Creating Constrained Paths
Rachel Holladay

Robots that work in unstructured human environments will have to adapt quickly to their surroundings and learn new skills daily. It is unrealistic to assume that experts will be able to program robots to do each of these skills. Instead, we want to be able to easily teach our robots new skills in a similar way that we would teach these skills to a child. Namely we provide an example of a new skill, thus giving the robot a demonstration, and want the robot to be able to generalize this skill to new situations. This concept, learning from demonstration is a popular and powerful technique for expanding robot capabilities [1, 2].

Imagine we wish our robot our draw the alphabet, beginning with drawing the letter ‘A’. We kinetically teach the robot how to make this motion by demonstrating this on the robot, thus providing it with a demonstration path $D$. Given a new environment, with possibly different endpoints, obstacles and scaling, we want to our robot to draw that ‘A’ as best it can. But what is the best it can do and how can achieve that?

One of our general purpose motion planners will output a path $P$ that we will assume is collision free. We want $P$ to be similar to $D$ in the sense that it achieves the desire goal, drawing the ‘A’. In particular we do not care if the drawn ‘A’ is a different size, in different location or at a different rotation. Hence we want translation invariant distance metrics between curves that will allow us to judge how similar two curves are.

Judging similarity between curves is a well studied topic that has been applied to classifying handwriting databases [3] and aligning curves for object and character recognition [4]. We propose to investigate 4 distance metrics, the well-studied Hausdorff and Frechet distances and well as two simplistic ones of our own design. A brief description of each is given below.

**Distance:** We discretize $D$ and $P$ in to $n$ evenly spaced waypoints. Our goal is minimize the distance between $D_i$ and $P_i$, so our metric is the sum squared difference between each $D_i$ and $P_i$ pair. This is a naive, but starting, metric that will attempt to force $P$ to exactly match $D$, thus not being invariant to any changes.

**Velocity:** Again we discretize $D$ and $P$ in to $n$ evenly spaced waypoints. In this case we compute the velocity from $D_i$ to $D_{i+1}$ and compare it to the velocity from $P_i$ to $P_{i+1}$. Once again the overall metric is the sum squared velocity and we hope to minimize this. The metric works to maintain the general shape of the curve and is invariant to various locations. This metric, however, is not rotation invariant, which could be desirable in certain scenarios.

**Hausdorff Distance:** This distance metric is defined such that two curves are close if every point on either set is close to some other point of the other curve [5, 6]. This is a commonly used and intuitive metric, however in certain cases it can fall to capture the shape of a curve well.

**Frechet Distance:** This metric is best described via analogy. Imagine an owner is walking along one curve and their dog is walking along the other. They each have control of their speed but they cannot go backwards. The Frechet distance is the minimum length of the leash between the dog and owner [7, 8]. This metric takes into account location and flow, which is why some champion it over the Hausdorff distance [9].

Minimizing each of these metrics corresponds to minimizing the distance between our $D$ and $P$ according to that metric. Each metric has their own set of advantages and disadvantages, so we will be testing the performance of each on a number of dummy problems such as drawing letters. We can use each metric by inputing it as a cost to TrajOpt, a trajectory optimization package [10], that produce a feasible plan with minimum cost.

**References**


