Distance Metrics and Algorithms for Task Space Path Optimization

Rachel M. Holladay[1] and Siddhartha S. Srinivasa[1]

Abstract— **We propose a method for generating a configuration space path that closely follows a desired task space path despite the presence of obstacles. We formalize closeness via two path metrics based on the discrete Hausdorff and Fréchet distances. Armed with these metrics, we can cast our problem as a trajectory optimization problem. We also present two techniques to assist our optimizer in the case of local minima by further constraining the trajectory. Finally, we leverage shape matching analysis, the Procrustes metric, to compare with respect to only their shape.**

I. Introduction

To create more capable robots in the home, we need our robots to be adaptable and able to quickly and easily learn new skills. In robotics a popular machine learning paradigm is teaching via demonstration, where someone teaches a robot a skill by showing the robot how to perform that skill [1]. By this, the robot programmers are freed from having to program specific tasks. Instead, the robot's collaborators can provide the required and desired skill set. This shifts the problem from programming specific tasks to programming the ability to teach and learn tasks.

This work focuses on skills where the robot needs to recreate some desired motion in the task space. For example, in Fig.1 a user provides a demonstration to the robot. We now want the robot to be able to recreate the reference path, shown on the right. While our motions are in task space, our robot plans in its configuration space. Therefore our goal is to generate a configuration space path that achieves a desired task space motion.

The naïve approach for creating the desired 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. Therefore by optimizing in full configuration space, we leverage its general ability to execute collision-free planning in various environments [2].

A prior approach is to use randomized planners that plan in joint space but use task space constraints to drive sampling and project actions [3–5]. These methods produce feasible trajectories that adhere to the constraints but are often sub-optimal. Incoporating task and joint space constraints has also been approached as a finite time nonlinear control problem [6]. Alternatively, some have generated the mapping from human motion to robot motion, allowing the robot to recreate the human's trajectories [7]. Instead, we wish to draw upon work in trajectory optimization to generate optimal paths that follow a task space path [8–10].

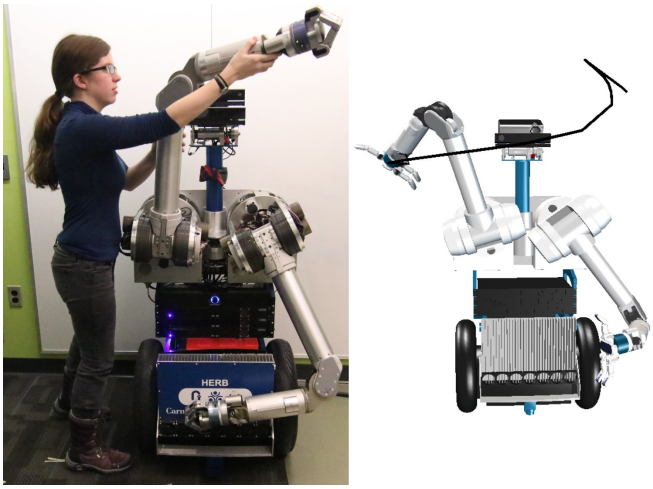


Fig. 1: On the left hand side, a user demonstrates a task space trajectory that is visualized on the right. Our goal is to enable the robot to be able to recreate the shape of the provided demonstration in the general setting.

Our key insight is that we can recreate task space demonstrations by optimizing a configuration space path with respect to a *cost function* defining the distance between the demonstration 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 compare two common shape matching metrics, the Hausdorff and the Fréchet distance.

We therefore make the following contributions:

1. Formulation as Optimization Problem. Using common distance metrics we formulate task space planning as a trajectory optimization problem.

2. Matching Algorithms. Trajectory optimization is succeptible to local minimum. This can lead the optimizer to generate paths that do not match the task space demonstration. To account for this we offer two methods, splitting and stapling. These methods guide the optimization to the correct basin of attraction by identifying critical points in the task space demonstration that serve as hard constraints to the optimizer.

3. EM-inspired Procrustes Analysis. Finally, often the demonstration does not have to have strong bindings to the particular task space location. Rather the

[1]Rachel M. Holladay and Siddhartha S. Srinivasa are with the Robotics Institute, Carnegie Mellon University. rmh@andrew.cmu.edu, siddh@cs.cmu.edu

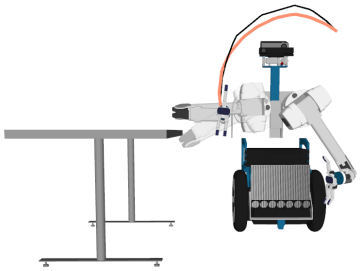


Fig. 2: Simply replaying recorded demonstrations, fails in new environments. Our planner is able to plan around obstacles.

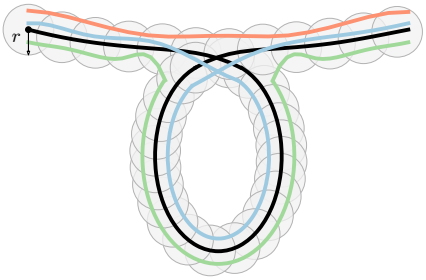


Fig. 3: The one way (orange) and two way (green) Hausdorff distances are constrained to be in the safe zone while the Fréchet distance (blue) forces us to pass through the safe zone in order.

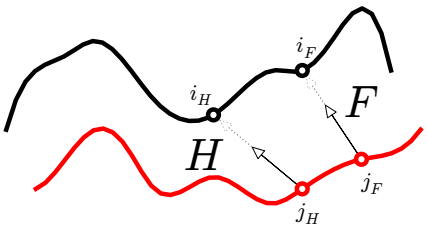


Fig. 4: The negative analytical gradient of the Fréchet and Hausdorff distance steps to decrease the distance.

demonstration 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.

Fig.2 shows an instance where our planner succeeds 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.

In enabling the robot to learn skills, we have only considered skills that are encoded as unique paths in task space. We therefore cannot capture skills that rely on sensory feedback or dynamic interaction.

II. Problem Statement

We work with a robot manipulator endowed with a configuration space $q \in \mathcal{C}$. We map configurations to the task space $x \in SE(3)$ via forward kinematics $x = FK(q)$. We are given a *reference path* either in configuration space $\bar{\xi}_q : [0,1] \to \mathcal{C}$ or in task space $\bar{\xi}_x : [0,1] \to SE(3)$. We will drop the path subscript wherever they can be used interchangeably. In general, we will only assume we have access to $\bar{\xi}_x$.

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

$$\xi^* = \arg\min_{\xi \in \Xi} ||\xi - \bar{\xi}|| \quad \text{s.t. constraints} \tag{1}$$

Two aspects of (1) are critical: (1) the distance metric used for estimating the closeness of curves, and (2) the constraints the robot faces in a new situation.

A. Distance Metrics

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. 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*.

One-Way Hausdorff Distance. The Hausdorff distance is a method for measuring how far apart two subsets of metric space are [11]. 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 [12–14]. Hence for paths we can formalize the one way Hausdorff distance as:

$$H(\xi_x, \bar{\xi}_x) = \sup_{y \in \xi} \inf_{\bar{y} \in \bar{\xi}} d(y, \bar{y}) \tag{2}$$

where, for d, we form a metric in $SE(3)$ via the Cartesian product of the Euclidean metric for $\mathbb{R}^3$ and the standard great circle solid angle metric for $SO(3)$. In practice, we compute the discrete Hausdorff distance using a set of waypoints sampled from the path.

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, 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 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 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*.

Fréchet Distance. The Fréchet distance captures the difference in flow between two curves [15]. 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 [16]. 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:

$$F(\xi_x, \bar{\xi_x}) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \left\{ d\Big(\xi_x(\alpha(t)),\, \bar{\xi_x}(\beta(t))\Big) \right\} \quad (3)$$

where d is defined as before. In practice, we sample discrete waypoints on the paths and compute the discrete Fréchet distance via dynamic programming [17, 18].

We can see with the blue path in Fig.3 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.

B. 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.4 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.4, the two distance metrics may select different (i,j) pairs. However, for both, the distance 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.4, we move our point of maximum violation closer to the reference path, thus decreasing our distance. Hence in Fig.4, gradient applied to each distance metric's (i,j) would decrease their P.

C. Constraints

We would like to address a broad range of constraints encountered in manipulation, including collisions with obstacles and other links of the robot, joint limits, and other dynamics. Here, we use the power of existing trajectory optimization algorithms, specifically TrajOpt [10] which handles a broad range of constraints types. Our goal is to formulate our distance metrics in a format compatible with these algorithms and harness their power.

III. 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. II-A 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.

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.

A. 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 [19]. 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.

B. Initial Results

We created a set of 24 demonstrations, some of which are shown in black in Fig.5. 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.5 shows each original demonstration in black, the the Fréchet optimization in blue and the Hausdorff optimization in orange. Each bar below compares

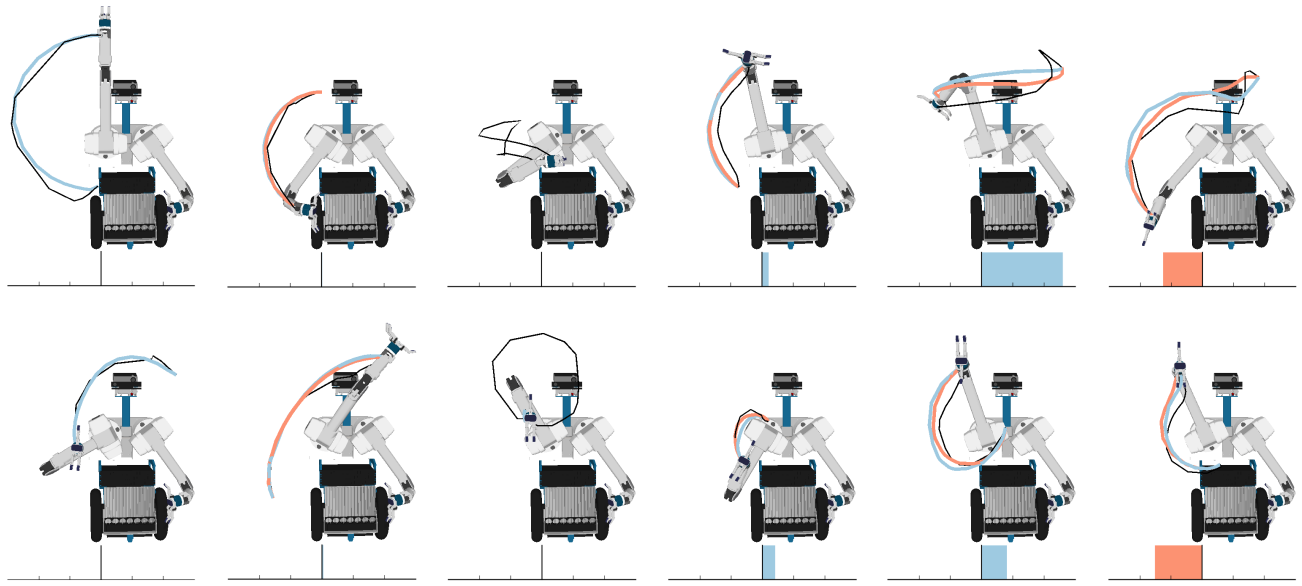


Fig. 5: The demonstration is shown in black. The path optimized according to the Hausdorff and Fréchet metrics are in orange and blue, respectively. The bars below show the difference between the Fréchet error of the two optimized paths.

the difference in Fréchet error between the Fréchet optimization and Hausdorff optimization. We elected to compare using the Fréchet error since the Fréchet metric restricts us to follow the curves of the reference path.

In Fig.5, 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.6, we show planning time as a function of the number of iterations that the optimizer runs. Each point represents a demonstration where corresponding demonstrations 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.5 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 demonstrations are difficult

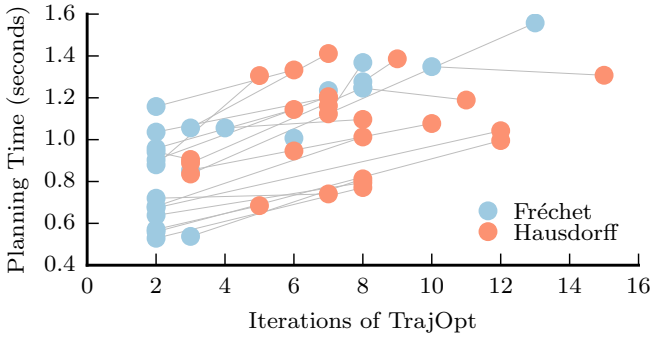


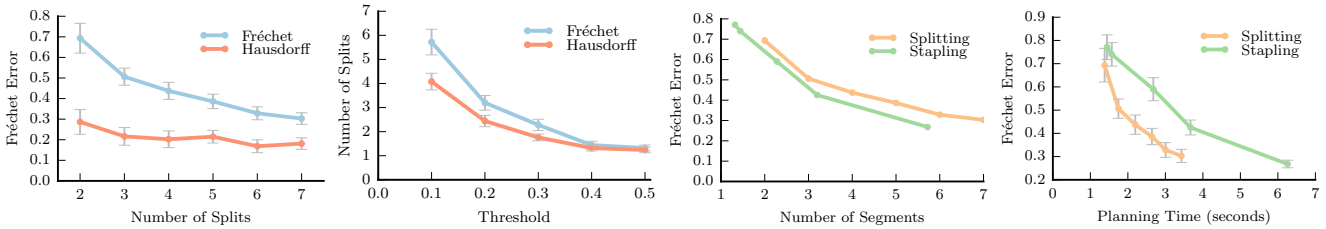
Fig. 6: Each point represents a demonstration and corresponding demonstrations optimized with each metric are connected via a grey segment. The Fréchet optimizations trend to take longer and planning time increases with the number of iterations.

for an optimizer to achieve. Many times, the optimizer falls into a local minimum that is different from our demonstration due to self-collisions or joint limits.

In order to produce more accurate demonstrations 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.

IV. 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



(a) Splitting: Error (b) Stapling: Segment Count (c) Comparing Segments for Error (d) Comparing Time for Error

Fig. 7: In joint space, as we increase the number of splits, the error decreases (a). Likewise, if we decrease the violation threshold when stapling, the number of segments increases (b). Comparing, stapling achieves a given Fréchet error in fewer segments (c) but a higher planning time (d) then splitting.

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: $\quad t_s = \frac{i}{k}$
5: $\quad t_e = \frac{i+1}{k}$
6: $\quad \xi^i = \text{PLANTO}(\bar{\xi_q}(t_s), \bar{\xi_q}(t_e), m)$
7: $\quad \xi_q = \text{COMBINETRAJS}(\xi_q, \xi^i)$
8: **return** ξ_q

Algorithm 2 StapleJoint

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

optimizing in joint space to generalize the to the more difficult problem of optimizing in task space.

A. 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.7a, 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. IV-C. 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.

B. 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 metric we begin with a path optimized to that metric m, like those created in Sec. III-B (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. II-A, 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.

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.7b, 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.

C. 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

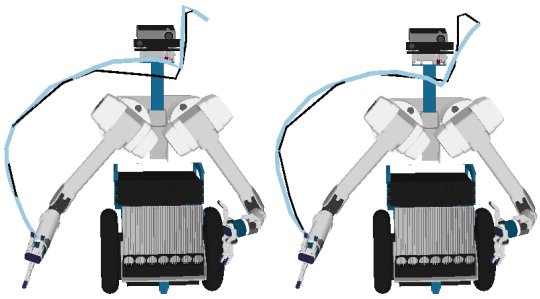


Fig. 8: In joint space: splitting with $k = 6$ (left) compared to stapling $\epsilon = 0.2$ (right).

violation, staples down the path at the point that is most useful to adhering to the threshold. Thus, as shown in Fig.7c, for some given target Fréchet error, the stapling method required fewer segments.

However, as illustrated in Fig.7d, 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.8 shows how stapling and splitting improve our performance, compared to the same path in Fig.5.

D. 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, 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 $\bar{\xi_x}$.

V. 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.

A. 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).

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: $\quad t_e = \frac{i+1}{k}$
6: $\quad Q_e$ = SampleIK($\bar{\xi_x}(t_e)$)
7: $\quad \xi^i$ = PlanTo(Q_s, Q_e, m)
8: $\quad \xi_q$ = CombineTrajs(ξ_q, ξ^i)
9: $\quad Q_s = \{\xi_q(1)\}$
10: **return** ξ_q

Algorithm 4 StapleTask

1: **Given:** Reference Path $\bar{\xi_q}$, metric m, threshold ϵ, 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: ξ_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: $\quad \hat{Q}_g$ = SampleIK($\bar{\xi_x}(p_i)$)
8: $\quad \xi_q^a$ = StapleTask($\bar{\xi_x}, Q_s, \hat{Q}_g, m$)
9: $\quad Q_s = \{\xi_q^a(1)\}$
10: $\quad \xi_q^b$ = StapleTask($\bar{\xi_x}, \hat{Q}_s, Q_g, m$)
11: $\quad$ **return** CombineTrajs($\xi_q^a), \xi_q^b$
12: **else**
13: $\quad$ **return** ξ_q

Similarly to in joint space, as the number of splits increases, the Fréchet error decreases, as seen in Fig.9a.

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).

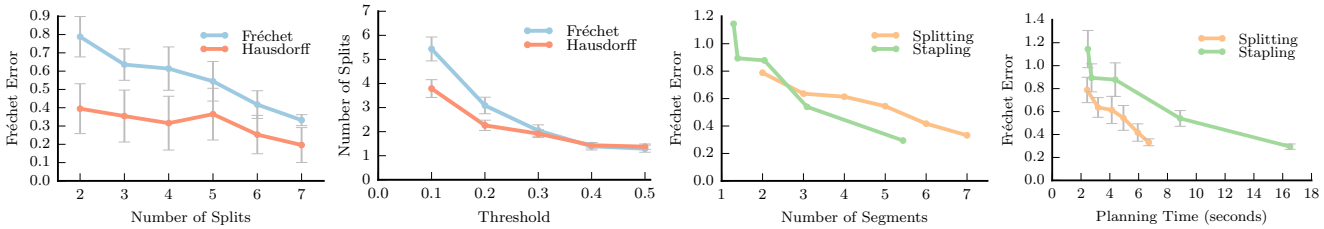
As we decrease our threshold, we increase the number of splits required, visualized in Fig.9b.

B. Splitting versus Stapling

As seen in Fig.9c, 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.9d. 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.

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.10, we can translate



(a) Splitting: Error (b) Stapling: Segment Count (c) Comparing Segments for Error (d) Comparing Time for Error

Fig. 9: In task space, as we increase the number of splits, the error decreases (a). Likewise, if we decrease the violation threshold when stapling, the number of segments increases (b). Across splitting and stapling, more segments leads to a decreased Fréchet error (c) For an allowance of planning time, splitting achieves a lower Fréchet error (d).

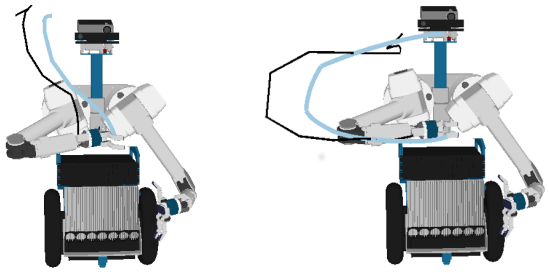


Fig. 10: The original reference path is given in black and the blue path is translated 10 cm.

Algorithm 5 ProcrustesConstraint

1: **Given:** Initial Path ξ_q^i, demonstration $\bar{\xi_x}$, metric m
2: p_0 = CORRESPOND($\xi_q^i, \bar{\xi_s}, m$)
3: **procedure** WHILE NOT MATCHES:
4: $\quad \xi_q^i$ = PROCRUSTES($\xi_q^i, \bar{\xi_s}$)
5: $\quad p_n$ = CORRESPOND($\xi_q^i, \bar{\xi_s}, m$)
6: $\quad$ matches = (p_0 == p_n)
7: $\quad p_0 = p_n$
8: **return** ξ_q

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.

VI. 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 metric from shape analysis [20–22]. 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.

A. 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 [23], 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.

VII. Discussion

We present a method for recreating paths by optimizing new instances to be close the reference path according to commonly used distance metrics. We present a splitting and stapling method for improving performance. Additionally, we show how Procrustes analysis can be used to optimizing only according to the demonstration's shape.

A. Limitations and Future Work

We can still move move one step further in concentrating on recreating a shape. 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 [24], often used in handwriting matching, that would allow us to measure distances with respect to a distribution [25].

We hope to pursue this exciting line of future work to better enable our robots to learn complex skills.

ACKNOWLEDGMENT

This material is based upon work supported by ONR BAA 13-0001 and CRA-W's CREU and CMU's SRC URO's undergraduate research grants. We would like to thank the members of the Personal Robotics Lab for helpful discussion and advice. We would especially like to thank Jennifer King and Christopher Dellin for their comments on drafts of this work.

REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *RAS*, vol. 57, no. 5, pp. 469–483, 2009.

[2] G. Ye and R. Alterovitz, "Demonstration-guided motion planning," in *ISRR*, vol. 5, 2011.

[3] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE ICRA*, pp. 625–632, IEEE, 2009.

[4] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *Robotics, IEEE Transactions on*, vol. 26, no. 3, pp. 576–584, 2010.

[5] Z. Yao and K. Gupta, "Path planning with general end-effector constraints: Using task space to guide configuration space search," in *IEEE/RSJ IROS*, pp. 1875–1880, IEEE, 2005.

[6] S. Seereeram and J. T. Wen, "A global approach to path planning for redundant manipulators," *IEEE TRA*, vol. 11, no. 1, pp. 152–160, 1995.

[7] G. Maeda, M. Ewerton, D. Koert, and J. Peters, "Acquiring and generalizing the embodiment mapping from human observations to robot skills," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 784–791, 2016.

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE ICRA*, pp. 489–494, IEEE, 2009.

[9] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE ICRA*, pp. 4569–4574, IEEE, 2011.

[10] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.," in *RSS*, vol. 9, pp. 1–10, Citeseer, 2013.

[11] F. Hausdorff and E. Brieskorn, *Felix Hausdorff-Gesammelte Werke Band III: Mengenlehre (1927, 1935) Deskripte Mengenlehre und Topologie*, vol. 3. Springer Science & Business Media, 2008.

[12] M.-P. Dubuisson and A. K. Jain, "A modified hausdorff distance for object matching," in *ICPR*, vol. 1, pp. 566–568, IEEE, 1994.

[13] D. P. Huttenlocher and K. Kedem, "Computing the minimum hausdorff distance for point sets under translation," in *Annual symposium on Computational geometry*, pp. 340–349, ACM, 1990.

[14] E. Belogay, C. Cabrelli, U. Molter, and R. Shonkwiler, "Calculating the hausdorff distance between curves," *Information Processing Letters*, vol. 64, no. 1, pp. 17–22, 1997.

[15] M. M. Fréchet, "Sur quelques points du calcul fonctionnel," *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, vol. 22, no. 1, pp. 1–72, 1906.

[16] E. W. Chambers, E. C. De Verdiere, J. Erickson, S. Lazard, F. Lazarus, and S. Thite, "Homotopic fréchet distance between curves or, walking your dog in the woods in polynomial time," *Computational Geometry*, vol. 43, no. 3, pp. 295–311, 2010.

[17] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[18] T. Eiter and H. Mannila, "Computing discrete fréchet distance," tech. rep., Citeseer, 1994.

[19] N. Ulrich and V. Kumar, "Passive mechanical gravity compensation for robot manipulators," in *IEEE ICRA*, pp. 1536–1541, IEEE, 1991.

[20] J. C. Gower, "Generalized procrustes analysis," *Psychometrika*, vol. 40, no. 1, pp. 33–51, 1975.

[21] C. Goodall, "Procrustes methods in the statistical analysis of shape," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 285–339, 1991.

[22] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[23] T. Benseghir, G. Malandain, and R. 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*, pp. 179–186, Springer, 2013.

[24] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences (Calcutta)*, vol. 2, pp. 49–55, 1936.

[25] N. Kato, M. Suzuki, S. I. Omachi, H. Aso, and Y. Nemoto, "A handwritten character recognition system using directional element feature and asymmetric mahalanobis distance," *IEEE TPAMI*, vol. 21, no. 3, pp. 258–262, 1999.