Understanding the Geometry of Workspace Obstacles in Motion Optimization

Nathan Ratliff[1,2], Marc Toussaint[2], and Stefan Schaal[1]

***Abstract*— What is it that makes movement around obstacles hard? The answer seems clear: obstacles contort the geometry of the workspace and make it difficult to leverage what we consider easy and intuitive straight-line Cartesian geometry. But is Cartesian motion actually easy? It's certainly well-understood and has numerous applications. But beneath the details of linear algebra and pseudoinverses, lies a non-trivial Riemannian metric driving the solution. Cartesian motion is easy only because the pseudoinverse, our powerhouse tool, correctly represents how Euclidean workspace geometry pulls back into the configuration space. In light of that observation, it reasons that motion through a field of obstacles could be just as easy as long as we correctly account for how those obstacles warp the geometry of the space. This paper explores extending our geometric model of the robot beyond the notion of a Cartesian workspace space to fully model and leverage how geometry changes in the presence of obstacles. Intuitively, impenetrable obstacles form topological holes and geodesics curve around them accordingly. We formalize this intuition and develop a general motion optimization framework called Riemannian Motion Optimization (RieMO) to efficiently find motions using our geometric models. Our experiments demonstrate that, for many problems, obstacle avoidance can be much more natural when placed within the right geometric context.**

I. Introduction

Cartesian motion for robot manipulation [22] is relatively well understood. The pseudoinverse solves an equality constrained quadratic program that calculates the configuration space motion that best makes the end-effector move in a particular motion. But obstacles complicate the mathematics. Obstacles introduce unstructured nonlinear and nonconvex constraints that don't fit well into either weighted pseudoinverse or more general Quadratic Programming (QP) formulations. Recent motion optimization strategies such as TrajOpt [20], CHOMP [18], STOMP [6], and iTOMP [3] have shown impressive performance modeling obstacles as costs and constraints over long time-horizons, but many types of obstacles, especially long thin obstacles or small poles, translate to ill-conditioned problems that are difficult to optimize or have many poor local minima. These problems are thought to be fundamentally hard requiring global optimization or exploration strategies such as Rapidly-exploring Randomized Trees (RRTs) or Probabilistic Road Maps (PRMs) [7].

This paper takes a step back and asks whether this observed ill-conditioning is fundamental to the underlying complexity of the workspace or simply an artifact of modeled geometric representation. Indeed, the mapping from the configuration space to the three-dimensional workspace is highly-nonlinear, yet pseudoinverse solutions invert that nonlinearity and make controlling linear end-effector motion easy.

Intuitively, we understand pretty well that the geometry of the workspace is strikingly non-Euclidean. Obstacles are "invalid" sections of the space, best viewed as non-existent. And straight lines are no longer Euclidean. It doesn't make sense for a straight line to pass through an invalid region; it must naturally curve around an obstacle. In the parlance of Riemannian Geometry, we say that obstacles are topological holes in the space and that geodesics curve to avoid those holes. This workspace geometry, too, is highly non-Euclidean. The nonlinearities of robotic motion don't end at the end-effector. They continue into the workspace.

This paper explores extending the geometric model of the robot beyond the simple Cartesian space of the end-effector to fully represent how the workspace geometry affects the overall geometry of the motion problem. We formalize the representation of workspace geometry as a Riemannian metric in the three-dimensional workspace surrounding the robot, and demonstrate an equivalence between Pullback metrics in Riemannian Geometry, and a form of partial Gauss-Newton approximation for representing objective curvature in Nonlinear Programming. This connection enables the implementation of geometrically aware motion optimizers that optimize a very general class of motion models efficiently using generic second-order constrained optimization solvers.

Our novel framework is called Riemannian Motion Optimization (RieMO). We introduce two related models to represent the non-Euclidean geometry of the workspace, show how to leverage them within motion optimization, and demonstrate experimentally that they can easily induce highly contorted non-linear workspace motion to avoid thin obstacles while retaining fast optimization speeds of around .5 seconds.[1] These optimization speeds are competitive with the state-of-the-art, but the purpose of this work is not to claim the fastest optimizer. This work aims to further understand the separation between motion problems that are fundamentally difficult and those that are relatively easy. Our tools generalize the mathematics of pseudoinverses, using ideas from Riemannian Geometry to represent and leverage the intuitively non-Euclidean geometry of the workspace within long-horizon motion optimization.

Section IV-B discusses some analysis proved in the extended version of this paper [17] that show a very close connection between Gauss-Newton approximations, which fully capture the first-order Riemannian geometry of this

[1]Autonomous Motion Department, Max Planck Institute for Intelligent Systems, Tübingen, Germany

[2]Machine Learning and Robotics Lab, University of Stuttgart, Germany

[1]These speed quotes are for unoptimized code running on a Linux virtual machine atop a 2012 Mac PowerBook.

problem, and the exact Hessian for terms modeling geodesic energy through non-Euclidean spaces. This result, along with the computational analysis of the general Motion Optimization framework discussed in Section III, demonstrates that Newton's method is efficient on non-Euclidean Motion Optimization problems, and the experimental demonstrations of Section V show that the optimized motions are simultaneously expressive and precise.

II. A QUICK REVIEW OF RELEVANT CONCEPTS FROM RIEMANNIAN GEOMETRY

This section reviews some of the tools from first-order Riemannian geometry we use in this paper. Our interest is in mathematical calculation and intuition, so we aren't concerned with rigorous details such as basis-free representations or definitions of smoothness and differentiability on manifolds. We present just the basic elements needed to understand this paper—for an excellent in-depth introduction to Smooth Manifolds and Riemannian Geometry, see [8].

A. Metrics, Geodesics, and Pullbacks

Core to the idea of Riemannian geometry, is a smooth map that maps each point $\boldsymbol{x} \in \mathcal{M}$ in the domain (where $\mathcal{M}$ is an n-dimensional smooth manifold) to a symmetric positive definite[2] matrix $\boldsymbol{A}(\boldsymbol{x})$. This map is known as the Riemannian metric.

Importantly, this metric at a given point $\boldsymbol{x}$ operates on the tangent space, the space of velocity vectors $\dot{\boldsymbol{x}} \in \mathcal{T}_{\mathcal{M}}$ at that point, to define a norm on incremental motions $\|\dot{\boldsymbol{x}}\|_{\boldsymbol{A}(\boldsymbol{x})} = \sqrt{\dot{\boldsymbol{x}}^T \boldsymbol{A}(\boldsymbol{x}) \dot{\boldsymbol{x}}}$. (Intuitively, $\boldsymbol{A}(\boldsymbol{x})$ stretches the space along its Eigenspectrum by amounts given by the square roots of its Eigenvalues. This norm more heavily penalizes directions along which the space is significantly stretched.) Such a norm allows us to define the length of a parameterized path $\boldsymbol{x} : \mathbb{R} \to \mathcal{M}$ as[3] $L(\boldsymbol{x}) = \int \|\dot{\boldsymbol{x}}\|_{\boldsymbol{A}(\boldsymbol{x})}\, dt$, which in turns allows us to define a geodesic as the minimum cost path between a pair of points

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \Xi(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\operatorname{argmin}} \frac{1}{2} \int \dot{\boldsymbol{x}}^T \boldsymbol{A}(\boldsymbol{x}) \dot{\boldsymbol{x}}\, dt, \tag{1}$$

where $\Xi(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is the set of all paths starting at $\boldsymbol{x}_1$ and ending at $\boldsymbol{x}_2$. Here we've replaced an explicit optimization over the length functional $L(\boldsymbol{x})$ with an equivalent optimization over a geodesic energy functional more amenable to motion optimization—the minimizers of the two functionals match.

We'll also make use of the concept of Pullback functions and Pullback metrics. Concretely, suppose $\phi : \mathcal{M} \to \mathcal{N}$ is a smooth mapping from one manifold $\mathcal{M}$ to another manifold $\mathcal{N}$. For any function $f : \mathcal{N} \to \mathbb{R}$ defined on that co-domain, there's an associated induced function defined on $\mathcal{M}$ called the Pullback[4] function h defined via composition as $h(\boldsymbol{x}) = f(\phi(\boldsymbol{x}))$. Simply put, the function acts by first mapping $\boldsymbol{x}$ from $\mathcal{M}$ to its associated point $\boldsymbol{z}$ in $\mathcal{N}$ using ϕ, and then evaluating the f there.

Similarly, if the manifold constituting the co-domain has a corresponding Riemannian metric $\boldsymbol{B}(\boldsymbol{z})$, then the map ϕ also induces an associated metric on $\mathcal{M}$ defined by $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{J}_\phi^T \boldsymbol{B}(\boldsymbol{z}) \boldsymbol{J}_\phi$, where $\boldsymbol{z} = \phi(\boldsymbol{x})$ and $\boldsymbol{J}_\phi$ is the Jacobian of ϕ evaluated at $\boldsymbol{x}$. This induced metric is called the Pullback metric; the formula is simply a generalization of the relation $\dot{\boldsymbol{z}} = \frac{d}{dt}\phi(\boldsymbol{x}) = \boldsymbol{J}_\phi \dot{\boldsymbol{x}}$, since $\|\dot{\boldsymbol{z}}\|_{\boldsymbol{B}}^2 = \dot{\boldsymbol{z}}^T \boldsymbol{B} \dot{\boldsymbol{z}} = \dot{\boldsymbol{x}}^T \left(\boldsymbol{J}_\phi^T \boldsymbol{B} \boldsymbol{J}_\phi \right) \dot{\boldsymbol{x}}$. These Pullbacks represent how first-order Riemannian geometry propagates between spaces. Of interest here, when the original metric on $\mathcal{N}$ is defined as a positive definite Hessian of a function $f : \mathcal{N} \to \mathbb{R}$, then this Pullback metric is equivalent to a partial Gauss-Newton Hessian approximation of the corresponding Pullback function (described in Section III-B).

B. The Geometry of Cartesian Control

Here we gain some intuition about the geometry of robot motion by showing how a Euclidean metric placed in the workspace *induces* a non-trivial non-Euclidean geometry on the underlying d-dimensional configuration space. The configuration space $\mathcal{C}$ and the workspace[5] $\mathbb{R}^3$ are two manifolds linked by the forward kinematics map $\phi : \mathcal{C} \to \mathbb{R}^3$ mapping a configuration to a point on the robot's end-effector. Let $\boldsymbol{J}_\phi$ be the Jacobian of ϕ. By itself, the Euclidean metric, represented by the identity matrix $\boldsymbol{I}$ in the workspace, pulls back to a metric of the form $\boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$. That metric is reduced rank for redundant manipulators ($d > 3$), and is therefore fundamentally non-Euclidean in the higher dimensional space. Even if we modify it to be $\boldsymbol{A}_\alpha(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi + \alpha \boldsymbol{I}$, the resulting metric is still non-Euclidean. (For a d-dimensional manifold $\mathcal{M}$ with metric $\boldsymbol{A}(\boldsymbol{q})$ to be fundamentally Euclidean, we need to be able to find a smooth mapping of the form $\psi : \mathcal{M} \to \mathbb{R}^d$ to a Euclidean space of the same dimension under which the Pullback is $\boldsymbol{A}(\boldsymbol{q})$ (specifically, $\boldsymbol{J}_\psi^T \boldsymbol{J}_\psi = \boldsymbol{A}(\boldsymbol{q})$). In this case, for $\boldsymbol{A}_\alpha(\boldsymbol{q})$, the most natural mapping is one that maps $\boldsymbol{q}$ to a *higher*-dimensional ($(d+3)$-dimensional) space $[\phi(\boldsymbol{q}); \alpha^{\frac{1}{2}} \boldsymbol{I}]$ (see Section IV-B for a more in depth discussion of such geometric mappings).)

In particular, following potential field gradients taken with respect to this metric produces motion consistent with the workspace metric. For instance, suppose we're following a simple attractor of the form $\|\boldsymbol{x}_g - \phi(\boldsymbol{q})\|$. The negative gradient of this attractor in the workspace is simply a unit vector pointing directly toward the goal. When we take the negative gradient in the configuration space with respect to the metric $\boldsymbol{A}_\alpha = \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$, we get (denoting the workspace negative gradient as $\boldsymbol{v}$)

$$\begin{aligned} -\nabla_{\boldsymbol{A}_\alpha} \|\boldsymbol{x}_g - \phi(\boldsymbol{q})\| &= (\boldsymbol{J}_\phi^T \boldsymbol{J}_\phi + \alpha \boldsymbol{I})^{-1} \boldsymbol{J}_\phi \boldsymbol{v} \\ &= \boldsymbol{J}_\phi^T (\boldsymbol{J}_\phi \boldsymbol{J}_\phi^T + \alpha \boldsymbol{I})^{-1} \boldsymbol{v}, \end{aligned}$$

which is the familiar (softened) pseudoinverse motion control rule that we know makes the robot's end-effector move (instantaneously) in the direction $\boldsymbol{v}$ (up to the softening factor

[2]There's a theoretical distinction between fully positive definite metrics and positive semi-definite metrics. Computationally, though, we can often utilize the latter in practice by simply replacing any inverse with a pseudoinverse. We'll see an example of that in the next section.

[3]With a slight abuse of notation, we often use the same variable, such as $\boldsymbol{x}$, to represent both a point in $\mathcal{M}$ and a trajectory through $\mathcal{M}$. That latter is understood in context by the presence of time indices or time derivatives.

[4]Often this function is denoted $h = \phi^* f$, but for our purposes, that notation is unnecessarily confusing.

[5]For simplicity here we take the workspace to be the space of all 3D point locations, but more generally it often includes orientations as well. In motion optimization, orientation constraints are often specified separately, so 3D point locations are sufficiently expressive.

approximation). The final line of this calculation can be verified by expanding it and the line above it in terms of the Jacobian's SVD.

Euclidean workspace metrics are already fundamentally non-Euclidean from the perspective of the configuration space. In that sense, here's nothing inherently special about Euclidean workspace metrics; Section IV leverages that intuition and derives some natural non-Euclidean metrics to better represent the inherent geometry of the workspace.

III. A General Motion Optimization Framework

Hessian information accounting for the kinematic structure of the trajectory has incrementally become an increasingly critical component of motions optimization [23], [18], [6], [3], [20]. The success of these methods have encouraged a shift away from global techniques toward more direct local optimization. These quick Hessian approximations leverage the graph structure of the motion problem to communicate gradient information across the full length of the trajectory at each iteration during approximate Newton step computations. We're finding a number of significant real-world problems for which local optimizers alone are able to find good paths starting from relatively naïve and generic initial hypotheses. Especially in human environments, where problems are intentionally simplified to be easily solvable by humans, who, themselves, often struggle with the same fundamental planning complexity limitations as robots, many everyday problems are quickly solvable by local optimization methods.

The motion optimization framework we use for this paper is largely inspired by the Kth-Order Markov Optimization (KOMO) framework [24]. The only conceptual difference is that our framework, which we call Riemannian Motion Optimization (RieMO), uses Gauss-Newton approximations only to avoid computing third-order tensors of differentiable maps when pulling Hessians back from the map's range space to its domain. All top-level cost functions (e.g. workspace cost-functions which are at the end-point of forward kinematics maps or similar workspace velocity norm terms) require explicit Hessian implementations under RieMO. These explicit Hessians correctly communicate the curvature of the workspace geometry, which the *partial* Gauss-Newton approximations then pull back to the configuration space.

Additionally, we don't assume the availability of full configuration goals (often supplied via inverse kinematics, an approach taken by optimizers such as CHOMP (although, see GSCHOMP [4]) and STOMP). All of our optimizations start simply from the zero-motion trajectory that doesn't move at all from its initial pose. We use the zero set of a goal constraint function (such as a function measuring the distance between the terminal configuration's end-effector and a Cartesian goal) to define the desired set of valid goal configurations implicitly as a modeled constraint.

Much attention in motion optimization has been given to speed. Which optimizer optimizes the motion model the fastest? Beyond the computational efficacy of individual components such as collision detection and sparse representations, fast convergence is largely a question of optimization: we know theoretically that increased utilization of second-order information translates to faster convergence [12]. Section III-A reviews a straightforward and very general methodology for accurate modeling and efficient optimization. This framework leverages fast band-diagonal linear system solvers to exploit problem structure during the calculation of Newton steps within a generic constrained Augmented Lagrangian optimizer. These tools promote a clean separation between modeling and optimization and allow this paper to focus primarily on the question of how to most accurately model workspace geometry and leveraged it efficiently within a motion optimization framework. The unconstrained inner loop solver uses a Levenberg Marquardt strategy with partial Gauss-Newton Hessian approximations. Section III-B discusses how these partial Gauss-Newton approximations act as Pullback metrics that communicate geometric information from the workspace to the configuration space; this generic methodology becomes a fully covariant algorithm for exactly representing the first-order Riemannian geometry of the problem.

Our optimizer solves simple Cartesian workspace (straight line end-effector) motion problems easily, even with joint limit constraints, surface penetration constraints, various velocity and acceleration penalizations in both the configuration space and the workspace, and other modeling considerations. Section IV shows that the *same system* can be easily modified to operate within a nontrivial Riemannian workspace geometry to also, with essentially no additional work, move freely among and around obstacles.

A. kth-order clique potentials and finite-differences in mapped spaces

This section reviews the basic function network formulation first used in KOMO [24], designed to expose the underlying band-diagonal structure of the objective's Hessian and facilitate the use of finite-differenced time derivatives through mapped (task) spaces. Our basic trajectory representation is $\xi = (\boldsymbol{q}_1 \ldots, \boldsymbol{q}_T)$; all time-derivatives are calculated using finite-differencing.

The cost objective terms or constraints at each time step along the trajectory depend on a collection of m differentiable maps (sometimes called task maps) $\boldsymbol{x}_i = \phi_i(\boldsymbol{q})$ mapping from the configuration space to some related task space. For instance, the forward kinematics map mapping the configuration to the end-effector location or an axis at a particular joint is one such task map, but we place no restriction beyond differentiability on what a task map might be.[6] These objective terms or constraint functions take the form

$$c_t(\xi) = \sum_{i=1}^{m} c_{ti}\left(\phi_i(\boldsymbol{q}_t), \frac{d}{dt}\phi(\boldsymbol{q}_t), \ldots, \frac{d^k}{dt^k}\phi(\boldsymbol{q}_t)\right). \quad (2)$$

Rather than taking finite-differences of configuration space variables $\boldsymbol{q}_t$ and transforming those approximate quantities using the kinematic equations describing how derivatives transform between mapped spaces, we calculate the finite-differences directly in the mapped space which make the calculations both more accurate and more straightforward. Specifically, denoting

$$\boldsymbol{x}_t^{(i)} = \phi_i(\boldsymbol{q}_t) \quad (3)$$

[6]Note that the identity map is a task map as well making the configuration space, itself, a task space as we define it.

and dropping the superscript for notational simplicity, we use the following linear map to calculate the velocity and acceleration variables in the mapped space:

$$\begin{bmatrix} \boldsymbol{x}_t \\ \dot{\boldsymbol{x}}_t \\ \ddot{\boldsymbol{x}}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_t \\ \frac{1}{\Delta t}(\boldsymbol{x}_t - \boldsymbol{x}_{t-1}) \\ \frac{1}{\Delta t^2}(\boldsymbol{x}_{t+1} + \boldsymbol{x}_{t-1} - 2\boldsymbol{x}_t) \end{bmatrix}. \tag{4}$$

Each task variable is a function of the corresponding configuration variable, so these first three time-derivatives are each functions of $\boldsymbol{x}_{t-1}, \boldsymbol{x}_t$, and $\boldsymbol{x}_{t+1}$. Since each $\boldsymbol{x}_t$ is a function of the corresponding $\boldsymbol{q}_t$, this clique of three time-derivative task variables is actually a function of $\boldsymbol{q}_{t-1}, \boldsymbol{q}_t$, and $\boldsymbol{q}_{t+1}$. An analogous relation holds for arbitrary kth-order derivatives.

Given this relationship, between time-derivatives in task space and the underlying kth-order clique of configuration space variables, we can generically describe the entire optimization problem as

$$\begin{aligned} \min_{\xi} \sum_{t=1}^{T} & c_t(\boldsymbol{q}_{t-1}, \boldsymbol{q}_t, \boldsymbol{q}_{t+1}) \\ \text{s.t.} \quad & \boldsymbol{g}_t(\boldsymbol{q}_{t-1}, \boldsymbol{q}_t, \boldsymbol{q}_{t+1}) \leq 0 \\ & \boldsymbol{h}_t(\boldsymbol{q}_{t-1}, \boldsymbol{q}_t, \boldsymbol{q}_{t+1}) = 0. \end{aligned} \tag{5}$$

Again, generalizations to kth-order derivatives and cliques are straightforward. Our experiments leverage only accelerations, so for expositional clarity we depict the equation[7] using simply $k = 2$.

The key to efficient optimization in this setting is to note that any k^{th}-order Markov network of functions has a band-diagonal Hessian of bandwidth kd, where d is the dimensionality of the configuration space. Solving band-diagonal systems is a well-studied problem [13] and, in this case, we can solve for inner loop Newton steps in time $O(T(kd)^2)$ (quadratic in the bandwidth, but *linear* in the time horizon) using well-tested and highly optimized tools from Linear Algebra packages such as LAPACK [10]. For most real-world problems, solving these band-diagonal systems is so cheap, relative to the computational expense of function evaluation across the trajectory, that this Newton step transformation is effectively free.

B. Partial Gauss-Newton approximations and Pullbacks

This section describes the partial Gauss-Newton Hessian approximation we use to avoid computing third-order tensors of differentiable maps, and relates it to Pullback metrics from Riemannian geometry. The theoretical results mentioned in Section IV-B show that these partial Gauss-Newton approximations, alone, provide the majority relevant curvature information for these problems.

At a high level, a function $c : \mathbb{R}^m \to \mathbb{R}$ defined on the codomain of a nonlinear differentiable map $\phi : \mathbb{R}^n \to \mathbb{R}^m$ can be pulled back into the nonlinear map's domain to form $\widetilde{c}(\boldsymbol{u}) = c(\phi(\boldsymbol{u}))$ as discussed in Section II-A. It's full Hessian is

$$\nabla^2_{\boldsymbol{u}} c(\phi(\boldsymbol{u})) = \boldsymbol{J}_\phi^T \nabla^2_{\boldsymbol{x}} c(\phi(\boldsymbol{u})) \boldsymbol{J}_\phi + \left[\frac{\partial \boldsymbol{J}_\phi}{\partial \boldsymbol{q}}\right] \nabla_{\boldsymbol{x}} c(\boldsymbol{x})|_{\phi(\boldsymbol{u})}.$$

The third-order tensor $\frac{\partial \boldsymbol{J}_\phi}{\partial \boldsymbol{q}} = \frac{\partial^2 \phi}{\partial \boldsymbol{q} \partial \boldsymbol{q}^T}$ in the equation is computationally expensive—it's equivalent to calculating m separate $n \times n$-dimensional Hessian matrices. Additionally, it can easily add indefinite components to the overall Hessian matrix. The first term, on the other hand is always positive (semi-)definite whenever $\nabla^2 c(\boldsymbol{x})$ is positive definite, so we can consider it a definiteness-preserving representation. In the partial Gauss-Newton approximation,[8] we simply throw away the second term, which accounts for additional curvature arising from the differentiable map ϕ, itself, and keep only that first, computationally simpler and better conditioned, term. That first term is the Pullback of the codomain's Hessian; it, alone, communicates the entirety of the first-order task space Riemannian geometry back to the differentiable map's domain. Experimentally, this term contributes enough information about the function's curvature to promote fast convergence in optimization. Section II-A discusses the geometry of such pullbacks in more detail.

Notice that task space functions defined over time derivatives, such as $c(\boldsymbol{x}, \dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}})$ are implemented as finite-differences in the task space as described in Section III-A. They, therefore, translate to a function defined over a clique of task space variables, and the resulting Hessian over that clique represents the associated task space Riemannian metric. That metric pulls back into the configuration space via the partial Gauss-Newton approximation as described above.

IV. Riemannian geometry of the workspace

We're now equipped to study the Riemannian geometry of the workspace, itself. We start with an abstract discussion on how these ideas integrate into Motion Optimization (Section IV-A) and then follow up with derivations of particular models of the workspace geometry we use in our experiments. Much of this discussion revolves around leveraging the energy form of the geodesic definition given in Equation 1.

A. Integrating workspace geometry into motion optimization

The objective portion of Equation 1 discretizes to

$$\psi(\xi) = \sum_{t=1}^{T} \frac{1}{2} \dot{\boldsymbol{x}}_t^T \boldsymbol{A}(\boldsymbol{x}_t) \dot{\boldsymbol{x}}_t \, \Delta t, \tag{6}$$

where $\dot{\boldsymbol{x}}_t = \frac{1}{\Delta t}(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})$ is a finite-differencing approximation to the workspace velocity. This discrete representation suggests that an effective way to integrate the workspace geometry into the motion optimization problem is to introduce a collection of new terms of the form

$$\psi_t(\boldsymbol{q}_{t-1}, \boldsymbol{q}_t) = \frac{1}{2\Delta t}(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})^T \boldsymbol{A}(\bar{\boldsymbol{x}}_t)(\boldsymbol{x}_t - \boldsymbol{x}_{t-1}), \tag{7}$$

[7]Notice this representation makes use of a pre-first configuration $\boldsymbol{q}_0$ and a post-last configuration $\boldsymbol{q}_{T+1}$. In practice, we assume that $\boldsymbol{q}_0$ is fixed and enforce that restriction as a hard constraint. We also typically use $\boldsymbol{q}_T$ as the final configuration and constrain it to achieving the goal. The post-last configuration $\boldsymbol{q}_{T+1}$ acts as an auxiliary configuration used just for terminal configuration acceleration calculations.

[8]The true Gauss-Newton approximation is appropriate only for generalized least-squares terms, and generally assumes the top-most (least-squares) curvature is very simple (constant and uncorrelated). This partial Gauss-Newton approximation may also be viewed as an *implicit* Gauss-Newton approximation, since for every function with positive definite $\nabla^2 c(\boldsymbol{x})$, there exists a mapping ψ under which $\nabla^2 c(\boldsymbol{x}) = \boldsymbol{J}_\psi^T \boldsymbol{J}_\psi$ (see [9]).

where $\boldsymbol{x}_t = \phi(\boldsymbol{q}_t)$ is the t^{th} configuration pushed through the forward kinematics map and $\bar{\boldsymbol{x}}_t = \frac{1}{2}(\boldsymbol{x}_t + \boldsymbol{x}_{t-1})$ is the midway points between $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$. Hessians of ψ_t can be complicated to calculate (they can sometimes be done efficiently by exploiting the structure of this objective—see the appendix of the extended version of this paper [17] for a calculation example for the metric defined in Section IV-C)—although Section IV-B presents a substantially simpler setting that's theoretically equivalent and easier to manipulate in practice. Conceptually, it helps first to make $\boldsymbol{A}(\boldsymbol{x})$ explicit in these expressions to understand how it integrates into the motion optimization problem.

The second place the Riemannian metric comes into play is in the goal constraint function or terminal potential. Since constraint functions in Augmented Lagrangian inner loop optimizations play effectively the same role as terminal potentials, we tailor this argument to just the latter.

Ideally, such a terminal potential should be a geometry aware attractor that pulls the robot along geodesics through the workspace. In the absence of obstacles, the Euclidean distance between the end-effector and the goal set works pretty well, but given environmental obstacles that warp the geometry, the potential function should be better informed. Pulling directly along a geodesic is usually hard, because finding that geodesic is an optimization problem in itself [11]. There are a number of ways to approximate the geodesic distance, though, that work pretty well in practice. The simplest of these is to use a metric weighted attractor terminal potential of the form

$$\phi_T(\boldsymbol{x}_T) = \frac{1}{2}(\boldsymbol{x}_T - \boldsymbol{x}_g)^T \boldsymbol{A}(\boldsymbol{x}_T)(\boldsymbol{x}_t - \boldsymbol{x}_g). \tag{8}$$

Section IV-B presents a more accurate form of attractor by defining it in a higher-dimensional geometric space.

With these modified terms the RieMO optimizer described in Section III automatically accounts for the Riemannian geometry of the workspace. In particular, the band-diagonal linear system solver ensures that local information about the curvature and gradient propagates efficiently across the entire trajectory during the Newton step calculation.

It's intuitively helpful to think of this band-diagonal solver as inference method for a Gaussian graphical model [21], [23]. Any symmetric positive definite linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is the first-order optimality criterion of a quadratic function, which in turn is the negative log-likelihood function of a Gaussian. Thus, inference in that Gaussian to find the mean (or mode) is equivalent to solving the system. Message passing for Gaussian graphical models has a nice and intuitive flavor; solving the linear system, especially using fast band-diagonal solvers that simply sweep once up and down the trajectory, is analogous to passing messages around about the geometry of the system until convergence. That said, linear algebra has a long history of time-critical application and the computational properties of special structures such as band-diagonality have been studied thoroughly. In our experience, it's typically much faster to use meticulously tuned and tested linear system solvers from thoroughly exercised toolkits such as LAPACK that exploit this common structure than it is to use the analogous graphical model inference methods.

B. Latent Euclidean representations and computational considerations

This section presents a useful geometric modeling construct that effectively extends the forward kinematic map nonlinearly beyond the workspace into a higher-dimensional latent space.

Any mapping of the form $\boldsymbol{z} = \phi_{\text{ws}}(\boldsymbol{x})$ into an unscaled Euclidean space defines a Pullback metric $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi$. This observation means that the generalized velocity term of the form $\dot{\boldsymbol{x}}^T \boldsymbol{A}(\boldsymbol{x})\dot{\boldsymbol{x}}$ can be equally well described as a Euclidean velocity through the map's co-domain:

$$\dot{\boldsymbol{x}}^T \boldsymbol{A}(\boldsymbol{x})\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}^T \boldsymbol{J}_\phi^T \boldsymbol{J}_\phi \dot{\boldsymbol{x}} = \left\| \frac{d}{dt}\phi_{\text{ws}}(\boldsymbol{x}) \right\|^2. \tag{9}$$

Thus, the discrete approximation to the geodesic energy objective in Equation 1 can be expressed

$$\begin{aligned}\psi(\xi) &= \sum_{t=1}^{T} \frac{1}{2} \left\| \frac{\phi_{\text{ws}}(\phi_{\text{fk}}(\boldsymbol{q}_t)) - \phi_{\text{ws}}(\phi_{\text{fk}}(\boldsymbol{q}_{t-1}))}{\Delta t} \right\|^2 \Delta t \\ &= \sum_{t=1}^{T} \frac{1}{2} \left\| \frac{\boldsymbol{f}(\boldsymbol{q}_t) - \boldsymbol{f}(\boldsymbol{q}_{t-1})}{\Delta t} \right\|^2 \Delta t,\end{aligned} \tag{10}$$

with $\boldsymbol{f} = \phi_{\text{ws}} \circ \phi_{\text{fk}}$, where ϕ_{fk} is the forward kinematics map. This version of the geodesic energy makes it easy to apply partial Gauss-Newton approximations during Hessian calculations. Importantly, Theorem 1 of the extended version of this paper [17] shows that the portion of the Hessian not captured by these partial Gauss-Newton approximations (the Pullbacks) vanishes quickly as the time-discretization becomes increasingly fine.

We call such a map $\phi_{\text{ws}}(\boldsymbol{x})$ a *geometric map*, and it's co-domain a *latent Euclidean* space. It acts as an embedding into a Euclidean space whose Pullback represents the desired geometry. The map should be a mapping to a strictly higher-dimensional Euclidean space as discussed in [15] and [17] (see also the Nash Embedding Theorem [9]). We'll see in Sections IV-C and IV-D that the nonlinear maps forming the geometric representations map the workspace into latent Euclidean spaces of strictly higher dimension.

Note that it's also straightforward to define attractors in this space of the form

$$\psi_g(\boldsymbol{x}) = \frac{1}{2}\|\phi_{\text{ws}}(\boldsymbol{x}) - \phi_{\text{ws}}(\boldsymbol{x}_g)\|^2 \tag{11}$$

pulling toward a goal point $\boldsymbol{x}_g$. Since ϕ_{ws} is a nonlinear map, its gradient field can be highly curved. Both of the concrete approaches discussed below use this representation; Section IV-D provides additional intuition behind their behavior.

C. Workspace cost-gradient stretching

Naturally, proximity to obstacles warps the space by stretching it increasingly in the direction of the obstacle. As we approach the obstacle, trajectories should become increasingly biased toward motion parallel to the surface, promoting motion along the obstacle's contours. Such a contour surface is generated by the isocontours of the cost function. The tangent space of these isocontours is the space orthogonal to the gradient direction $\nabla c(\boldsymbol{x})$ at a point $\boldsymbol{x}$, so

we posit that the right approach is to stretch the space in the direction of that gradient. Such a warping of the workspace (when pulled back into the workspace itself) is given by the Riemannian metric

$$\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{I} + \lambda \nabla c \nabla c^T, \tag{12}$$

where $\lambda \geq 0$ is a positive scaling factor defining the strength of the second term. The identity term simply represents Euclidean geometry, while the cost gradient outer product term stretches the space in the direction of the gradient. It's insightful to think through the operation of such a metric by looking at how it manifests in the corresponding norm on velocities:

$$\begin{aligned} f(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \dot{\boldsymbol{x}}^T \boldsymbol{A}(\boldsymbol{x}) \dot{\boldsymbol{x}} \\ &= \dot{\boldsymbol{x}}^T \left(\boldsymbol{I} + \lambda \nabla c \nabla c^T \right) \dot{\boldsymbol{x}} \\ &= \|\dot{\boldsymbol{x}}\|^2 + \lambda \left(\nabla c^T \dot{\boldsymbol{x}} \right)^2. \end{aligned} \tag{13}$$

That first term is just the Euclidean velocity squared norm. So if we remove the influence of the second time entirely ($\lambda = 0$), it reduces to just the Euclidean velocity squared norm alone. But adding that second term adds a component that measures the squared inner product between $\dot{\boldsymbol{x}}$ and the gradient direction ∇c. That inner product is the projection coefficient of $\dot{\boldsymbol{x}}$ in the direction ∇c, scaled by the norm of ∇c, so we can interpret that term as measuring the squared size of the component of the velocity collinear with this gradient direction in a way that increases with increasing cost. Since ∇c is generally orthogonal to the surface, this term penalizes velocity components misaligned with the obstacle's contours increasingly as our point of evaluation gets closer to the obstacle. In other words, we can interpret the metric $\boldsymbol{A}(\boldsymbol{x})$ as stretching the space in the direction of obstacles increasingly strongly as $\boldsymbol{x}$ approaches the obstacle.

A natural geometric mapping whose Pullback represents this geometry is

$$\phi^c_{\text{ws}}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x} \\ \lambda^{\frac{1}{2}} c(\boldsymbol{x}) \end{bmatrix} \quad \text{with} \quad \frac{\partial \phi^c_{\text{ws}}}{\partial \boldsymbol{x}} = \begin{bmatrix} \boldsymbol{I} \\ \lambda^{\frac{1}{2}} \nabla c^T \end{bmatrix},$$

It's straightforward to see that the corresponding Pullback metric matches Equation 12.

This geometric map ϕ^c_{ws} makes it easy to use partial Gauss-Newton Hessian calculations and a latent Euclidean attractor as a geometry aware terminal potential as described in Section IV-B.

D. Globalized local coordinates

The previous section augments the workspace with an extra dimension given by a workspace cost function. That's a useful representation when a natural differentiable cost-based environment representation exits, but here we deconstruct the workspace geometry in more detail and derive a more general geometric representation by combining local coordinate representations. While the cost-based representation relies on the cost gradient to define a single direction of most relevant stretch, the representation we examine here keeps track of all relevant obstacles simultaneously, blending continuously between local representations as a function of obstacle proximity.

Locally, close to an obstacle, it's usually relatively easy to find a mapping $\phi(\boldsymbol{x})$ that maps $\boldsymbol{x}$ into a space where geodesics through the workspace geometry (smooth obstacle avoiding paths) are straight lines. An example, one we use as a canonical experimental representation below, is the cylindrical coordinates of a pole. The straight line path between two points in cylindrical coordinates (i.e. the convex combination of the two points) never gets closer to the pole than the smallest radius of the two points. Thus, the straight line, when mapped back to the workspace, wraps naturally around the pole with no effort. Notice that this particular mapping is just a curvilinear coordinate transform, although we make no restrictions on the dimensionality of these local coordinate maps.

However, if we have multiple objects, each with non-linear local coordinate transformations $\phi_i(\boldsymbol{x})$ mapping $\boldsymbol{x}$ to a linear manifold where geodesics are easy to compute (think multiple poles each with its own cylindrical coordinate system), then the question becomes how should these local transformations be combined to make a globally consistent representation of the geometry.

More formally, suppose each of k objects $o_1, \ldots, o_k$ has an associated local mapping $\phi_i : \mathbb{R}^3 \to \mathcal{M}^3$ representing a simple change of coordinates to a representation where the metric is more natural. Locally, we can represent geodesics of this geometry by mapping two points $\boldsymbol{x}$ and $\boldsymbol{x}_g$ through ϕ_i and following a straight line trajectory through the mapped space. More globally, though, we need to formally mitigate the relative influences of multiple objects properly.

We can combine these coordinate systems by representing all of the transformations simultaneously in a single vector of higher dimension on the order of $3k$, weighing each coordinate system by a weight $\alpha_i(\boldsymbol{x})$ that reflects proximity to the object. To be concrete, suppose we have a differentiable distance function $d_i(\boldsymbol{x})$ for each object o_i that's positive outside the object and negative inside. And suppose that there is an ambient Euclidean geometry (represented by a $k+1$st *identity* transform $\phi_{k+1}(\boldsymbol{x}) = \boldsymbol{x}$) with corresponding *constant* distance function $d_{k+1}(\boldsymbol{x}) = 1$. The specific weighting functions $\alpha_i(\boldsymbol{x})$ we use in the experiments below are constructed as multinomials using these distance functions as *energy* functions $\alpha_i(\boldsymbol{x}) \propto e^{-d_i(\boldsymbol{x})/\sigma_i}$ for each $i = 1, \ldots, k+1$. Note that the length scales $\sigma_i > 0$ indicate the distance at which the ambient Euclidean geometry begins to dominate the multinomial weights.

Defining the combined coordinate transform as

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \widetilde{\phi}_1(\boldsymbol{x}) \\ \vdots \\ \widetilde{\phi}_{d+1}(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \alpha_1(\boldsymbol{x})\phi_1(\boldsymbol{x}) \\ \vdots \\ \alpha_{d+1}(\boldsymbol{x})\phi_{d+1}(\boldsymbol{x}) \end{bmatrix}. \tag{14}$$

Assuming each co-domain is Euclidean (any constant metric in the co-domain can be absorbed into the mapping itself) constructs a mapping from the workspace $\mathbb{R}^3$ to a $3(k+1)$-dimensional Euclidean space. We can calculate the resulting Pullback into the domain $\mathbb{R}^3$ by writing out the expression

for the instantaneous velocity norm:

$$f(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \left\| \frac{d}{dt}\phi(\boldsymbol{x}) \right\|^2 = \frac{1}{2}\dot{\boldsymbol{x}}^T \underbrace{\left(\sum_{i=1}^{k+1} \boldsymbol{C}_i(\boldsymbol{x})^T \boldsymbol{C}_i(\boldsymbol{x}) \right)}_{\boldsymbol{A}_\phi(\boldsymbol{x})} \dot{\boldsymbol{x}},$$

where $\boldsymbol{C}_i(\boldsymbol{x}) = \alpha_i \boldsymbol{J}_{\phi_i} + \phi_i \nabla \alpha_i^T$. The implied full Pullback metric is $\boldsymbol{A}_\phi(\boldsymbol{x})$ indicated in the equation.

As with the cost gradient workspace geometry representation of Section IV-C, since this workspace geometric representation is defined by a geometric map, we can easily leverage partial Gauss-Newton Hessian calculations and geometry aware terminal potential as described in Section IV-B.

Note that since $\phi(\boldsymbol{x})$ defines a 3-dimensional nonlinear embedding into a $3(k+1)$-dimensional Euclidean space, the attractor does not precisely follow geodesics across the curved surface in the embedded space, but the natural gradient flow through the workspace of this attractor with respect to the Pullback metric works pretty well in practice as an informative approximation to the geodesic flow.

V. Experimental demonstrations

We implemented both the cost-based workspace metric of Section IV-C and the globalized local coordinates metric of Section IV-D for problems of avoiding vertical poles of varying sizes extending from a tabletop. Thin obstacles, such as these poles, are usually hard for optimization-based motion planning because their physical extent is small relative to size of the robot. The resulting optimization problem, especially with discrete time and sparse robot body representations, is usually very ill-conditioned and difficult to solve under existing modeling and optimization methodologies. The geometric modeling components we introduce here, though, effectively broaden each obstacle's physical extent into the surrounding workspace and act to guide the optimizer more naturally around the obstacles during optimization to achieve final optimized motions inherently biased toward workspace geodesics.

A. Motion cost components

Our motion optimization problem is tailored to a single arm of the robot pictured in Figure 1 (upper left), and consists of a collection of cost objectives and constraints within the RieMO framework described in Section III. The cost terms trade off dynamics, kinematic smoothness, and posture criteria, while the constraints prevent joint limit violations and obstacle penetration, while enforcing goal success. Specifically, we use the following terms:

- **Configuration space derivatives penalties.** $f(\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \alpha_1 \|\dot{\boldsymbol{q}}\|^2 + \alpha_2 \|\ddot{\boldsymbol{q}}\|^2$.
- **Task space velocity penalties.** $f(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{1}{2}\| \frac{d}{dt}\phi(\boldsymbol{x})\|^2$, where $\phi : \mathbb{R}^3 \to \mathbb{R}^n$ is a mapping to a higher-dimensional workspace geometric space (see Section IV-D). Ideally, these penalties should be added to a collection of key points along the robot's body, but for these experiments, we add them only to the end-effector.
- **Joint limit proximity penalties.** $f_i(q_i) = \left(\max\{0, q_i - (q_{\max} - \epsilon), (q_{\min} + \epsilon) - q_i\}\right)^2$, where $\epsilon > 0$ is a joint limit margin.
- **Posture potentials.** $f(\boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{q} - \boldsymbol{q}_{\text{default}}\|^2$ pulling toward a neutral default configuration $\boldsymbol{q}_{\text{default}}$.

And we also use the following constraints

- **Joint limit constraints.** We have explicit constraints on the joint limits that prevent them from being violated in the final solution.
- **Obstacle constraints.** All objects are modeled as analytical inequality constraints with a margin. The end-effector, first knuckle, and second knuckle use margins of 0cm, 1cm, and 6cm, respectively; the lower and upper wrist joints use margins of 14cm and 17cm, respectively.
- **Goal constraint.** Reaching the goal is enforced as a zero distance constraint on the goal proximity distance function (see Section IV). This strategy generalize the goal set ideas described in [4].

B. Results

Figure 1 shows a collection of arm trajectories moving around and between thin cylindrical obstacles planned for MPI's Apollo platform. The top row shows the behavior of the cost-gradient workspace Riemannian metric described in Section IV-C tracing a path around a column on the table. In this case, since our attractor is build around scaled Euclidean distances, we can only move up to half way around the column before the attractor chooses a topologically distinct path. However, the metric in conjunction with the metric-weighted attractor is consistently able to move smoothly around the column, tracing naturally across its surface. The figure shows a bigger column here for visual impact, but the method performs just as well on the smaller column.

The bottom row of the figure demonstrates motions highly contorted by two poles in the environment. The environmental geometry in these executions is represented by combining local cylindrical coordinate systems as described in Section IV-D. Our cylindrical coordinate system doesn't wrap around from π to $-\pi$ in θ, and instead chooses the sign of θ based on which way the arm is wrapped around the pole. This choice of sign encodes the topological homotopy class created by the physical constraint that the arm must connect to the fixed torso one way or the other around the obstacle. Any terminal potential build on Euclidean distances would take the arm along a topologically distinct path around the back of the pole, but this spiraling cylindrical coordinate system implements a nice heuristic consistent with the robot's physical topological constraints, and chooses a successful highly non-Euclidean motion of the end-effector (and full arm configuration) around the pole.

Our focus here has been on modeling the motion optimization problem, so we implemented the optimization system making engineering choices to promote correctness and ease of use over speed. Even so, optimization times are typically between .3 and 1 second depending on the complexity of the problem, running on a Linux virtual machine atop a 2012 Mac PowerBook. We emphasize here, that our optimizer does not decompose the problem into separate workspace geodesic optimizations with subsequent tracking optimizations. These trajectories are all generated by a single optimization on an objective that builds the above described geometric workspace model directly into its potential functions.

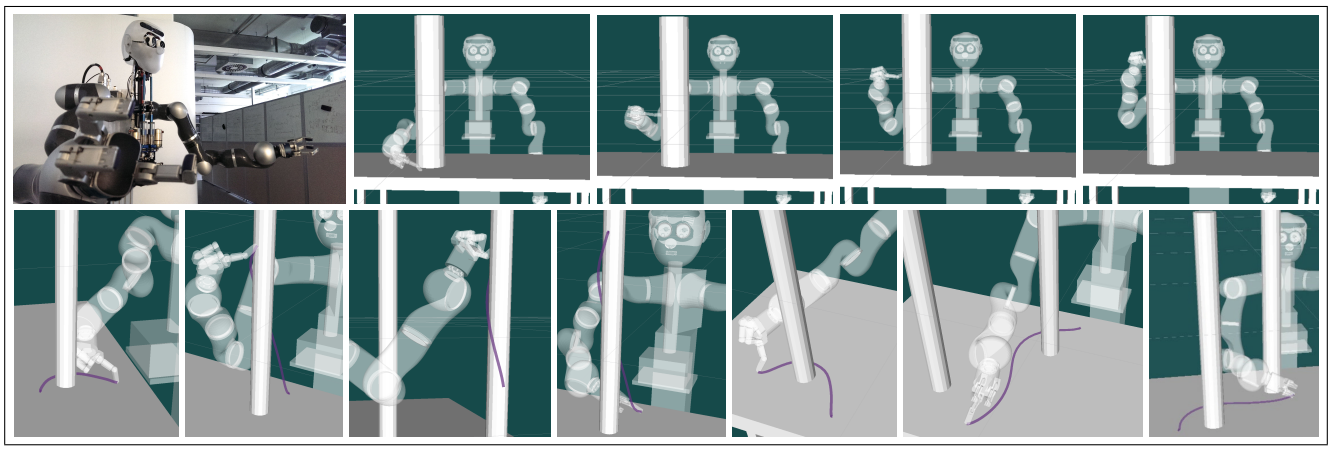

Fig. 1. Top (left): The Apollo dual-arm manipulation platform at the Max Planck Institute for Intelligent Systems modeled in our experiments. Top (remaining): Apollo using the cost-gradient metric and attractor described in Section IV-C to trace a path around a column. Bottom: A collection of trajectories found using the combined local transformation metric and corresponding attractor as described in Section IV-D. Each image shows the final configuration with a purple line indicating the end-effector trajectory.

VI. Conclusions and Future Work

In the 80's and 90's researchers spent significant effort to understand the geometry and fundamental complexity of motion generation. The differential geometric insights from [19] and related work led to significant advances in our theoretical understanding of the problem. More recently, a lot of effort has gone into the more practical side of developing useful tools and techniques to further the practice of robot motion generation. Interestingly, within that work, largely motivated by what works best, a common thread has emerged around the fundamental importance of optimization and second-order information. This paper takes a view that optimization is, and will remain, a centerpiece of motion generation. It has, therefore, been our goal here to understand how the optimizers we use communicate information about problem geometry across the trajectory and between different mapped spaces during the calculation of a Newton step, and how we can leverage those properties to better model motion through the non-Euclidean geometry of the workspace. Moving forward we hope to build on these ideas, leveraging fast combinatoric algorithms for approximating geodesic flow through the discretized workspace (voxel grid) to create more expressive geometrically cognizant attractors that obviate the need for the approximate terminal potentials we use here.

References

[1] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Comput.*, vol. 10, no. 2, pp. 251–276, Feb. 1998. [Online]. Available: http://dx.doi.org/10.1162/089976698300017746

[2] J. A. D. Bagnell and J. Schneider, "Covariant policy search," in *Proceeding of the International Joint Conference on Artifical Intelligence*, August 2003.

[3] J. P. Chonhyon Park and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[4] A. Dragan, N. Ratliff, and S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *2011 IEEE International Conference on Robotics and Automation*, May 2011.

[5] S. Hassani, *Mathematical Physics: A Modern Introduction to its Foundations*, 2nd ed. Springer, 2013.

[6] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011. [Online]. Available: http://www-clmc.usc.edu/publications/K/kalakrishnan-ICRA2011.pdf

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

[8] J. M. Lee, *Introduction to Smooth Manifolds*, 2nd ed. Springer, 2002.

[9] J. Nash, "The imbedding problem for Riemannian manifolds," *Ann. Math.*, vol. 63, pp. 20–63, 1956.

[10] Netlib, "Lapack 3.4: Linear algebra package," 2011. [Online]. Available: http://www.netlib.org/lapack

[11] L. Noakes, "A global algorithm for geodesics," *J. Australian Math. Soc. (Series A)*, vol. 64, pp. 37–50, 1998.

[12] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2006.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 9 2007.

[14] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *In Proceedings of the International Conference on Robotics and Automation*, 1993, pp. 802–807.

[15] N. Ratliff, "Learning geometric reductions for planning and control," in *ICML Workshop on Robot Learning*, June 2013.

[16] ——, "Multivariate calculus II: The geometry of smooth maps," 2014, lecture notes: Mathematics for Intelligent Systems series.

[17] N. Ratliff, M. Toussaint, and S. Schaal, "Riemannian motion optimization," Max Planck Institute for Intelligent Systems, Tech. Rep., 2015.

[18] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[19] E. Rimon and D. Koditschek, "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," *Transactions of the American Mathematical Society'*, vol. 327, no. 1, pp. 71–116, 1991.

[20] J. D. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *In the proceedings of Robotics: Science and Systems (RSS)*, 2013.

[21] O. Shental, D. Bickson, P. H. Siegel, J. K. Wolf, and D. Dolev, "Gaussian belief propagation solver for systems of linear equations," in *IEEE Int. Symp. on Inform. Theory (ISIT)*, Toronto, Canada, July 2008.

[22] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, 2nd ed. Springer, 2010.

[23] M. Toussaint, "Robot trajectory optimization using approximate inference," in *(ICML 2009)*. ACM, 2009, pp. 1049–1056.

[24] ——, "Newton methods for k-order Markov constrained motion problems," *CoRR*, vol. abs/1407.0414, 2014. [Online]. Available: http://arxiv.org/abs/1407.0414