Learning Geometric Reductions for Planning and Control

Nathan Ratliff RATLIFF.NATHAN@GMAIL.COM
Google Pittsburgh, 6425 Penn Ave # 700, Pittsburgh, PA 15206

Abstract

Obstacles in an environment inherently affect the its geometry. A straight line between two points is no longer optimal and may not even be feasible. This paper explores that idea directly and develops algorithms to learn and efficiently leverage the underlying geometry of the environment for planning and control. Specifically, we uncover a problem's geometry by formulating the problem as one of finding a diffeomorphism (smooth and invertible mapping) from the configuration space to a latent Euclidean space where curved geodesics in the configuration space map to well understood and computationally efficient straight lines in the latent space. Diffeomorphisms in this context act to reduce more complex problems to simpler more fundamental and easily controlled ones. We provided some preliminary experimental results using these techniques and discuss their advantages, limitations, and generalizations.

1. Introduction

Obstacles inherently affect an environment's geometry. At one extreme, obstacles explicitly tear holes the configuration space (LaValle, 2006) by invalidating entire sets of configurations. At the very least, they warp the geometry by redefining geodesics. No longer is a straight line the shortest distance between two points; trajectories must bend around obstacles to avoid collision.

This paper capitalizes on these ideas to explore techniques for integrating learned representations of the underlying environment's geometry, specifically how that geometry is warped by obstacles, into simple greedy controllers. We use ideas from Riemannian geometry to reduce superficially complex problems to their fundamental simplicity by learning smooth invertible maps that link the curved geometry in the configuration space to the well-understood geometry of a latent Euclidean space where geodesics become computationally efficient straight lines.

We study two algorithm that leverage these ideas: *Latent Geodesic Euclideanization* and *Latent Euclidean Value Iteration*. The former explicitly learns a mapping to a latent space where curved geodesics become straight, while the latter exploits the computational efficiency of Euclidean distances to run a value iteration type algorithm that learns a compact representation of all value functions (to any goal) of an environment simultaneously. These value functions are encoded in the Riemannian structure the latent Euclidean mapping defines across the configuration space. In both cases, the algorithms uncover a mapping to a latent Euclidean space that simplifies the problem significantly.

2. Diffeomorphisms and Riemannian Geometry

This section reviews some basic ideas from Riemannian Geometry, the generalized study of curves and surfaces in n-dimensional space. There are many ways to characterize how a space curves, but for our purposes we will focus on the geometry that arises by identifying the space in question with another, in this case specifically Euclidean, space using a smooth nonlinear map.

Let $\phi : \mathcal{C} \rightarrow \mathbb{R}^d$ be a smooth bijective mapping from a d-dimensional configuration space $\mathcal{C}$ to a d dimensional Euclidean space $\mathbb{R}^d$. ϕ may be viewed as inducing a non-Euclidean geometry on $\mathcal{C}$ that mimics the well understood Euclidean geometry in $\mathbb{R}^d$ where distances and inner products are computationally simple.

Specifically, under this non-Euclidean geometry, the distance between any two points $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{C}$ is defined as their distance in the mapped space $\|\phi(\mathbf{q}_1) - \phi(\mathbf{q}_2)\|$ and, if we denote a trajectory through $\mathcal{C}$ as $\mathbf{q} : [0, 1] \rightarrow \mathcal{C}$, differentiating the mapped trajectory

ICML 2013 Workshop on Robot Learning — 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. Copyright 2013 by the author(s).

$\mathbf{x}(t) = \phi(\mathbf{q}(t))$ shows that velocity vectors $\dot{\mathbf{q}}$ in $\mathcal{C}$ are linked to velocity vectors $\dot{\mathbf{x}}$ in $\mathbb{R}^n$ through $\dot{\mathbf{x}} = \mathbf{J}_\phi(\mathbf{q})\dot{\mathbf{q}}$, where $\mathbf{J}_\phi(\mathbf{q})$ is the mapping's Jacobian at $\mathbf{q}$. In particular, since the inner product between two velocity vectors $\dot{\mathbf{x}}_1, \dot{\mathbf{x}}_2$ at a point $\mathbf{x} = \phi(\mathbf{q})$ is just $\dot{\mathbf{x}}_1^{\mathrm{T}}\dot{\mathbf{x}}_2$, we see that this mapping induces a specific notion of inner product on the tangent space[1] $\mathcal{C}$ defined by the Jacobian:

$$\dot{\mathbf{x}}_1^{\mathrm{T}}\dot{\mathbf{x}}_2 = \dot{\mathbf{q}}_1^{\mathrm{T}}\left(\mathbf{J}_\phi^{\mathrm{T}}(\mathbf{q})\mathbf{J}_\phi(\mathbf{q})\right)\dot{\mathbf{q}}_2 = \dot{\mathbf{q}}_1^{\mathrm{T}}\mathbf{A}\dot{\mathbf{q}}_2,$$

where $\mathbf{A} = \mathbf{J}_\phi^{\mathrm{T}}\mathbf{J}_\phi$. (Moving forward, as we do here, explicit dependences on $\mathbf{q}$ are typically hidden to simplify the notation.)

This matrix $\mathbf{A}(\mathbf{q})$, which changes smoothly from point to point, is known as the Riemannian metric of the manifold $\mathcal{C}$ (Lee, 1997). In this case, since it is an implied metric defined by the mapping ϕ, it is known as the *pullback* metric. $\mathbf{A} = \mathbf{J}_\phi^{\mathrm{T}}\mathbf{J}_\phi$ is the metric induced on $\mathcal{C}$ by *pulling* the Euclidean metric back from $\mathbb{R}^n$ onto $\mathcal{C}$ through ϕ. In this work, we often refer to the range space $\mathbb{R}^n$ as a *latent Euclidean* space and to the mapping ϕ itself as a *latent Euclideanization* of the space $\mathcal{C}$.

Note also that under such a smooth bijective map ϕ, which in the context of smooth manifolds is often referred to as a *diffeomorphism* from $\mathcal{C}$ to $\mathbb{R}^d$, geodesics (i.e. shorted paths) are mapped to geodesics. This means that although a straight line throught $\mathbb{R}^d$ may be mapped to a severely curved trajectory through $\mathcal{C}$ under ϕ^{-1}, that curve is guaranteed to be the shortest path through $\mathcal{C}$ between the end points under the induced geometry since Euclidean geodesics are straight lines. On the flip side, if ϕ is constructed so that geodesics in $\mathcal{C}$ obey obstacle constraints in the environment, then computing smooth collision free trajectories is as simple as mapping a start and goal point through ϕ into $\mathbb{R}^d$, finding the straight line trajectory between those two points, and mapping it back.

3. Control in a diffeomorphic space

Suppose we have such a diffeomorphism (smooth invertible map) $\phi : \mathcal{C} \to \mathbb{R}^d$. One simple way to construct a planning algorithm that leverages the geometry of the latent Euclidean space, as we alluded above, is, given start and goal points $\mathbf{q}_s$ and $\mathbf{q}_g$, map those points into the latent space $\mathbf{x}_s = \phi(\mathbf{q}_s)$ and $\mathbf{x}_g = \phi(\mathbf{q}_g)$, find the straight line geodesic $\mathbf{x} : [0,1] \to \mathbb{R}^n$ through the mapped Euclidean space defined as $\mathbf{x}(\alpha) = (1-\alpha)\mathbf{x}_s + \alpha\mathbf{x}_g$ for $\alpha \in [0,1]$, and explicitly map that geodesic back into the configuration space using the inverse mapping $\phi^{-1}(\mathbf{x}) : [0,1] \to \mathcal{C}$ to form $\mathbf{q}(\alpha) = \phi^{-1}((1-\alpha)\mathbf{x}_s + \alpha\mathbf{x}_g)$.

That's certainly the most straightforward way to generate such a geodesic trajectory through $\mathcal{C}$, but it requires an explicit representation of the inverse function ϕ^{-1}. Unfortunately, most learning algorithms do not simultaneously construct the inverse mapping. One would need to approximate the inverse map separately in order to run this procedure, and errors in that approximation can lead to artifacts in the final plan.

Alternatively, since many control algorithms, especially simple gradient-based method, can be adapted to linearized non-Euclidean geometries as represented by the Riemannian metric, one can leverage the induced geometry while avoiding the explicit use of an inverse map by pulling the latent Euclidean metric back into the configuration space through ϕ as described in Section 2. To illustrate these properties, this paper explores a simple gradient based control method that builds an attractor in the latent space. The attractor's objective is

$$p(\mathbf{q}) = \|\phi(\mathbf{q}_{\mathrm{g}}) - \phi(\mathbf{q})\|, \tag{1}$$

and we can minimize it by taking steps in the direction of the negative gradient evaluated with respect to the pullback metric $\mathbf{A}(\mathbf{q}) = \mathbf{J}_\phi{}^{\mathrm{T}}\mathbf{J}_\phi$:

$$\begin{aligned}\nabla_{\mathbf{A}} p(\mathbf{q}) &= -\mathbf{A}^{-1}\mathbf{J}_\phi \frac{\phi(\mathbf{q}_{\mathrm{g}}) - \phi(\mathbf{q})}{\|\phi(\mathbf{q}_{\mathrm{g}}) - \phi(\mathbf{q})\|} \\ &= -(\mathbf{J}_\phi^{\mathrm{T}}\mathbf{J}_\phi)^{-1}\mathbf{J}_\phi^{\mathrm{T}}\widehat{\Delta\mathbf{x}},\end{aligned}$$

where $\widehat{\Delta\mathbf{x}} = (\mathbf{x}_{\mathrm{g}} - \mathbf{x})/\|\mathbf{x}_{\mathrm{g}} - \mathbf{x}\|$ is the normalized direction vector pointing from the mapped current point $\mathbf{x} = \phi(\mathbf{q})$ to the mapped goal point $\mathbf{x}_{\mathrm{g}} = \phi(\mathbf{q}_{\mathrm{g}})$.

As we will see below, this algorithm provably mimics the behavior of running the controller directly in the latent Euclidean space, without requiring an explicit representation of the inverse map ϕ^{-1}. The only approximation employed is the linearization of each step. Essentially, this algorithm may be viewed as a first-order Euler method for finding integral curves of the differential equation implied in the configuration space by the goal potential's gradient in the latent space. Higher order numerical integration techniques such as the Runge Kutta method can offer tighter approximations if the above first-order steps prove insufficient.

Note that, although we presented it generally above, the matrix $\mathbf{J}_\phi^{\dagger} = (\mathbf{J}_\phi^{\mathrm{T}}\mathbf{J}_\phi)^{-1}\mathbf{J}_\phi^{\mathrm{T}}$, which is the well studied pseudoinverse of $\mathbf{J}_\phi$, reduces to $\mathbf{J}_\phi{}^{-1}$ in our case since ϕ, itself, is invertible.

[1]The *tangent space* is the space of velocity vectors or tangents to curves at a point.

3.1. Theoretical results and limitations

One nice property of diffeomorphisms is that they do not affect the number, location, or classification of critical points of an objective function. Specifically, if $\phi : \mathcal{C} \rightarrow \mathbb{R}^d$ is a diffeomorphism, and $h = f \circ \phi$ is a function defined on $\mathcal{C}$ by composition with a function f defined in $\mathbb{R}^d$, then any local optimum x^* of f maps to a local optimum of h through ϕ^{-1} and vice versa. Moreover, the second order optimality conditions of the functions at those points also match (Lee, 1997; Nocedal & Wright, 2006). In the case of the greedy control algorithm defined above, since the goal potential is convex in the latent Euclidean space, the gradient based procedure remains globally convergent when pulled back into the configuration space despite any seemingly extreme contortions introduced by the diffeomorphic map.

Latent Euclideanizations, however, although convenient when applicable, are fundamentally limited in scope. Not all manifolds can be isometrically embedded into a same dimensional space. To be fully general, the Nash Embedding theorem (Lee, 1997) implies that you need at least one extra dimension.

Intuitively, the Eigenspace of the Riemannian metric dictates how the manifold is stretched or contracted at any given point. Representing that contortion explicitly as a diffeomorphism into a same-dimensional space may not be possible because the mapping would have to adhere to a natural "fabric conservation" property—stretching the space in one location must contract the space in another. If the metric does not have such a global conservation property, the "extra fabric" must pop out into an extra dimension, such as into the third dimension when dealing with a two-dimensional surface.

This paper addresses only the more limited case of same-dimensional mappings, but an active direction for future work includes generalizing these ideas to mappings into one extra dimension so that the manifold becomes an embedded hypersurface. Interestingly, the same greedy control algorithm derived in Section 3 can be used to move along the such a hypersurface allowing us to study the behavior of control across canonical manifolds of this sort along with methods of learning diffeomorphisms that reduce more complicated problems to these simpler scenarios.

4. Learning a diffeomorphism

This section addresses how we can efficiently represent diffeomorphisms and how we can used those ideas to develop training algorithms to learn diffeomorphisms from demonstrated behavior.

4.1. Diffeomorphism representations

Generally, not all smooth mappings from one manifold onto to another are diffeomorphisms. In particular, they can fail to be one-to-one, and hence fail to be invertible. So when faced with learning a diffeomorphism, the obvious question is how do we define a hypothesis class that guarantees a trained mapping is diffeomorphic.

The approach we take here is analyzed in (Dupuis & Grenander, 1998) and has been used previously in the context of dimensionality reduction (Walder & Schölkopf, 2008). Intuitively, this approach suggests that mappings built from vector flows that transport points in the space from one region to another over time are provably guaranteed to be diffeomorphisms.

Explicitly, let $v(\mathbf{x}, t)$ be a time varying vector field that for a fixed time t defines a field of small perturbations to $\mathbf{x}$ across the entire domain. Imagine starting at a point $\mathbf{x}_0$ at step $k = 0$ and moving a small dt along the vector $v(\mathbf{x}_0, 0)$. This gives us $\mathbf{x}_1 = \mathbf{x}_0 + dt\ v(\mathbf{x}_0, 0)$. From there imagine moving a step further along the vector $dt\ v(\mathbf{x}_1, dt)$ defined at the new point $\mathbf{x}_1$ at the current time dt, and so on, moving at each step k along the time-scaled vector $dt\ v(\mathbf{x}_k, k\ dt)$. This procedure maps out Euler steps of the vector flow induced by this field; we can write the continuous version of this process recursively as

$$\phi(\mathbf{x}, t) = \mathbf{x} + \int_0^t v(\phi(\mathbf{x}, s), s) ds. \qquad (2)$$

The induced diffeomorphism is defined as the mapping that takes a point to its final flow destination. The diffeomorphism is, therefore, given explicitly by $\phi(\mathbf{x}) = \phi(\mathbf{x}, 1)$.

Clearly, implementing the integral as a function parameterization is hard, but one may discretize the process by using a sequence of identity centered transformation functions of the form $\phi_t(\mathbf{x}) = \mathbf{x} + u_t(\mathbf{x})$ and constructing each subsequent stage of the transform as $\psi_{t+1}(\mathbf{x}) = \phi_t(\psi_t(\mathbf{x}))$, viewing each subsequent $u_t(\mathbf{x})$ as a small perturbation to the space. This construction essentially suggests forming the diffeomorphism as a sequence of small consecutive perturbations. Since this parameterization is a discrete approximation, it is not fully guaranteed to represent a diffeomorphism, but it works well in practice as we will see below.

Algorithm 1 Train Diffeomorphism

```
**Input:** initial demonstrations $\Xi_0 = \{\xi_i^0\}_{i=1}^N$, morphic algorithm $\mathcal{A}$, training algorithm $\mathcal{T}$, initial mapping $\psi_0$ (typically the identity).
**for** $t = 0$ **to** $T - 1$ **do**
    Morph $\mathcal{D}_t = \{(\xi_i^t, \mathcal{A}(\xi_i^t))\}_{i=1}^N$.
    Train $\phi_{t+1}^* = \mathcal{T}(\mathcal{D}_t)$
    Update map $\psi_{t+1} = \phi_{t+1}^* \circ \psi_t$
    Update demonstrations
        $\Xi_{t+1} = \{\phi_{t+1}^*(\xi_i^t)\}_{i=1}^N = \{\psi_{t+1}(\xi_i^0)\}_{i=1}^N$
**end for**
```

4.2. Latent Geodesic Euclideanization

Suppose there is an algorithm $\mathcal{A}$ that takes as input a collection of points $\xi \subset \mathcal{C}$ and morphs them in some way so as to iteratively optimize an energy function. An example of this that we will explore momentarily is a trajectory straightening algorithm that takes a curved trajectory and produces a straighter one. We assume that the algorithm is well behaved in the sense that the steps are of incrementally decreasing energy, and that repeated application of the algorithm will converge toward a minimum energy solution over time. In what follows, we denote the application of the algorithm as $\xi_{\text{new}} = \mathcal{A}(\xi)$.

Suppose also that we have a nonlinear function approximation algorithm $\mathcal{T}$ that takes a data set $\mathcal{D} = \{(\mathbf{q}_i, \mathbf{x}_i)\}$ and returns the best fit (e.g. in the least squares sense) function approximator ϕ^* from a set of candidate hypotheses $\mathcal{H}$. We denote the action of this function approximation algorithm as $\phi^* = \mathcal{T}(\mathcal{D})$.

Algorithm 1 gives a training algorithm that learns a diffeomorphism guided by the behavior of the morphic algorithm $\mathcal{A}$. In words, this algorithm starts from a set of initial demonstrations $\Xi = \{\xi_i\}_{i=1}^N$ and the identity mapping $\psi(\mathbf{q}) = \mathbf{q}$. At each iteration it observes how the morphic algorithm $\mathcal{A}$ perturbes the points of each input ξ_i^t thereby forming a data set of all perturbation pairs across the demonstrations $\mathcal{D} = \bigcup_i \{(\mathbf{x}, \mathbf{x}') \mid \mathbf{x} \in \xi_i^t, \mathbf{x}' \in \mathcal{A}(\xi_i^t)\}$. It then updates the mapping as $\psi_{t+1} = \phi_t^* \circ \psi_t$ where $\phi_t^* = \mathcal{T}(\mathcal{D})$. Each successive mapping approximator operates recursively as $\psi_{t+1}(\mathbf{q}) = \phi_t^*(\psi_t(\mathbf{q}))$ further perturbing the output of the previous composition of perturbation mappings. After each iteration, the demonstration set is updated to be the morphed version of the set as in $\xi_i^{t+1} = \psi_{t+1}(\xi_i) = \phi_t^*(\xi_i^t)$.

Choosing an identity centered representation for each perturbation function $\phi_t(\mathbf{x}) = \mathbf{x} + u_t(\mathbf{x})$ as suggested in Section 4.1 approximates a class of diffeomorphisms as can be seen by recursively expanding the expression for ψ_t in terms of ϕ_t.

Moreover, choosing the morphic algorithm $\mathcal{A}$ to be a trajectory straightening method will coax the algorithm to learn a diffeomorphism that takes curved geodesic demonstrations to straight lines thereby finding a representation of a latent Euclideanization. We call the variant of the above training algorithm that uses a trajectory straightening morphic algorithm Latent Geodesic Euclideanization. The experiments detailed in Section 5.1 use this variant, specifically with a straightening algorithm modeled off the CHOMP trajectory optimization algorithm (Ratliff et al., 2009) designed to iteratively minimize the global acceleration across the trajectory while simultaneously retaining average interpoint distances (so that the learned mapping implicitly encodes velocity profiles as well).

4.3. Latent Euclidean Value Iteration

If $\phi : \mathcal{C} \to \mathbb{R}^d$ is a diffeomorphism to a Euclidean space, then since geodesic distances become Euclidean distances which are trivial to compute. For every goal point $\mathbf{q}_{\text{g}}$, the mapping ϕ therefore induces a very simple but expressive value function on $\mathcal{C}$ given by $V(\mathbf{q}) = \|\phi(\mathbf{q}) - \phi(\mathbf{q}_{\text{g}})\|$.

This connection to value functions suggests that we can explore the use of traditional value function learning techniques such as value iteration (Sutton & Barto, 1998) to learn the best fit diffeomorphism for a given cost function in the environment. Additionally, since the diffeomorphism encodes value functions efficiently as Euclidean distances in a latent space, we can extend the ideas of value iteration to learn all value functions for the environment simultaneously by encoding them into the mapping ϕ.

Algorithm 2 presents an algorithm we call Latent Euclidean Value Iteration that performs a generalized form of value iteration with function approximation for this setting. In this algorithm, $\mathcal{A}$ denotes the discrete set of actions that we can take from any state in $\mathcal{C}$, and we denote the environment's cost function as $c : \mathcal{C} \times \mathcal{A} \to \mathbb{R}$. We also assume we have access to a discrete training distribution of states $\mathcal{D} = \{(\mathbf{q}_i)\}_{i=1}^N$ where we want the value function to be accurate and a distribution of goals $\mathcal{G}$ that we will generalize from.

The basic idea behind the algorithm is straightforward. At each iteration, we perform a backup at each point $\mathbf{q}_i \in \mathcal{D}$ to get an incrementally more accuracte estimate of the cost-to-go value to each goal $\mathbf{q}_{\text{g}} \in \mathcal{G}$. Given that information, we learn a new stage ϕ_{t+1} to perturb the space in the right direction. We do so by creating a least squares data set designed to perturb the

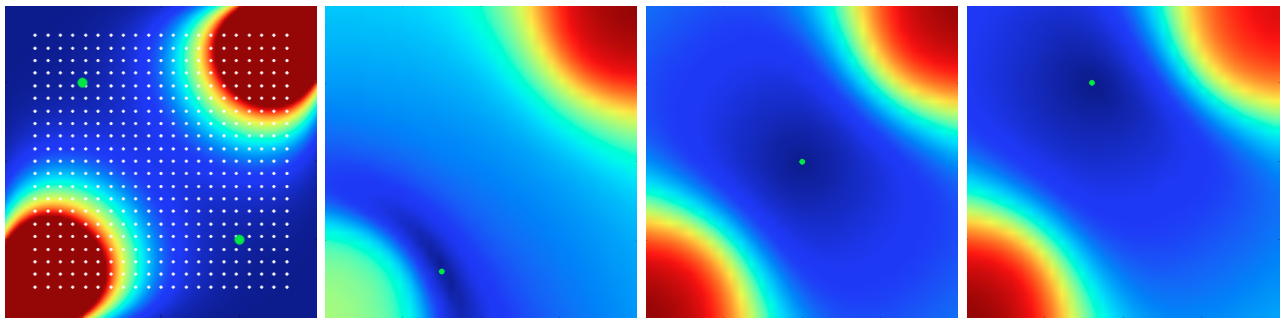

Figure 1. From left to right we have the environment's cost map along with the configuration set (grey points) and the goal set (green points), and three learned value functions for the region.

Algorithm 2 Latent Euclidean Value Iteration

```
**Input:** Configuration and action spaces $\mathcal{C}$ and $\mathcal{A}$, cost function $c : \mathcal{C} \times \mathcal{A} \Rightarrow \mathbb{R}$, sampled state $\mathcal{D} = \{(\mathbf{q}_i)\}_{i=1}^N$ and goal $\mathcal{G}$ distributions.
**Initialize:** All values $\{v_i^0(\mathbf{q}_\mathrm{g})\}$ to zero and $\psi_0 : \mathcal{C} \to \mathbb{R}^d$ to the identity.
for $t = 0$ to $T - 1$ do
    for each $\mathbf{q}_i, \mathbf{q}_\mathrm{g}$ pair in $\mathcal{D} \times \mathcal{G}$ do
        $v_i^{t+1}(\mathbf{q}_\mathrm{g}) \leftarrow \min_{\mathbf{a} \in \mathcal{A}} c(\mathbf{q}_i, \mathbf{a}) + \|\psi_t(\mathbf{q}_i + \mathbf{a}) - \psi_t(\mathbf{q}_\mathrm{g})\|$
    end for
    Train $\phi_{t+1}$ on $\{(\psi_t(\mathbf{q}_i),\ \psi_t(\mathbf{q}_i) + \eta \Delta^g_{\mathbf{q}_i}\psi_t\}_{\mathcal{D}\times\mathcal{G}}$ and $\{(\psi_t(\mathbf{q}_\mathrm{g}),\ \psi_t(\mathbf{q}_\mathrm{g}) - \eta \Delta^g_{\mathbf{q}_i}\psi_t\}_{\mathcal{D}\times\mathcal{G}}$
    Compose $\psi_{t+1} = \phi_{t+1} \circ \psi_t$
end for
```

transformed points $\psi_t(\mathbf{q}_i)$ and $\psi_t(\mathbf{q}_\mathrm{g})$ away from each other proportionally to the discrepancy between the previously predicted value and newly updated value. In Algorithm 2, the perturbation direction is denoted $\Delta^g_{\mathbf{q}_i}\psi_t = \mathbf{norm}(\psi_t(\mathbf{q}_i) - \psi_t(\mathbf{q}_\mathrm{g}))$ (the normalized direction vector from the transformed goal to the transformed state). Once ϕ_{t+1} is trained, the updated value function in full becomes $\psi_{t+1} = \phi_t \circ \psi_t$.

5. Experiments

This section presents some preliminary experiments exploring these ideas and algorithms. In all of these experiments, we used kernelized least squares kernel regression with radial basis kernels as our mapping approximator (Schölkopf & Smola., 2002).

5.1. Latent Geodesic Euclideanization

Section 4.2 proposes an algorithm that explicitly takes a collection of curved geodesics in a d-dimensional space and learns a diffeomorphism to $\mathbb{R}^d$ under which those geodesics become straight lines. Here we demonstrate the algorithm using a simple 2D point mass motion problem.

Figure 2 on the left shows some learned trajectories found using the greedy control algorithm of Section 3 (colored) over the top of the (grey) training geodesics optimized using the CHOMP motion planning algorithm (Ratliff et al., 2009) optimized to avoid a higher cost region from the right (not shown here). The middle subplot of that figure demonstrates a learned obstacle avoidance behavior on a planar 3 DOF manipulator trained using the Latent Geodesic Euclideanization algorithm on a single trajectory demonstration that avoided the obstacle in question. We used the learned latent Euclidean space to define a Riemannian metric in the workspace and pulled it back through the arm's kinematic map to induce an associated metric for our controller in the configuration space. We then leverage that metric by running a covariant gradient control algorithm that greedily pulls the end-effector toward a goal.

We also used the latent Euclideanization algorithm to learn a latent space that encoded minimum energy trajectories. Figure 3 shows some generalized trajectories predicted after learning from a single minimum energy demonstration. Statistics taken over 20 sampled trajectories, show a 24% decrease in energy on average over straight line trajectories when greedily planning under the learned latent Euclidean space. That's a decrease from an average energy of 0.50 for trajectories planned as straight lines through the configuration space to an average energy of 0.38 for trajectories planned as straight lines through the learned latent Euclidean space. None of the predicted trajectories was higher energy than their configuration space counterpart.

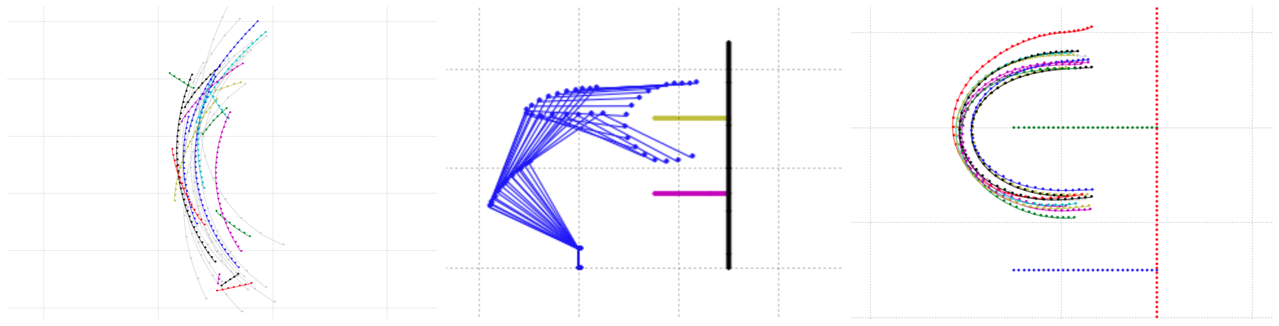

Figure 2. The leftmost plot shows a basic demonstration of Latent Geodesic Euclideanization learning from optimized geodesics. The remaining two subplots show the generalization performance of a learned workspace metric and the planning behavior of a greedy arm controller performing under the corresponding pullback. See Section 5.1 for details.

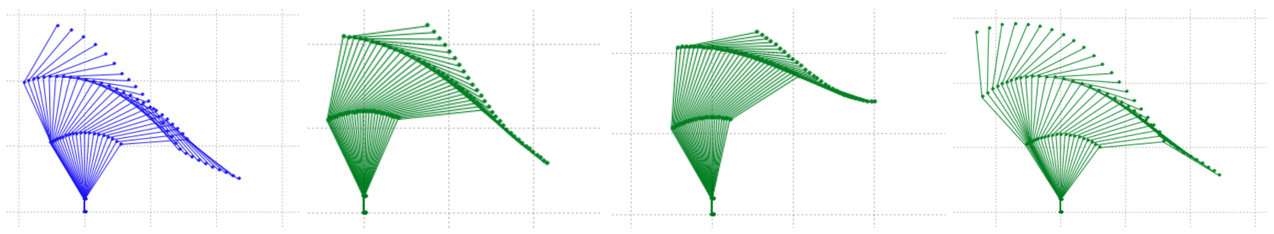

Figure 3. The first trajectory from the left is the minimum energy trajectory demonstration. The three remaining trajectories are predicted nearby trajectories between waypoint generated as geodesics under the learned Riemannian metric.

5.2. Latent Euclidean Value Iteration

Here we demonstrate the efficient value function representational power of diffeomorphisms to a Euclidean space by running the Latent Euclidean Value Iteration algorithm on a planar planning problem. Figure 1 gives the environment's cost map with the start and goal distributions used for training, as well as a three generalized value functions for the environment calculated using straight-line Euclidean distances through the learned latent Euclidean space.

6. Conclusions

Uncovering the underlying geometry of a planning or control problem by learning a diffeomorphism to a geometrically simple space can reduce seemingly complex problems to much simpler ones. At one extreme, using latent Euclideanization techniques, we can reduce the problem to a Euclidean geometry space where straight line trajectories are sufficient for obstacle avoidance and behavior generation.

This paper presents some preliminary results moving in that direction, but the algorithms presented here are only proofs of concept demonstrating some simple ways in which we might leverage these ideas. More substantial experiments in practical settings, as well as explorations into stronger function approximators such as deep networks that can aid generalization to new problems by correlating environment features with the learned diffeomorphisms, will further substantiate these ideas.

Additionally, one active area of future work, motivated in Section 3.1, is to expand beyond the relatively restrictive diffeomorphisms from d-dimensional configuration spaces to d-dimensional latent spaces to mappings that isometrically embed manifolds into higher dimensional spaces for increased expressivity. We have been exploring the diffeomorphic reduction of problems to hypersurface representations that explicity model obstacles as tears in the space. Progress in this direction will greatly expand the generality of this approach.

References

Dupuis, P. and Grenander, U. Variational problems on flows of diffeomorphisms for image matching. *Quartly of Applied Mathematics, LVI*, pp. 587–600, 1998.

LaValle, S. M. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

Lee, John M. *Riemannian Manifolds: An Introduction To Curvature*. Springer, 1997.

Nocedal, Jorge and Wright, Stephen. *Numerical Optimization*. Springer, 2006.

Ratliff, Nathan, Zucker, Matthew, Bagnell, J. Andrew (Drew), and Srinivasa, Siddhartha. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

Schölkopf, Bernhard and Smola., Alex J. *Learning with Kernels*. MIT Press, 2002.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.

Walder, Christian and Schölkopf, Bernhard. Diffeomorphic dimensionality reduction. In D Koller, D Schuurmans, Y Bengio and Bottou, L (eds.), *Proceedings of Neural Information Processing Systems 22*, December 2008.