf. Automated Planning and Scheduling*, pp. 254–258, 2010.

[133] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 1470–1477, 2011.

[134] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," *Proc. IEEE Conf. Decision and Control*, pp. 7681–7687, Dec. 2010.

[135] D. J. Balkcom and M. T. Mason, "Time optimal trajectories for bounded velocity differential drive vehicles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 199–217, 2002.

[136] I. Havoutis and S. Ramamoorthy, "Motion planning and reactive control on learnt skill manifolds," *Int. Journal of Robotics Research (IJRR)*, vol. 32, no. 9-10, pp. 1120–1150, 2013.

[137] I. A. Şucan and L. E. Kavraki, "Mobile manipulation: Encoding motion planning options using task motion multigraphs," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 5492–5498, May 2011.

[138] L. P. Kaelbling and T. Lozano-Pérez, "Unifying perception, estimation and action for mobile manipulation via belief space planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 2952–2959, May 2012.

[139] R. B. Rusu, I. A. Sucan, B. P. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pp. 4245–4252, Oct. 2009.