Following Paths in Task Space: Distance Metrics and Planning Algorithms

Rachel Holladay

May 5, 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Advisor:
Siddhartha Srinvasa, CMU RI

Submitted in partial fulfillment of the requirements for the SCS Undergraduate Honors Senior Thesis.

Abstract

Many of our everyday jobs we imagine robots accomplishing are defined via a variety of task-specific constraints. In order for robots to perform these tasks, the robot's motion planners must respect these constraints. While a robotic manipulator moves and plans in its joint or configuration space, many constraints are naturally defined in task space. We focus on the specific constraint asking the robot's end effector (hand) to trace out a shape.

Formally, our goal is to produce a configuration space path that closely follows a desired task space path despite the presence of obstacles. This thesis proposes distance metrics for formally defining closeness and planning algorithms that efficiency leverage these definitions. Adapting metrics from computational geometry, we show that the discrete Fréchet distance metric is an effective and natural tool for capturing the notion of closeness between two paths in task space.

We then introduce two algorithmic approaches for efficiently planning with this metric. The first is a trajectory optimization approach that directly optimizes to minimize the Fréchet distance between the produced path and the desired task space path. The second approach searches along a configuration space embedding graph of the task space path. Finally, we evaluate these approaches through real robot and simulation experiments.

Acknowledgements

I could not have asked for a better undergraduate advisor then Siddhartha Srinviasa. Only a few months into my first year you let a clueless, overly enthusiastic eighteen year into your lab and it has changed my life. You have taught me, guided me, provided me every opportunity and will forever be my Sensei. I am also very grateful to Oren Salzman for working alongside during part of this project, introducing me to the world of computational geometry and filling up an absurd number of whiteboards with drawings of graphs.

I have been incredibly lucky to work in the Personal Robotics Lab for the last four years with amazing co-workers and friends. Thank you to Aaron J., Aaron W., Aditya, Ariana, Brian H., Brian O., Clint, Daqing, Gilwoo, Hanjun, Jennifer, JS, Laura, Liz, Matt, Mike D., Rosario, Shen, Shushman, Stefanos, Pras and Pyry. Special thanks to Anca for generously mentoring during my first two years, Jen for teaching how to make robots mostly work, Mike K. for pushing me to always be better, Chris for sharing his expertise on graphs, Henny for guiding me professionally and Shervin for teaching me the importance of de-stressing. I am grateful to all of my fellow undergraduates in lab, especially Jimmy, Vinitha, Vikram, Neil and Don. Special thanks to Jean Harpley and Keyla Cook for keeping the lab running and Matt Mason for kindly offering his wisdom.

Research does happen in a vacuum and I am indebted to my friends, family and undergraduate peers for keeping me sane. Wing, thank you for being the greatest randomly selected roommate to ever exist and for getting me through these four years. Ariana, we took almost every computer science class together and I am not sure I could have graduated without you. Spoon Gang, thank you for being a ragtag group of friends that both supported me and pushed me, although sometimes in random directions. Ananya, you are the best partner in crime I could ask for.

Finally, nothing would be possible without my amazing family: my infinitely supportive Mom and Dad, my beloved older sister and brother, Annie and Ben, and my womb buddy, Sam. I wouldn't be who I am without your love.

Funding

The thesis contains work supported by the U.S. Office of Naval Research (grant number BAA 13-0001); Computing Research Association's Collaborative Research Experiences for Undergraduates; Carnegie Mellon's SRC Undergraduate Research Opportunities funding; and Carnegie Mellon's Summer Undergraduate Research Fellowship.

Contents

List of Figures

1 Introduction

Our goal is to enable robots to complete complex tasks, like clearing off the dinner table or pouring glasses of water. Many of these tasks have task-specific constraints, like not tipping over plates of food while we clear off the table or carefully pouring liquid without spilling. In order to build robots to successfully accomplish these tasks, we need robots with motion planners that reason about and respect these constraints in addition to joint limits and collision constraints.

In this thesis we focus on a particular type of constraint: following a reference path in task space. While we formally define this later, we can informally define it as constraining a robots hand, known as an end effector, to trace out a path.

For example, in Fig.1.1 a user provides a demonstration to a robot on top, who can then recreate that demonstration, as seen on the bottom, where the path is shown in black. Demonstration could encapsulate a task, such as opening a door, as learning from demonstrations is a popular robotic paradigm (Argall et al., 2009).

Following a path, whether demonstrated or directly specified, can also be used to allow robots to recreate handwritten letters or other line drawings (der Einreichung, 2016). Robot planners that follow paths are especially suited for the industrial application of following arc welding paths (Ahmad and Luo, 1989). This is an area of intense interest as almost 13% of robots shipped were employed for arc welding application in 2011 (Larkin et al., 2016).

In certain application, like arc welding, we may only be given the path as a series of positions that we want our robot to trace out. However, in the case of demonstrations, we might also be given the joint poses of the robot that achieved the motion. The naïve solution for creating the task space motion would be to replay the recorded demonstration. However, this would prevent us from generalizing the motion and would fail given any clutter in the scene.

For example, Fig.1.2 shows an instance where our planner succeeds

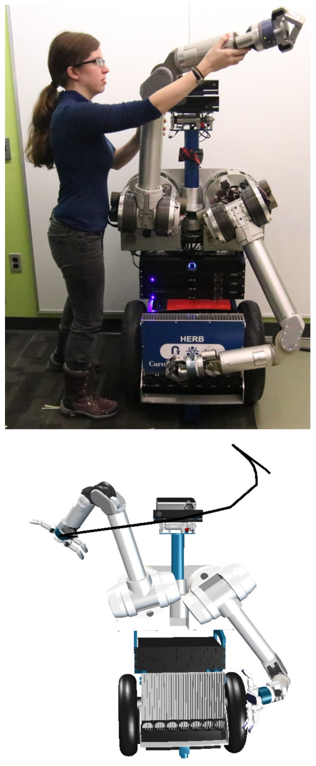
Figure 1.1: On the top, a user demonstrates a task space trajectory that is visualized on the bottom. Our goal is to enable the robot to be able to recreate the shape of the provided demonstration in the general setting.

when the naïve solution of replaying the trajectory would fail. The demonstration, shown in black, was gathered in a blank environment. The starting position of this trajectory is shown as the translucent robot arm. In the new environment that includes a table, this trajectory collides with the table and is therefore infeasible. By planning with the task space motion, we are able to avoid the table and successfully reproduce the desired motion, shown in orange.

Therefore, we will plan in the robots full configuration space to leverage its general ability to execute collision-free planning in various environments (Ye and Alterovitz, 2011).

We next provide definition and a more formal specification of our problem.

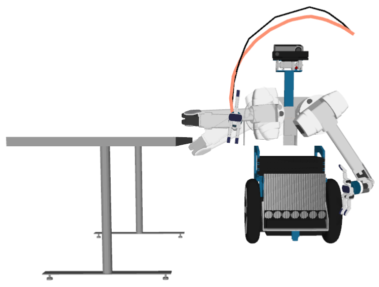
Figure 1.2: Simply replaying recorded reference paths, fails in new environments. Our planners are able to plan around obstacles.

1.1 *Problem Specification*

Before providing our problem statement we formally define configuration space and task space. A configuration, q, of our robot completely describes the location of the robot. The configuration space, $\mathbb{C}$, is the set of all configurations (Lozano-Perez, 1990). In the case of a manipulator, the configuration space is equal to the joint space since we can completely describe the location of the arm by the joint angles. A path in configuration space is detonated by $\xi : [0, 1] \to \mathbb{C}$.

Task space is the space defined by the pose of the robot's end effector, $SE(3)$. A path in task space is denoted as $\xi : [0, 1] \to SE(3)$.

We operate with paths, which do not specify velocities or timing. This is in contrast to trajectories, which are parameterized by time. For a trajectory of length t we define trajectories in configuration space and task space, $\xi : [0, t] \to \mathbb{C}$ and $\xi : [0, t] \to SE(3)$ respectively. Before execution, we therefore need to time our generated paths into trajectories. We use a standard parabolic smoother and retimer (Hauser and Ng-Thow-Hing, 2010).

The robot induces a forward kinematics and an inverse kinematics mapping. Forward kinematics maps configurations to task space, $x = FK(q)$. Inverse kinematics maps a location in task space to a set of robot configurations, $Q = IK(x)$ such that $Q = \{q^1, q^2, ..q^k\}$. In an abuse of notation we will use $FK(\xi)$ to map a configuration space path into a task space path.

Equipped with these definitions, we can formally define our problem.

We are given a robotic manipulator, endowed with a configuration space $q \in \mathcal{C}$, and a *reference path* in task space, $\bar{\xi} : [0, 1] \to SE(3)$.

Our goal is to produce the *closest* path that matches the reference path subject to constraints on the system:

$$\xi^* = \arg\min_{\xi \in \Xi} ||\xi - \bar{\xi}|| \quad \text{s.t. constraints} \tag{1.1}$$

The formulation of (1.1) raises several interesting questions. Namely, how do we define the distance between two paths and how to do plan in this space. This thesis addresses both questions.

To answer the first question we explore two distance metrics from computational geometry: the Hausdorff and Fréchet metrics. While both can be used to capture the distance between two paths, we show that the Fréchet serves as a more natural tool because it encodes the flow the path.

We present two planning methods. The key insight of our first method is that we can recreate task space reference path by optimizing a configuration space path with respect to a cost function defining the distance between the reference path and the task space motion achieved by the path. By formalizing a cost function we are able to frame our problem as a trajectory optimization problem. We also introduce two techniques to assist our optimizer's performance.

One limitation of this trajectory optimization approach is that our assistance techniques search only within a one dimensional space. Therefore, for our second planning technique we define a two dimensional search space where we sample on the reference path. From these samples we construct and search along a cross product graph.

1.2 *Summary of Contributions*

This thesis proposes using distance metrics for formally defining closeness and planning algorithms that efficiency leverage these definitions. Specifically, we contribute:

- Adapting metrics from computational geometry, we show that the discrete Fréchet distance metric is an effective and natural tool for capturing the notion of closeness between two paths in task space (Sec. 3).
- We first formulate the planning problem under a trajectory optimization framework that directly optimizes to minimize the Fréchet distance between the produced path and the desired task space path (Sec. 4).
- We then reformulate our problem as a randomized sample-based graph search problem on the cross product between the reference path and a configuration space embedding of the reference path (Sec. 5).

We also include a discussion on related work both with respect to task space constraints and search in motion planning (Sec. 2). We conclude the thesis with a discussion of future research questions and reflections (Sec. 6).

2
Background

To place our work within context, we first overview previous work in planning with task space constraints, also referred to as Cartesian or end effector constraints, Sec. 2.1. We then review previous work on searching in configuration space, which provides background for our second method, Sec. 2.2.

2.1 Task Space Constraints

Task space constraints seem to have initially emerged as a by-product of planning with redundant manipulators. With robot manipulators that have more degrees of freedom then the number of degrees of freedom in the task space, we have an infinite number of solutions for end effector poses. This infinity allows us to place additional constraints on our end effector pose.

Seereeram and Wen formulated feasible, rather than optimal, planning with redundant manipulators as a finite time nonlinear control problem that, as a by-product, allowed for task space constraints (Seereeram and Wen, 1995). These constraints had to be inequality constraints and were enforced via quadratic programming. Following a path has also been formulated as a series of equality constraints (Ahmad and Luo, 1989) or posed as a Particle Swarm Optimization problem (Xu et al., 2008).

Yao and Gupta introduced two methods for handling end effector constraints (Yao and Gupta, 2007), Adapted-RGD (Randomized Gradient Descent) and ATACE (Alternate Task Space and Configuration Space Exploration). Adapted RGD performed gradient descent on closed chain kinematics, temporarily breaking the chain and then enforcing the constraint. ATACE searches for poses in task space and connects poses through a local planner in configuration space. Our method interweaves task space and configuration space with similar inspiration, but we propose a different graph configuration and search method. Stillman later compared an extension of Adapted-RGD and showed the projection method to be more efficient (Stilman, 2010).

Previously, Berenson et al. defined constraints as manifolds in configuration space and enforced constraints by projecting back on to the constraint manifold (Berenson et al., 2009). Oriolo et al. also sampled on the manifold and specifically defined following task space constraints as the problem of Motion Planning along End-effector Paths (MPEP) (Oriolo et al., 2002).

Projecting between spaces has also been used to achieve fast planning in incredible high dimensional space by exploring actions in the low dimensional task space (Shkolnik and Tedrake, 2009).

Approached as a Cartesian path planning problem in the 1970s, Paul generated a trajectories that is are piecewise straight lines with parabolic transitions on knot points (Paul, 1979). In contrast, Froissart and Mechler prioritized that the path is continuous in its first two derivatives (Froissart and Mechler, 1993). By doing so they reduced the mechanical oscillation in the arm, a critical specification for their applications of arc welding, gluing, laser cutting and high pressure washing.

Path following has even been approach via a genetic algorithm, although this was only shown for a two-dimensional arm (Tian and Collins, 2004).

Moving beyond specific task space constraints for manipulators allows us to explore even more general problems. For example, Zacharias et al. expanded the search space to leverage an arm and mobile base for following for three dimensional trajectories with the tool tip at the end of the arm (Zacharias et al., 2009). Instead of using one trajectory, der Einreichung created a distribution of reference trajectories provded by human demonstrations. Similar in style to our planning approach discussed in Sec. 4, they use trajectory optimization to generate a trajectory that minimize the distance between trajectory and distribution according to Kullback-Leibler metric (der Einreichung, 2016).

2.2 *Motion Planning and Search*

Our second planning technique leverages searching in configuration space, a popular paradigm in robotics. The high dimensional configuration space is sampled to create a graph, eliminating the need to represent the space explicitly. Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT) are two fundamental sample-based motion planners (Kavraki et al., 1996; LaValle and Kuffner, 2001).

Building off of these we can use search methods, like Dikjsta or A* to improve performance (Dijkstra, 1959; Hart et al., 1968). Both the

PRM and RRT were extended with A* to PRM* and RRT* respectively to achieve asymptotically optimality (Karaman and Frazzoli, 2011).

In most of these methods you use inverse kinematics to project into configuration space and place landmarks (Ahuactzin and Gupta, 1998). These landmarks are then connected through varying technique to create a graph that is then searched through for a solution. Typically these connections are made by a local planner (Bessiere et al., 1993; Mazer et al., 1998)

There has been a considerable amount of work in deciding what landmarks to connect via a local planner and how to sample points. A simple method is to sample randomly and connect nodes according to the *k*-nearest neighbor (Horsch et al., 1994). A more guided approach to to grow the graph by picking each new milestone as far away as possible from the current milestones with the goal of further exploring the space (Ahuactzin and Gupta, 1999).

Another sampling method would be to select candidate points that are designed to make it easier to get through narrow passages created by collisions. Amato and Wu use this as motivation to sample points uniformly on the surface of a configuration space obstacle (Amato and Wu, 1996).

Kavraki explored a variety of ways to sample landmarks to improve the search, such as adding landmarks close to other landmarks that have few neighbors, adding landmarks where existing landmarks are far apart, adding landmarks to areas where the local planner failed to connect other landmarks, etc. (Kavraki and Latombe, 1994; Kavraki et al., 1996).

There have been two directions of exploration for improving the efficiency of these methods. The first is to delay collision checking as much as possible.

Before using an edge in our graph, we need to collision check it to ensure we produce a collision-free path. However, collision checking is an expensive operation. By performing these checks lazily, only when we believe an edge will become part of our candidate path, we can save computation (Bohlin and Kavraki, 2000). These kind of lazy evaluation approaches have previously been applied to graph search and can be adapted to planning (Cohen et al., 2015; Dellin and Srinivasa, 2016).

A second interesting question is how to efficiently search a dynamic graph. Kallman proposed dynamic roadmaps that handle when edges can become temporarily unavailable due to collision but where we want to reuse paths that may be broken in several locations (Kallman and Matarić, 2004).

Koenig et al. proposed Lifelong Planning A* (LPA*) as a way to combine the merits of informed and incremental search (Koenig et al., 2004). We can see that as we build or change a graph, we can continuously use this kind of incremental search method to update our solution. We discuss incremental updates as an area of future work in Sec. 6.

3
Distance Metrics

In order to create a path that closely follows our reference path we need a distance metric for defining the closeness of two curves. We explore two distance metrics from computational geometry that are commonly used for shape matching: the Hausdorff distance and the Fréchet distance. Below, we explore each of them as metrics for capturing the distance between paths.

3.1 Hausdorff Distance

Our goal is to produce a path that is close to our reference path. One natural way to quantify closeness is to require that each point on our path be close to a corresponding point on the reference path.

For example, consider the reference path shown in black in Fig.3.1. We can place an r-disc ball around each point in our path and union these balls together to create a *safe zone*, as shown. If each point on our path is within some r distance to our reference path, then our path is contained within our safe zone. This metric corresponds to the *one-way Hausdorff distance*.

The Hausdorff distance is a method for measuring how far apart two subsets of metric space are (Hausdorff and Brieskorn, 2008). Given that an adversary picks a point on one shape, the Hausdorff distance is the longest distance you would be forced to travel to get to any point on the other shape. Although originally formulated as metric for shapes, the one-way Hausdorff distance can also be applied to curves, point sets and objects (Dubuisson and Jain, 1994; Huttenlocher and Kedem, 1990; Belogay et al., 1997). Hence for paths we can formalize the one way Hausdorff distance as:

$$H(\xi_x, \bar{\xi}_x) = \sup_{y \in \xi} \inf_{\bar{y} \in \bar{\xi}} d_{TS}(y, \bar{y}) \tag{3.1}$$

where d_{TS} is a distance metric between points in task space defined in Sec. 3.3

When we consider the Hausdorff distance for paths, if every point on the path was to find its closest neighbor on the reference path, the Hausdorff distance is the longest neighbor to neighbor distance.

Therefore, if our path and our reference path have a Hausdorff distance less then r, then our path is entirely contained with our r-disc safe zone. We can see that for the orange path in Fig.3.1, each point is forced to travel at least r distance from its closest point on the black reference path.

By constraining our path to be within some threshold, r, according to the one-way Hausdorff distance, we are insuring that our path is within the reference path's r-disc safe zone. While our orange path in Fig.3.1 lies within the safe zone, it fails to capture the exact flow of the reference path, shortcutting the center loop.

One way to insure that the path better matches the reference path is to, in addition to requiring each point on the path be close to a point on the reference point, also require that each point on the reference path is close to a point on the path. This corresponds to the *two way Hausdorff distance* for paths. In Fig.3.1 the two way Hausdorff distance is shown in green. While the reference path (in black) and the two way Hausdorff path (in green) are close to each other, the path fails to capture the true shape, instead traversing through the loop in the reverse order.

To follow our reference path, we want to constrain our path to pass through our r-disc balls *in order*. This requirement motivates using the *Fréchet distance*.

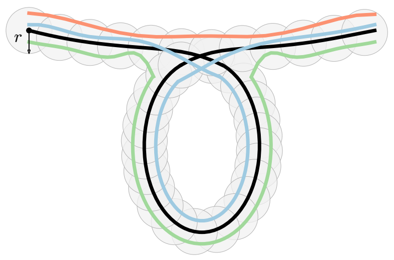

Figure 3.1: The one way (orange) and two way (green) Hausdorff distances are constrained to be in the safe zone while the Fréchet distance (blue) forces us to pass through the safe zone in order.

3.2 *Fréchet Distance*

The Fréchet distance captures the difference in flow between two curves (Fréchet, 1906). The Fréchet distance is commonly explained through an analogy, where a dog is walking along ξ at speed parameterization α and its owner is walking along $\bar{\xi}$ at speed parameterization β (Chambers et al., 2010). The two are connected via a leash. The Fréchet distance is the shortest possible leash via some distance metric d such that there exists a parameterization α and β so that the two stay connected and move monotonically. More formally:

$$F(\xi_x, \bar{\xi_x}) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \left\{ d_{TS}\Big(\xi_x(\alpha(t)), \bar{\xi_x}(\beta(t))\Big) \right\} \tag{3.2}$$

where d_{TS} is defined as before.

We can see with the blue path in Fig.3.1 that the Fréchet's monotoncity requirement forces it to follow each of the balls in order, thus capturing the flow of the reference path. Therefore, the one-way Hausdorff distance captures constraining our result to be contained within our safe zone. In contrast the Fréchet distance captures following the r-discs of the safe zone in order.

Additionally, in previous comparisons between the Fréchet and hausdorff, the Fréchet distance has been shown to better capture the similarity between paths (Alt, 2009; Larios et al., 2015; Alt et al., 2004).

3.3 *Computing Distance Metrics*

We next quickly detour to discuss implementation details of our approach.

There are two possibilities for our definition of the distance between two points, $x, y \in SE(3)$, $d_{TS}(x, y)$. A common metric in motion planning is the the Cartesian product of the Euclidean metric for $\mathbb{R}^3$ and the standard great circle solid angle metric for $SO(3)$ (Sucan et al., 2012). We examined this metric and just the Euclidean metric for $\mathbb{R}^3$. For the experiments presented throughout the thesis, we concentrated on the latter metric.

In practice, computing the continuous Fréchet and Hausdorff is difficult. In fact, calculating the continuous Fréchet with a similar metric involves solving an optimization problem (Wenk et al., 2010). Additionally, our path representation is given as a series of waypoints. If our path is not given as a series of waypoints we can sample our path to convert to this representation. We therefore use the discrete Hausdorff and the discrete Fréchet (F_d), the latter of which is typically calculated by dynamic programming (Alt and Godau, 1995; Eiter and Mannila, 1994).

Adapting our Fréchet metaphor to the discrete case, we replace the dog and owner to a pair of frogs, still attached by a leash, that can only hop along stones (our waypoints) (Agarwal et al., 2014).

3.4 *Gradient Interpretation*

The gradient of the one-way Hausdorff or Fréchet distances push us into the safe zone or ordered safe zone, respectively. Let's say we have a path and reference path with a discrete number of fixed waypoints. Then, the analytical negative gradient of either distance metric is the unit magnitude pointing towards the furtherest point on the reference path.

Consider the reference path in black in Fig.3.2 and the candidate path in red. Given a Hausdorff or Fréchet distance of P, there must exist a point i on the reference path and j on the path such that $d(i, j) = P$. This pair represents our point of maximum violation, which we will assume is unique. As seen in Fig.3.2, the two distance metrics may select different (i, j) pairs. However, for both, the dis-

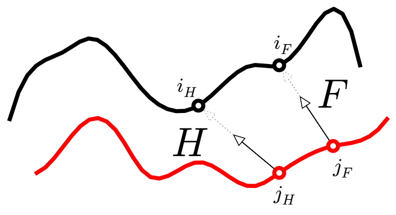

Figure 3.2: The negative analytical gradient of the Fréchet and Hausdorff distance steps to decrease the distance.

tance is determined by their maximum violation.

Since the point at maximum deviation determines the distance, the gradient of the distance metric with respect to the candidate path is equivalent to the the gradient at just our maximum deviation point.

By taking a step in the negative direction of this gradient, shown as the black arrow in Fig.3.2, we move our point of maximum violation closer to the reference path, thus decreasing our distance. Hence in Fig.3.2, gradient applied to each distance metric's (i, j) would decrease their P.

4
Direct Optimization

We first formulate following at task space path as a trajectory optimization problem. Our key insight is that we can recreate task space reference path by optimizing a configuration space path with respect to a cost function defining the distance between the reference path and the task space motion achieved by the path. Our cost function is defined by the distance metrics discussed in Sec. 3.

This approach has been published and presented at IROS 2016 (Holladay and Srinivasa, 2016).

We find that trajectory optimization is susceptible to local minimum. This can lead the optimizer to generate paths that do not match the reference path. To account for this we offer two methods, splitting and stapling (discussed in Sec. 4.2 and Sec. 4.3). These methods guide the optimization to the correct basin of attraction by identifying critical points in the reference path that serve as hard constraints to the optimizer.

Leveraging a different computation geometry metric, Procrustes Analysis, we explore a related problem. Often the reference path does not have to have strong bindings to the particular task space location. Rather the reference path is meant to provide a general task space shape that the robot should produce, i.e. tracing out the letter 'A'. We provide an optimization method for recreating only the shape (Sec. 4.4).

Finally, we explore the limitations of our optimization approach in Sec. 4.5. This directly motivates our sample based search approach discussed in the next chapter.

4.1 A Preliminary Experiment

We conducted a preliminary experiment to explore the problem. We first gather demonstrations of trajectories on a robot that serve as our reference paths. Then we use the distance metrics discussed in Sec. 3 as cost functions that penalize the trajectory for being far from the goal in addition to constraints that prevent self-collision and respect joint limits.

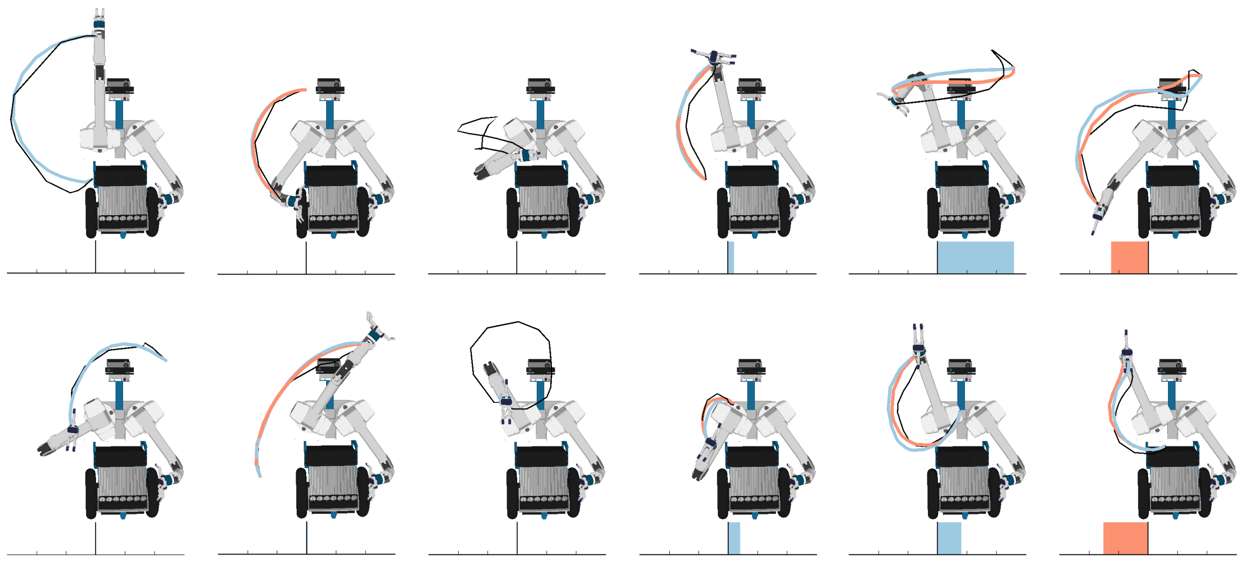

Figure 4.1: The reference path is shown in black. The path optimized according to the Hausdorff and Fréchet metrics are in orange and blue, respectively. The bars below show the difference between the Fréchet error of the two optimized paths.

Clearly, the demonstrated path is one (of possibly many) global minimum for the optimization. But we are interested in its *basin of attraction*. Specifically, we explore how well a completely uninformed initialization, like a straight line from start to goal, can bend and twist itself to get to the reference path. By solving this problem, instead of relying on the demonstrated path $\bar{\xi}_q$, we can more generally recreate shapes. This general problem proves to be surprisingly challenging.

Gathering Demonstrations

In order to gather demonstrations from our robot HERB, we placed it in gravity compensation mode such that an operator could move the arms freely to create the desired motion (Ulrich and Kumar, 1991). We recorded joint angles at 100Hz. that serve as waypoints for the reference path $\bar{\xi}_q$. This was then converted to $\bar{\xi}_x$ using inverse kinematics, since our distane metrics operate in task space and we want to only rely on having $\bar{\xi}_x$ in the general case.

Initial Results

We created a set of 24 demonstrations, some of which are shown in black in Fig.4.1. We then optimize using the Hausdorff or Fréchet metric as the cost function. We used TrajOpt's default stopping criteria and computed distances with respect to the end effector position, not the full arm pose.

Fig.4.1 shows each reference path in black, the the Fréchet optimization in blue and the Hausdorff optimization in orange. Each bar below compares the difference in Fréchet error between the Fréchet optimization and Hausdorff optimization. We elected to compare using theFréchet error since the Fréchet metric restricts us to follow the curves of the reference path.

In Fig.4.1, we see three categories. The first three columns show paths where the error for the Fréchet and Hausdorff optimization are the same, hence their difference is zero. In the first of these two columns, both optimization produce the nearly same path. In the third column the path begin and end at the same location. Thus the empty path is at a local minimum.

The fourth and fifth column show paths where the Fréchet optimization produced a higher Fréchet error. In these cases the Hausdorff optimization produced paths that were closer to the reference paths, which were generally monotone. In the sixth column, the Hausdorff paths have a higher error, and the Fréchet paths better corresponded to the shape of the path.

In Fig.4.2, we show planning time as a function of the number of iterations that the optimizer runs. Each point represents a reference path where corresponding reference path optimized with the Fréchet and Hausdorff metric are connected via a grey line segment. We see two trends. First, planning time increases with more iterations of TrajOpt, which is expected. Second, it generally takes longer to optimize according to the Fréchet metric, even when there are less iterations required. This is to be expected as the Fréchet metric, computed used dynamic programming, is generally more time consuming.

While the paths in Fig.4.1 capture the shape to an extent, they often fail to capture the shape entirely. Our optimization process does not drive our cost to zero in part because these reference path are difficult for an optimizer to achieve. Many times, the optimizer falls into a local minimum that is different from our provided demonstration due to self-collisions or joint limits.

In order to produce more accurate paths we therefore assist TrajOpt via two methods: *split* and *staple*. Both of these methods add more constraints to to our problem, thus moving our basin of attraction to new locations.

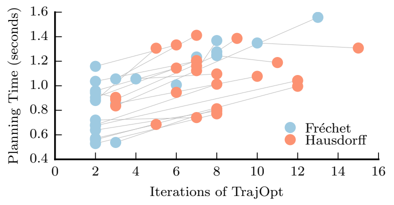

Figure 4.2: Each point represents a reference path and corresponding reference path optimized with each metric are connected via a grey segment. The Fréchet optimizations trend to take longer and planning time increases with the number of iterations.

4.2 *Optimizing in Joint Space*

In order to provide assistance to our optimizer we present two techniques, splitting and stapling, that further constrain the path. Splitting segments the path in a predefined way, while stapling segments through a more intelligent method. We begin by examining both in the robot's joint space since it serves as an easier problem. This assumes we have access to $\bar{\xi}_q$. We later relax this to only having $\bar{\xi}_x$, and use our insights from optimizing in joint space to generalize the to the more difficult problem of optimizing in task space.

Algorithm 1 SplitJoint

```
1: Given: Reference Path $\bar{\xi}_q$, metric $m$, split count $k$
2: Initialize: $\xi = \emptyset$
3: procedure FOR I=0:K
4:     $t_s = \frac{i}{k}$
5:     $t_e = \frac{i+1}{k}$
6:     $\xi^i = \text{PlanTo}(\bar{\xi}_q(t_s), \bar{\xi}_q(t_e), m)$
7:     $\xi_q = \text{CombineTrajs}(\xi_q, \xi^i)$
8: return $\xi_q$
```

Split Method

To move our basin of attraction, and therefore ease our problem, we split our path in k segments and optimize on each segment.

The algorithm for the general k is given in Algorithm 1. We loop through k calculating the starting and end configurations (Line 4, Line 5). We then plan between these two segments (Line 6) and combine sequentially (Line 7).

As we can see in Fig.4.3, the more pieces we segment the path into, the lower our Fréchet error. This is expected since with more segments, more constraints are imposed on the optimizer.

For each path we could select a k that balances between cost and computation time. And in fact, we will see this tradeoff later in Sec. 4.2. However, it is unclear how to choose this k in the general case. Instead we want our optimization method to select some k and place the partitions where they are most needed. This intuition inspires the stapling method, described in the following section.

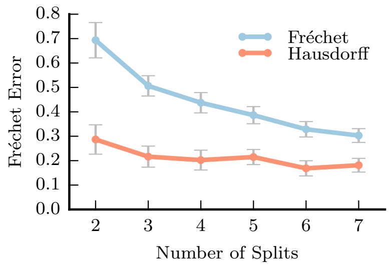

Figure 4.3: In joint space, as we increase the number of splits, the error decreases.

Staple Method

Instead of picking some k which splits our path into evenly spaced segments, we want to concentrate on points in our path that need extra help from the optimizer. Therefore, we present the Stapling Algorithm, in joint space, in Algorithm 2, that dynamically selects the points of the path to staple.

Using either the Hausdorff or Fréchet we begin with a path optimized to that metric m, like those created in Sec. 4.1 (Line 3). For either metric, Hausdorff or Fréchet , we then find the point of of the path that errors furthest from our reference path: the point of maximum violation (Line 4).

Since the Fréchet distance is the minimum length leash, as described in Sec. 3, we find the location that forces that distance and declare that to be our point of maximum violation. For the Hausdorff distance, this maximum point is the furthest you could be forced to travel to go from one curve to another.

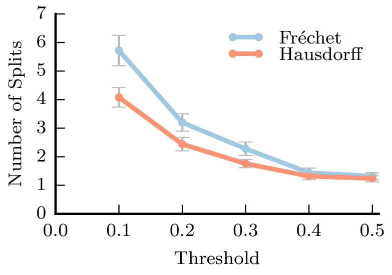

Figure 4.4: If we decrease the violation threshold when stapling, the number of segments increases.

Algorithm 2 StapleJoint

```
1:  Given: Reference Path $\bar{\xi}_q$, metric $m$, threshold $\epsilon$, start point $t_s$, end
    point $t_e$
2:  Initialize: $t_s = 0, t_e = 1$
3:  $\xi_q$ = PLANTO($\bar{\xi}(t_s), \bar{\xi}_q(t_e), m$)
4:  $v_t, t_i$ = FINDMAXVIOLATION($\bar{\xi}_q, \xi_q, m$)
5:  if $v_t > \epsilon$ then
6:      $\xi_q^a$ = STAPLEJOINT($\bar{\xi}_q, t_s, t_i, m$)
7:      $\xi_q^b$ = STAPLEJOINT($\bar{\xi}_q, t_i, t_e, m$)
8:      return COMBINETRAJS($\xi_q^a, \xi_q^b$)
9:  else
10:     return $\xi_q$
```

We then staple that point of maximum violation to the reference path, similarly to the splitting method, and recurse on the two pieces: the reference path from the start to the staple point and the reference path from the staple point to the end. We repeat this iteratively until the maximum violation is below some threshold (Line 5-Line 10).

As we vary the threshold of what violation is considered acceptable, we see in Fig.4.4, the number of segments increases. While it does segment the path like the splitting algorithm, the key difference is that stapling places the segments intelligently.

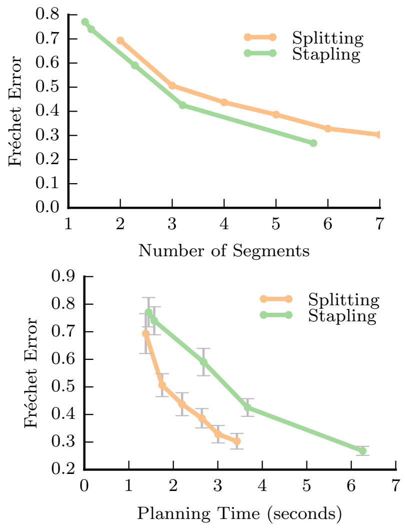

Figure 4.5: Comparing, stapling achieves a given Fréchet error in fewer segments (top) but a higher planning time (bottom) then splitting.

Splitting versus Stapling

The splitting algorithm evenly spaces its allocated split count, ignoring the path's shape. In contrast, the stapling algorithm, by finding the point of maximum violation, staples down the path at the point that is most useful to adhering to the threshold. Thus, as shown in Fig.4.5, for some given target Fréchet error, the stapling method required fewer segments.

However, as illustrated in Fig.4.5, the splitting algorithm can generally achieve a lower Fréchet error than the stapling algorithm in the same time allotment. The recursive nature of the stapling algorithm leads it have to repeat optimizations in trying to satisfying the violation threshold. In contrast, the splitting algorithm handles each segment once.

Fig.4.6 shows how stapling and splitting improve our performance, compared to the same path in Fig.4.1.

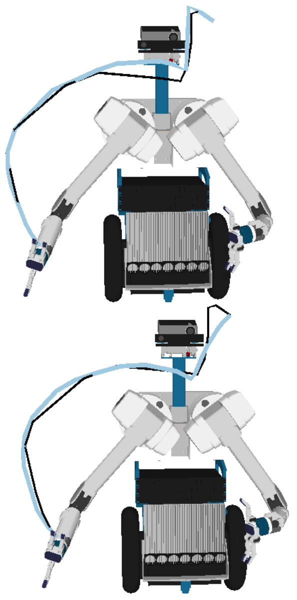

Figure 4.6: In joint space: splitting with $k = 6$ (top) compared to stapling $\epsilon = 0.2$ (bottom).

Limitations

Optimizing in joint space presents a fundamental limitation. Since we rely on indexing into the path, we must have a joint space representation, $\bar{\xi}_q$. This restricts us to using the original demonstration,

Algorithm 3 SplitTask

```
1:  Given: Reference Path $\bar{\xi}_q$, metric $m$, split count $k$
2:  Initialize: $Q_s$ = SampleIK($\bar{\xi}_x(0)$)
3:  Initialize: $\xi_q = \emptyset$
4:  procedure for i=0:k
5:      $t_e = \frac{i+1}{k}$
6:      $Q_e$ = SampleIK($\bar{\xi}_x(t_e)$)
7:      $\xi^i$ = PlanTo($Q_s, Q_e, m$)
8:      $\xi_q$ = CombineTrajs($\xi_q, \xi^i$)
9:      $Q_s = \{\xi_q(1)\}$
10: return $\xi_q$
```

preventing us from generalizing the motion to translate or rotate in task space. To generalize our motion, we advance to optimizing in the task space, using only $\bar{\xi}_x$.

4.3 *Optimizing in Task Space*

While we previously explored the splitting and stapling algorithms in joint space, we now lift those algorithms to task space. Since our robot has a redundant manipulator, there are multiple inverse kinematic solutions to a given point in task space. Therefore, we solve for these inverse kinematic solutions and plan to this set. We detail the changes this consideration implies for both splitting and stapling.

Split and Staple Methods

Splitting: We describe the splitting method in task space in Algorithm 3. In contrast to splitting in joint space, where we plan to a specific configuration, in task space we compute the inverse kinematic solutions for that point in task space (Line 2 and Line 6) and plan to that set.

Once we compute the path for the first segment, the ending configuration of the first segment determines the configuration for the beginning of the next segment (Line 9).

Similarly to in joint space, as the number of splits increases, the Fréchet error decreases, as seen in Fig.4.7.

Stapling: The stapling method in task space in Algorithm 4. Here, as in splitting in task space, we must compute the inverse kinematics set to plan to (Line 2, Line 3, Line 7). Again, before continuing to the next segment, we need to start where the last segment left off (Line 9).

As we decrease our threshold, we increase the number of splits required, visualized in Fig.4.8.

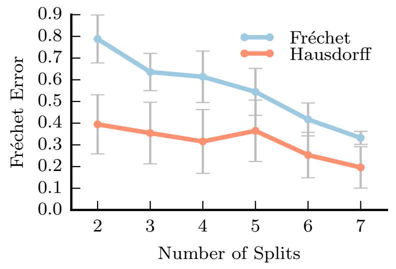

Figure 4.7: In task space, as we increase the number of splits, the error decreases.

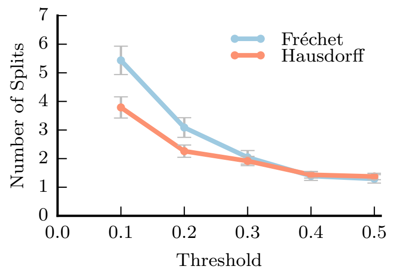

Figure 4.8: In task space, if we decrease the violation threshold when stapling, the number of segments increases.

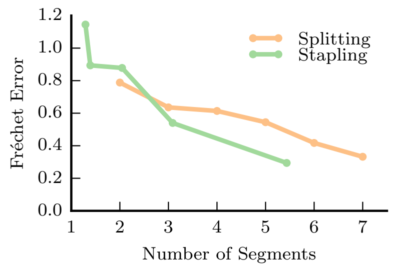

Figure 4.9: Across splitting and stapling, more segments leads to a decreased Fréchet error.

Algorithm 4 StapleTask

```
1:  Given: Reference Path $\bar{\xi}_q$, metric $m$, threshold $\epsilon$, start set $Q_s$, goal set $Q_q$
2:  Initialize: $Q_s$ = SampleIK($\bar{\xi}_x(0)$)
3:  Initialize: $Q_e$ = SampleIK($\bar{\xi}_x(1)$)
4:  $\xi_q$ = PlanTo($Q_s, Q_e, m$)
5:  $v_i, p_i$ = FindMaxViolation($\bar{\xi}_x, \xi_q, m$)
6:  if $v_i > \epsilon$ then
7:      $\hat{Q}_g$ = SampleIK($\bar{\xi}_x(p_i)$)
8:      $\xi_q^a$ = StapleTask($\bar{\xi}_x, Q_s, \hat{Q}_g, m$)
9:      $\hat{Q}_s = \{\xi_q^a(1)\}$
10:     $\xi_q^b$ = StapleTask($\bar{\xi}_x, \hat{Q}_s, Q_g, m$)
11:     return CombineTrajs($\xi_q^a$), $\xi_q^b$
12: else
13:     return $\xi_q$
```

Splitting versus Stapling

As seen in Fig.4.9, for a given number of segments, stapling has, on average, a smaller Fréchet error. This is explainable by the fact that stapling intelligently selects where to segment the path, while splitting selects its segments blindly.

Like in joint space, splitting achieves a lower Fréchet error given a time allotment as seen in Fig.4.10. However both take longer then their joint space counterparts since we must compute the inverse kinematic solution set and plan to a configuration in that set, as opposed to planning to one configuration.

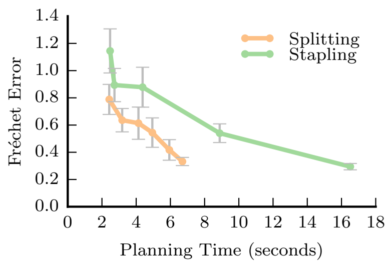

Figure 4.10: For an allowance of planning time, splitting achieves a lower Fréchet error.

We can now easily generate a path that follows a new task space path formed by warping the original reference path. For example, in Fig.4.11, we can translate our original task space path by 10 cm and use our optimization to generate a new path.

However, even this places the burden on the user to determine where to place the path. Instead, we would like to be able to recreate the shape of the path at any location.

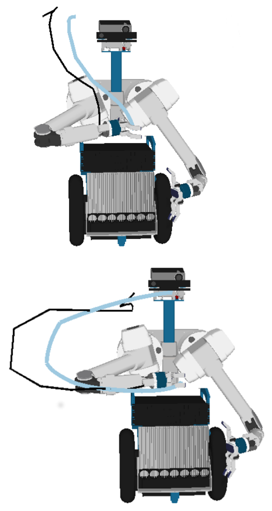
Figure 4.11: The original reference path is given in black and the blue path is translated 10 cm.

4.4 Procrustes Analysis

So far we have required that our reference path have specific bindings to poses in workspace. Often, you may want to specify only a general shape and allow the robot to find any path that follows the shape path. For example, you may want to draw a circle, with no particular preference for where in task space the circle is drawn.

In essence, we wish to focus on the path's shape, ignoring any translations or rotations. To achieve this we draw upon the Procrustes

Algorithm 5 ProcrustesConstraint

```
1: Given: Initial Path $\xi_q^i$, reference path $\bar{\xi}_x$, metric $m$
2: $p_0$ = CORRESPOND($\xi_q^i, \bar{\xi}_s, m$)
3: procedure WHILE NOT MATCHES:
4:     $\xi_q^i$ = PROCRUSTES($\xi_q^i, \bar{\xi}_s$)
5:     $p_n$ = CORRESPOND($\xi_q^i, \bar{\xi}_s, m$)
6:     matches = ($p_0 == p_n$)
7:     $p_0 = p_n$
8: return $\bar{\xi}_q$
```

metric from shape analysis (Gower, 1975; Goodall, 1991; Schönemann, 1966). The Procrustes distance metric first attempts to optimally superimpose two curves on top of each other before assessing the distance between them. By superimposing the two curves, the relative differences in placement are ignored.

Algorithm

To evaluate a cost within our optimization loop, we iteratively perform an expectation-maximization inspired algorithm (Algorithm 5). This algorithm compares the output of the optimizer at iteration i, ξ_q^i to the reference path $\bar{\xi}_x$.

First we use either the Fréchet or Hausdorff metric to compute the correspondence between the path's waypoints (Line 2). Since the discrete Fréchet distance computes the shortest leash between each of the points on our line we can map points on one line to their partners on the other line. Likewise for the Hausdorff metric we can find the point on curve that each corresponding point travels to. Next we compute the scaled Procrustes analysis on each of the curves, which moves ξ_q^i to be as close as possible to $\bar{\xi}_x$ (Line 4).

We use our distance metric to recompute the correspondences (Line 5). If we have converged to an optimal transform, then these correspondences will have not changed within the loop. Otherwise we repeat the process. Upon converging, we have now moved the two paths to be as close as possible. We use metric m to evaluate the distance between the two paths and return this to the optimizer as the cost of ξ_q^i.

The process is similar in style to that described in (Benseghir et al., 2013), although using different metrics.

Since our stapling and splitting algorithms rely on binding to task space, we cannot apply these methods to our current Procrutes constraint formulation.

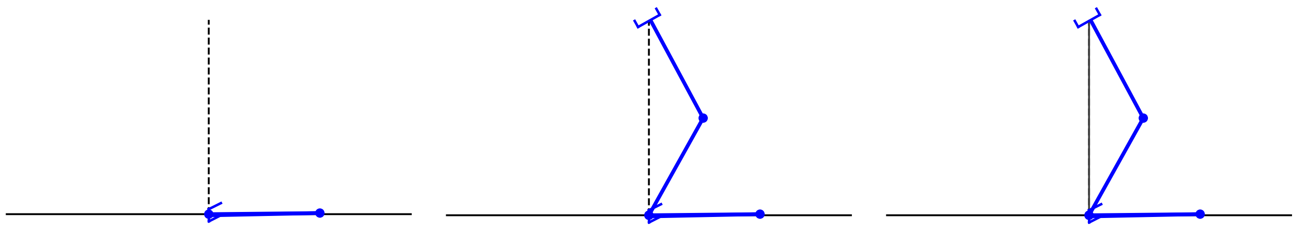

Figure 4.12: This shows the progression of our planner on an RR arm such that we result in a path that closely matches our reference path. The reference path is the black dotted line and the generated path is given in dark gray.

4.5 *Limitations*

In Sec. 4.2 and Sec. 4.3 we introduced the splitting and stapling algorithms for reducing the Fréchet error in our generated paths. While these methods had a degree of success, they critically suffered from the ability to sample only one inverse kinematic solution. However, there is a space of multiple IK solutions which may admit different paths. In this section, we demonstrate why this is problematic on a simplified 2 degree of freedom arm. This motivates our second algorithm, described in Sec. 5.

To illustrate this limitation we use a planar two-link arm with two rotational joints, known as an RR Arm and seen in Fig.4.12 and Fig.4.13. The robot's workspace is defined as the set of position reachable by our end effector in the (x, y) plane. For many locations in the robot's workspace, there exist multiple feasible configurations, commonly refereed to as 'elbow up' and 'elbow down' or the 'left' and 'right' solution. We will use the latter terminology. While these two configuration have the same endpoint in task space, they are quite distant in configuration space.

We will step through two possible execution of our optimization planner using the stapling strategy described in Sec. 4.3. In both cases our reference path $\bar{\xi}$ is the straight line.

We first consider the case shown in Fig.4.12. We begin by selecting the starting configuration for the first task space point, shown in the leftmost column. Considering we have two possible configurations, the left and right, we choose which one with equal probability since our inverse kinematic sampling is random. In this case, we chose the right handed configuration. We next choose our ending configuration, shown in the second panel. In this case, we are lucky because we selected the right handed configuration again. Since our starting and ending locations are in the same homotopic class. Therefore our optimizer will find the path shown in grey in the rightmost column, which exactly matches the reference path. Thus Fig.4.12 shows a successful instance of the planner.

We next consider the case illustrated in Fig.4.13. Again, we select the starting configuration as the left handed solution (shown in the first column) and but select the ending configuration as the right

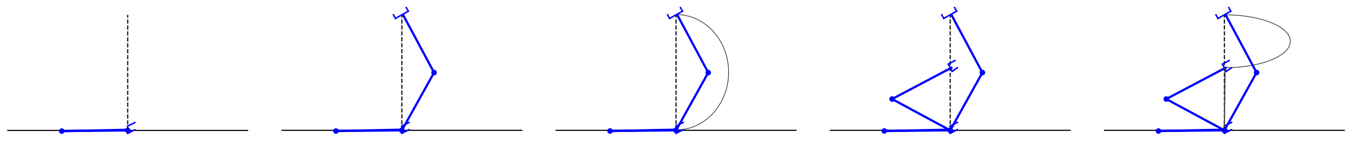

Figure 4.13: This shows the progression of our planner on an RR arm such that we result in a path that does not closely match our reference path. The reference path is the black dotted line and the generated path is given in dark gray.

handed solution (shown in the second column). The result of our optimization planning between these two configurations is shown in the third column. Rather then exactly following the reference path we have to detour to reposition our arm on the opposite side.

Our stapling algorithm will find the midpoint as the location of maximum violation and select a configuration to staple to. With 50% probability we select the left handed configuration, shown in the fourth column. Our planning is now broken into two segments, the results of which are shown in the last column. For the first half of our path we are planning between two configurations on the same side, leading to a straight line that exactly matches our reference path. However, in the top half of our trajectory we are still forced to switch between the left handed configuration and the right handed configuration. And, in fact, we have a smaller distance to make this detour, leading to a larger detour. Each time we staple, we are force to travel through that configuration, preventing us from ever recovering from our mistake.

While we illustrate this point on a two degree-of-freedom arm, this because even more problematic on a higher degree of freedom arm, such as HERB's seven degree arm.

Therefore our failure comes from randomly selecting one possible IK solution from a set of many and then forcing our commitment. Instead, our next method proposes search over sets of inverse kinematic solutions.

5 Cross Product Search

We saw in Sec. 4.5 that our trajectory optimization approach suffered from the fact that it only used one inverse kinematic solution when there exist multiple. To correct this we present a second planning approach that is a randomized sample based search planning method that allows us to search over the entire space of inverse kinematic solutions.

This work was conducted in collaboration with Oren Salzman.

We begin by restating our problem specification and formalizing helpful data structures (Sec. 5.1).

Our first step is to sample our reference path for inverse kinematics solutions and create a layered graph L (Sec. 5.2). This graph represented a sampling of our configuration search space. From this, we create a graph Φ that is the cross product of our layered graph L and our reference path (Sec. 5.4). In this cross product graph, finding a path with the minimal Fréchet value to our reference path corresponds to finding the minimum bottleneck path. We can compute the minimum bottleneck path with a simple variant of Dijkstra's graph search algorithm.

We conclude this section with experiments verifying our approach and comparing its performance to the trajectory optimization approach presented in Sec. 4. Fig.5.1 shows the reference path in black and out output of our planner in red.

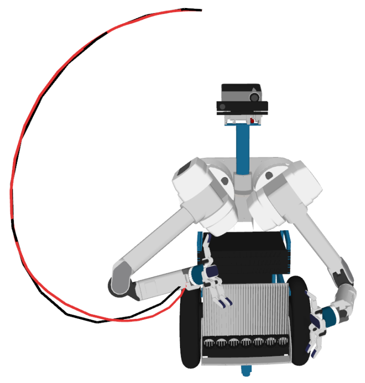
Figure 5.1: Our reference path is given in black and the output of our planner is given in red. For this plan, our layered graph used four waypoints and four inverse kinematic solutions.

5.1 Problem Statement

We are given a robot and a reference path in task space, $\bar{\xi}$ that is a polyline given by a sequence of waypoints $R \subseteq SE(3)$. As we will see shortly it will be convenient to treat $\bar{\xi}$ as a (one-dimensional) graph $G_{\bar{\xi}} = (V_{\bar{\xi}}, E_{\bar{\xi}})$ where $V_{\bar{\xi}}$, are the waypoints and an edge $e \in E_{\bar{\xi}}$ connects subsequent waypoints.

Our objective is to create a collision-free path $\xi \in \mathbb{C}$ whose forward kinematics maps to a path, $FK(\xi)$, in task space that follows $\bar{\xi}$ as close as possible, where "follows" is defined formally below. Similarly to

$\bar{\xi}$, our produced path ξ is a polyline represented by a sequence of waypoints. We are given a discriminative black box collision detector that, given a point in configuration space, returns whether or not the robot would be in collision.

As described in Sec. 3.3, we use the discrete Fréchet , (F_d) to compare the distances between two paths in task space to see how well one follows the other. Using the Fréchet distance we can restate our goal as find a collision free path ξ such that $\min\limits_{\xi \in \Xi} F_d(\bar{\xi}, \xi)$ where Ξ is the set of all task space paths whose begin and end task space poses are the same as $\bar{\xi}$.

5.2 Generating a Set of Candidate Path

The first step in our approach is to sample for candidate paths.

Consider the set of all configuration space paths that *exactly* map to our reference path $\bar{\xi}$. The set of all configurations along the path is:

$$S = \{q \in \mathbb{C} \,|FK(q) \in \bar{\xi}\}. \tag{5.1}$$

In other words, S is the set of all configurations that map to a point in task space on our reference path.

Alternatively we can define S as the inverse kinematics of all points along our reference path. Hence:

$$S = \bigcup_{\alpha \in [0,1]} IK(\bar{\xi}(\alpha)). \tag{5.2}$$

Thus S is a two-dimensional space parametrized by α, the location of a point in task space along $\bar{\xi}$ and k, the inverse kinematic solution of this point.

We sample this space to create a graph $L = (V_L, E_L)$ that will allow us to search for a path in configuration space. To leverage the two-dimensional structure we construct a layered graph L embedded in configuration space where each layer is a set of IK solutions for a waypoint.

Our graph is parameterized via two numbers: n, the number of waypoints we sample, and k, the number of inverse kinematic solutions at each point. These two numbers entirely capture our two-dimensional search space. We consider the fixed graph L_k^n and discuss densifying and expanding the graph in Sec. 5.5.

To construct our graph, we begin by sampling n waypoints in task space along our reference trajectory: $\{r_1...r_n\} \subseteq R$. To construct our graph, we begin by sampling n waypoints in task space along our reference trajectory: $\{r_1...r_n\} \subseteq R$. At each waypoint r_j, we generate up to k inverse kinematic solutions: $\{q_j^1...q_j^k\}$. Each configuration q_j^i

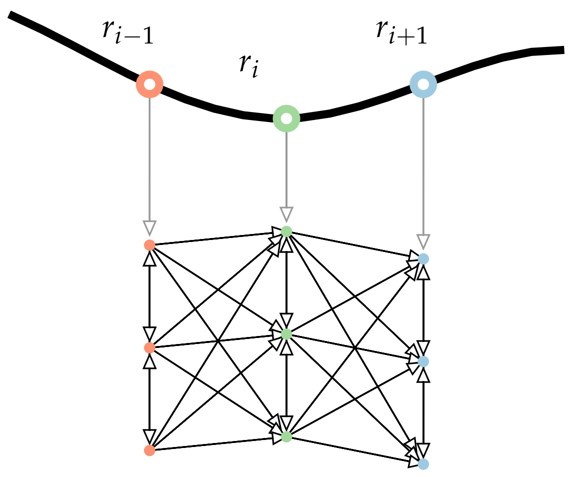

Figure 5.2: We show a layered graph for a portion of a reference path for $L_{k=3}^{n=3}$. We have a directed graph where we want to find a path from the left to right.

is a vertex in our graph L. Namely, the vertex set of our graph V_L is defined as: $V_L = \{q_i^j | 1 \leq i \leq n \text{ and } 1 \leq j \leq k\}$.

We next want to define our edge set, E_L. Each vertex in a layer of IK solutions connects to every vertex in the subsequent layer and to every vertex its own layer. Intuitively this allows us to pass through every waypoint, with the freedom to select any IK solution for that waypoint. More formally, our edge set is defined as:

$$\begin{aligned} E_L = &\{(q_j^{i_1}, q_{j+1}^{i_2}) | 1 \leq j_1 \leq n, 1 \leq j_2 \leq n\} \\ &\cup \{(q_j^{i_1}, q_j^{i_2}) | 1 \leq i_1 \leq n, 1 \leq i_2 \leq n\} \end{aligned} \tag{5.3}$$

Each of these edge represents a subpath of our entire path. Each subpath is a straight line segment in configuration space between the two configurations. We delay collision checking of these paths and will discuss that in further detail later.

A layered graph, $L_{k=3}^{n=3}$, for a portion of a path is shown in Fig.5.2.

We can restate our original objective described in Sec. 5.1 in terms of our layered graph. Our goal is to find the (shortest) path, ξ_L from any vertex in the first layer, q_i, to any vertex in our last layer, q_n, such that we minimize $F_d(\bar{\xi}, FK(\xi_L))$.

One naïve option would be to enumerate all candidate paths, ξ_c and compute $F_d(\bar{\xi}, \xi_c)$. However, there are $\Omega(n^k)$ candidate paths. We cannot evaluate Fréchet on individual edges, which would decrease the computation, because the Fréchet metric is a metric on paths, not on individual edges.

Instead we adapt a method that searches a cross product space between our layered graph and reference path.

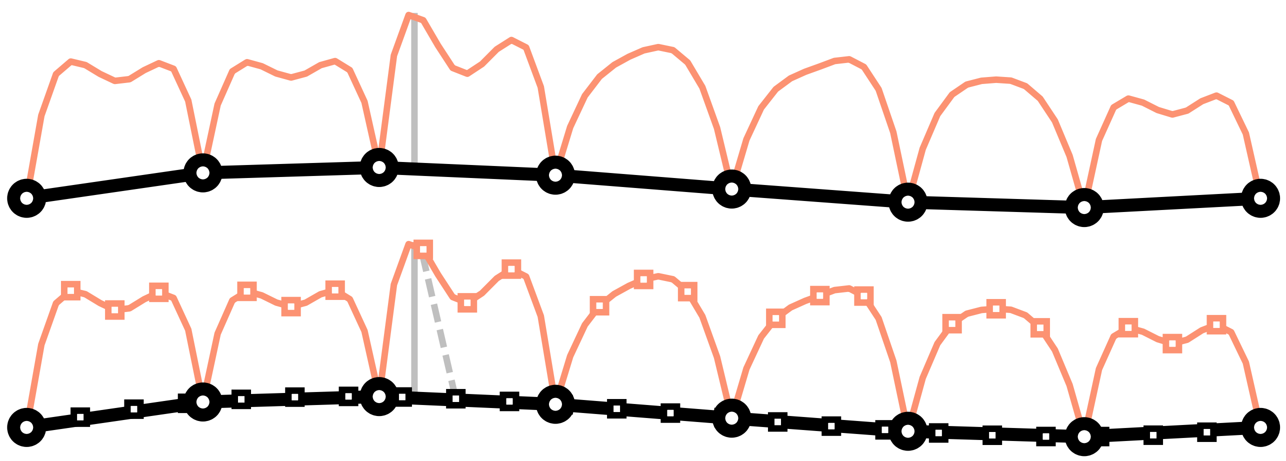
Figure 5.3: We show two levels of discretization. On the top we show the reference path in black and a candidate path in task space in orange. The continuous Fréchet is given at the grey line but because of our discretization the discrete Fréchet is 0. On the bottom we show a finer level of discretization that allows us to more accurately estimate the true Fréchet with the discrete Fréchet which is shown as the dotted grey line.

5.3 *Illustrative Example*

Consider a $L^n_{k=1}$. It is effectively a line graph of a candidate path. We use our inverse kinematic function to project each configuration into task space, ξ_c. We can see $\bar{\xi}$ in black and ξ_c in orange in top row of Fig.5.3. The continuous Fréchet would find the gray line as the point of maximum violation, the longest leash. Our Fréchet value is some value f_c. However, according to our discrete Fréchet $F_d(\bar{\xi}, \xi_c) = 0$.

This is quite alarming - is our Fréchet distance broken? Luckily, no. Our problem stems from our choice of discretization. The waypoints that represent ξ_c we chosen to have the same task space location as $\bar{\xi}$, shown as the black circles. Therefore our waypoints in task space are exactly the same, resulting in a distance of zero.

In order to more accurately capture the flow of each path we need to *subsample each edge*. Remembering from Sec. 5.2, each edge is a straight line path in configuration space. We can sample configurations along this path and use inverse kinematics to generate subsampled points in configuration space. Therefore, in our layered graph we replace each edge with a path composed of these subsampled points, shown in the bottom of Fig.5.3.

Our continuous Fréchet finds the longest leash at the solid grey line and the discrete Fréchet find the longest leash between two points, shown by the dotted gray line. As we increase our discretization, our discrete Fréchet better approximates the continuous Fréchet .

Therefore, by subsampling, we increase the discretization, allowing the discrete Fréchet to more accurately capture the flow of the path.

5.4 *Compute the Closest Path*

Our goal is to find a path in task space, ξ_L embedded in our layered graph L such that $F_d(\bar{\xi}, \xi_L)$. Both our graph and path are simplicial complexes, a mathematical set that is a generalization of points, polygonal curves, straight line graphs, meshes, triangulations, etc. Har-Peled and Raichel introduced an algorithm for computing the Fréchet distance between two curves on complexes by considering their cross product space (Har-Peled and Raichel, 2014). Therefore, our instance is a restricted case of their problem and we present our adaption below.

Cross Product Graph

We create a new graph $\Phi = (V_\Phi, E_\Phi)$ that is the cross product of our two graphs $G_{\bar{\xi}}$ and L. Each vertex in V_Φ is a tuple, (r, q) such that $r \in V_{\bar{\xi}}$ and $q = V_L$.

We next define the edge set, E_Φ where edges exist between two tuples if either or both elements are adjacent to each other in their respective graph. More formally:

$$\begin{aligned} E_\Phi = \{ & ((r_{m_1}, q_{i_1}^{j_1}), (r_{m_2}, q_{i_2}^{j_2})) \mid \text{if} \\ & ((q_{i_1}^{j_1}, q_{i_2}^{j_2}) \in E_L \text{ and } r_{m_1} = r_{m_2}) \text{ or} \\ & (q_{i_1}^{j_1} = q_{i_2}^{j_2} \text{ and } (r_{m_1}, r_{m_2}) \in E_{\bar{\xi}}) \text{ or} \\ & ((q_{i_1}^{j_1}, q_{i_2}^{j_2}) \in E_L \text{ and } (r_{m_1}, r_{m_2}) \in E_{\bar{\xi}}) \} \end{aligned} \tag{5.4}$$

We can interpret this graph structure by drawing comparisons between it and the typical dynamic programming approach for calculating the discrete Fréchet metric (Eiter and Mannila, 1994). Let us consider the simplest case with $L_{k=1}^n$, where there is only one inverse kinematic solution per waypoint and the graph represents one path.

Using the dynamic programming approach, we create a matrix of the waypoints in the reference path versus the path embedded in the graph. The dynamic programming solution finds the path from one end of the matrix to the other that minimize the the Fréchet value. This is equivalent to searching a graph of their cross product, which has the same matrix structure. By creating this cross product graph we can generalize the same approach to a graph with multiple paths.

Given our graph Φ, we assign a cost to each edge (referred to as elevation in (Har-Peled and Raichel, 2014)). For a vertex (r, q) we define it's cost as $C(r, q) = d_{TS}(r, FK(q))$. The cost of an edge is then the maximum of the cost of it's endpoints.

Bottleneck Search

With our cross product graph ϕ, we next want to find the path is the lowest Fréchet value. Given our construction of our graph this is equivalent finding the path whose maximum edge cost is minimal, or *bottleneck shortest path*. While there are efficient algorithms for this, linear in the number of edges (Har-Peled and Raichel, 2014), we can use a simple variant of Dijkstra's graph search (Dijkstra, 1959).

We briefly describe both algorithms and compare their performance. First we review the bottleneck search method from (Har-Peled and Raichel, 2014).

Let s be the starting node and t be the end node. Assuming s and t are connected, we proceed in a recursive manner. At each stage we compute the median edge weight, E_{med}. If s and t are connected in the graph $G_{\leq med}$, a graph with all edges above the median weight removed, then we recurse on this smaller graph. In this case we know the bottleneck path has cost equal to or below E_{med} Otherwise, our cost is above E_{med}. We contract the connected components of $G_{\geq med}$ down to vertices and recurse on this graph. Once we have a constant number of edges, we find the minimum bottleneck path via brute force.

This search results in the minimal possible Fréchet value, f_{Φ}, but not the path that produces that value. Therefore, we perform a post-processing step to reconstruct the path. We prune the the original graph, prior to contractions and deletions, of any edge who has a Fréchet value greater than f_{Φ}. Since we know there exists a path of Fréchet cost f_{Φ}, any edge with a higher cost will not be useful to us. From this pruned graph, we conduct a simple shortest path search weighted by the Fréchet cost of the edge.

We compare this alongside a variant of Dijkstra. Normally Dijkstra estimates the cost to vertex v coming vertex u as the distance to u plus the cost of the edge (u, v). However, we are not interested in the distance but the bottleneck. Therefore we swap the addition, the sum of costs, for a $max()$ such that the cost$(v) = max($cost(v), cost$(u, v))$. Therefore the bottleneck is the maximum of either the cost of the current edge or the bottleneck cost of the path leading to this edge.

While Dijkstra's algorithm, and our variant, is asymptotically larger, $\mathcal{O}(|E| + |V|log|V|)$, we have empirically found it to be faster. We vary the number of inverse kinematic solutions and waypoints and compare the two algorithms across the number of vertices in Fig.5.4. Here the linear in edges algorithm is labeled Har-Peled after an author of (Har-Peled and Raichel, 2014). We see that our Dijsktra variant takes less time and grows at a slower rate.

Our other motivation for using the Dijkstra-variant is that it allows for more efficient updates, which we discuss as an area of future work in Sec. 5.5.

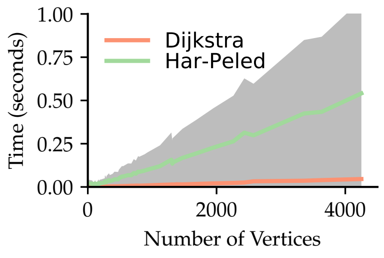

Figure 5.4: We compare search time in the algorithm presented in (Har-Peled and Raichel, 2014) as compared to our variant of Dijkstra. We see that the Dijkstra variant both takes less time and grows at a slower rate.

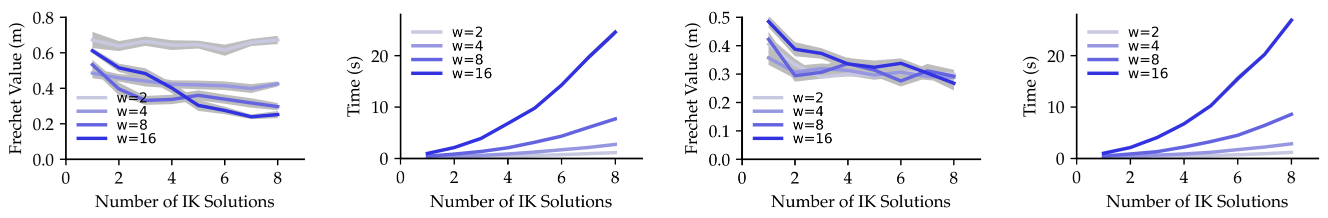

Figure 5.5: We compare the time and Fréchet value across differenet along our two parameters: number of IK solutions and number of waypoints.

Experimental Validation

In the following section we briefly summarize the results of this algorithm. We look forward to conducting more testing as an area of future work in conjunction with graph updates which we will discuss in Sec. 5.5.

When planning for a particular reference path our algorithm has three knobs that can turned. The first of which is the number of waypoints, n. The second is the number inverse kinematic solutions, k. Finally, we can also specify the degree of discretization has mentioned in Sec. 5.3. For the following experiments we maintained a constant level of discretization and varied n and k.

Since sampling inverse kinematic solutions is a randomized process, our planner is also randomized. Therefore, for any particular set of parameters, we average across many runs of the planner. Additionally, each reference path is of varying length, shape and therefore difficulty. As such, we cannot average across different reference path inputs. Therefore, we show the results for a few randomly selected reference paths.

We vary both k, the number of IK solutions, and n, the number of waypoints. We compare across two metrics. The first is the total planning time, measured in seconds. The second is the discrete Fréchet distance between the reference path $\bar{\xi}$ and the path outputted by the planner ξ, measured in meters.

The results for two reference paths are shown in Fig.5.5, with the two graphs on the left corresponding to one path and the two on the right showing the results for the other reference path.

We first consider the timing information. Unsurprisingly, the planning time increases as we have more inverse kinematic solutions and more waypoints because in both dimensions we need to sample more poses and search a larger graph. Another interesting note is that our time profiles between two different graphs are remarkably similar. This is because although we are solving different problems, the specific of the problem do not effect our planning time because each operation is the same length and we are operating on a fixed graph. This also explains why our error bars are nearly nonexistent.

We next consider our Fréchet values of our produced graph. We see a similar two trends: the Fréchet decreases as we have more inverse kinematic solutions and more waypoints. Adding more waypoints further constraints and defines the reference path while more IK solutions give more possible path segments to chose from.

5.5 Future Work: Densification

Our planner considers a fixed layered graph, L_k^n. Our next immediate step of future work is how we can incrementally explore our space and improve our solution by densifying our graph. Given a solution on our initial graph, we would like to sample more points in such a way that we, hopefully, improve our solution. This introduces two sampling questions: how do we sample new points and where do we sample these points. To answer the first question, we can augment our layered graph, and therefore our cross product graph, by sampling more inverse kinematic solutions in a layer, by adding a layer or by subsampling existing edges. To answer the where question, the obvious reply is to densify at the location of the highest bottleneck, since this is where the leash of our Fréchet is longest. This serves only as a heuristic thought, as the Fréchet is not a local property. Therefore, to balance this out and insure we explore the search space, we can both densify at the area of highest bottleneck and randomly throughout the graph.

Following each densification, we would research our graph for the new solution. However, we do not want to restart our search from scratch. Instead, we would hope to reuse as much information from our previous search as possible. By using Dijkstra as our search method we can leverage method like LPA* to perform efficient incremental search (Koenig et al., 2004).

Therefore, following our initial search we can incrementally densify to create an anytime algorithm. We look forward to implementing and experimentally verifying this densification process.

6
Conclusion

We conclude this thesis by summarizing our contributions, discussing future work and offering a few concluding remarks. The goal of this thesis was to enable a robot to follow a reference path in task space. Doing so involves answering two question: how to do measure the distance between two paths and, using this, how to we create paths that minimize their distance to our reference path.

To answer the first question we leverage tools from computational geometry to quantify the distance between two paths in Sec. 3. Specifically, we show that the Fréchet distance is an straightforward and natural metric for comparing distances. Equipped with this metric we then present two planning strategies.

In our first planning strategy we pose this as a trajectory optimization problem where our cost is equally our distance metric (Sec. 4). To assist our optimizer, we presented two methods for constraining the trajectory. However, by constraining our trajectory, we limited our inverse kinematic search space, which posed a significant limitation.

Therefore, to overcome this limitation we developed a graph search algorithm, given in Sec. 5, that sampled and searched over a set of inverse kinematic solutions. By transforming our problem into a series of graph structures, we could find the path with the minimal Fréchet compared to our reference path via a simple variant of Dijkstra's search algorithm.

Throughout the thesis we provide illustrative examples and simulation experiments that verify and demonstrate the efficacy of our approach.

6.1 Future Work

Taking a step back, we can still generalize further in recreating a shapes in task space. Imagine that someone is trying to show the robot how to draw the letter 'A'. This person may have a distribution of 'A' that they consider an acceptable representation. Therefore, in learning this skill, we want to optimize not be close to a single demonstration, but close to the distribution of demonstrations. We believe we could use an alternative metric, the Mahalanobis distance metric (Mahalanobis, 1936), often used in handwriting matching, that would allow us to measure distances with respect to a distribution (Kato et al., 1999).

Both of our methods concentrate on following a particular path, or as discussed above, a set of paths. However, there may be other task space constraints that we would want to incoporate. For example, when carrying a glass full of water, we typically keep the glass upright to not spill the contents. This constraint specifies that we maintain an object's orientation, but does not encode any particular path. One way to encode this constraint with our optimization method is specify a cost function that defines the distance between the current orientation and the desired one. For a series of possible task space constraints, it would be interesting to explore how generalizable the techniques in this thesis generalize.

6.2 *Concluding Remarks*

This thesis takes a step towards developing metrics and planners that handle task space constraints. Notably, we leverage tools from computational geometry to develop efficient motion planners and are looking forward to finding more connections between these two fields.

As discussed in Sec. 6.1, we hope to take this as a stepping stone to develop robust, efficient planners that accommodate a variety of constraints. In conjunction, we need to insure that the way we specify these constraints is intuitive and user-friendly, creating seamless interactions between the robot and its human collaborators. By creating robots that can plan with constraints, we open new robotic possibilities.

Bibliography

Pankaj K Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

Shaheen Ahmad and Shengwu Luo. Coordinated motion control of multiple robotic devices for welding and redundancy coordination through constrained optimization in cartesian space. *TRA*, 5(4): 409–417, 1989.

Juan Manuel Ahuactzin and Kamal Gupta. A motion planning based approach for inverse kinematics of redundant robots: the kinematic roadmap. *Expert Systems with Applications*, 14(1):159–167, 1998.

Juan Manuel Ahuactzin and Kamal K Gupta. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *TRA*, 15(4):653–669, 1999.

Helmut Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, pages 235–248. Springer, 2009.

Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.

Nancy M Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *ICRA*, volume 1, pages 113–120. IEEE, 1996.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *RAS*, 57(5): 469–483, 2009.

E Belogay, Carlos Cabrelli, Ursula Molter, and Ron Shonkwiler. Calculating the hausdorff distance between curves. *Information Processing Letters*, 64(1):17–22, 1997.

Thomas Benseghir, Grégoire Malandain, and Régis Vaillant. Iterative closest curve: a framework for curvilinear structure registration application to 2d/3d coronary arteries registration. In *Medical Image Computing and Computer-Assisted Intervention*, pages 179–186. Springer, 2013.

Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *ICRA*, pages 625–632. IEEE, 2009.

Pierre Bessiere, J-M Ahuactzin, E-G Talbi, and Emmanuel Mazer. The "ariadne's clew" algorithm: global planning with local methods. In *IROS*, volume 2, pages 1373–1380. IEEE, 1993.

Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *ICRA*, volume 1, pages 521–528. IEEE, 2000.

Erin Wolf Chambers, Eric Colin De Verdiere, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010.

Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *Annual Symposium on Combinatorial Search*, 2015.

Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. *arXiv*, 2016.

Tag der Einreichung. Combining human demonstrations and motion planning for movement primitive optimization. Master's thesis, Technische Universitat Darmstadt, 2016.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Marie-Pierre Dubuisson and Anil K Jain. A modified hausdorff distance for object matching. In *ICPR*, volume 1, pages 566–568. IEEE, 1994.

Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

M Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22(1):1–72, 1906.

Cyrille Froissart and Pierre Mechler. On-line polynomial path planning in cartesian space for robot manipulators. *Robotica*, 11(03): 245–251, 1993.

Colin Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society*, pages 285–339, 1991.

John C Gower. Generalized procrustes analysis. *Psychometrika*, 40(1): 33–51, 1975.

Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *TALG*, 10(1):3, 2014.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Felix Hausdorff and Egbert Brieskorn. *Felix Hausdorff-Gesammelte Werke Band III: Mengenlehre (1927, 1935) Deskripte Mengenlehre und Topologie*, volume 3. Springer Science & Business Media, 2008.

Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *ICRA*, pages 2493–2498. IEEE, 2010.

Rachel M Holladay and Siddhartha S Srinivasa. Distance metrics and algorithms for task space path optimization. In *IROS*, 2016.

Th Horsch, F Schwarz, and Henning Tolle. Motion planning with many degrees of freedom-random reflections at c-space obstacles. In *ICRA*, pages 3318–3323. IEEE, 1994.

Daniel P Huttenlocher and Klara Kedem. Computing the minimum hausdorff distance for point sets under translation. In *Annual symposium on Computational geometry*, pages 340–349. ACM, 1990.

M Kallman and Maja Matarić. Motion planning using dynamic roadmaps. In *ICRA*, volume 5, pages 4399–4404. IEEE, 2004.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *IJRR*, 30(7):846–894, 2011.

Nei Kato, Masato Suzuki, Shin'ichiro Omachi, Hirotomo Aso, and Yoshiaki Nemoto. A handwritten character recognition system using directional element feature and asymmetric mahalanobis distance. *Transactions on Pattern Analysis and Machine Intelligence*, 21(3):258–262, 1999.

Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration for fast path planning. In *ICRA*, pages 2138–2145. IEEE, 1994.

Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *TRA*, 12(4):566–580, 1996.

Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy. Incremental heuristic search in ai. *AI Magazine*, 25(2):99, 2004.

Nikolaos Larios, UOA GR, Christos Mitatakis, and Dimitrios Gunopulos. Evaluating distance measures for trajectories in the mobile setting. 2015.

Nathan Larkin, Andrew Short, Zengxi Pan, and Stephen van Duin. Automatic program generation for welding robots from cad. In *AIM*, pages 560–565. IEEE, 2016.

Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *IJRR*, 20(5):378–400, 2001.

Tomas Lozano-Perez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2: 49–55, 1936.

Emmanuel Mazer, Juan Manuel Ahuactzin, and Pierre Bessiere. The ariadne's clew algorithm. *JAIR*, 9:295–316, 1998.

Giuseppe Oriolo, Mauro Ottavi, and Marilena Vendittelli. Probabilistic motion planning for redundant robots along given end-effector paths. In *IROS*, volume 2, pages 1657–1662. IEEE, 2002.

Richard Paul. Manipulator cartesian path control. *Transactions on Systems, Man, and Cybernetics*, 9(11):702–711, 1979.

Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

Sanjeev Seereeram and John T Wen. A global approach to path planning for redundant manipulators. *TRA*, 11(1):152–160, 1995.

Alexander Shkolnik and Russ Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *ICRA*, pages 2061–2067. IEEE, 2009.

Mike Stilman. Global manipulation planning in robot joint space with task constraints. *Transactions on Robotics*, 26(3):576–584, 2010.

Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

Lianfang Tian and Curtis Collins. An effective robot trajectory planning method using a genetic algorithm. *Mechatronics*, 14(5): 455–470, 2004.

Nathan Ulrich and Vijay Kumar. Passive mechanical gravity compensation for robot manipulators. In *ICRA*, pages 1536–1541. IEEE, 1991.

Carola Wenk et al. Geodesic fréchet distance inside a simple polygon. *TALG*, 7(1):9, 2010.

Wenfu Xu, Cheng Li, Bin Liang, Yu Liu, and Yangsheng Xu. The cartesian path planning of free-floating space robot using particle swarm optimization. *International Journal of Advanced Robotic Systems*, 5(3):301–310, 2008.

Zhenwang Yao and Kamal Gupta. Path planning with general end-effector constraints. *RAS*, 55(4):316–327, 2007.

Gu Ye and Ron Alterovitz. Demonstration-guided motion planning. In *ISRR*, volume 5, 2011.

Franziska Zacharias, Wolfgang Sepp, Christoph Borst, and Gerd Hirzinger. Using a model of the reachable workspace to position mobile manipulators for 3-d trajectories. In *Humanoids*, pages 55–61. IEEE, 2009.

Index