AEROSP 575 Final Project

Review of CHOMP: Covariant Hamiltonian Optimization for Motion Planning

Justin Diep
Electrical and Computer Engineering
University of Michigan, Ann Arbor
Ann Arbor, Michigan
jjdiep@umich.edu

Yash Kapoor
Aerospace Engineering
University of Michigan, Ann Arbor
Ann Arbor, Michigan
ykapoor@umich.edu

***Abstract*—In this paper, we review a seminal publication in the 2013 International Journal of Robotics Research titled CHOMP: Covariant Hamiltonian Optimization for Motion Planning by Zucker et al., a method for for trajectory optimization invariant to reparametrization. CHOMP uses functional gradient techniques to iteratively improve the quality of an initial trajectory, optimizing a functional cost equation that trades off between a smoothness and an obstacle avoidance component. CHOMP can be used to locally optimize feasible trajectories, as well as to solve motion planning queries, converging to low- cost trajectories even when initialized with infeasible ones. It uses Hamiltonian Monte Carlo to alleviate the problem of convergence to high-cost local minima (as well as for probabilistic completeness), and is capable of respecting hard constraints along the trajectory. Lastly the paper presents extensive results from experiments with CHOMP on manipulation and locomotion tasks, using 7-DOF manipulators and a rough-terrain quadruped robot (Barret Technology's WAM arm and Boston Dynamics's LittleDog).**

In this report, we will cover the motivation and background of the problem that CHOMP addresses, the mathematical model and optimization problem formulation that the algorithm uses, as well as the experiments and results that they used to demonstrate its efficacy. Based on an analysis of their methods and results, we evaluate the methods and results produced in the original paper and compare it to prior research conducted in the motion planning and trajectory optimization fields in order to ultimately judge whether the objective of the paper was met.

***Index Terms*—trajectory optimization, motion planning, robotics**

I. Introduction

A. Motivation

A fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move. The terms motion planning and trajectory planning are often used for these kinds of problems. Motion planning (also known as the navigation problem or the piano mover's problem) is used to find a sequence of valid configurations that moves the robot from the source to destination. A basic motion planning problem is to compute a continuous path that connects a start configuration $\mathcal{S}$ and a goal configuration $\mathcal{G}$, while avoiding collision with known obstacles. The robot and obstacle geometry is described in a 2D or 3D workspace, while the motion is represented as a path in (possibly higher-dimensional) configuration space $\mathcal{C}$.

Motion planning is important in modern day robotics for several reasons. First, of all, in order to develop autonomous robots that can interact in the physical world and handle even mundane tasks, algorithms must be developed for robots to bridge low level individual actions and control inputs in order to solve high-level, possibly dangerous tasks that can be in potentially hazardous or remote environments that are difficult to access for humans. Another reason is that, due to limited processing and computational power on most hardware robotic platforms, better algorithms are needed in order to be more efficient and and have more robust implementations for more sophisticated problems in robotics. The last reason is that, given the widespread successes of planning algorithms in several industries and academic disciplines, including robotics, manufacturing, drug design, and aerospace applications, the field remains highly in demand and has rapidly grown in recent years and may have many more fascinating applications on the horizon.

B. Background

The deficiencies in existing high-dimensional motion planning algorithms led Zucker et al. to the development of CHOMP: Covariant Hamiltonian Optimization for Motion Planning. According to the authors in their original paper, ”in domains sparsely populated by obstacles, the heuristics used by sampling-based planners to navigate 'narrow passages' can be needlessly complex; furthermore, additional post-processing is required to remove the jerky or extraneous motions from the paths that such planners generate”, resulting in them deeming such algorithms to be ”simultaneously overpowered and underpowered” [2]. In their paper published in the International Journal of Robotics Research in 2013, Zucker et al. proposed a new motion planning algorithm that would utilize trajectory optimization techniques in high-dimensional spaces while being invariant to reparameterization. The CHOMP algorithm, is a variational strategy that uses functional gradient techniques to iteratively improve the quality of an initial trajectory in order to optimize a functional that trades off between smoothness $\mathcal{U}_{smooth}$ and obstacle avoidance $\mathcal{U}_{obs}$. In conjunction with a Hamiltonian Monte Carlo (HMC) algorithm that perturbs the minimal trajectory in order to restart the process as well as to provide a sampling procedure for the trajectory distribution, the algorithm attempts to solve for an optimal trajectory while incorporating dynamics, smoothness, and obstacle avoidance in its objective.

The algorithm was developed under two central tenets:

1) Gradient information is often available and can be computed inexpensively:
CHOMP computes $\mathcal{U}_{smooth}$ and a $\mathcal{U}_{obs}$ in its objective function. $\mathcal{U}_{smooth}$ is a generalization of the trajectory smoothness in the space of the trajectories. $\mathcal{U}_{obs}$ is a line integral of the sclar cost field c, and generalizes a cost field using the workspace and the body points of the robot with the help of Euclidean Distance Transforms (EDT). This enables the algorithm to compute functional gradients efficiently.
2) Invariance to parameterization:
Invariance guarantees identical behavior independent of the type of parametrization used, and ensures the algorithms chosen respect the underlying problem geometry. This motivated the choice of variational methods for optimization. The functional gradient is the gradient of the functional $\mathcal{U}$ with respect to the trajectory. Perturbations of the trajectory are then in terms of the endowed norm. This gives a clearly defined rule for modifying the gradient, making it covariant to reparametrization.

These two central tenets informed the development of the algorithm as it was successfully implemented on several robotic platforms including the including the *HERB* 1.0 and 2.0 mobile manipulation platforms, the LittleDog quadruped, Carnegie Mellon's *Andy* Autonomous Robotics Manipulation-Software (ARM-S) platform, and the Willow Garage PR2 robot. In their research on these platforms, they found CHOMP to be an effective planning algorithm of first resort, surpassing randomized planners in performance for typical problems. They describe their extensive experiments in several different simulated environments with varying clutter, and compare CHOMP with other popular motion planning algorithms in bidirectional RRT and RRT* algorithms. Ultimately, they concluded that trajectory planning methods such as those used in CHOMP can be effective in playing a complementary role to solving simple high-dimensional motion planning problems quickly and effectively, while further acting to smooth and improve the results of other techniques locally.

II. The CHOMP Algorithm: Math Model and Optimization Formulation

In this section, we review the details of the CHOMP algorithm, a trajectory optimization procedure based on covariant gradient descent. The end goal of the algorithm is to find smooth (without unnecessary motion), collision-free trajectories for complex robotic systems with many degrees of freedom through the configuration space $\mathbb{R}^m$ between two prespecified end points $q_{init}, q_{goal} \in \mathbb{R}^m$. The authors note that in real implementation, the trajectory is first discretized into a set of n waypoints $q_1, ..., q_n$ (excluding the endpoints) where then the dynamical quantities are computed via finite differencing for the sake of computational efficacy.

A. Mathematical Models

1) Simplified Robot Model: The robot's body is made up of a set of points, $\mathcal{B} \subseteq \mathbb{R}^w$. Using the forward kinematics function $x(q,u) : \mathbb{R}^d \to \mathcal{B}$ gives the point $u \in \mathcal{B}$ of the robot, given its configuration q. The set $\mathcal{B}$ can be approximated by a set of geometric primitives which enclose the body of the robot, such as spheres, capsules, and polyhedra. This simplification was made in the experiments conducted in this paper as a swept-sphere capsule was fitted to each link of the robot manipulator, with a discrete set of spheres for each capsule then generated. In this way, the nearest distance to any point in $\mathcal{B}$ was made simpler to compute. This is because for spheres, the distance to any point is given by the distance to the center of the sphere minus its radius.

2) Trajectory: A trajectory $\xi : [0,1] \to \mathcal{C} \subset \mathbb{R}^d$ is considered as a smooth function mapping time to robot configurations. Since ξ is a function, the objective criterion to be optimized is represented as an objective *function* $\mathcal{U} : \Xi \to \mathbb{R}$ which then maps each trajectory ξ in the space of trajectories Ξ to a real number. Thus, to achieve the goal of a short, dynamically sound (low acceleration and jerk) and collision-less trajectory, the objective must also take into account obstacle proximity or collision criteria besides the dynamical quantities of the trajectory. It is defined that such a trajectory ξ is collision-free if for every configuration $q_i \in \xi$, for each point $x_u \in x(q_i, u)$, the distance from that point to any obstacle is greater than some threshold $\epsilon > 0$.

3) Obstacle Representations: Let the function $\mathcal{D} : \mathbb{R}^w \to \mathbb{R}$ provide the distance from any point in the workspace to the surface of the nearest obstacle. Assuming all obstacles are closed objects with finite volume, if the point $x \in \mathcal{R}^w$ is inside of an obstacle, $\mathcal{D}(x)$ is negative, whereas if x is outside of all obstacles, $\mathcal{D}(x)$ is positive, and $\mathcal{D}(x)$ is zero if the point x lies on the boundary of an obstacle. In practice, the distance function $\mathcal{D}$ can be implemented using geometric obstacle primitives (such as boxes, spheres, and cylinders), or it can be pre-computed and stored as a discrete array of distance values using a Euclidean Distance Transform algorithm. Efficient algorithms to compute the discrete Euclidean Distance Transform include P. Felzenswalb and D. Huttenlocher's $\mathcal{O}(n)$ algorithm [3], and a $\mathcal{O}(n \log n)$ algorithm from E.G. Gilbert et al. [4], starting from a boolean obstacle grid. The EDT for both the obstacle grid d and its complement $\bar{d}$ is computed and the signed distance field is given by the difference of these two fields, i.e. $\mathcal{D}(x) = d(x) - \bar{d}(x)$. These algorithms also provide a straightforward way to store an approximation to the gradient of the distance field, $\nabla\mathcal{D}(x)$ via finite differencing over the field's contents.

Many of the objects with which robots must interact cannot be easily represented as collections geometric primitives. The objects are almost never convex and typically have complex shape. In addition, the authors note that the efficiency of many modern robot perception systems is due to leveraging certain domain knowledge of the problem, frequently involving detailed models of the known objects in the environment. In an effort to leverage a symbiotic relationship of planning and perception, the they propose using the same object models for efficient distance field computation.

In particular, the authors note and exploit the fact that the environment representations based on distance fields are compositional under the min operation. For every available object model, a high-resolution distance field and its gradients are computed via an extensive, but off-line computation in free space and with respect to a certain frame of reference of the object, F_O. During planning, a perception process generates a model of the environment which includes a set $\mathcal{O}$ of objects and their poses, expressed with homogeneous transforms $T_{F_O}^{F_W}$ in world frame F_W. Then the distance field computation is reduced to a minimization across a set of distance field primitives pre-computed for each object in $\mathcal{O}$.

$$\mathcal{D}(x) = \min_{O \in O} (T_{F_O}^{F_W})^{-1} x \tag{1}$$

Hierarchical representations, such as the k-d tree, may be utilized to speed up this computation. In their experiments, Zucker et al. use a combination of these approaches. For static elements of the environment, a distance field is pre-computed and stored, and for dynamic or sensed objects, oriented bounding boxes as well as pre-computed distance fields are used. For raw sensor data such as point clouds, an occupancy grid is kept track of over the entire workspace, and is also used to generate a distance field.

B. Optimization Problem Formulation

CHOMP is designed to produce high quality trajectories for complex robotic systems with many degrees of freedom which are both smooth—without unnecessary motion—and collision free. The goal was to develop an algorithm that produces short, dynamically sound (low acceleration and jerk) trajectories efficiently in a way that is not affected by the arbitrary choice of trajectory representation, which in turn influenced how the objective functional is derived.

1) Objective Functional: The objective function here separately measures two complementary aspects of the motion planning problem. It first penalizes a trajectory based on criteria such as velocities and accelerations to obtain smooth trajectories. Secondly, it penalizes proximity to objects in the environment to encourage trajectories that clear obstacles by a prescribed amount. These are denoted by $\mathcal{F}_{smooth}$ and $\mathcal{F}_{obs}$, and the objective is simply defined as:

$$\mathcal{U}[\xi] = \mathcal{F}_{obs}[\xi] + \lambda \mathcal{F}_{smooth}[\xi] \tag{2}$$

The trajectory ξ is a function that maps time to the configurations of the robot. We will assume that the initial configuration $\xi(0) = q_0$ and final configuration $\xi(1) = q_1$ are fixed. $\mathcal{F}_{smooth}$ measures dynamical quantities across the trajectory such as the integral over squared velocity norms:

$$\mathcal{F}_{smooth}[\xi] = \frac{1}{2} \int_0^1 \left\| \frac{\partial}{\partial t} \xi(t) \right\|^2 dt \tag{3}$$

While $\mathcal{F}_{smooth}$ encourages smooth trajectories, the objective's other term $\mathcal{F}_{obs}$, encourages collision free trajectories by penalizing parts of the robot that are close to obstacles, or already in collision. Let $\mathcal{B} \subset \mathbb{R}^3$ be the set of points on the exterior body of the robot and let $x : \mathcal{C} \times \mathcal{B} \to \mathbb{R}^3$ denote the forward kinematics, mapping the configuration of a robot $q \in Q$ and a particular body point $u \in \mathcal{B}$ to a point $x(q, u)$ in the workspace. Furthermore, let $c : \mathbb{R}^3 \to \mathbb{R}$ be a workspace cost function that penalizes the points inside and around the obstacles. the obstacle objective $\mathcal{F}_{obs}$ is an integral that collects the cost encountered by each workspace body point on the robot as it sweeps across the trajectory. Specifically, it computes the arc length parametrized line integral of each body point's path through the workspace cost field and integrates those values across all body points:

$$\mathcal{F}_{obs}[\xi] = \int_0^1 \int_{\mathcal{B}} c(x(\xi(t), u)) \left\| \frac{\partial}{\partial t} x(\xi(t), u) \right\| du\, dt \tag{4}$$

Here, the workspace cost function c is multiplied by the norm of the workspace velocity of each body point, transforming what would otherwise be a simple line integral into its corresponding arc-length parametrization. The arc-length parametrization ensures that the obstacle objective is invariant to re-timing the trajectory (i.e. moving along the same path at a different speed). Alternatively, it may be useful to define $\mathcal{F}_{obs}$ to choose the *maximum* cost over the body rather than the integral over the body:

$$\mathcal{F}_{obs}[\xi] = \int_0^1 \max_{u \in \mathcal{B}} c(x(\xi(t), u)) \left\| \frac{\partial}{\partial t} x(\xi(t), u) \right\| dt \tag{5}$$

Intuitively, this formulation minimizes the largest violation of the obstacle constraint at each iteration.

2) Functional Gradients: To optimize such objective criteria, gradient methods were used. In this case, a *functional gradient* – the generalization of the notion of a direction of steepest ascent in the functional domain. That is, $\bar{\nabla}\mathcal{U}$ is the perturbation $\phi : [0, 1] \mapsto \mathbb{R}^d$ that maximizes $\bar{\nabla}\mathcal{U}[\xi + \epsilon\phi]$ as $\epsilon \to 0$. Let the objective function take the form $\mathcal{U}[\xi] = \int v(\xi(t), \xi'(t))dt$. Considering the perturbation ϕ applied to ξ by a scalar amount ϵ to arrive at the new function $\bar{\xi} = \xi + \epsilon\phi$. Then, holding ξ and ϕ. constant, we may regard $\mathcal{U}[\bar{\xi}]$ to be a function $f(\epsilon)$, with

$$\frac{df}{d\epsilon} = \int \left(\frac{\partial v}{\partial \bar{\xi}} \cdot \frac{\partial \bar{\xi}}{\partial \epsilon} + \frac{\partial v}{\partial \bar{\xi}'} \cdot \frac{\partial \bar{\xi}'}{\partial \epsilon} \right) dt \tag{6}$$

Considering the second term in the parenthesis, integrating by parts, and applying the constraint that $\phi(0) = \phi(1) = 0$, we find

$$\int \frac{\partial v}{\partial \bar{\xi}'} \cdot \frac{\partial \bar{\xi}'}{\partial \epsilon} dt = - \int \frac{d}{dt} \left(\frac{\partial v}{\partial \bar{\xi}} \right) \cdot \frac{\partial \bar{\xi}'}{\partial \epsilon} dt \tag{7}$$

So, overall

$$\frac{df}{d\epsilon} = \int \left(\frac{\partial v}{\partial \bar{\xi}} - \frac{d}{dt} \frac{\partial v}{\partial \bar{\xi}'} \right) \cdot \frac{\partial \bar{\xi}}{\partial \epsilon} dt \tag{8}$$

When ϵ =0, the directional derivative of $\mathcal{U}[\xi]$ is basically taken in the direction ϕ, and the above equation becomes,

$$\left. \frac{df}{d\epsilon} \right|_0 = \int \left(\frac{\partial v}{\partial \epsilon} - \frac{d}{dt} \frac{\partial v}{\partial \xi'} \right) \cdot \phi \, dt \tag{9}$$

The perturbation ϕ that maximizes this integral is the one which is proportional to the quantity in the parenthesis above, and hence the functional gradient is defined as:

$$\bar{\nabla}\mathcal{U}[\xi] = \frac{\partial v}{\partial \xi} - \frac{d}{dt}\frac{\partial v}{\partial \xi'} \tag{10}$$

This can be extended to include higher derivatives of ξ. The functional gradient vanishes at the critical points of $\mathcal{U}$, which includes the minima we are finding. Steepest descent is run on the objective function by iteratively taking steps in the direction of the negative functional gradient:

$$\xi_{i+1} = \xi_i - \eta_i \bar{\nabla}\mathcal{U}[\xi] \tag{11}$$

until a local minimum has been reached. The functional gradient of the smoothness objective is:

$$\bar{\nabla}\mathcal{F}_{smooth}[\xi](t) = -\frac{d^2}{dt^2}\xi(t), \tag{12}$$

and the functional gradient of the obstacle objective is given by:

$$\bar{\nabla}\mathcal{F}_{obs}[\xi] = \int_{\mathcal{B}} J^T \|x'\| \left[(I - \hat{x}'\hat{x}'^T)\nabla c - c\kappa\right] du, \tag{13}$$

where $\kappa = \|x'\|^{-2}(I - \hat{x}'\hat{x}'^T)x''$ is the curvature vector along the workspace trajectory traced by a particular body point, x' and x'' are the velocity and acceleration of a body point, $\hat{x}'$ is the normalized velocity vector, and J is the kinematic Jacobian at that body point. To simplify notation, we suppress dependencies on time t and body point u in these expressions. The matrix $(I - \hat{x}'\hat{x}'^T)$, is a projection matrix that projects workspace gradients orthogonally to the trajectory's direction of motion. It acts to ensure that the workspace gradient, as an update direction, does not directly manipulate the trajectory's speed profile.

3) Functionals and Covariant Gradients for Trajectory Optimization: The objective function $\mathcal{U}[\xi] = \mathcal{F}_{obs}[\xi] + \lambda\mathcal{F}_{smooth}[\xi]$ is manifestly invariant to the choice of parametrization since both the obstacle term $\mathcal{F}_{obs}$ and the smoothness term $\mathcal{F}_{smooth}$ depend only on physical traits of the trajectory. This section discusses at an abstract functional level the principle behind removing this dependence on the parametrization. The approach used revolves around properly defining a norm on trajectory perturbations to depend solely on the dynamical quantities of the trajectory, in terms of an operator A:

$$\|\xi\|_A^2 = \int \sum_{n=1}^{k} \alpha_n (D^n \xi(t))^2 dt \tag{14}$$

where D^n is the nth order derivative operator and $\alpha_n \in \mathbb{R}$ is a constant. Similarly, the inner product on the space of trajectories is defined by measuring the correlation between trajectory derivatives:

$$\langle \xi_1, \xi_2 \rangle = \int \sum_{n=1}^{k} \alpha_n (D^n \xi_1(t))(D^n \xi_2(t)) dt \tag{15}$$

A at this point serves primarily to distinguish this norm and inner product from the Euclidean norm, but it's notational purpose is clarified in the sub-sections below. For simplicity of presentation, the authors do not discuss the role of A in its general form beyond saying that it is an abstract operator formed of constituent differential operators D as $A = D^\dagger D$ so that $\|f\|_A^2 = \langle f, Af \rangle = \int (Df(t))^2 dt$. The pertubations are measured to a trajectory in terms of how much the perturbation changes the overall velocity profile of the trajectory. Functional gradients taken with respect to this definition of norm are covariant to $re-parametrization$ of the trajectory in the sense that if the trajectory's parametrization is changed, there is still a clearly defined rule for changing the norm as well so that it still measures the same physical quantity. The functional gradient that arises under this invariant norm $\|\cdot\|_A$ differs from the Euclidean functional gradient only in that it is transformed by the inverse of A:

$$\bar{\nabla}_A \mathcal{U}[\xi] = A^{-1} \bar{\nabla}\mathcal{U}[\xi] \tag{16}$$

The mathematical presentation is simplified by deriving results in terms of their finite-dimensional counterparts in a waypoint parametrization. Using techniques from the analysis of $direct\ methods$ in the Calculus of Variations, it can be shown that the limit of these results as the discretization becomes increasingly fine converges to the functional variants; however, finite-dimensional linear algebra is often simpler than its infinite-dimensional counterpart. The infinite-dimensional theory behind CHOMP constitutes a theoretical framework that facilitates deriving these algorithms for very different forms of trajectory representation such as spline parametrization or reproducing kernel Hilbert space parametrization.

4) Waypoint Trajectory Parametrization: There are many valid representations for parametrization, but a uniform discretization is chosen which samples the trajectory function over equal time steps of length $\Delta t : \xi \approx (q_1^T, q_2^T, ..., q_n^T)^T \in \mathbb{R}^{n \times d}$, with q_0 and q_{n+1} the fixed starting and ending points of the trajectory. Using this waypoint parametrization, the smoothness objective can be written as a series of finite differences:

$$\mathcal{F}_{smooth}[\xi] = \frac{1}{2}\sum_{t=1}^{n+1} \left\| \frac{q_{t+1} - q_t}{\Delta t} \right\|^2 \tag{17}$$

With a finite differencing matrix K and vector e, this can be rewritten as:

$$\mathcal{F}_{smooth}[\xi] = \frac{1}{2}\|K\xi + e\|^2 = \frac{1}{2}\xi^T A \xi + \xi^T b + c \tag{18}$$

with $A = K^T K$, $b = K^T e$, and $c = e^T e/2$. The prior term is quadratic in ξ with Hessian A and gradient b. Here, the matrix A can be shown to measure the total amount of acceleration in the trajectory.

5) Gradient Descent: Having a finite dimensional parametrization of the trajectory ξ and the gradient of $\mathcal{U}$ with respect to each element q_t of ξ, the gradient descent can now be implemented. An iterative update is defined that starts from an initial trajectory ξ_0 and computes the refined trajectory ξ_{i+1} given the previous trajectory ξ_i. Typically, a naïve initial trajectory ξ_0 consisting of a straight line path in configuration space is chosen. This path is not in general feasible; however, unlike prior trajectory optimizers, CHOMP is often able to find collision free trajectories nonetheless. To get the update rule, the Lagrangian form of an optimization problem is solved that attempts to maximize the decrease in our objective function subject to keeping the difference between ξ_i and ξ_{i+1} small. The covariant update rule defined seeks to make small changes in the average acceleration of the resulting trajectory, instead of simply making small changes in the (arbitrary) parameters that define it. A first order Taylor series approximation is used to linearize $\mathcal{U}$ around ξ_i:

$$\mathcal{U}[\xi] \approx \mathcal{U}[\xi_i] + (\xi - \xi_i)^T \bar{\nabla}\mathcal{U}[\xi_i] \tag{19}$$

The optimization problem is now defined as:

$$\xi_{i+1} = arg \min_{\xi} \mathcal{U}[\xi_i] + (\xi - \xi_i)^T \bar{\nabla}\mathcal{U}[\xi_i] + \frac{\eta}{2} \left\| \xi - \xi_i \right\|_M^2 \tag{20}$$

The term $\left\| \xi - \xi_i \right\|_M^2 = (\xi - \xi_i)^T M (\xi - \xi_i)$ is the squared norm of the change in trajectory with respect to a metric tensor M, and the regularization coefficient η determines the trade-off between minimizing $\mathcal{U}$ and step size. M is chosen to be the matrix A defined previously. If the term on the right hand side of the above equation is differentiated with respect to ξ and set the result to zero, the **update rule** is derived which is:

$$\xi_{i+1} = \xi_i - \frac{1}{\eta} A^{-1} \bar{\nabla}\mathcal{U}[\xi_i] \tag{21}$$

This update rule is $covariant$ in the sense that the change to the trajectory that results from the update is a function only of the trajectory itself, and not the particular representation used (e.g. waypoint based), at least in the limit of small step size and fine discretization. Analyzing the behavior of the CHOMP update rule in the general case is very difficult to characterize. However, by considering in a region around a local optimum sufficiently small that $\mathcal{F}_{obs}$ is convex insight can be found into the performance of both standard gradient methods and the CHOMP rule. The overall CHOMP objective function is $strongly\ convex$ – that is, it can be lower-bounded over the entire region by a quadratic with curvature A. Gradient descent minimizes an uninformed, isotropic quadratic approximation while more sophisticated methods, like Newton steps, compute tighter lower bounds using a Hessian. In the case of CHOMP, the Hessian need not exist as the obstacle objective $\mathcal{F}_{obs}$ may not even be twice differentiable, however a quadratic lower bound can be formed using A. This is much tighter than an isotropic bound and leads to a correspondingly faster minimization of the objective function. The most straightforward termination criteria for CHOMP is to terminate when the magnitude of the gradient $\bar{\nabla}\mathcal{U}[\xi]$ falls below a predetermined threshold. The algorithm may terminate with a trajectory that is not feasible, and furthermore CHOMP will fail to report if no feasible path exists. Finally, it should be noted that although the A matrix may be quite large, it is also sparse, with a very simple band diagonal structure. Hence, algorithms such as Thomas' backsubstitution method can be used to solve the system $Ax = b$ in time $O(n)$ rather than a naïve matrix inverse which requires $O(n^3)$ time.

6) Cost Weight Scheduling: The weights in $\mathcal{U}[\xi] = \mathcal{F}_{obs}[\xi] + \lambda \mathcal{F}_{smooth}[\xi]$ are useful for varying the emphasis on either of the sum components during optimization via a process referred to as $weight\ scheduling$, similar to gain scheduling in non-linear control. At the early stages in optimization, it is advantageous to give most of the weight to collision avoidance and similar costs and nearly remove the influence of the smoothness cost. As optimization progresses and trajectory obstacle cost significantly reduces, the weight is shifted to the smoothness cost to ascertain that the trajectory remains attractive with respect to dynamics measures. Even while the optimizer focuses on obstacle avoidance, the covariant trajectory updates still maintain smoothness. This measure may make the overall cost function more robust to local minima. For example, if the initial trajectory is already smooth but in collision, then the action of the collision cost will be to modify the trajectory to bring it out of collision. If this modification would cause the trajectory to be less smooth, the smoothness cost will in fact try to undo that change and to restore smoothness, thereby bringing the trajectory back into collision.

III. Methodology

In this section, we review Zucker et al's approach in deriving CHOMP. Their approach was to first define the mathematical properties of the space of trajectories (quantities such as norm and inner product) and the objective function entirely in terms of physical aspects of the trajectory so that mathematical objects and calculations depend only on the physics of the trajectory and not on subtle representational nuances of the problem. Leveraging the idea of covariance to reparametrization from Riemannian geometry, they derived CHOMP as a steepest descent optimization algorithm that in principle operates directly on the trajectories themselves, and not on particular representations of those mathematical objects. Changing trajectory parametrization, therefore, does not change the algorithm's behavior. In practice, they used a simple discretized waypoint representation of the trajectory for its computational efficacy. By design, the covariant notion of gradient ensures that the behavior of our implementation, modulo approximation error induced by the reduction to finite-dimensions, closely matches the theoretically prescribed behavior.

In addition to this representational invariance, another key contribution of CHOMP is the use of workspace gradient information to allow obstacles to "push" the robot into collision free configurations. Approaches to trajectory optimization traditionally rely on separate planners to find a feasible trajectory first before attempting any optimization. CHOMP, however, uses this workspace gradient information to find solutions even when the initial trajectory is infeasible. CHOMP can be conceptualized as minimizing an objective over a set of paths through the workspace, all jointly constrained by the configuration space trajectory and the robot kinematics, along with a potential measuring the size of the configuration space trajectory itself. In this view, and each path corresponds to the motion of a single point u on the robot's body, and the objective function acts to push these workspace paths away from obstacles.

A. Hamiltonian Monte Carlo

We review the Hamiltonian Monte Carlo (HMC) method which is a sampling based technique that leverages gradient information. The algorithm has strong ties to the simple momentum-based optimization procedures commonly used to avoid local minima in neural network training, and has straightforward generalizations to *simulated annealing* that re-frame it as an optimization procedure. This technique is an efficient way to better explore the full space of solutions and to be more robust to local minima. Additionally, the sampling perspective makes CHOMP *probabilistically complete* by entertaining all options with probability dependent on the trajectory cost assigned by the cost function.

1) Intuition: With the objective function $\mathcal{U}(\xi)$, an associated probability distribution can be constructed that respects the contours of the function relative to its global minimum as:

$$p(\xi;\alpha) \propto \exp(-\alpha\mathcal{U}(\xi)) \tag{22}$$

This distribution reflects the cost-tradeoffs among hypotheses by assigning high probability to low cost trajectories and low probability to high cost trajectories. Here, $\alpha > 0$ adjusts how flat the distribution is going to be: as $\alpha \to 0$, the distribution becomes increasingly uniform, and as $\alpha \to \infty$, the distribution becomes increasingly peaked around the global minimum.

HMC uses a combination of gradients and momenta to efficiently search through the space of trajectories and explicitly sample from the distribution $p(\xi;\alpha)$. To understand how it works, imagine the graph of the scaled objective $\alpha\mathcal{U}(\xi)$ as a landscape. Throughout this section, ξ is considered to be a single point in an infinite-dimensional space; momenta and dynamics act on this point within this infinite-dimensional space analogously to their behavior on a small ball in the more familiar three-dimensional space of everyday life. This ball can start rolling from any point in this space with a particular initial velocity, and, in a perfect frictionless world, the ball will continue rolling forever with constant total energy. At times the ball will transfer that energy entirely to potential energy by pushing uphill into higher cost regions, and at times the ball will convert much of its energy into kinetic energy by plummeting down into local minima, but the principle of energy conservation dictates that its total energy always remain constant.

The total energy of the system, accordingly, is a function of two things: how high the ball starts, and how fast it starts. By changing the amount of total energy the ball starts with, the amount of time the ball spends moving quickly through local minima is modulated relative to the amount of time it spends pushing up into higher cost regions. If it is set off with very little energy, either by starting it low or giving it very little initial momentum, the ball will easily get stuck in some local bowl and not have enough energy to escape. On the other hand, if it is set off will a lot of energy, either by placing it very high in the beginning or by throwing it very hard in some direction, the ball will have enough energy to shoot out of local minima and visit many distinct regions as it travels.

The following sections formalize these ideas as HMC, in terms of both its use as a sampler and as an optimizer using simulated annealing.

2) Hamiltonian Monte Carlo: Let γ denote a variable that represents the momentum of a trajectory ξ. The trajectory is considered as a single infinite-dimensional point in the space of trajectory functions. Accordingly, the momentum refers to how the entire trajectory changes over time. Following the analogy from above, the energy of the system is defined by both its potential energy $U(\xi)$ and its kinetic energy $K(\gamma) = \frac{1}{2}\gamma'\gamma$. Rather than addressing the marginal $p(\xi)$ directly, the Hamiltonian Monte Carlo operates on the joint probability distribution between the trajectory variable ξ and its momentum γ:

$$p(\xi,\gamma) \propto \exp(-\mathcal{U}(\xi) - K(\gamma)) = \exp(-H(\xi,\gamma)) \tag{23}$$

In this way, the probability of any given system configuration is related to its total energy. Low energy configurations have high probability while high energy configurations have low probability. An algorithm that successfully samples from the joint distribution also implicitly gives us samples from the marginal since the two random variables are independent. Simply throwing away the momentum samples leaves us with samples from the desired marginal $p(\xi)$. In physics, $\mathcal{U}(\xi)$ and $K(\gamma)$ are the potential energy and kinetic energy, respectively, and the combined function $H(\xi,\gamma)$ is known as the Hamiltonian of the dynamical system. $H(\xi,\gamma)$ reports the total energy of the system, and since energy is conserved in a closed system physically simulating the system is central to the algorithm. Physical simulations move the system from one infinite-dimensional point ξ_t to another ξ_{t+1}, the latter likely in a very different region of the space, without a significantly changing the total energy H. Any observed change stems solely from errors in numerical integration. The following system of first-order differential equations models the system dynamics:

$$\begin{cases} \frac{d\xi}{dt} = \gamma \\ \frac{d\gamma}{dt} = -\nabla \bar{U}(\xi) \end{cases} \tag{24}$$

These equations are referred to as the instantaneous update equations. Considering ξ as a particle with momentum γ, this system simply restates the physical principles that a particle's change in position is given by its momentum, and its change in momentum is governed by the force from the potential field, which have been defined as $\mathcal{U}(\xi)$ in the problem. An analysis of this system demonstrates that all integral curves conserve total energy. This observation indicates that if $(\xi(t), \gamma(t))$ is a solution to the system above, the value of the Hamiltonian $H(\xi(t), \gamma(t))$ is always the same independent of t. In terms of the joint distribution in Equation 23, this constancy along solution paths implies these system solutions also trace out iso-contours of the Hamiltonian $H(\xi, \gamma)$. Simulating the Hamiltonian dynamics of the system, therefore, moves from one point $(\xi(0), \gamma(0))$ to another point $(\xi(t), \gamma(t))$, where $\xi(0)$ and $\xi(t)$ may be very different from one another, without significantly changing the probability $p(\xi, \gamma) \propto \exp(-H(\xi, \gamma))$. The Hamiltonian Monte Carlo algorithm capitalizes on the system's conservation of total energy by leveraging the decomposition $p(\xi, \gamma) \propto exp(-\mathcal{U}(\xi))exp(-K(\gamma))$. If an exact simulation of the dynamical system exactly, then the following sampling procedure would be exact: (1) sample the momentum term from the Gaussian $p(\gamma) \propto exp(-\frac{1}{2}\gamma^T\gamma)$; and (2) simulate the system for a random number of iterations from its current ξ_t to get the new sample ξ_{t+1}. However, this is not possible due to numerical inaccuracies in approximate integration that may play a significant role. Thus, the Hamiltonian Monte Carlo algorithm is essentially the above procedure with an added rejection step to compensate for the lost accuracy from numerical integration. Most presentations of Hamiltonian Monte Carlo use the second-order leapfrog method of numerical integration to simulate the system dynamics because of its relative simplicity and its reversibility property (running it forward for T iterations and then backward for T iterations gets you back where you started, up to floating point precision), the latter of which is of theoretical importance for Markov Chain Monte Carlo algorithms. The leapfrog method updates are given

$$\begin{cases} \gamma_{t+\frac{\epsilon}{2}} = \gamma_t - \frac{\epsilon}{2}\nabla\bar{\mathcal{U}}(\xi_t) \\ \xi_{t+\epsilon} = \xi_t + \epsilon\gamma_{t+\frac{\epsilon}{2}} \\ \gamma_{t+\epsilon} = \gamma_{t+\frac{\epsilon}{2}} = \gamma_{t+\frac{\epsilon}{2}} - \frac{\epsilon}{2}\nabla\bar{\mathcal{U}}(\xi_{t+\epsilon}) \end{cases} \tag{25}$$

The full numerically robust Hamiltonian Monte Carlo sampling algorithm therefore iterates the following three steps:

- Sample an initial trajectory momentum γ: Sample a random initial trajectory momentum from the Gaussian formed by the marginal kinetic energy term $p(\gamma) \propto \exp(-K(\gamma)) = exp(\frac{-1}{2}\gamma^T\gamma)$
- Simulate the system: Simulate the system for a random number of iterations using the leapfrog method of numerical integration given in Equation 25.
- Reject: Compensate for errors in numerical integration. If the final point is more probable than the initial point, accept it without question. Otherwise, reject it with probability $p(\xi_{t+1}, \gamma_{t+1})/p(\xi_t, \gamma_t)$.

3) Theoretical Considerations: The authors briefly explore some of the theoretical considerations describing both why the instantaneous update rule is correct and how to properly account for arbitrary constant metrics A on the space of trajectories.

By analyzing the time derivative of H, one can see that the instantaneous update rule in System 24 does not change the Hamiltonian $H(\xi, \gamma)$:

$$\begin{aligned} \frac{d}{dt}H(\xi,\gamma) &= \sum_{i=1}^{d}(\frac{d\xi_i}{dt}\frac{\partial H}{\partial \xi_i} + \frac{d\gamma_i}{dt}\frac{\partial H}{\partial \gamma_i}) \\ &= \sum_{i=1}^{d}(\frac{d\xi_i}{dt}\frac{\partial E}{\partial \xi_i} + \frac{d\gamma_i}{dt}\gamma_i) \\ &= \sum_{i=1}^{d}(\gamma_i\frac{\partial E}{\partial \xi_i} - \frac{\partial E}{\partial \xi_i}d\gamma_i) = 0 \end{aligned} \tag{26}$$

Given the covariant update rule with metric A, one might suspect that the following system is the analogous instantaneous HMC update rule under a constant metric A:

$$\begin{cases} \frac{d\xi}{dt} = \gamma \\ \frac{\gamma}{dt} = -A^{-1}\nabla\bar{\mathcal{U}}(\xi) \end{cases} \tag{27}$$

The system is covariant using a simple change of variable argument. Using $H_A(\xi, \gamma) = E(\xi) + \frac{1}{2}\gamma^T A\gamma$ as the Hamiltonian, let $\tilde{\xi} = A^{\frac{1}{2}}\xi$ and $\tilde{\gamma} = A^{\frac{1}{2}}\gamma$ be a change of variable transformation such that Euclidean inner products in $\tilde{\xi}$ and $\tilde{\gamma}$ are A-inner products in the original space. The Euclidean update rules given by Equation 24 of this modified Hamiltonian in the transformed space are $\frac{d\tilde{\xi}}{dt} = \tilde{\gamma}$ and $\frac{d\tilde{\gamma}}{dt} = -\bar{\nabla}_{\tilde{\xi}}\mathcal{U}(A^{-\frac{1}{2}}\tilde{\xi})$. By substituting the identities

$$\begin{gathered} \frac{d\tilde{\xi}}{dt} = A^{\frac{1}{2}}\frac{d\xi}{dt}, \\ \frac{d\tilde{\gamma}}{dt} = A^{\frac{1}{2}}\frac{d\gamma}{dt}, \\ \bar{\nabla}_{\tilde{\xi}}\mathcal{U}(\xi) = A^{-\frac{1}{2}}\bar{\nabla}_{\xi}\mathcal{U}(\xi), \end{gathered} \tag{28}$$

into the covariant system 27 reduces it to the Euclidean system in terms of $\tilde{\xi}$ and $\tilde{\gamma}$, Hamiltonian Monte Carlo prescribes Euclidean sampling of the momenta. So, in terms of the original variables ξ and γ, since $\frac{1}{2}\tilde{\gamma}^T A\gamma$, the analogous rule under H_A is to sample from $p_A(\xi) \propto exp(-\frac{1}{2}\gamma^T A\gamma)$. This alteration is intuitive since this distribution p_A gives high probability to smooth momenta trajectories and low probability to non-smooth trajectories.

The final altered algorithm reads:

- **Sample an initial momentum:** Sample γ_t from $p_A(\gamma) \propto exp\frac{1}{2}\gamma^T A\gamma$.
- **Simulate the system:** Use to leapfrog method to simulate the system in Equation 27
- **Reject:** If the final point is more probable than the initial point, accept it without question. Otherwise, reject it with probability $p_A(\xi_{t+1}, \gamma_{t+1})/p_A(\xi_t, \gamma_t)$.

4) Optimization and simulated annealing: Simulated annealing can be used to turn these efficient gradient-based sampling algorithms into optimization procedures that better explore the space of trajectories to avoid bad local minima. *Simulated annealing* builds off the observation that the family of distributions $p(\xi, \gamma; \alpha) \propto \exp(-\alpha H(\xi, \gamma))$ ranges from the uniform distribution at $\alpha = 0$ to the true distribution at $\alpha = 1$ and then toward a distribution increasingly peaked around the global minimum as $\alpha = \infty$. As α increases, sampling from the distribution should sample increasingly close to the global minimum. Generally, the larger α becomes, the regions around the local / global minima become narrower which generally increases the burn-in time of the sampler. However, $simulated annealing$ circumvents these long burn-in periods by stepping from $\alpha = 0$ toward larger α's in small steps to coax the samples toward the minima incrementally over time. Effective practical variants on this robust optimization procedure may relax theoretical rigor in lieu of a simple strategy for combining momentum-based updates with periodic perturbations to the momentum to efficiently skip past local minima and quickly converge on a globally strong solution.

IV. Results

A. Experimental Results on Manipulator Planning Tasks

CHOMP's performance on common motion planning problems for a typical robotic manipulator have been evaluated. Such problems are posed in a high-dimensional state space, but the distribution of obstacles in the robot's workspace is structured.

1) Experimental Design: Day-to-day manipulation task of grasping in cluttered environments have been explored. For the majority of the experiments, a canonical grasping in clutter problem is used: the robot is tasked with moving to grasp a bottle placed on a table among a varying number of obstacles. The following hypothesis have been tested:

H1: CHOMP can solve many structured, day-to-day manipulation tasks, and it can do so very fast.

H2: For a large number of structured, day-to-day manipulation tasks, CHOMP obtains a feasible, low-cost trajectory in the same time that an RRT obtains a feasible, high-cost trajectory.

CHOMP, RRT and RRT* have been compared here. To ensure fairness of the comparison, experiments were conducted in a standard simulation environment – OpenRAVE version 0.6.4, and use the standard implementations for RRT (bi-directional RRT-Connect) and RRT* from the Open Motion Planning Library (OMPL) version 0.10.2. HERB, the Home Exploring Robot Butler, was used as the experimental platform. Each algorithm is run on each problem 20 times, for 20 seconds each. The RRT shortcuts the path (by the shortcutting method in OpenRAVE) until the final time is reached. At each time point the algorithm is checked if it has found a feasible solution, and this is used in the path length for cost to ensure a fair comparison, biased towards the randomized planners: this is the cost that the RRT shortcutting method optimizes, but not directly the cost that CHOMP optimizes. Instead, CHOMP minimizes sum squared velocities, while pulling the trajectory far from obstacles.

Grasping in clutter problems were created with varying features: starting configurations, target locations and clutter distributions. The problems were divided into a training and testing set, such that no testing problem has any common features with a training one. This is important, since it tests true generalization of the parameters to different problems. The training set was used to adjust parameters for all algorithms, giving each the best shot at performing well. With 170 testing problems, leading to 3400 runs of each algorithm.

2) Time to Produce a Feasible Solution: Validating H1, CHOMP succeeded on about 80% of the problems, with an average time of 0.34s (SEM = 0.0174). On problems where both CHOMP and RRT both succeed, CHOMP found a solution that was 2.6 seconds faster, and the difference is quite significant (as indicated by a paired t-test, t(2586) = 49.08, p $<$ 0.001). CHOMP has a lower success rate than an RRT on these problems. **When it does succeed, it does so faster.**

The CHOMP times do not include the time to compute the Signed Distance Field from the voxelized world. The SDF computation takes an average of 0.1 seconds.

3) Collision Checking - The Grain of Salt: The time taken by the RRT highly depends on the time it takes to perform collision checks. This implementation uses OpenRAVE for collision checking, and the average time for a check was approximately 444 microseconds which is faster than the times reported in the benchmark comparison for an easier problem, indicating that the results for the RRT are indicative of its typical performance. The RRT may be improved with recent, more advanced collision checkers. Example, if collision checking were 5 times faster, the difference in success rate would be much higher in favor of the RRT, and the planning time when both algorithms succeed would become comparable, with an estimated average difference of only 0.2s in favor of CHOMP. **H2**, as will be see in following section, would remain valid: CHOMP produces a low-cost feasible trajectory in the same time that an RRT produces a high-cost feasible trajectory.

4) Cost and Feasibility Comparison when the RRT Returns its First Solution: 3067 of the 3400 RRT runs yielded feasible trajectories. For every successful RRT run, a CHOMP trajectory from the same problem at the time point when the RRT obtained a solution was retrieved. In 78% of the cases, the CHOMP trajectory was feasible, and its path length was on average 57% lower. This difference in path length was indeed significant (t(2401) = 65.67, $p < 0.001$): in 78% of the cases, in the same time taken by an RRT to produce a feasible solution, CHOMP can produce a feasible solution with significantly lower cost (**H2**). Note that the CHOMP trajectories evaluated here were not the ones with the smallest path length: the algorithm optimizes a combination of a smoothness and an obstacle cost function. Therefore, CHOMP is increasing the path length even after the trajectory becomes feasible, in order to keep it far from obstacles.

5) Time Budgets: Planners are often evaluated within fixed time budgets. Each planner is allowed a fixed planning time, and it's result is evaluated. It was found that the relative performance of CHOMP and RRT depends greatly on the time budget allotted. In case of CHOMP, iterations were run until such time that a final full-trajectory collision check will finish before the given budget; the check is then performed, and the result is reported. Note that a CHOMP trajectory can oscillate between feasible and infeasible during optimization; it may be the case that an infeasible CHOMP result was in fact feasible at an earlier time, but the algorithm is unaware of this because it only performs the expensive collision check right before it returns. For the RRT, we simply stop planning or shortcutting at the end of the budget. Time budgets were evaluated of 1, 2, 3, 5, 10, and 20 s.

The results illustrate the differences between the planners. For short time budgets (< 5 s), the deterministic CHOMP has a higher success rate than the RRT; however, it flattens quickly, and does not exceed 75% for any budget. The RRT continues to improve, with a 90.2% success rate within the longest budget. Across all feasible solutions for all budgets, CHOMP significantly outperforms the RRT when evaluated by path length.

6) HMC: The deterministic CHOMP algorithm resulted in a feasible path after a 20 s time budget with a 74.7% success rate. A further 5.9% of problems yielded no successful paths at any point during the runs of any of the planners tested, and are thus classified as unsolvable here. The Hamiltonian Monte Carlo component of CHOMP aims to improve performance on the remaining 19.7% of the cases. HMC was implemented by adding a trajectory momentum term to the optimizer as described. The momentum of the trajectory was repeatedly resampled after a random number of iterations given by an exponential distribution with parameter $\lambda_{\text{exp}} = 0.02$. Momenta were sampled from the distribution $p(\gamma) \propto \exp(-\frac{1}{2}\alpha\gamma^T\gamma)$, with the parameter α increasing exponentially as $\alpha = 100\exp(0.02k)$ with k the iteration number. On those problems in which CHOMP failed, the addition of HMC resulted in a 56% success rate during the 20 s time budget.

7) RRT:* The performance of CHOMP and Bi-Directional RRT-Connect was compared to the RRT* implementation in OMPL. The RRT* range parameter was set equal to that of the RRT. Other algorithm parameters were also chosen as directed by the implementation documentation. RRT* had a 5.97% success rate on testing suite of clutter problems. When it succeeded, it found its first feasible solution after an average of 6.34 s, and produced an average path length of 11.64 rad.

B. Beyond Grasping in Clutter

The experiments so far focused on grasping an object surrounded by clutter. But how does CHOMP perform on more complex tasks, defined by narrow spaces? To explore this question, the algorithm's performance was investigated on the problem setup: The start and the goal for a complex problem of reaching into the back of a narrow microwave. The robot is close to the corner of the room, which makes the problem particularly challenging because it gives the arm very little space to move through. Running CHOMP and BiRRT-Connect for 8 different scenarios, CHOMP was able to solve 7 of the 8 scenarios, taking an average of 1.04 seconds. The RRT had a total success rate of 67.1%, taking an average of 63.36 seconds to first-feasible when it succeeds. On the problem for which CHOMP failed, the RRT had a 10% success rate. A collision check here took an average of 2023 microseconds.

C. Fast Replanning Configuration

Simulated and real-robot experimental results are presented in the context of efficient re-planning due to unforeseen changes in the environment or moving obstacles. Uptil now, configuration of the CHOMP algorithm with pre-computed compositional distance fields and weight scheduling is done. This configuration, particularly well suited for re-planning, is henceforth referred to as CHOMP-R.

In comparison to the original CHOMP, a simulated model of a manipulator robot that operates in a set of scenes that include dense clutter due to a collection of 50 randomly placed box obstacles of random size is utilized. Ten different scenes were generated and 100 planning queries were performed by selecting random initial and final arm configurations. The baseline implementation was executed on these planning queries, and the results were considered the control group. The methods described in this paper were then attempted also, and the results were compared to this control group. Three different planner configurations were compared to CHOMP:

- The standard signed distance field was replaced with the compositional distance field.
- The cost function weights were scheduled.
- CHOMP-R: both object distance primitives and weight scheduling were combined.

D. Experiments on a Robotic Arm

This section presents experimental results for the implementation of CHOMP on Barrett Technology's WAM. Zucker et al. demonstrate the efficacy of their technique on a set of tasks representative of the type of tasks that may be commonly encountered in a home manipulation setting.

1) Collision heuristic: However, the home setting is different from the settings encountered in legged locomotion in that the obstacles are often thin. While initially, they used a heuristic based on the signed distance field under which the obstacles themselves specify how the robot should remove itself from collision which worked well when the obstacle is a large body, the heuristic was found to fail for smaller obstacles. An initial straight-line trajectory through the configuration space often contains configurations that pass entirely through an obstacle. In that case, the naive workspace potential works tends to simultaneously push the robot out of collision on one side, but also, to pull the robot further through the obstacle on the other side.

To avoid this behavior, an indicator function was added to the objective function that makes all workspace terms that appear after the first collision along the arm vanish. This indicator factor can be written mathematically as $I(\min_{j\leq i} d(x_j(q))$, and implemented simply by ignoring all terms after the first collision while iterating from the base of the body out toward the end effector for a given time step along the trajectory.

Ultimately, this heuristic suggests that the workspace gradients encountered after the first collision of a given configuration are invalid and should therefore be ignored. Because the base of the robotic arm is always collision free, the region along the arm prior to the first collision will work to pull the rest of the arm out of collision. In the experiments conducted by Zucker et al., this heuristic works well to pull the trajectory free of obstacles commonly encountered in the home environment.

2) Experimental Results: To evaluate the efficacy of CHOMP and its probabilistic variants as a replacement for planning on a variety of everyday household manipulation problems, 15 different configurations were chosen in a given scene representing various tasks such as picking up an object from the table, placing an object on a shelf, or pulling an item from a cupboard. Using these start/goal points, 105 planning problem were generated consisting of planning between all pairs of end configurations. In this implementation, each link of the robot arm was modeled as a straight line, which was subsequently discretized into 10 evenly spaced points to numerically approximate the integrals over u in fobs. The voxel space used was a discretization of 50 × 50 × 50, and we used the MATLAB's $bwdist$ for the distance field computation. Under this resolution, the average distance field computation time was about .8 seconds. For each problem, CHOMP was run for 400 iterations (approximately 12 seconds), although the core of the optimization typically completed within the first 100 iterations (approximately 3 seconds). However, we made little effort to make our code efficient; we stress that our algorithm is performing essentially the same amount of work as the smoother of a two stage planner, without the need for the initial planning phase. CHOMP successfully found collision-free trajectories for 99 of the 105 problem. We additionally compared the performance of CHOMP when initialized to a straight-line trajectory through configuration space to its performance when initialized to the solution of a bi-directional RRT. Surprisingly, when CHOMP successfully finds a collision free trajectory, straight-line initialization typically outperforms RRT initialization. On average, excluding those problems that CHOMP could not solve, the log-objective value achieved when starting from a straight-line trajectory was approximately .5 units smaller than than achieved when starting from the RRT solution on a scale that typically ranged from 17 to 24. This difference amounts for approximately 3% of the entire log-objective range spanned during optimization.

Figure 5 depicts an example of the objective progressions induced by each of these initialization strategies. We note that in our experiments, setting $A = I$ and performing Euclidean gradient descent performed extremely poorly. Euclidean gradient descent was unable to successfully pull the trajectory free from the obstacles.

E. Implementation on a Quadruped Robot

In this section, we discuss the CHOMP implementation on the Quadruped LittleDog robot built by Boston Dynamics Inc. which was designed to traverse over standardized terrains quickly and robustly. The approach to robotic legged locomotion is decomposed the problem into a footstep planner which informs the robot where to place its feet as it traverses the terrain, and a footstep controller which generates full-body trajectories to realize the planned footsteps. CHOMP was used as a critical component of the footstep controller. Footsteps for the LittleDog robot consist of a stance phase, where all four feet have ground contact, and a swing phase, where the swing leg is moved to the next support location. During both phases, the robot can independently control all six degrees of trunk position and orientation via the supporting feet. Additionally, during the swing phase, the three degrees of freedom for the swing leg may be controlled. For a given footstep, CHOMP was run as coordinate descent, alternating between first optimizing the trunk trajectory ξ_T given the current swing leg trajectory ξ_S, and subsequently optimizing ξ_S given the current ξ_T on each iteration. The initial trunk trajectory is given by a Zero Moment Point (ZMP) preview controller and the initial swing leg trajectory is generated by interpolation through a collection of knot points intended to guide the swing foot a specified distance above the convex hull of the terrain.

To construct the signed distance fields (SDF) representation of the terrain, Zucker et al. first scan-converted triangle mesh models of the terrain into a discrete grid representation. To determine whether a grid sample lies inside the terrain, they shot a ray through the terrain and use the even/odd rule. When running CHOMP with LittleDog, domain knowledge was exploited by adding a prior to the workspace potential function $c(x)$. The prior was defined as penalizing the distance below some known obstacle-free height when the swing leg is in collision with the terrain. Its effect in practice was to add a small gradient term that sends colliding points of the robot upwards regardless of the gradient of the SDF. For the trunk trajectory, in addition to the workspace obstacle potential, the objective function included terms which penalize kinematic reachability errors (which occur when the desired stance foot locations are not reachable given desired trunk pose) and which penalize instability resulting from the computed ZMP straying towards the edges of the supporting polygon. Penalties from the additional objective function terms were also multiplied through A^{-1} when applying the gradient, just as the workspace potential was. CHOMP was represented as an exponential map vector corresponding to a differential rotation with respect to the "mean orientation" of the trajectory. The exponential map encodes an (axis, angle) rotation as a single vector in the direction of the rotation axis whose magnitude is the rotation angle. The mean orientation is computed as the orientation halfway between the initial and final orientation of the trunk for the footstep.

Timing for the footstep is decided by a heuristic which is evaluated before the CHOMP algorithm is run. Typical footstep durations run between 0.6 s and 1.2 s. The trajectories were discretized at the LittleDog host computer control cycle frequency, which is 100 Hz. Currently, trajectories are pregenerated before execution because in the worst-case, optimization can take slightly longer (by a factor of about 1.5) than execution to return a collision free trajectory; however, CHOMP typically would converge after less than 0.5s. It was found that the initial trajectory for the footstep is not always feasible; however, the CHOMP algorithm is almost always able to find a collision-free final trajectory, even when the initial trajectory contains many collisions.

V. Research Discussion

A. Prior Research Comparison

1) Sampling-Based Planning: Sampling-Based Planning has become popular in the domain of high-dimensional motion planning, including manipulation planning. The Probabilistic Roadmap (PRM) and Expansive Space Trees, works by Barraquand and Latombe were well-suited for path planning in $\mathcal{C}$-spaces with multiple degrees of freedom [5]. Rapidly-exploring Random Trees (RRT) were also applied to such constraints, and proved to be successful for general high-dimensional planning. These approaches typically work in a two-steps: first a collision-free path is discovered without any consideration of the cost function, and then it is improved by applying certain heuristics. One popular approach is the so-called "shortcut" heuristic, which picks pairs of configurations along the path and invokes a local planner to attempt to replace the intervening sub-path with a shorter one. Methods such as medial axis and "partial" shortcuts have also proven effective. However, they do not guarantee optimality of the planned path nor the existence of one, and even when it does it will result in jerky, non-smooth motion trajectories, requiring post-processing or smoothing. The CHOMP algorithm does not always guarantee convergence, but however will generally find a low-cost solution faster than RRT or RRT* as shown in the experimental results with the robotic arm. The performance and optimization of the CHOMP algorithm can also be improved by leveraging these complementary research efforts by acting as a post-processor of the sample-based trajectories.

2) Resolution Complete Approaches: The application of the A* algorithm for high-dimensional problems proved unsuitable. A major problem with the A* was that the heuristics resulted in the examination of large portions of the configuration space. Likhachev et al. presented an effected method to update the A* algorithm while smoothly reducing heuristic inflation allowing resolution complete search for a broader variety of problems [6]. The discretization choices made by these approaches presents a clear contrast with the CHOMP algorithm. While resolution complete approaches necessarily discretize states, actions, and time, CHOMP allows states to vary continuously. This is a key benefit insofar as gradient information from the workspace can be used to continuously deform trajectories around obstacles, and trajectory smoothness is directly related to the resolution of the underlying state and action discretizations in the resolution complete setting. The authors however do stress at multiple points in the paper that CHOMP is not suitable for all tasks. Difficult problems featuring bottlenecks and other such "narrow passages" may indeed require complete planners; however, they believe that CHOMP is well suited for solving the types of problems described in their experimental designs.

3) Trajectory Optimization: Despite a rich theoretical history and successful applications, most notably in the control of spacecraft and rockets, trajectory optimization techniques have had limited success in motion planning. Much of this may be attributed to two causes: the large computational cost for evaluating objective functions and their higher order derivatives in high-dimensional spaces, and the presence of local minima when considering motion planning as a (generally non-convex) continuous optimization problem. Khatib pioneered the application of the artificial potential field for real-time obstacle avoidance [7]. The method's sensitivity to local minima has been addressed in a range of related work. Analytical navigation functions that are free of local minima have been proposed for some specific environments by Rimon and Koditschek [8]. Quinlan and Khatib further extended numerical methods in motion replanning by modeling the trajectory as a mass-spring system, an elastic band, and performing replanning by scanning back and forth along the elastic, while moving one mass particle at a time [9]. The method of Stochastic Trajectory Optimization for Motion Planning (STOMP) by Kalakrishnan et al. obtains gradient information using sample trajectories [10], whereas CHOMP exploits the availability of the gradient information. The use of functional gradients have also been studied. Possible advantage of CHOMP over many of these methods may be in its comparative computational speed and use of HMC to help improve performance in light of the sensitivity to local minima issues that other methods like artificial potential fields are affected by.

4) Objective Learning: Dalibard and Laumond [11] use PCA to discover promising directions for exploring an RRT. Vernaza and Lee [12] propose a method, similar to block-coordinate descent in finite-dimensional optimization, that optimizes a trajectory over a lower-dimensional subspace while leaving other dimensions intact. While our approach also lends itself well to learning techniques by virtue of its formal mapping between planner parameters and its performance, it does not make any assumptions on the structure of the objective function and therefore can perform well even in the cases where other techniques that rely on structure would not be readily applicable. In other publications using CHOMP, research has been done to demonstrate how the algorithm can exploit the parameter-performance mapping to train the cost function in a manner reminiscent of imitation learning techniques [13].

5) Optimal Control: CHOMP can also be related to optimal control of robotic systems. Instead of focusing on computing feasible paths, the goal is to construct trajectories which optimize over a variety of dynamic and task-based criteria.

Differential dynamic programming is a popular approach in this category and has been applied to a number of complex systems [14]. The techniques presented in this paper however may enable such methods to naturally consider obstacle avoidance as part of optimizing the control. Also by virtue of avoiding the explicit computation of second-order dynamics, one may argue that CHOMP is likely to be more efficient in many relevant scenarios.

With some similarity to the approach stated above, ILQG [15] uses quadratic approximations to the objective function and is able to incorporate nonlinear control constraints.However, many high-dimensional planning problems, such as in manipulation, pose computational challenges to such methods. Moreover, ILQG and similar approaches design an affine feedback control law, whereas CHOMP owes much of its runtime efficiency to producing a single trajectory that satisfies constraints.

Other optimal control approaches [16] require a description of configuration space obstacles, which is problematic in the context of high-dimensional planning problems, whereas, CHOMP demonstrates attractive computational efficiency and convergence even in cluttered obstacle environments.

B. Research Critique

In analyzing and reviewing this paper, we noted the amount of detail and consideration involved in the derivation and formulation of the CHOMP algorithm. From basic principles of physics, fundamental optimization techniques, to computational efficiency, the authors were technically correct and well thought out in developing this method. From analyzing their extensive results and testing on multiple robotic platforms across different applications, we believe that the CHOMP algorithm proposed is feasible for practical implementation, under the use cases they presented.

However, in terms of being a ideal, general-purpose motion planning implementation, CHOMP does have its limitations as noted by the authors. CHOMP inherits the problems of optimization in high-dimensional spaces on non-convex cost func- tions. Finding the global optimum may be intractable, and the optimizer can converge to high-cost local minima. Complex problems (described by narrow passages) are difficult to solve if the optimizer is initialized with a trajectory far from a collision free solution. This high-cost minima issue can be alleviated to a certain extent through exploration via HMC, which makes CHOMP probabilistically complete). Another method they propose is to remedy this is through learning from previous experiences to initialize the optimizer in a good basin of attraction.

Another issue we noted was that while the general premise behind the derivation of their objective functional was reasonable, the current cost may in fact not be specific enough or difficult to tune to the user's preference, leading potentially to varying levels of performance compared to expectation. For example, a trajectory that is in collision briefly but stays far away from obstacles on average can have lower cost than a trajectory that never collides, but stays close to obstacles. This cost function also does not necessarily create trajectories that are what a human observer would want or expect from the robot, and in fact different observers have different expectations. CHOMP also depends on several parameters, which require tuning, like the trade-off between obstacle and smoothness cost or the step size schedule. Very little of parameter tuning methodology was described and thus it could be difficult to replicate results. Gradient step size constraints (such as Armijo's Rule) as well as obstacle and smoothness cost parameter searching or optimization perhaps could have been described. Without correct tuning, it is possible that CHOMP will not perform to the originally set metrics of obstacle avoidance and non-jerky trajectory planning.

An additional limitation relates to handling constraints. While additional research into CHOMP has shown that it can handle trajectory-wide constraints, hard constraints can be time-consuming (they require matrix inversion at every iteration), and soft constraints (in the form of penalty functions) which lead to poor solutions when the penalty and the cost have opposing gradients.

Finally, one thing we noticed was that the authors don not characterize what level of performance to expect for different types of problems of varying difficulty. Without this, it is difficult to say whether this method is indeed superior to other trajectory planning methods (especially such as the now popular Stochastic Trajectory Optimization for Motion Planning (STOMP) or other optimization based methods) and it would be interesting to see a direct comparison between CHOMP and these methods on the same task.

VI. Conclusion

This work presents a powerful new trajectory optimization procedure that solves a much wider range of problems than previous optimizers, including many to which randomized planners are traditionally applied. The key concepts that contribute to the success of CHOMP all stem from utilizing superior notions of geometry. The experiments in this paper show that this algorithm substantially outperforms alternatives such as RRT* and improves performance on real world robotic systems.

There are a number of important issues that have not been addressed in this paper. First, in choosing a priori a discretization of a particular length, the optimizer is effectively being constrained to consider only trajectories of a predefined duration. A more general tool should dynamically add and remove samples during optimization. The discretization-free functional representation discussed in this review will provide a theoretically sound avenue through which trajectories of differing time lengths can be accommodated.

Additionally, while the HMC algorithm provides a well founded means of adding randomization during optimization, there are still a number of problems under which this technique flounders in local minima under finite time constraints.

With regards to future work, ways could be considered in which these optimization concepts can be more fundamentally integrated into a practical complete planning framework.

Finally, the algorithm proposed is promising in that it is amenable to new machine learning techniques. Most randomized planners are unsuitable for well formulated learning algorithms because it is difficult to formalize the mapping between planner parameters and planner performance. As seen in this paper, CHOMP can perform well in many areas previously believed to require complete planning algorithms; since the algorithm explicitly specifies its optimization criteria, a learner can exploit this connection to more easily train the cost function in a manner reminiscent of recent imitation learning techniques. This connection can be explored in detail as future work.

Acknowledgment

Many thanks to Professor Kolmanovsky and Nan Li for teaching AEROSP 575 this semester!

References

[1] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, J.A. Bagnell, S.S. Srinivasa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning " International Journal of Robotics Research, May 2013.

[2] N. Ratliff, M. Zucker, J. A. Bagnell, S. Srinivasa, "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning", 2009 IEEE International Conference on Robotics and Automation, Kobe International Conference Center, Kobe, Japan, May 12-17, 2009.

[3] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. Technical Report TR2004-1963, Cornell University, 2004.

[4] Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation, 4(2):193–203, 1988. doi: 10.1109/56.2083.

[5] Jerome Barraquand and Jean-Claude Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. Proc. of the IEEE International Conference on Robotics and Automation, pages 1712–1717, 1990.

[6] B.J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In IEEE International Conference on Robotics and Automation, pages 2902–2908. IEEE, 2010.

[7] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1):90–98, 1986.

[8] Elon Rimon and Daniel E. Koditschek. Exact robot navigation using cost functions: the case of distinct spherical boundaries in En. In Proc. of the IEEE International Conference on Robotics and Automation, pages 1791–1796, 1988. doi: 10.1109/ROBOT.1988.12325.

[9] Sean Quinlan and Oussama Khatib. Elastic Bands: connecting path planning and control. In Proc. Conf. IEEE Int Robotics and Automation, pages 802–807, 1993. doi: 10.1109/ROBOT.1993.291936.

[10] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In Proc. IEEE Int Robotics and Automation (ICRA) Conf, pages 4569–4574, 2011. doi: 10.1109/ICRA.2011.5980280.

[11] S. Dalibard and J. Laumond. Control of probabilistic diffusion in motion planning. In Proceedings of the International Workshop on Algorithmic Foundations of Robotics, pages 467–481, 2009.

[12] P. Vernaza and D. Lee. Learning dimensional descent for optimal motion planning in high- dimensional spaces. In Proc. of the Association for the Advancement of Artificial Intelligence (AAAI) Conference, 2011.

[13] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt. Boosting structured prediction for imitation learning. In Workshop on Neural Information Processing Systems, 2006.

[14] David H. Mayne and David Q. Jacobson. Differential dynamic programming. New York: American Elsevier Pub. Co., 1970.

[15] E. Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In Proceedings of the American Control Conference, 2005.

[16] Zvi Shiller and Steven Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. IEEE Transactions on Robotics and Automation, 6(7):785–797, 1991.