CHOMP: Covariant Hamiltonian Optimization for Motion Planning

Matt Zucker[1], Nathan Ratliff[2], Anca D. Dragan[3],
Mihail Pivtoraiko[4], Matthew Klingensmith[3], Christopher M. Dellin[3],
J. Andrew Bagnell[3], Siddhartha S. Srinivasa[3]

[1] Department of Engineering
Swarthmore College
Swarthmore PA
mzucker1@swarthmore.edu

[2] Google, Inc.
Pittsburgh PA
ratliffn@google.com

[3] The Robotics Institute
Carnegie Mellon University
Pittsburgh PA
adragan@cs.cmu.edu
mklingen@andrew.cmu.edu
cdellin@cs.cmu.edu
{*dbagnell, siddh*}*@cs.cmu.edu*

[4] Dept. of Mechanical Engineering
and Applied Mechanics
University of Pennsylvania
Philadelphia PA
mihailp@seas.upenn.edu

Abstract

In this paper, we present CHOMP (Covariant Hamiltonian Optimization for Motion Planning), a method for trajectory optimization invariant to reparametrization. CHOMP uses functional gradient techniques to iteratively improve the quality of an initial trajectory, optimizing a functional 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 (and for probabilistic completeness), and is capable of respecting hard constraints along the trajectory. We present extensive experiments with CHOMP on manipulation and locomotion tasks, using 7-DOF manipulators and a rough-terrain quadruped robot.

1 Introduction

We propose a trajectory optimization technique for motion planning in high-dimensional spaces. The key motivation for trajectory optimization is the focus on producing *optimal motion* — incorporating dynamics, smoothness, and obstacle avoidance in a mathematically precise objective. 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.

Our algorithm CHOMP, short for *Covariant Hamiltonian Optimization for Motion Planning*, is a simple variational strategy for achieving good trajectories. This approach to motion planning builds on two central tenets:

- **Gradient information is often available and can be computed inexpensively.**

  Robot motion planning problems share a common objective of producing smooth motion while avoiding obstacles. We formalize this notion in terms of two objective functionals: a smoothness term $\mathcal{U}_{smooth}[\xi]$ which captures dynamics of the trajectory, and an obstacle functional $\mathcal{U}_{obs}[\xi]$ which captures the requirement of avoiding obstacles and preferring margin from them.

  We define the smoothness functional naturally in terms of a metric in the space of trajectories. By doing so, we generalize many prior notions of trajectory smoothness, including springs and dampers models of trajectories [61]. Each of these prior notions are just one type of valid metric in

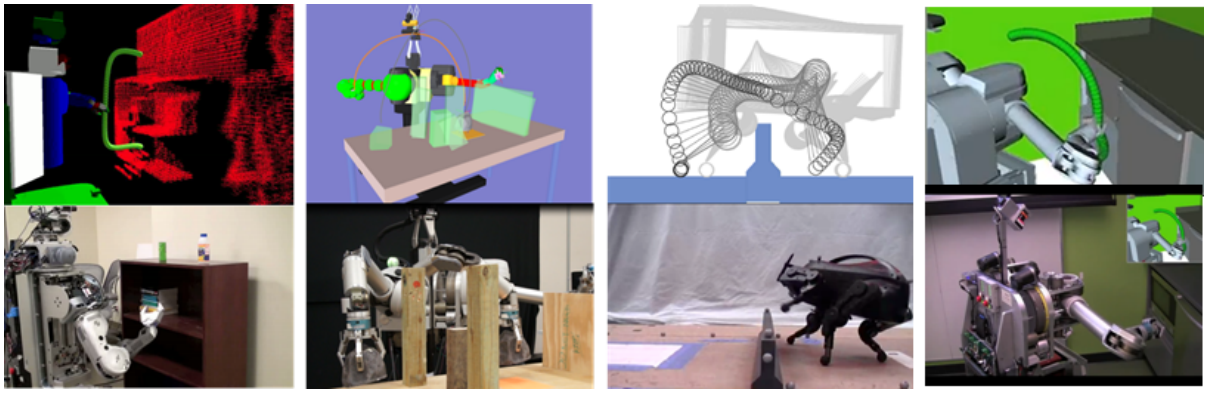


Figure 1: From left: CHOMP running on the Willow Garage PR2, CMU ARM-S Andy, LittleDog, and HERB platforms. The top row illustrates the simulation environments and planned trajectories, and the bottom row demonstrates real-world experiments executing the planned trajectories.

the space of trajectories. With this generalization, we are able to include higher-order derivatives or configuration-dependent metrics to define smoothness.

The obstacle functional is developed as a line integral of a scalar cost field c, defined so that it is invariant to re-timing. Consider a robot arm sweeping through a cost field, accumulating cost as is moves. Regardless of how fast or slow the arm moves through the field, it must accumulate the exact same cost.

As a consequence, the two functionals are complementary: the obstacle functional governs the shape of the path, and the smoothness functional governs the timing along the path.

The physical intuition of an arm sweeping through a cost field, hints at a further simplification. Instead of computing the cost field in the robot's high-dimensional configuration space, we compute it in its workspace (typically 2-3 dimensional) and use body points on the robot to accumulate workspace cost to compute $\mathcal{U}_{obs}[\xi]$. By framing the obstacle cost in terms of obstacle distance fields in the robot's workspace, we are able to exploit several recent innovations in the computation of Euclidean Distance Transforms (EDT). With this, we are able to compute functional gradients efficiently for complex real-world scenes (Sec. 4) to enable online replanning.

- **Trajectory optimization should be invariant to parametrization.** Our central tenet is that a trajectory is a geometric object unencumbered by parametrization. We treat it as a point in a possibly infinite dimensional space (here, a Hilbert space, but easily generalizable to a Riemannian manifold, see Sec. 3). Invariance guarantees identical behavior independent of the type of parametrization used, and ensures the algorithms we chose respect the underlying problem geometry.

  This motivates our choice of variational methods for optimization. Here, a trajectory ξ is expressed a function, mapping time t to configurations q, for example, and the optimization objective $\mathcal{U}$ is expressed as a functional: a function $\mathcal{U}[\xi]$ of the trajectory function ξ. The functional gradient is then the gradient of the functional $\mathcal{U}$ with respect to the trajectory ξ, another geometric term. The metric structure of the trajectory space enables us to precisely define *perturbations* of the trajectory in terms of the endowed norm. This gives us a clearly defined rule for modifying the gradient making it *covariant* to reparametrization.

Like other optimization techniques for non-convex objectives, functional gradient optimization will descend to a local minimum. We use the Hamiltonian Monte Carlo (HMC) algorithm to perturb the minimal trajectory to restart the process, as well as to provide a sampling procedure for a natural distribution over trajectories. In keeping with our central tenet, HMC is made parametrization invariant, and adds a momentum term, to perturb the trajectory out of its current basin of attraction (Sec. 5). Imagine the trajectory to be a ball sliding in an uneven landscape. Instead of a random restart, which

would pick up the ball from its local minimum and randomly place it elsewhere, HMC gives the ball a little kick of momentum, pushing it out of its current basin to explore elsewhere.

Our machinery also allows us to generalize our algorithm to constraints on the entire trajectory which, like keeping a coffee mug upright, are crucial in robotics applications. We show in Sec. 6 how the constrained functional gradient can be naturally formulated as an unconstrained step to minimize the objective followed by a Newton's method-style descent back on to the constraint manifold. For the specific case of goal manifolds, for example those induced by the rotational symmetry of objects for grasping, this reduces to a simple and efficient update that makes the algorithm faster still in practice.

CHOMP has been successfully implemented on several robotic platforms (Fig. 1) including the *HERB* 1.0 and 2.0 mobile manipulation platforms [72], the LittleDog quadruped [84], Carnegie Mellon's *Andy* Autonomous Robotics Manipulation-Software (ARM-S) platform, and the Willow Garage PR2 robot. In our research on these platforms, we have come to see CHOMP as the planning algorithm of first resort, surpassing randomized planners in performance for typical problems. We describe our extensive experiments in several different simulated environments with varying clutter, comparing CHOMP with the bidirectional RRT and RRT* algorithms in Sec. 7 and our experience with running CHOMP on the LittleDog platform in greater detail in Sec. 8.

Our experience and successes with implementing CHOMP on several robot platforms suggests that many real-world problems are, in fact, not hard to solve, especially with gradient information. For example, although LittleDog traversed extremely challenging terrain, nearly all terrains shared the commonality that moving a foot upward could remove it from collision. This simple gradient information is automatically conveyed to CHOMP via the obstacle functional. Likewise, most ARM-S tests involved objects placed on a table. Here again, moving up sufficiently above the table succeeds in getting the robot out of collision. In both cases gradient information makes a seemingly tricky motion planning problem relatively easy to solve.

Without question, trajectory planning methods – even with the probabilistically complete generalization we present – are not a panacea. They play a limited role in high-dimensional motion planning, and we would not advocate them as a complete solution for robotics problems. Rather, we propose that such techniques play a complementary role by solving simple problems quickly and effectively, and acting to smooth and improve the results of other techniques locally. Our focus on understanding the structure and geometry of motion planning problems in the real world, and developing a set of algorithms that respect natural parametrization invariances has enabled us to demonstrate good and fast performance on many problems.

2 Prior Work

The problem of high dimensional motion planning has been studied in great detail over the last several decades. We discuss below several types of approaches and their relationship to CHOMP.

2.1 Sampling-Based Planning

Sampling based approaches have become popular in the domain of high-dimensional motion planning, including manipulation planning. Seminal works by Barraquand and Latombe [3] and others demonstrated solvers for impressively difficult problems. The Probabilistic Roadmap (PRM) and Expansive Space Trees [33] methods were shown to be well-suited for path planning in $\mathcal{C}$-spaces with many degrees of freedom [41], and with complex constraints, including kinodynamic [9, 32, 44, 46]. Rapidly-exploring Random Trees (RRT) [50] have also been applied to differential constraints, and were shown to be successful for general high-dimensional planning [47].

These approaches typically work in a two-step fashion: first a collision-free path is discovered without regard for any measure of cost, 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 [11, 40]. Methods such as medial axis and "partial" shortcuts have also proven effective [28].

Randomized approaches are understood to be probabilistically complete [50], but capable of solving many challenging problems quite efficiently [7]. Such approaches typically do not explicitly optimize

an objective function, although Karaman and Frazzoli [39] suggest such optimization in the asymptotic sense. As the sampling-based planners became increasingly well understood in recent years, it was suggested that randomization may not, by itself, account for their efficiency [51]. Branicky et al. [7] show that quasi-random sampling sequences can accomplish similar or better performance than their randomized counterparts.

The method proposed here represents a departure from this motion planning paradigm in that sampling is performed directly in the space of trajectories, and numerical optimization is performed explicitly given the relevant measures of optimality. A complementary sampling process attempts to achieve a similar degree of exploration to that attainable by classical sampling based methods.

Similar to sampling based and search based methods [53], CHOMP may benefit from pre-computed, compositional representations of the environment [6, 29, 54]. Efficient hierarchical representations have also been proposed by Faverjon [24], Quinlan [60] and others. It is also straight-forward to pre-compute accurate, high-resolution distance fields and gradients for the known objects in the environment. The computer graphics community has developed an array of high-performance methods for computing distance fields and proximity queries for arbitrary meshes, including parallelized algorithms implemented using graphics hardware [23, 48, 71, 73]. Recent methods are capable of processing dynamic scenes in real-time. The performance of the CHOMP algorithm can be trivially improved by leveraging these complementary research efforts.

2.2 Resolution Complete Approaches

Although naïve application of A* is typically unsuitable for high-dimensional planning problems, current research has made advances in applying forward search to systems with many degrees of freedom. Until recently, a major problem with A* and related algorithms had been that admissible heuristics result in examination of prohibitively large portions of the configuration space, whereas inflated heuristics cause significantly suboptimal behavior. Likhachev et al. [52] present a framework for efficiently updating an A* search while smoothly reducing heuristic inflation, allowing resolution complete search in an anytime fashion on a broader variety of problems than previously computable. Additional work has examined inducing smoother motion while reducing the cardinality of the action set [13], as well as re-using partial plans discovered by past searches [59].

The choice of discretization made by these approaches presents a clear contrast with CHOMP. While resolution complete approaches necessarily discretize states, actions, and time, CHOMP allows states to vary continuously. We believe 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. Again, we stress that CHOMP is not suitable for all tasks. Difficult problems featuring bottlenecks and other such "narrow passages" may indeed require complete planners; however, we believe that CHOMP is well suited for solving the types of problems described in sections 7 and 8.

2.3 Trajectory Optimization

A number of numerical trajectory optimization techniques have also been applied in this domain. Khatib [43] pioneered the application of the artificial potential field for real-time obstacle avoidance. 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 [67].

Quinlan and Khatib [61] 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. An extensive effort is made to construct a model of the free space, and the cost function contains terms that attempt to control the motion of the trajectory particles along the elastic. Brock and Khatib [8] further extend this technique. Much of the complexity of explicit simulation of physical quantities is alleviated by the method proposed herein via *covariant* gradient descent, introduced by Ratliff et al. [65]. Obstacles are considered directly in the workspace of the robot, where the notions of distance and inner product are more natural.

Kalakrishnan et al. [38] apply motion sampling techniques in a context similar to ours. Their method, STOMP, obtains gradient information using trajectory samples. In contrast, our approach exploits the availability of the gradient. Our assumption that it exists is satisfied in most relevant cases and does not hinder the application of the method in a variety of relevant contexts, such as collision avoidance and workspace constraint satisfaction, as we demonstrate later on.

Functional gradient based approaches have also been studied by Žefran et al. [79] in the context of motion planning for systems in which the dynamic equations describing the evolution of the system change in different regions of the state space. They utilize a control theory point of view and focus on the planning of open loop trajectories that can be used as nominal inputs for control.

2.4 Objective Learning

A number of methods approach difficult, high-dimensional planning problems by learning the structure of the objective function in order to simplify the problem. Dalibard and Laumond [15] use PCA to discover promising directions for exploring an RRT. Vernaza and Lee [78] 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. CHOMP, nevertheless, can exploit the parameter-performance mapping to train the cost function in a manner reminiscent of imitation learning techniques [62, 63].

Jetchev and Toussaint [35] seek to train a regressor to predict trajectories given information about a problem instance. Here, *problem instance* denotes a combination of task-specific information, such as initial and goal poses, and environment information, such as a coarse occupancy grid enumerating free workspace. The authors show that informed predictions greatly boost the performance and convergence time of optimization algorithms. Although the authors in this case used a variety of other optimizers such as DDP [55], AICO [77], and RPROP [34], we view the trajectory prediction research as complementary in that it establishes that informed priors can overcome local minima and trade-off offline learning for online optimization.

2.5 Optimal Control

CHOMP is also related to optimal control of robotic systems. Instead of focusing on computing feasible paths, however, our goal is to directly construct trajectories which optimize over a variety of dynamic and task-based criteria. Few current approaches to optimal control are equipped to handle obstacle avoidance, though. Differential dynamic programming [55] is a popular approach in this category and has been applied to a number of complex systems. The techniques presented herein may enable such methods to naturally consider obstacle avoidance as part of optimizing the control. We also note that by virtue of avoiding the explicit computation of second-order dynamics, CHOMP is likely to be more efficient in many relevant scenarios.

In some similarity to the proposed approach, ILQG [75] 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, such as [70], require a description of configuration space obstacles, which can be prohibitive to create in the context of high-dimensional planning problems. Many optimal controllers which do handle obstacles are framed in terms of mixed integer programming, which is known to be an NP-hard problem [69]. Approximate optimal algorithms exist, but so far, only consider very simple obstacle representations. As our results illustrate, CHOMP demonstrates attractive computational efficiency and convergence even in cluttered obstacle environments.

3 Theory

CHOMP is designed to produce high quality trajectories for complex robotic systems with many degrees of freedom which are both *smooth*—eliminating unnecessary motion—and *collision free*. Our 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. CHOMP satisfies these properties and can be viewed as a generalization of earlier work leveraging analogies to masses, springs, and repulsive forces in trajectory optimization developed in e.g., [8, 60]

In this work, we consider a trajectory $\xi : [0,1] \to \mathcal{C} \subset \mathbb{R}^d$ as a smooth function mapping time to robot configurations.[1] Since ξ is a function, any objective criterion that we may like to optimize is best represented as an objective *functional* $\mathcal{U} : \Xi \to \mathbb{R}$ which maps each trajectory ξ in our space of trajectories Ξ to a real number. Ideally, we want our objective to account for obstacle proximity or collision criteria as well as dynamical quantities of the trajectory such as smoothness.

Our approach, which we discuss in detail in Section 3.3, is to define the mathematical properties of our space of trajectories (quantities such as norm and inner product) and our objective functional entirely in terms of physical aspects of the trajectory so that mathematical objects and calculations (e.g. gradients and algorithmic convergence criteria) 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 (see [30]) we derive 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, we work with 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 [49]. 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. See Fig. 2 for an illustration.

3.1 The Objective Functional

Our objective functional separately measures two complementary aspects of the motion planning problem. It first penalizes a trajectory based on dynamical criteria such as velocities and accelerations to encourage smooth trajectories. Simultaneously, it penalizes proximity to objects in the environment to encourage trajectories that clear obstacles by a prescribed amount. We denote these two terms in our presentation as $\mathcal{F}_{smooth}$ and $\mathcal{F}_{obs}$, respectively, and define our objective simply as their weighted sum:

$$\mathcal{U}[\xi] = \mathcal{F}_{obs}[\xi] + \lambda \mathcal{F}_{smooth}[\xi]. \tag{1}$$

As stated above, the trajectory ξ is a function mapping time to robot configurations. In this section we assume the initial configuration $\xi(0) = q_0$ and final configuration $\xi(1) = q_1$ are fixed, although Section 6 relaxes this assumption to allow the final configuration to lie anywhere on a smooth manifold of goal points.

[1] We consider time to range from 0 to 1 here without loss of generality. Dilating the time by some constant factor changes the updates only by scaling the gradient contribution from $\mathcal{F}_{smooth}$ by a constant factor. One can interpret this scaling factor as absorbed into our trade-off parameter λ.

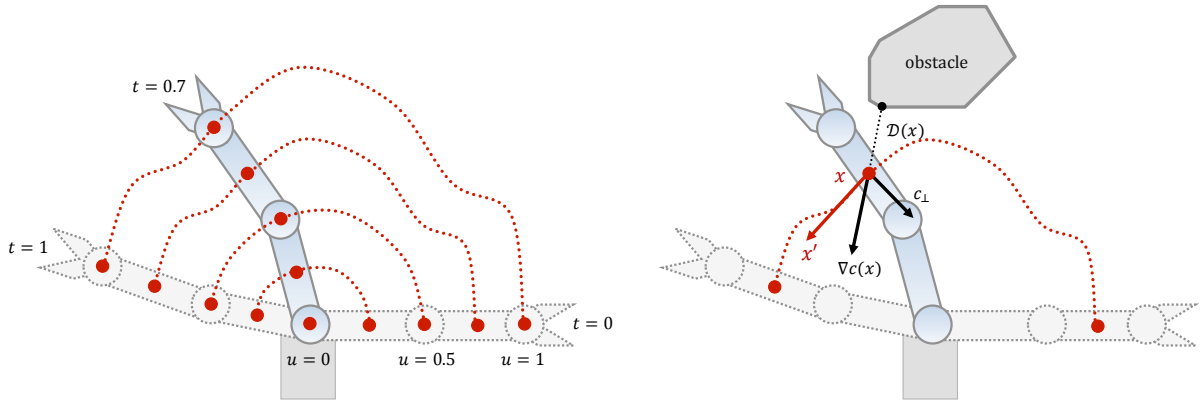


Figure 2: Trajectory optimization for a robot can be considered as minimizing a potential over a set of workspace paths which are jointly constrained by the robot kinematics. *Left:* CHOMP seeks to minimize the obstacle potential $c(x)$, where $x(\xi(t), u)$ is a workspace position indexed by the configuration $\xi(t)$ at time t and robot body position u. *Right:* A single body point $x(\xi(t), u)$ is examined, and its velocity x' is shown, along with the minimum distance to an obstacle $\mathcal{D}(x)$, and the gradient of the collision cost at the body point $\nabla c(x)$, with orthogonal projection to the trajectory $c_\perp$.

$\mathcal{F}_{smooth}$ measures dynamical quantities across the trajectory such as, for example, the integral over squared velocity norms:

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

Our theory extends straightforwardly to higher-order derivatives such as accelerations or jerks, although we simplify the notation and presentation here by dealing only with squared velocities.

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 a robot configuration $q \in Q$ 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. As discussed in Section 3.4 we typically define this workspace cost function in terms of the Euclidean distance to the boundaries of obstacles.

Accordingly, 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:

$$\mathcal{F}_{obs}[\xi] = \int_0^1 \int_{\mathcal{B}} c\big(\, x(\xi(t), u)\,\big) \left\| \frac{d}{dt} x\big(\xi(t), u\big) \right\| du\, dt \tag{3}$$

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).

Quinlan [61] used a similar obstacle objective in his foundational elastic bands work, and provides a useful introduction with derivations of similar gradients in [60]; however our obstacle objective, unlike Quinlan's, integrates with respect to arc-length in the workspace as opposed to the configuration space. This simple modification represents a fundamental change: instead of assuming the geometry in the configuration space is Euclidean, we compute geometrical quantities directly in the workspace where Euclidean assumptions are more natural. Intuitively, this observation means that the objective functional

has no incentive to directly alter the trajectory's speed through the workspace for any point on the robot. Section 3.2 discusses this property in terms the role the workspace gradient plays in perturbing the trajectory during a steepest descent update.

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:

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

Intuitively, this formulation minimizes the largest violation of the obstacle constraint at each iteration. From iteration to iteration, this maximally colliding point may shift to other body elements as the algorithm incrementally decreases these large violations with each step. While this formulation is less demanding computationally per iteration, it may require a greater number of iterations, since each of these iteration is less informed.

3.2 Functional Gradients

Gradient methods are natural candidates for optimizing such objective criteria since they leverage crucial first-order information about the shape of the objective functional. In our case, we consider the *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 $\epsilon \to 0$. Here, we summarize the functional gradient for a particular family of functionals, following the derivation in [60].

Let our functional take the form $\mathcal{U}[\xi] = \int v(\xi(t), \xi'(t))\, dt$. Consider 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

$$\frac{df}{d\epsilon} = \int \left(\frac{\partial v}{\partial \bar{\xi}} \cdot \frac{\partial \bar{\xi}}{\partial \epsilon} + \frac{\partial v}{\partial \bar{\xi}'} \cdot \frac{\partial \bar{\xi}'}{\partial \epsilon} \right) dt \tag{5}$$

Consider the second term in the parentheses above. Integrating by parts, and applying the constraint that $\phi(0) = \phi(1) = 0$ (i.e. the displacement leaves the endpoints of $\bar{\xi}$ fixed), we find

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

So overall,

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

When $\epsilon = 0$, we are essentially taking a directional derivative of $\mathcal{U}[\xi]$ in the direction ϕ, and the equation above becomes

$$\left. \frac{df}{d\epsilon} \right|_0 = \int \left(\frac{\partial v}{\partial \xi} - \frac{d}{dt} \frac{\partial v}{\partial \xi'} \right) \cdot \phi\, dt \tag{8}$$

The perturbation ϕ that maximizes this integral (subject to, say, a finite Euclidean norm) is the one which is proportional to the quantity in the parentheses above, and hence the functional gradient is defined as

$$\bar{\nabla}\mathcal{U}[\xi] = \frac{\partial v}{\partial \bar{\xi}} - \frac{d}{dt} \frac{\partial v}{\partial \bar{\xi}'}. \tag{9}$$

As shown in [60], this can naturally be extended to incorporate higher derivatives of ξ. Furthermore, the functional gradient vanishes at critical points of $\mathcal{U}$, which include the minima we seek. In fact, the criterion $\bar{\nabla}\mathcal{U} = 0$ is the Euler-Lagrange equation, which is central in the study of calculus of variations [27]. Hence, we can run steepest descent on the objective functional by iteratively taking steps in the direction of the negative functional gradient:

$$\xi_{i+1} = \xi_i - \eta_i \bar{\nabla}\mathcal{U}[\xi]. \tag{10}$$

until we reach a local minimum.

Since the objective is the sum of the prior and obstacle terms, we have $\bar{\nabla}\mathcal{U} = \bar{\nabla}\mathcal{F}_{obs} + \lambda\bar{\nabla}\mathcal{F}_{smooth}$. The functional gradient of the smoothness objective in (2) is given by

$$\bar{\nabla}\mathcal{F}_{smooth}[\xi](t) = -\frac{d^2}{dt^2}\xi(t), \tag{11}$$

and the functional gradient of the obstacle objective in (3) is given by

$$\bar{\nabla}\mathcal{F}_{obs}[\xi] = \int_{\mathcal{B}} J^{\mathrm{T}}\|x'\|\left[(I - \hat{x}'\hat{x}'^{\mathrm{T}})\nabla c - c\kappa\right] du, \tag{12}$$

where $\kappa = \|x'\|^{-2}(I - \hat{x}'\hat{x}'^{\mathrm{T}})x''$ 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 $(I - \hat{x}'\hat{x}'^{\mathrm{T}})$, 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 (see Figure 2 for an illustration).[2]

3.3 Functionals and Covariant Gradients for Trajectory Optimization

Our objective functional defined in Section 3.1 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. However, the steepest descent algorithm portrayed in Section 3.2, while attractive in its simplicity, unfortunately does depend on the trajectory's representation since the functional gradient at its core is derived from a Euclidean norm. (Specifically, it assumes the trajectory function is represented in terms of a basis of Dirac delta functions so that the t^{th} component is $\langle f,\ \delta_t\rangle = \int f(\tau)\delta(t-\tau)d\tau = f(t)$).

This section discusses at an abstract functional level the principle behind removing this dependence on the parametrization. Our approach 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:

$$\|\xi\|_A^2 = \int \sum_{n=1}^{k} \alpha_n (D^n\xi(t))^2 dt, \tag{13}$$

where D^n is the nth order derivative operator and $\alpha_n \in \mathbb{R}$ is a constant. More concretely, for $k = 1$, since D^1 operates simply by taking the first derivative, the norm in Equation 13 reduces to $\|\xi\|_A^2 = \int \xi'(t)^2 dt$. Similarly, we can define an inner product on the space of trajectories by measuring the correlation between trajectory derivatives:

$$\langle \xi_1,\ \xi_2\rangle = \int \sum_{n=1}^{k} \alpha_n \big(D^n\xi_1(t)\big)\big(D^n\xi_2(t)\big) dt, \tag{14}$$

A at this point serves primarily to distinguish this norm and inner product from the Euclidean norm, but it's notational purpose is clarified below in Sections 3.4 and 3.5 the concrete context of our finite-dimensional waypoint parametrization. For simplicity of presentation, we will 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 $\|f\|_A^2 = \langle f,\ Af\rangle = \int (Df(t))^2 dt$ [30].[3]

Our experiments primarily use the $k = 1$ (total squared velocity) variant. In other words, we measure perturbations to a trajectory in terms of how much the perturbation changes the overall velocity profile

[2] Alternatively, the functional gradient of (4) simply evaluates the quantity inside this integral at the maximizer u.

[3] Throughout this paper we assume A is constant, but in full generality A constitutes a Riemannian metric on the manifold of trajectories, which allows A to vary smoothly across the space. For instance, if we define A to account for the configuration of robot masses in its measurement of dynamical quantities, the form of A depends on kinematic Jacobians along the trajectory which vary as the trajectory deforms.

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 we change the trajectory's parametrization, we have a clearly defined rule for changing the norm as well so that it still measures the same physical quantity. Functional gradients computed under this norm, which we denote $\bar{\nabla}_A \mathcal{U}$, therefore, constitute update directions fundamental to trajectories, themselves, and not their choice of representation.

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:

$$\bar{\nabla}_A \mathcal{U}[\xi] = A^{-1} \bar{\nabla} \mathcal{U}[\xi]. \tag{15}$$

For much of what follows, both in this section and in Section 6, we simplify the mathematical presentation by deriving results in terms of their finite-dimensional counterparts in a waypoint parametrization (see Section 3.4). Borrowing techniques from the analysis of *direct methods* in the Calculus of Variations (particularly the Method of Finite Differences [27]), one can show that the limit of these results as our discretization becomes increasingly fine converges to the prescribed functional variants; however, finite-dimensional linear algebra (such as matrix inversion) is often substantially simpler than its infinite-dimensional counterpart (such as deriving an operator's Green's function). We emphasize here, however, that 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 [68].

3.4 Waypoint Trajectory Parametrization

Although so far we have remained unencumbered by a particular parametrization of the trajectory ξ, numerically performing functional gradient descent on (1) mandates a parametrization choice. As mentioned above, there are many valid representations, but we choose a uniform discretization which samples the trajectory function over equal time steps of length Δt: $\xi \approx (q_1^{\mathrm{T}}, q_2^{\mathrm{T}}, \ldots, q_n^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{n \times d}$, with q_0 and q_{n+1} the fixed starting and ending points of the trajectory.

Using this waypoint parametrization, we can write the smoothness objective (2) as a series of finite differences:

$$\mathcal{F}_{smooth}[\xi] = \frac{1}{2} \sum_{t=1}^{n+1} \left\| \frac{q_{t+1} - q_t}{\Delta t} \right\|^2 \tag{16}$$

With a finite differencing matrix K and vector e (which handles boundary conditions q_0 and q_{n+1}), we can rewrite (16) as

$$\mathcal{F}_{smooth}[\xi] = \frac{1}{2} \|K\xi + e\|^2 = \frac{1}{2} \xi^{\mathrm{T}} A \xi + \xi^{\mathrm{T}} b + c \tag{17}$$

with $A = K^{\mathrm{T}}K$, $b = K^{\mathrm{T}}e$, and $c = e^{\mathrm{T}}e/2$. Hence, 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.

Computation of the obstacle gradient for the waypoint parametrization is a relatively straightforward modification of (12): the set $\mathcal{B}$ is discretized, and the integral is replaced with a summation. Furthermore, to compute x' and x'', the workspace velocity and position of a point, we use central differences of the elements q_t of ξ and map them through the kinematic Jacobian J.

3.5 Gradient Descent

Equipped with a finite dimensional parametrization of the trajectory ξ and the gradient of $\mathcal{U}$ with respect to each element q_t of ξ, we can now perform gradient descent. Here we define an iterative update rule that starts from an initial trajectory ξ_0 and computes a refined trajectory ξ_{i+1} given the previous trajectory ξ_i. We typically start with a naïve initial trajectory ξ_0 consisting of a straight line path in configuration space. As mentioned previously, this path is not in general feasible; however, unlike prior trajectory optimizers, CHOMP is often able to find collision free trajectories nonetheless.

To derive the update rule, we will solve the Lagrangian form of an optimization problem [1] that attempts to maximize the decrease in our objective function subject to keeping the difference between

ξ_i and ξ_{i+1} small. As discussed in Section 3.3, the choice of a metric on the space of trajectories (i.e. determining what "small" means) is critical. The covariant update rule we define 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.

We begin by linearizing $\mathcal{U}$ around ξ_i via a first order Taylor series approximation:

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

The optimization problem is now defined as

$$\xi_{i+1} = \arg\min_{\xi} \mathcal{U}[\xi_i] + (\xi - \xi_i)^{\mathrm{T}} \bar{\nabla} \mathcal{U}[\xi_i] + \frac{\eta}{2} \|\xi - \xi_i\|_M^2 \tag{18}$$

The term $\|\xi - \xi_i\|_M^2 = (\xi - \xi_i)^{\mathrm{T}} M (\xi - \xi_i)$ 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. In the typical Euclidean case we would have $M = I$; instead, we choose M to be the matrix A defined in the previous subsection. Again, this means we prefer perturbations which add only small amounts of additional acceleration to the overall trajectory. If we differentiate the term on the right hand side of (18) above with respect to ξ and set the result to zero, we arrive at the update rule

$$\xi_{i+1} = \xi_i - \frac{1}{\eta} A^{-1} \bar{\nabla} \mathcal{U}[\xi_i] \tag{19}$$

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.

We gain additional insight into the computational benefits of the covariant gradient based update by considering the analysis tools developed in the on-line learning/optimization literature, especially [31, 83]. 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 we can gain insight into the performance of both standard gradient methods (including those considered by, e.g. [61]) and the CHOMP rule.

Under these conditions, 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 [64]. The authors of [31] show how gradient-style updates can be understood as sequentially minimizing a local quadratic approximation to the objective function. 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 our obstacle objective $\mathcal{F}_{obs}$ may not even be twice differentiable, however we may still form a quadratic lower bound using A. This is much tighter than an isotropic bound and leads to a correspondingly faster minimization of our objective. In particular, in accordance with the intuition of adjusting large parts of the trajectory due to the impact at a single point we would generally expect it to be $O(n)$ times faster to converge than a standard, Euclidean gradient based method that adjusts a single point due an obstacle.

We have used a variety of termination criteria for CHOMP, but the most straightforward is to terminate when the magnitude of the gradient $\bar{\nabla} \mathcal{U}[\xi]$ falls below a predetermined threshold. Of course, the algorithm may terminate with a trajectory that is not feasible (i.e. there remain collisions), and furthermore CHOMP will fail to report if no feasible path exists. Despite the lack of completeness, deciding whether any particular path is feasible is straightforward since the workspace cost function (which must be evaluated at each point over the trajectory at each iteration) is aware of distances to obstacles. After CHOMP terminates, a higher-level planner or coordination scheme may decide how to proceed based on the quality and feasibility of the path.

Finally, we note that although the A matrix may be quite large, it is also sparse, with a very simple band diagonal structure. Hence we can use an algorithm such as Thomas' backsubstitution method for tridiagonal matrices [14] to solve the system $Ax = b$ in time $O(n)$ rather than a naïve matrix inverse which requires $O(n^3)$ time.

3.6 Cost Weight Scheduling

The weights in the sum in (3.1) 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 [42]. Early in optimization, it is beneficial 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, as discussed in Section 3.5.

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. Indeed, the optimization may be at a local minimum of this sort right at the outset, for example if the initial trajectory is a line in joint space. Experimental results in Figure 23 suggest that weight scheduling indeed improves performance in practice – both in terms of better planner convergence and the reduced number of required iterations.

4 CHOMP Obstacle Cost

For many robotic applications, obtaining the CHOMP collision cost is straightforward. Consider the case of a many-linked robot in the workspace $\mathbb{R}^w$. Assume that the robot's configuration can be fully described by d parameters, that is, the robot's configuration is $q \in \mathbb{R}^d$. Assume also that obstacles are well-defined subsets of the workspace . Now, given a trajectory through space, we are able to define its cost by considering the configuration of the robot at each point in time, and looking at the distance from points on the robot's body to obstacles.

4.1 Modeling the Robot and Obstacles

To make the optimization problem tractable, we must first develop an efficient representation of the robot and obstacles in the environment. We do this by making simplifying assumptions about the robot's structure, and the structure of obstacles.

4.1.1 Simplified Robot Model

The robot's body is made up of a set of points, $\mathcal{B} \subseteq \mathbb{R}^w$. Now, let the forward kinematics function $x(q, u) : \mathbb{R}^d \to \mathcal{B}$ give the point $u \in \mathcal{B}$ of the robot, given its configuration q.

In practice, 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. For our experiments, we fit a swept-sphere capsule to each link of the robot manipulator, and then generate a discrete set of spheres for each capsule. In this way, the nearest distance to any point in $\mathcal{B}$ can be made simple to compute. For spheres, the distance to any point is given by the distance to the center of the sphere minus its radius.

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 $\epsilon > 0$.

4.1.2 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 $x \in \mathbb{R}^w$ 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

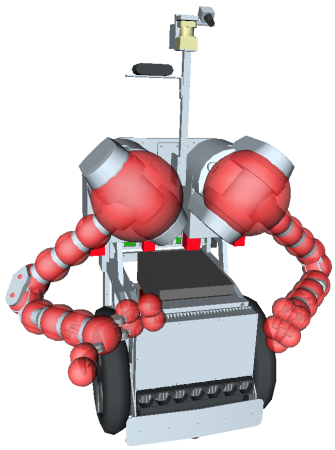
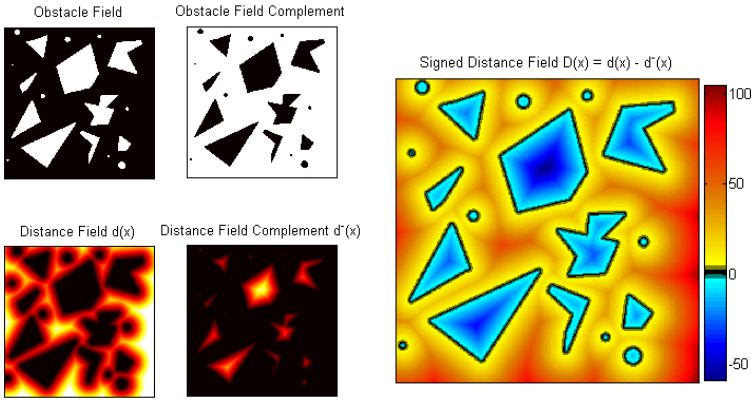


Figure 3: A simplified representation of the HERB Robot used in our experiments. We simply fit a small set of spheres over the links of the robot arm.



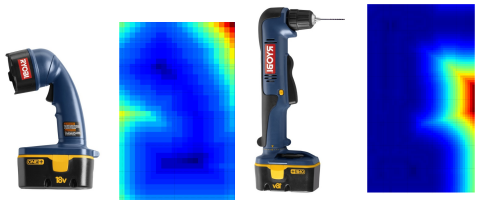
(a) Field Construction (b) Signed Distance Field

Figure 4: The signed distance field is created as the difference between two distance transforms created over a binary grid where obstacles are set to 1, and its complement, where obstacles are set to zero, so that $\mathcal{D}(x) = d(x) - \bar{d}(x)$. Here, brighter colors represent higher values.

Euclidean Distance Transform include P. Felzenswalb and D. Huttenlocher's $O(n)$ algorithm[25], and a $O(n \log n)$ algorithm from E.G. Gilbert et al. [26], starting from a boolean obstacle grid. We compute the EDT for both the obstacle grid d and its complement $\bar{d}$. The signed distance field is given by the difference of these two fields, i.e. $\mathcal{D}(x) = d(x) - \bar{d}(x)$, as shown in Fig. 4. 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 of geometric primitives. The objects are almost never convex and typically have complex shape, as illustrated in Figure 5. In addition, we 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, we propose utilizing the same object models for efficient distance field computation.

In particular, we 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



(a) (b) (c) (d)

Figure 5: The figures shows two examples of objects with complex, non-convex shapes that would be difficult to represent accurately with simple geometric primitives. The corresponding object distance primitives are shown on the right in the figure.

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 $\mathcal{O}$.

$$\mathcal{D}(\mathbf{x}) = \min_{O \in \mathcal{O}} \left(T_{F_O}^{F_W} \right)^{-1} \mathbf{x} \tag{20}$$

Hierarchical representations, such as the k-d tree [4], may be utilized to speed up this computation.

For our experiments, we use a combination of these approaches. For static elements of the environment, we pre-compute and store a distance field, and for dynamic or sensed objects, we use oriented bounding boxes as well as pre-computed distance fields. For raw sensor data such as point clouds, we simply keep track of an occupancy grid over the entire workspace, and use this occupancy grid to generate a distance field. Depending on the relative speeds of the distance transform algorithm, reading files from the hard disk, and analytic distance computations to primitives, using any one of these methods in place of the other may provide a performance boost.

4.2 Obstacle Cost Formulation

Now, we define a obstacle cost function which penalizes the robot for being near obstacles. As in Ratliff et al. [65], we define the general cost function $c : \mathbb{R}^w \to \mathbb{R}$ (the general cost of any point in the workspace) as

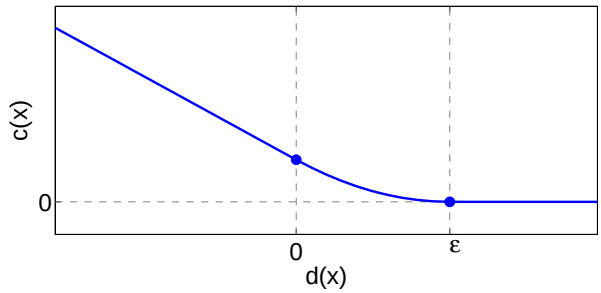


Figure 6: A plot of (21). Note that the cost of a point in the workspace smoothly drops to zero as a distance of the allowable threshold ε is reached.

$$c(x) = \begin{cases} -\mathcal{D}(x) + \frac{1}{2}\varepsilon, & \text{if } \mathcal{D}(x) < 0 \\ \frac{1}{2\varepsilon}(\mathcal{D}(x) - \varepsilon)^2, & \text{if } 0 < \mathcal{D}(x) \leq \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

And define $\nabla c : \mathbb{R}^w \to \mathbb{R}^n$ as the gradient $\frac{\partial c}{\partial x}$. Note that since c depends only on $\mathcal{D}$ and x, $\nabla(c)$ can easily be computed via finite differencing in the distance field.

It is also possible to modify our workspace cost function to attain some arbitrary desired Euclidean motion. For example, given a certain point on the robot body, $u \in \mathcal{B}$, it is natural to define a cost function that evaluates to 0 if an element of u's position and orientation vectors satisfies the desired constraint and to quadratic error if it does not. Using the z-dimension of position as an example, and supposing we desire to limit u's pose such that its z-coordinate is greater than $z_{\min}$, we can define a workspace cost potential similar to (21) :

$$c(z) = \begin{cases} (z - z_{\min})^2 & \text{if } z < z_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

If the desired interval is closed, another instance of (22) can be used for the other boundary, e.g. $z_{\max}$. We can now define the obstacle cost of a trajectory ξ. Assume that ξ can be expressed as a continuous time function $\xi(t) : [0, 1] \to \mathbb{R}^d$, which gives the configuration of the robot at time t. We can then express the obstacle cost as a functional over ξ.

One formulation of the obstacle cost might be as the integral over the entire trajectory of the collision-cost of each trajectory point. While intuitive, this formulation ignores the velocity of points through obstacles (which prevents the obstacle cost from being invariant to reparametrization), so instead we must define the obstacle cost as:

$$\mathcal{F}_{\text{obs}}[\xi] = \int_0^1 \int_{u \in \mathcal{B}} c\big(x(\xi(t), u)\big) \left\| \frac{d}{dt} x(\xi(t), u) \right\| \, \mathrm{d}u \, \mathrm{d}t \tag{23}$$

We then compute the functional gradient of the obstacle cost as

$$\bar{\nabla}\mathcal{F}_{\text{obs}}[\xi] = \frac{\partial v}{\partial \xi} - \frac{d}{dt}\frac{\partial v}{\partial \xi'} \tag{24}$$

Where v is everything in the time integral in (23) [65]. Applying to (23), we obtain

$$\bar{\nabla}\mathcal{F}_{\text{obs}}[\xi] = \int_{u \in \mathcal{B}} J^{\mathrm{T}} \Big(\|x'\| \big((I - \hat{x'}\hat{x'}^{\mathrm{T}}) \nabla c - c\kappa \big) \Big) \tag{25}$$

Where x' is the first derivative of x, $\hat{x'}$ denotes the normalized vector $\frac{x'}{\|x'\|}$, κ is the curvature of the trajectory [65], defined by

$$\kappa = \frac{1}{\|x'\|^2} (I - \hat{x'}\hat{x'}^{\mathrm{T}}) x'' \tag{26}$$

and J is the Jacobian of the point $u \in \mathcal{B}$ on the robot given its configuration $\frac{\partial}{\partial q} x(\xi(t), u)$. These terms are visualized for a robot in $\mathbb{R}^2$ in Fig. 2.

Here, (25) represents an integration in time and workspace positions over the entire trajectory, taking body-point velocities into account. Notice also that due to the curvature term κ, the gradient is orthogonal to the workspace motion of each point on the robot's body.

In practice, the obstacle cost and gradient are computed by first by discretizing the time interval into T steps, with $\Delta T = \frac{1}{T}$ time between them, iterating over the discrete set of body primitives, summing all of the pre-computed obstacle cost gradients (21), and passing the result through the kinematic Jacobian at each discrete timestep along the trajectory. Velocities are approximated via finite differencing, and stored for each timestep in the trajectory.

That is, assume the time interval of the trajectory is $\mathbf{t} = \{t_0, t_1, \ldots, t_T\}$, and assume that the set of body points is a set of primitives: $\mathcal{B} = \{u_1, u_2, \ldots, u_b\}$, where the distance from the primitive i at timestep j to any obstacle in the environment can be computed in constant time by $\mathcal{D}\big(x(u_i, t_j)\big)$. Now, the workspace velocity of a body point is simply $x'_{i,j} = \frac{x(u_i, t_j) - x(u_i, tj-1)}{\Delta T}$. The Jacobian for that body point at time j can likewise be computed and stored, as $J_{i,j}$, as well as the collision cost and its gradient.

Then, the gradient term (25) can be approximated as

$$\bar{\nabla}\mathcal{F}^j_{\text{obs}} \approx \sum_{u_i \in \mathcal{B}} J^{\mathrm{T}}_{i,j} \Big(\|x'_{i,j}\| \big((I - \hat{x'_{i,j}} \hat{x'_{i,j}}^{\mathrm{T}}) \nabla c_{i,j} - c_{i,j}\kappa \big) \Big) \tag{27}$$

Where $\bar{\nabla}\mathcal{F}^j_{\text{obs}}$ is a d-dimensional vector representing the functional gradient of the trajectory at discrete timestep j. In this way, the functional gradient $\bar{\nabla}\mathcal{F}^j_{\text{obs}}$ can be approximated in its discrete form as a $d \times T$ matrix, where the j^{th} column is the discrete approximation of the functional gradient over time interval j (27).
Using this approximation of the obstacle term, for each iteration of CHOMP, we first do forward kinematics calculations for each body point on the robot, and simultaneously store the Jacobian evaluated at each point, as well as the collision cost and its gradient. It then becomes straightforward to compute the vector in the configuration space which pushes the robot out of collision.

4.2.1 Joint Limits

One practical concern is the fact that after the gradient update described in Section 3.2, the robot's configuration might be outside of hard joint limits. A naive way to solve this problem might be to clamp the robot's configuration such that it remains inside joint limits. However, this is not ideal, as it makes the trajectory of the robot non-smooth. As in Ratliff et al [65], we handle joint limits by smoothly projecting joint violations using the metric described in section 3.

Suppose that the robot has joint limits defined by $q_{\min} \in \mathbb{R}^d$, and $q_{\max} \in \mathbb{R}^d$. Along each point in the trajectory ξ, if the configuration is outside of the allowed bounds, we compute the closest L_1 projection to the bounds defined by $\{q_{\min}, q_{\max}\}$. Do this for each point in the trajectory. Call this the *violation trajectory*, ξ_v.

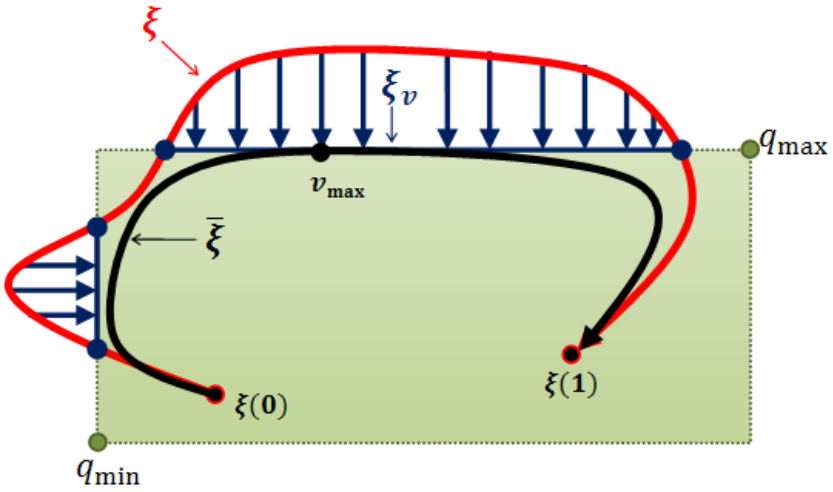


Figure 7: The configuration space is shown as $\mathbb{R}^2$. The trajectory ξ briefly exits the joint limit subspace (shown as a box), and re-enters. The violation trajectory ξ_v, is shown as a set of vectors projecting ξ back onto the surface of the joint limit subspace. The point with the largest violation, $v_{\max}$ is also shown. Note that ξ_v is zero whenever ξ is inside the joint limit subspace. The resulting trajectory $\bar{\xi}$ is also shown.

Then, until we are inside the joint limits, or until we have reached a maximum number of iterations for resolving joint limits, we subtract a scaled version of the violation trajectory from the current trajectory after passing it through the smoothness metric, i.e.:

$$\bar{\xi} = \xi + \xi_v^* \tag{28}$$

where $\xi_v^* = A^{-1}\xi_v$, scaled in such a way so that it exactly cancels out the largest joint violation along the original trajectory. That is, the element of ξ_v^* with the largest absolute value (called $v_{\max}$) equals the element of ξ_v with the largest absolute value. We can do this and still retain smoothness because of the invariance of the covariant gradient update to the scale of the update vector.

5 Hamiltonian Monte Carlo

Many recent approaches to motion planning [49] center around sampling to ensure probabilistic completeness. These methods choose the most promising solution from a set of feasible candidates sampled from an implicitly defined distribution over paths. When a problem has many possible solutions, especially those distinct homotopy classes of solutions, trajectory distributions offer a more comprehensive representation of the breadth of the solution space. Sampling from trajectory distributions can be a good way avoid the local optima that plague greedy gradient procedures.

Modeling *distributions* over a space of paths is a problem of fundamental significance to a number of fields such as statistical physics [10], finance [45], human behavior modeling [81, 82], and robust and stochastic control [2, 74, 76]. In all of these applications, *Gibbs distributions* over trajectories $p(\xi) \propto \exp\{-\mathcal{U}(\xi)\}$ play a central role since they may be regarded as an optimal representation of the uncertainty intrinsic to decision making [80].

In this section, we review the Hamiltonian Monte Carlo (HMC) method [56, 57], a sampling 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, in the context of CHOMP, 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.

5.1 Intuition

Given an objective $\mathcal{U}(\xi)$, we can construct an associated probability distribution that respects the contours of the function relative to its global minimum as:

$$p(\xi; \alpha) \propto \exp\{-\alpha \mathcal{U}(\xi)\}. \tag{29}$$

This distribution reflects the cost-tradeoffs among hypotheses by assigning high probability to low cost trajectories and low probability to high cost trajectories. The parameter $\alpha > 0$ adjusts how flat the distribution is: as α approaches 0 the distribution becomes increasingly uniform, and as α approaches ∞ the distribution becomes increasingly peaked around the global minimum.[4]

Hamiltonian Monte Carlo 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, we consider ξ 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. We can start a ball 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 (its initial potential energy), and how fast it starts (its initial kinetic energy). By changing the amount of total energy the ball starts with, we modulate the amount of time the ball spends moving quickly through local minima relative to the amount of time it spends pushing up into higher cost regions. If we set it off with very little energy, either by starting it low or giving it very little initial momentum (or both), the ball will easily get stuck in some local bowl and not have enough energy to escape. On the other hand, if we set it off will a lot of energy, either by placing it very high in the beginning or by throwing it very hard in some direction (or both), the ball will have enough energy to shoot out of local minima and visit many distinct regions as it travels.

[4] There are number of technical criteria, such as measurability and differentiability, that we gloss over in this presentation in favor of simplicity and stronger intuition. See [56] for the details.

The way we schedule these energy levels over time (i.e. whether we choose them randomly, monotonically decrease them to zero, or some combination of the two), dictates how we may interpret the behavior of the ball and the distribution of locations the ball visits in its lifetime. If we randomly catch the ball and send it off in some other direction with a random (normally distributed) speed, then one can show that the distribution of points where we catch the ball converges to the Gibbs distribution $p(\xi)$. On the other hand, if we consistently decrease the ball's energy by again catching the ball randomly and but this time sending it off again in random directions now with less energy on average, the ball will consistently be losing energy over time which means it will generally move downhill. Eventually, the ball will have essentially no energy at all, at which point it must be in a local minimum of the energy landscape. This energy profile defines an optimization procedure, one which starts by exploring a multitude of local minima before finally converging with high probability on a relatively deep basin.

The following sections formalize these ideas as Hamiltonian Monte Carlo [56, 57], in terms of both its use as a sampler (Section 5.2) and as an optimizer using simulated annealing (Section 5.4). Section 5.3 reviews some theoretical analysis behind the general Hamiltonian Monte Carlo algorithm and briefly derives the algorithm for a constant metric A.

5.2 Hamiltonian Monte Carlo

Let γ denote a variable that represents the momentum of a trajectory ξ. As we note above, in this context we think of the trajectory as a single infinite-dimensional point in the space of trajectory functions. Accordingly, the momentum refers to how the entire trajectory changes over time (such as how quickly it morphs from a straight-line into an S-shaped trajectory).

Following the analogy from above, the energy of the system is defined by both its potential energy $\mathcal{U}(\xi)$ and its kinetic energy $\mathcal{K}(\gamma) = \frac{1}{2}\gamma'\gamma$. We will review the basic algorithm in terms of Euclidean metrics here, and then later generalize it in Section 5.3.2 to arbitrary constant metrics A. Rather than addressing the marginal $p(\xi)$ directly, Hamiltonian Monte Carlo [56] operates on the joint probability distribution between the trajectory variable ξ and its momentum γ:

$$p(\xi, \gamma) \propto \exp\{-\mathcal{U}(\xi) - \mathcal{K}(\gamma)\} = \exp\{-H(\xi, \gamma)\}. \tag{30}$$

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 $\mathcal{K}(\gamma)$ 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:

$$\begin{cases} \frac{d\xi}{dt} = \gamma \\ \frac{d\gamma}{dt} = -\bar{\nabla}\mathcal{U}(\xi) \end{cases}. \tag{31}$$

Throughout this presentation we refer to these equations 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 defined as $\mathcal{U}(\xi)$ for our problem. A straightforward analysis of this system demonstrates that all integral curves conserve total energy (see Section 5.3.1). This observation indicates that if $(\xi(t), \gamma(t))$ is a solution to System 31 the value of the Hamiltonian $H(\xi(t), \gamma(t))$ is always the same independent of t.

In terms of our joint distribution in Equation 30, this constancy along solution paths implies these system solutions also trace out isocontours of the Hamiltonian $H(\xi, \gamma)$. Simulating the Hamiltonian

dynamics of the system, therefore, moves us 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 $p(\xi, \gamma) \propto \exp\{-\mathcal{U}(\xi)\} \exp\{-\mathcal{K}(\gamma)\}$. If computers were able to simulate 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}. Unfortunately, though, numerical inaccuracies in approximate integration may play a significant role. 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 [56]. The leapfrog method updates are given by

$$\begin{cases} \gamma_{t+\frac{\epsilon}{2}} = \gamma_t - \frac{\epsilon}{2}\bar{\nabla}\mathcal{U}(\xi_t) \\ \xi_{t+\epsilon} = \xi_t + \epsilon\gamma_{t+\frac{\epsilon}{2}} \\ \gamma_{t+\epsilon} = \gamma_{t+\frac{\epsilon}{2}} - \frac{\epsilon}{2}\bar{\nabla}\mathcal{U}(\xi_{t+\epsilon}) \end{cases}. \tag{32}$$

It is common to see these equations written as presented, but in practice it is more efficient when chaining multiple leapfrog steps together to combine the last half-step momentum update of the current iteration and the first half-step update of the next iteration to void extraneous function and gradient evaluations since those steps together simply amount to a full step update.

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

1. **Sample an initial trajectory momentum γ:** Sample a random initial trajectory momentum from the Gaussian formed by the marginal kinetic energy term $p(\gamma) \propto \exp\{-\mathcal{K}(\gamma)\} = \exp\{-\frac{1}{2}\gamma^T\gamma\}$.
2. **Simulate the system:** Simulate the system for a random number of iterations[5] using the leapfrog method of numerical integration given in Equation 32.
3. **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 $p(\xi_{t+1}, \gamma_{t+1})/p(\xi_t, \gamma_t)$.

5.3 Theoretical considerations

This section briefly explores some of the theoretical considerations describing both why the instantaneous update rule is correct and how we can properly account for arbitrary constant metrics A on the space of trajectories.

5.3.1 Brief analysis

By analyzing the time derivative of H, one can see that the instantaneous update rule in System 31 does not change the Hamiltonian $H(\xi, \gamma)$:

$$\begin{aligned} \frac{d}{dt}H(\xi,\gamma) &= \sum_{i=1}^{d}\left(\frac{d\xi_i}{dt}\frac{\partial H}{\partial \xi_i} + \frac{d\gamma_i}{dt}\frac{\partial H}{\partial \gamma_i}\right) = \sum_{i=1}^{d}\left(\frac{d\xi_i}{dt}\frac{\partial E}{\partial \xi_i} + \frac{d\gamma_i}{dt}\gamma_i\right) \\ &= \sum_{i=1}^{d}\left(\gamma_i\frac{\partial E}{\partial \xi_i} - \frac{\partial E}{\partial \xi_i}\gamma_i\right) = 0, \end{aligned} \tag{33}$$

[5] Technically, since Monte Carlo algorithms require reversibility, some presentations suggest at this point randomly choosing whether to simulate the system forward or backward in time. In our case, though, since the marginal over momenta is symmetric, randomly resampling the momentum at each iteration automatically provides reversibility.

where we arrive at Equation 33 by use the instantaneous update equations.

5.3.2 HMC with constant metric A

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:

$$\begin{cases} \frac{d\xi}{dt} = \gamma \\ \frac{d\gamma}{dt} = -A^{-1}\bar{\nabla}\mathcal{U}(\xi) \end{cases} . \tag{34}$$

We can see that this 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 $\widetilde{\xi} = A^{\frac{1}{2}}\xi$ and $\widetilde{\gamma} = A^{\frac{1}{2}}\gamma$ be a change of variable transformation such that Euclidean inner products in $\widetilde{\xi}$ and $\widetilde{\gamma}$ are A-inner products in the original space. The Euclidean update rules given by Equation 31 of this modified Hamiltonian in the transformed space are $\frac{d\widetilde{\xi}}{dt} = \widetilde{\gamma}$ and $\frac{d\widetilde{\gamma}}{dt} = -\bar{\nabla}_{\widetilde{\xi}}\mathcal{U}(A^{-\frac{1}{2}}\widetilde{\xi})$. Substituting the identities

$$\frac{d\widetilde{\xi}}{dt} = A^{\frac{1}{2}}\frac{d\xi}{dt}, \quad \frac{d\widetilde{\gamma}}{dt} = A^{\frac{1}{2}}\frac{d\gamma}{dt}, \quad \text{and} \quad \bar{\nabla}_{\widetilde{\xi}}\mathcal{U}(\xi) = A^{-\frac{1}{2}}\bar{\nabla}_{\xi}\mathcal{U}(\xi), \tag{35}$$

into the covariant system 34 reduces it to the Euclidean system in terms of $\widetilde{\xi}$ and $\widetilde{\gamma}$. Therefore, under these instantaneous update equations, H_A never changes.

Moreover, in terms of $\widetilde{\xi}$ and $\widetilde{\gamma}$ Hamiltonian Monte Carlo prescribes Euclidean sampling of the momenta. So, in terms of our original variables ξ and γ, since $\frac{1}{2}\widetilde{\gamma}^T\widetilde{\gamma} = \frac{1}{2}\gamma^T A\gamma$, the analogous rule under H_A is to sample from $p_A(\xi) \propto \exp\{-\frac{1}{2}\gamma^T A\gamma\}$. In retrospect, 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

1. **Sample an initial momentum:** Sample γ_t from $p_A(\gamma) \propto \exp\{-\frac{1}{2}\gamma^T A\gamma\}$.
2. **Simulate the system:** Use to leapfrog method to simulate the system in Equation 34.
3. **Reject:** If the final point is more probable than the initial point, accept it without question. Otherwise, reject it with probability $p_A(\xi_{t+1}, \gamma_{t+1})/p_A(\xi_t, \gamma_t)$.

5.4 Optimization and simulated annealing

Finally, we can use *simulated annealing* 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 $\alpha \to \infty$. 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 αs in small steps to coax the samples toward the minima incrementally over time (see [56]).

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 (including entire resamplings thereof) to efficiently skip past local minima and quickly converge on a globally strong solution.

6 Constraints

In many real-world problems, the ability to plan a trajectory from a starting configuration to a goal configuration that avoids obstacles is sufficient. However, there are problems that impose additional

constraints on the trajectory, like carrying a glass of water that should not spill, or lifting a box with both hands without letting the box slip. In this section, we derive an extension of the original CHOMP algorithm that can handle trajectory-wide equality constraints, and show its intuitive geometrical interpretation. We then focus on a special type of constraint, which only affects the endpoint of the trajectory. This type of constraint enables the optimizer to plan to a set of possible goals rather than the typical single goal configuration, which adds more flexibility to the planning process and increases the chances of converging to a low-cost trajectory.

6.1 Trajectory-Wide Constraints

We assume that we can describe a constraint on the Hilbert space of trajectories in the form of a nonlinear differentiable vector valued function $\mathcal{H} : \Xi \to \mathbb{R}^k$, for which $\mathcal{H}[\xi] = 0$ when the trajectory ξ satisfies the required constraints.

At every step, we optimize the regularized linear approximation of $\mathcal{U}$ from (18), subject to the nonlinear constraints $\mathcal{H}[\xi] = 0$:

$$\begin{aligned} \xi_{i+1} = \arg\min_{\xi \in \Xi} \; & \mathcal{U}[\xi_t] + \bar{\nabla}\mathcal{U}[\xi_i]^T(\xi - \xi_i) + \frac{\eta_i}{2}\|\xi - \xi_i\|_A^2 \\ s.t. \;\; & \mathcal{H}[\xi] = 0 \end{aligned} \tag{36}$$

We first observe that this problem is equivalent to the problem of taking the unconstrained solution in (19) and projecting it onto the constraints. This projection, however, measures distances not with respect to the Euclidean metric, but with respect to the Riemannian metric A. To show this, we rewrite the objective:

$$\min \mathcal{U}[\xi_i] + \bar{\nabla}\mathcal{U}[\xi_i]^T(\xi - \xi_i) + \frac{\eta_i}{2}\|\xi - \xi_i\|_A^2 \Leftrightarrow$$

$$\min \bar{\nabla}\mathcal{U}[\xi_i]^T(\xi - \xi_i) + \frac{\eta_i}{2}(\xi - \xi_i)^T A(\xi - \xi_i) \Leftrightarrow$$

$$\min \left(\xi_i - \frac{1}{\eta_i}A^{-1}\bar{\nabla}\mathcal{U}[\xi_i] - \xi\right)^T A \left(\xi_t - \frac{1}{\eta_i}A^{-1}\bar{\nabla}\mathcal{U}[\xi_i] - \xi\right)$$

.

The problem can thus be written as:

$$\begin{aligned} \xi_{t+1} = \arg\min_{\xi \in \Xi} \; & \|\xi_t \overbrace{- \frac{1}{\eta_i}A^{-1}\bar{\nabla}\mathcal{U}[\xi_i]}^{\textbf{unconstr. (19)}} - \xi\|_A^2 \\ s.t. \;\; & \mathcal{H}[\xi] = 0 \end{aligned} \tag{37}$$

This interpretation will become particularly relevant in the next section, which uncovers the insight behind the update rule we obtain by solving (36).

To derive a concrete update rule for (36), we linearize $\mathcal{H}$ around ξ_i: $\mathcal{H}[\xi] \approx \mathcal{H}[\xi_i] + \frac{\partial}{\partial \xi}\mathcal{H}[\xi_i](\xi - \xi_i) = C(\xi - \xi_i) + b$ where $C = \frac{\partial}{\partial \xi}\mathcal{H}[\xi_i]$ is the Jacobian of the constraint functional evaluated at ξ_t and $b = \mathcal{H}[\xi_i]$. The Lagrangian of the constrained gradient optimization problem in (36), now with linearized constraints, is $\mathcal{L}_g[\xi, \lambda] = \mathcal{U}[\xi_i] + \bar{\nabla}\mathcal{U}[\xi_i]^T(\xi - \xi_i) + \frac{\eta_i}{2}\|\xi - \xi_i\|_A^2 + \lambda^T(C(\xi - \xi_i) + b)$, and the corresponding first-order optimality conditions are:

$$\begin{cases} \bar{\nabla}_\xi \mathcal{L}_g = \bar{\nabla}\mathcal{U}[\xi_i] + \eta_i A(\xi - \xi_i) + C^T\lambda = 0 \\ \bar{\nabla}_\lambda \mathcal{L}_g = C(\xi - \xi_i) + b = 0 \end{cases} \tag{38}$$

Since the linearization is convex, the first order conditions completely describe the solution, enabling the derivation of a new update rule in closed form. If we denote $\frac{\lambda}{\eta_i} = \gamma$, from the first equation we get $\xi =$

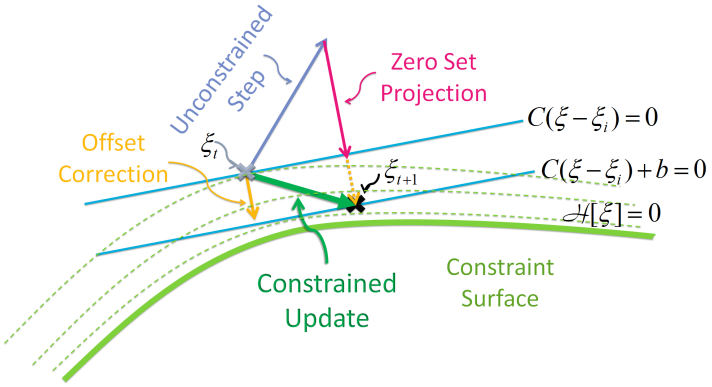


Figure 8: The constrained update rule takes the unconstrained step and projects it w.r.t. A onto the hyperplane through ξ_t parallel to the approximated constraint surface (given by the linearization $C(\xi - \xi_t) + b = 0$). Finally, it corrects the offset between the two hyperplanes, bringing ξ_{t+1} close to $\mathcal{H}[\xi] = 0$. *Image reproduced from* [21].

$\xi_i - \frac{1}{\eta_i} A^{-1}\bar{\nabla}\mathcal{U}[\xi_i] - A^{-1}C^T\gamma$. Substituting in the second equation, $\gamma = (CA^{-1}C^T)^{-1}(b - \frac{1}{\eta_i}CA^{-1}\bar{\nabla}\mathcal{U}[\xi_i])$. Using γ in the first equation, we solve for ξ:

$$\xi = \xi_i \overbrace{-\frac{1}{\eta_i}A^{-1}\bar{\nabla}\mathcal{U}[\xi_i]}^{\textbf{unconstr. (19)}} \overbrace{+\frac{1}{\eta_i}A^{-1}C^T(CA^{-1}C^T)^{-1}CA^{-1}\bar{\nabla}\mathcal{U}[\xi_i]}^{\textbf{zero set projection}} \overbrace{-A^{-1}C^T(CA^{-1}C^T)^{-1}b}^{\textbf{offset}} \tag{39}$$

The labels on the terms above hint at the goal of the next section, which provides an intuitive geometrical interpretation for this update rule.

6.2 Geometrical Interpretation

Looking back at the constrained update rule in (39), we can explain its effect by analyzing each of its terms individually. Gaining this insight not only leads to a deeper understanding of the algorithm, and relates it to an intuitive procedure for handling constraints in general. By the end of this section, we will have mapped the algorithm indicated by (39) to the projection problem in (37): take an unconstrained step, and then project it back onto the feasible region.

We split the update rule in three parts, depicted in Fig. 8: take the unconstrained step, project it onto a hyperplane that passes through the current trajectory and is parallel to the approximation of the constraint surface, and finally, correct the offset between these two hyperplanes:

1. The first term computes the **unconstrained** step: smooth the unconstrained Euclidean gradient $\bar{\nabla}\mathcal{U}[\xi_i]$ through A^{-1} and scale it, as in (19). Intuitively, the other terms will need to adjust this step, such that the trajectory obtained at the end of the iteration, ξ_{i+1}, is feasible. Therefore, these terms implement the projection onto the constraint with respect to A, as shown in (37).

2. Linearizing $\mathcal{H}$ provides an approximation of the constraint surface, given by $C(\xi - \xi_i) + b = 0$. The current trajectory, ξ_i, lies on a parallel hyperplane, $C(\xi - \xi_i) = 0$. When ξ_i is feasible, $b = 0$ and the two are identical, intersecting the constraint surface at ξ_i. What the second term in the update rule does is to **project** the unconstrained increment onto the **zero set** of $C(\xi - \xi_i)$ with respect to the metric A, as depicted in Fig. 8. Formally, the term is the solution to the problem that minimizes the adjustment to the new unconstrained trajectory (w.r.t. A) needed to satisfy

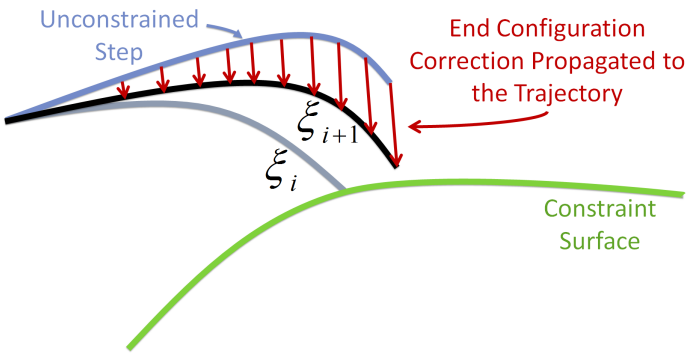


Figure 9: One iteration of the goal set version of CHOMP: take an unconstrained step, project the final configuration onto the constraint surface, and propagate that change to the rest of the trajectory. *Image reproduced from* [21].

$C(\xi - \xi_i) = 0$:

$$\min_{\Delta\xi} \ \frac{1}{2}\|\Delta\xi\|_A^2 \qquad (40)$$
$$s.t. \ \ C\left(\left(\xi_i - \frac{1}{\eta_i}A^{-1}g_t + \Delta\xi\right) - \xi_i\right) = 0$$

Therefore, the second term projects the unconstrained step onto the zero set of $C(\xi - \xi_i)$. If $b \neq 0$, the trajectory is still not on the approximation to the constraint surface, and the third step makes this correction.

3. The first two steps lead to a trajectory on $C(\xi - \xi_i) = 0$, at an **offset** from the hyperplane that approximates the feasible region, $C(\xi - \xi_i) + b = 0$. Even if the Euclidean gradient $\bar{\nabla}\mathcal{U}[\xi_i]$ is 0 and the previous two terms had no effect, the trajectory ξ_i might have been infeasible, leading to $b \neq 0$. The third term subtracts this offset, resulting in a trajectory that lies on the approximate constraint surface. It is the solution to the problem that minimizes the adjustment to ξ_i (again, w.r.t. the Riemmanian metric A) such that the trajectory gets back onto the target hyperplane:

$$\min_{\Delta\xi} \ \frac{1}{2}\|\Delta\xi\|_A^2 \qquad (41)$$
$$s.t. \ \ C\left(\ (\xi_i + \Delta\xi) - \xi_i\ \right) + b = 0$$

As Fig. 8 shows, adding the third term to the result of the previous two steps (ξ_i when the unconstrained step is zero, somewhere else along $C(\xi - \xi_i) = 0$ otherwise) brings the trajectory onto the approximation of the constraint surface.

Overall, the algorithm can be thought of as first taking an unconstrained step in the direction dictated solely by the cost function, and then projecting it onto its guess of the feasible region in two steps, the last of which aims at correcting previous errors. For the special case of endpoint constraints, which the next section addresses, the projection further simplifies to a purely Euclidean operator, which is then smoothed through the matrix A.

6.3 Goal Set Constraints

Goal sets are omnipresent in manipulation: picking up objects, placing them on counters or in bins, handing them off — all of these tasks encompass continuous sets of goals. Sampling-based planners do exist that can plan to a goal set [5]. However, the CHOMP algorithm described thus far plans to a single goal configuration rather than a goal set. This single goal assumption limits its capabilities: goal sets enlarge the space of candidate trajectories, and, as Sec. 7.3 will show, enable the optimizer to converge

to better solutions. In this section, we use the trajectory parametrization from Sec. 3.4 to explain the algorithm above in the context of goal sets.

In order to exploit goal sets, the trajectory endpoint, which is a constant in the original CHOMP, becomes a variable. That is, we use trajectory functions ξ defined on $(0, 1]$ as opposed to $(0, 1)$. When using vectors of waypoints as a trajectory parametrization, this leads to a small change in the finite differencing matrix K and the vector e from Sec. 3.4.

The goal set variant of CHOMP thus becomes a version of the constrained CHOMP from (36), in which the trajectories satisfying $\mathcal{H}(\xi) = 0$ (the constraint function under the waypoint parametrization) are the ones that end on the goal set. Constraints that affect only the goal are a special case of trajectory constraints, for which $\mathcal{H}(\xi) = \mathcal{H}_n(q_n)$ (the constraint is a function of only the final configuration of the trajectory). Therefore, a large portion of the update rule will focus on the last configuration. Since $C = [0, \ldots, 0, \widetilde{C}]$, in this case C only affects the last block-row of A^{-1}, which we denote by $B_n \in \mathbb{R}^{d \times nd}$. Also note that the last $d \times d$ block in A^{-1}, B_{nn}, is in fact equal to βI_d, since there are no cross-coupling terms between the joints. Therefore, the update rule becomes:

$$\xi_{i+1} = \xi_i - \frac{1}{\eta_i} A^{-1} \nabla \mathcal{U}(\xi_i) + \frac{1}{\eta_i \beta} B_n^T \widetilde{C}^T (\widetilde{C} \widetilde{C}^T)^{-1} \widetilde{C} B_n \nabla \mathcal{U}(\xi_i) + \frac{1}{\beta} B_n^T \widetilde{C}^T (\widetilde{C} \widetilde{C}^T)^{-1} b \tag{42}$$

Although not the simplest version of this update rule, this form lends itself to an intuitive geometrical interpretation. The goal set CHOMP algorithm, as depicted in Fig. 9, follows the "take an unconstrained step and project it" rule, only this time the projection is much simpler: it is a configuration-space projection with respect to the *Euclidean metric*, rather than a trajectory-space projection with respect to the *Riemmanian metric* A. The same projection from Fig. 8 now applies only to the end-configuration of the trajectory. To see this, note that $\frac{1}{\eta_i} B_n \nabla \mathcal{U}$ gets the unconstrained step for the end configuration from $\frac{1}{\eta_t} A^{-1} \nabla \mathcal{U}$, a $d \times d$ vector, and $\widetilde{C}^T (\widetilde{C} \widetilde{C}^T)^{-1} \widetilde{C}$ projects it onto the row space of $\widetilde{C}$. This correction is then propagated to the rest of the trajectory, as illustrated by Fig. 9, through $\frac{1}{\beta} B_n^T$. The entries of B_n, on each dimension, interpolate linearly from 0 to β. Therefore, $\frac{1}{\beta} B_n^T$ linearly interpolates from a zero change at the start configuration to the correction at the end point. Since B_n^T multiplies the last configuration by β, $\frac{1}{\beta}$ scales everything down such that the endpoint projection applies exactly.

So far in this section, we showed that the projection onto a linearized version of the goal set constraint simplifies to a two step procedure. We first project the final configuration of the trajectory onto the linearized goal set constraint with respect to the Euclidean metric in the configuration space, which gives us a desired perturbation Δq_n of that final configuration. We then smooth that desired perturbation linearly back across the trajectory so that each configuration along the trajectory is perturbed by a fraction Δq_n; in particular for our prior, the perturbation at the configuration at t is $\Delta q_t = t/n \Delta q_n$. One can show that the linearization of the constraints is not required in general. For any goal set, the projection of a trajectory onto the set may be decomposed as prescribed above.[6]

7 Experimental Results on Manipulator Planning Tasks

In this section, we evaluate CHOMP's performance on common motion planning problems for a typical robotic manipulator. Such problems are posed in a high-dimensional state space (e.g. that of 7-dimensional arm configurations), but the distribution of obstacles in the robot's workspace is structured.

We begin by focusing on the problem of planning to a pre-grasp pose among clutter (Section 7.1). We show that CHOMP can solve most such tasks quickly and obtain good solutions, and compare its performance to sampling-based algorithms like RRT or the more recent, optimality-driven, RRT* [39]. CHOMP has a lower success rate than the RRT; however, when it succeeds it does so more quickly, and

[6] Specifically, one can show that for any single point in the goal set, the magnitude of the optimal projection is proportional to the Euclidean distance between the final configurations of the original trajectory and the projected trajectory. Therefore, the optimal projection is dictated by that Euclidean distance. Moreover, the optimal projection onto each of those given points in the goal set is given by the smoothing procedure described above.

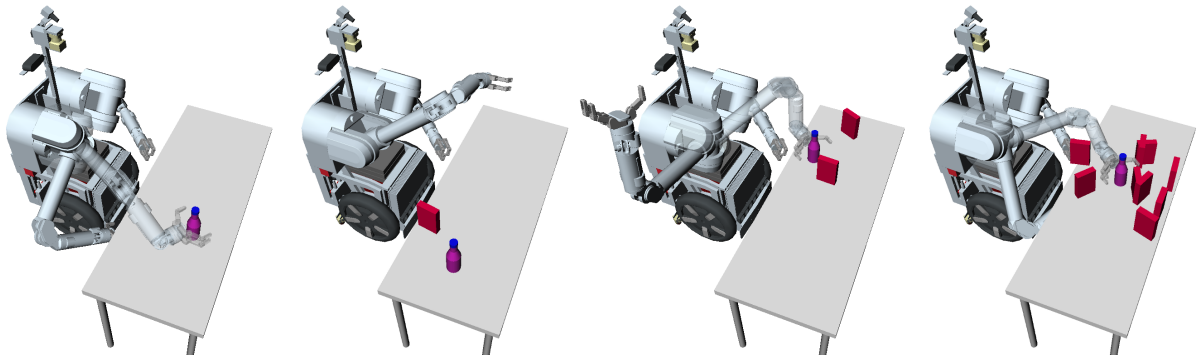


Figure 10: Grasping in clutter scenes, with different starting configurations, target object locations, and clutter distribution (from left to right: no clutter, low, medium and high clutter).

directly produces a trajectory of much lower cost. We also show that when initializing the deterministic version of CHOMP with a straight-line trajectory fails, using Hamiltonian Monte Carlo leads to successful, low-cost trajectories.

We then focus on extensions to CHOMP which allow it to be used for a more general class of problems. In Section 7.3, we apply the constrained planning approach formalized in Section 6 to goal sets, enabling the the algorithm to converge to better solutions. In Section 7.5, we show that certain modifications for efficiency enable CHOMP to perform trajectory replanning in dynamic environments.

7.1 Comparison on Common Motion Planning Tasks for Manipulation

Here, we focus on a motion planning problem which arises in common manipulation tasks: *planning to a pre-grasp pose among clutter*. A pre-grasp pose is an arm configuration which positions the hand in a pre-grasp position relative to an object. To solve such problems in high-dimensional spaces, a historical dichotomy exists between trajectory optimization (e.g. CHOMP) and sampling-based approaches (e.g. the Rapidly-exploring Random Tree). Recently, algorithms such as RRT* have brought optimization to sampling-based planners. We are interested in evaluating the performance of CHOMP on motion planning problems commonly encountered in real-world manipulation domains, and comparing it with such sampling-based approaches.

It is important to observe that the experiments presented in this section are not "apples to apples" comparisons in that we are juxtaposing a (local) optimization algorithm with global randomized search algorithms. Obviously the effectiveness of our approach depends strongly upon the inherent structure underlying the planning problem, including the sparsity and regularity of obstacles. Certainly, there exist any number of maze-like motion planning problems for which CHOMP is ill-suited. However, as we hypothesize below, it can come to fill some of the space which has until recently been occupied by sampling-based methods; hence, we feel the comparison between heterogeneous systems is well motivated.

7.1.1 Experimental Design

We explore day-to-day manipulation task of grasping in cluttered environments. For the majority of our experiments, we use a canonical grasping in clutter problem: the robot is tasked with moving to grasp a bottle placed on a table among a varying number of obstacles, as in Fig. 10.

We test the following hypotheses:

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.*

We compare CHOMP, RRT and RRT*. To ensure fairness of the comparison, we conduct the experiments in a *standard* simulation environment – OpenRAVE [17] 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[7]. We use HERB, the Home Exploring Robot Butler, as our experimental platform. We run each algorithm on each problem 20 times, for 20 seconds each (single-thread CPU time). The RRT shortcuts the path (using the shortcutting method available in OpenRAVE) until the final time is reached. We measure at each time point if the algorithm has found a *feasible* solution, and we use 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 (which correlates to, but is different from, path length), while pulling the trajectory far from obstacles.

We created grasping in clutter problems with varying features: starting configurations, target locations and clutter distributions. We split the problems into a training and testing set, such that no testing problem has any common features with a training one. This is important, as it tests true generalization of the parameters to different problems. We used the training set to adjust parameters for all algorithms, giving each the best shot at performing well. We had 170 testing problems, leading to 3400 runs of each algorithm. Below, we present the results for the deterministic version of CHOMP vs. RRT, and then discuss the benefits of HMC and the comparison with RRT*.

7.1.2 Time to Produce a Feasible Solution

Validating **H1**, CHOMP (the deterministic version) succeeded on about 80% of the problems, with an average time of 0.34s ($SEM = 0.0174$). On problems where both CHOMP and RRT succeed, CHOMP found a solution 2.6 seconds faster, and the difference is statistically significant (as indicated by a paired t-test, $t(2586) = 49.08$, $p < 0.001$). *Overall, CHOMP has a lower success rate than an RRT on these problems. When it does succeed, it does so faster.* See Fig. 11 for the paired time comparison.

The CHOMP times do not include the time to compute the Signed Distance Field from the voxelized world (which the robot acquires in practice through a combination of cached voxelized static environments and voxel grids obtained online via laser scans). The SDF computation takes an average of 0.1 seconds.

7.1.3 Collision Checking - The Grain of Salt

The time taken by the RRT heavily depends on the time it takes to perform collision checks. Our implementation uses OpenRAVE for collision checking, and the average time for a check was approximately 444 microseconds (averaged over 174 million checks) – see Fig. 12 for the distribution. This is faster than the times reported in the benchmark comparison from [66] for an easier problem, indicating that our results for the RRT are indicative of its typical performance. However, the RRT may be improved with recent, more advanced collision checkers (e.g. FCL [58]). *For example, if collision checking were 5 times faster (an optimistic estimate for state-of-the-art performance), 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 we will see in following section, would remain valid: for many problems (namely 78%), CHOMP produces a low-cost feasible trajectory in the same time that an RRT produces a high-cost feasible trajectory.

7.1.4 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, we retrieved the CHOMP trajectory from the same problem at the time point when the RRT obtained a solution. 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**).* See Fig. 13 for the cost vs. cost comparison. Note that that the CHOMP trajectories evaluated here were not the ones with the smallest path length: the algorithm is optimizing a combination of a smoothness and an obstacle cost. Therefore, CHOMP is increasing the path length even after the trajectory becomes feasible, in order to keep it far from obstacles.

[7]The bug fix for RRT* (path improvement) in 0.11.1 did not alter the results on our problems, for our time intervals. We did verify that the issue was indeed fixed by finding problems on which the RRT* did eventually improve the path.

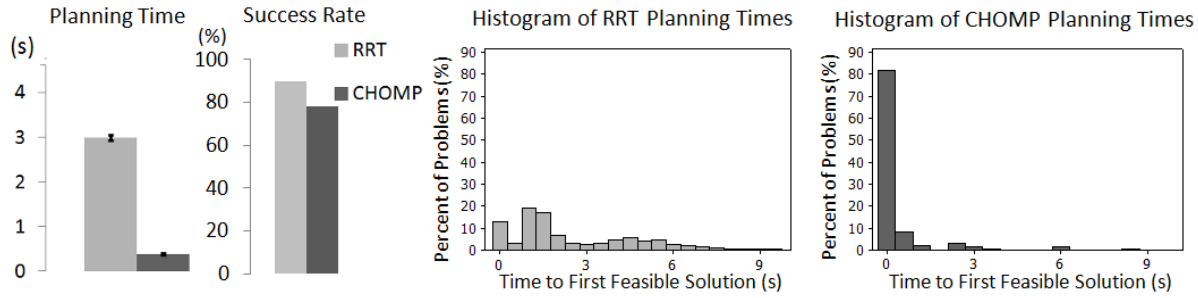


Figure 11: From left to right: a paired time comparison between RRT and CHOMP when both algorithms succeed, success rates for both algorithms within the 20 s time interval and the planning time histograms for both algorithms. In the time comparison chart on the left, each data point is one run of the RRT algorithm vs. the discrete run of CHOMP on a problem. Due to the large number of data points, the standard error on the mean is very small.

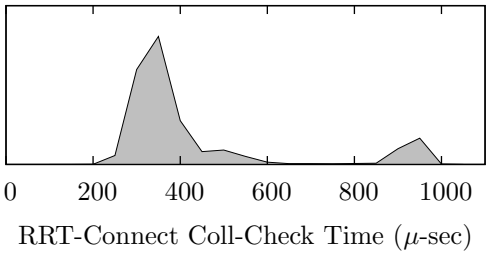


Figure 12: Distribution of per-problem average collision times over all test problems.

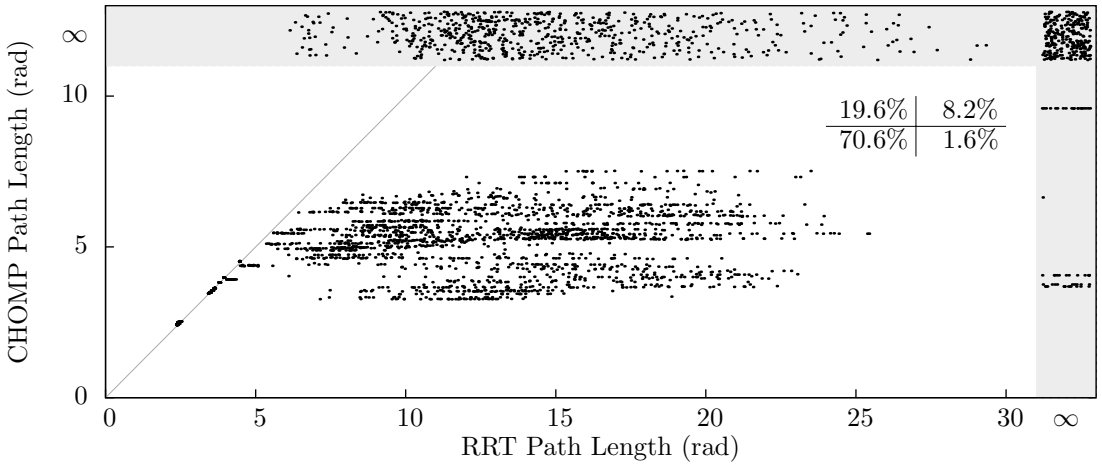


Figure 13: Comparison of path lengths of CHOMP and RRT at the RRT first-feasible time. Each RRT run is paired with the deterministic CHOMP run for its problem, and the pair is plotted here by path length. Infeasible paths are assigned an infinite path length; the grey bands show the distribution of lengths when one of the results was infeasible. The table also shows the distribution of feasible paths found by each planner.

Time Budget	Success (Percentage) RRT	CHOMP	Average Path Length (radians) RRT	CHOMP
1 s	16.5	24.7	4.37	3.69
2 s	47.0	68.2	6.85	4.89
3 s	57.1	70.6	6.64	4.94
5 s	66.3	74.1	6.69	5.00
10 s	88.0	74.7	6.79	5.03
20 s	90.2	74.7	6.58	5.03

Table 1: Comparison of CHOMP and RRT for different time budgets.

7.1.5 Time Budgets

In practice, planners are often evaluated within fixed time budgets. Here, we allow each planner a fixed planning time, and evaluate its result (for both feasibility and path length). We found that the relative performance of CHOMP and RRT depends greatly on the time budget allotted (and of course, on the collision checker). For CHOMP, we run iterations 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. We evaluated time budgets of 1, 2, 3, 5, 10, and 20 s. The summary of these results is shown in Table 1. Snapshots of the distribution of each pair of planning runs are also shown in Fig. 14.

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 plateaus quickly, and does not exceed 75% for any budget. The addition of HMC can improve this performance further (see Section 7.1.6). 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.

7.1.6 HMC

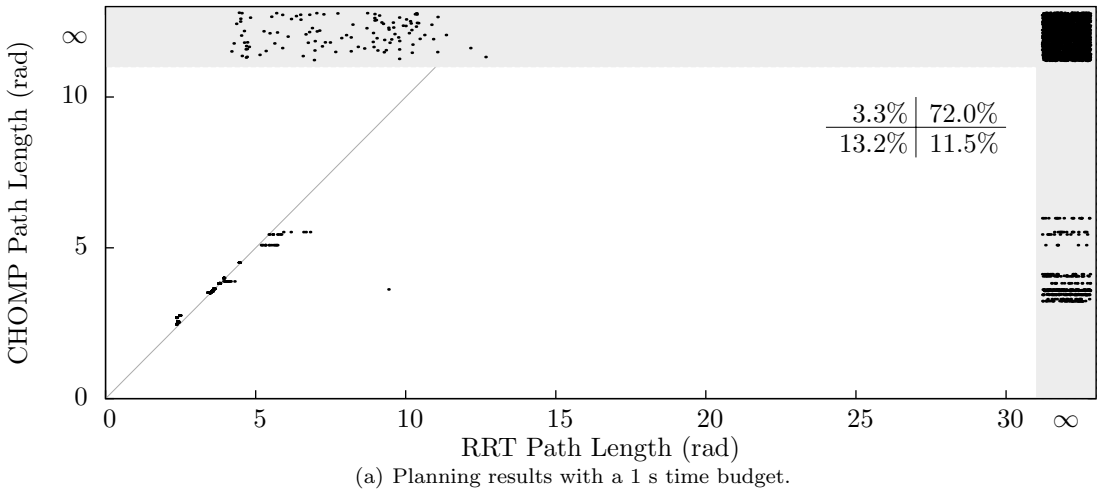
The deterministic CHOMP algorithm resulted in a feasible path after a 20 s time budget with a 74.7% success rate (see Fig. 15) on our testing suite of clutter problems. A further 5.9% of problems yielded no successful paths at any point during the runs of any of the planners we tested, and are thus classified as unsolvable here. The Hamiltonian Monte Carlo component of CHOMP aims to improve performance on the remaining 19.7% of the cases.

We implemented HMC by adding a trajectory momentum term to the optimizer as described in Section 5. 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 $p(\gamma) \propto \exp\left(-\frac{1}{2}\alpha\gamma^T\gamma\right)$, with the parameter α increasing exponentially as $\alpha = 100\exp(0.02\,k)$ with k the iteration number.

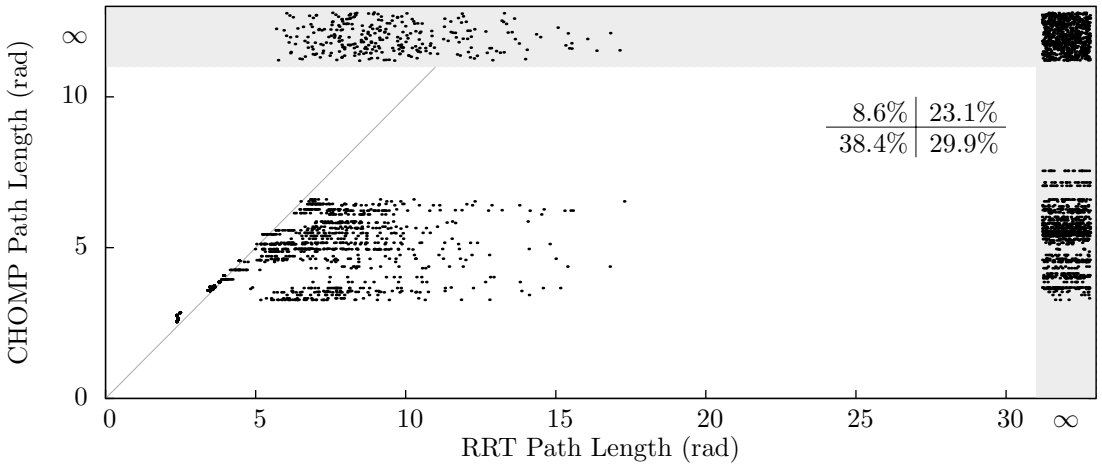
On those solvable problems on which deterministic CHOMP failed, the addition of HMC resulted in a 56% success rate during the 20 s time budget. Fig. 15 shows an example problem and also illustrates the behavior of the algorithm.

7.1.7 RRT*

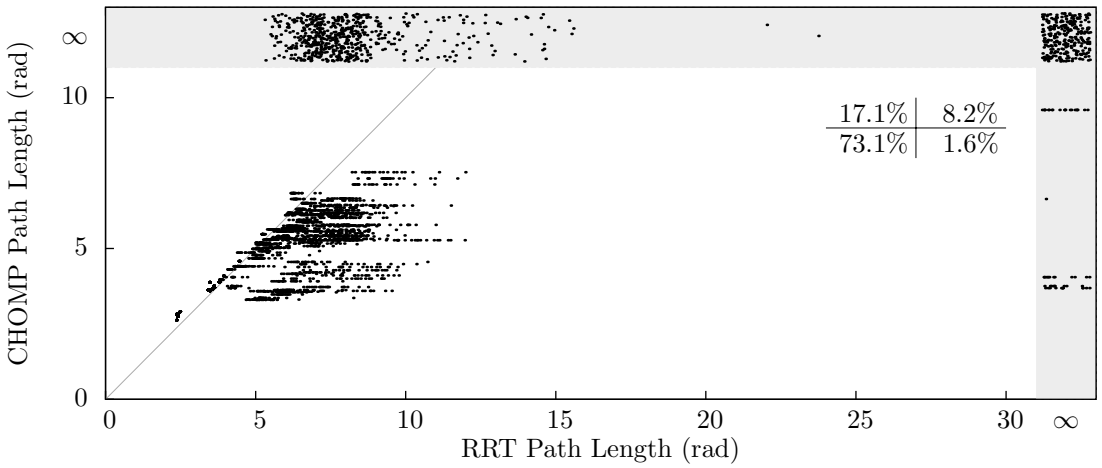
We compared the performance of CHOMP and Bi-Directional RRT-Connect to the RRT* implementation in OMPL. The RRT* range (step size) parameter was set equal to that of the RRT (corresponding to a workspace distance of 2 cm). We chose other algorithm parameters (goal bias, ball radius constant, and max ball radius) as directed by the implementation documentation. RRT* had a 5.97% success rate on our testing suite of clutter problems. When it did succeed, it found its first feasible solution after an



(a) Planning results with a 1 s time budget.

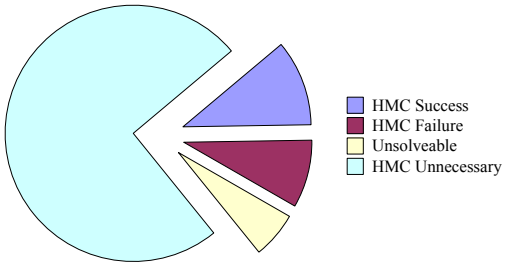


(b) Planning results with a 2 s time budget.

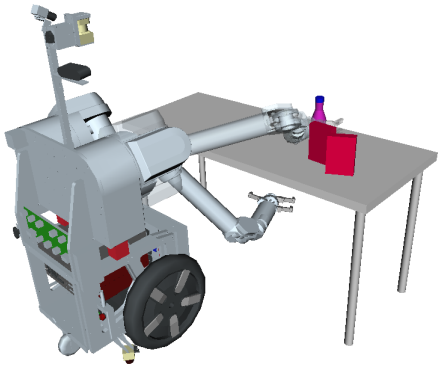


(c) Planning results with a 20 s time budget.

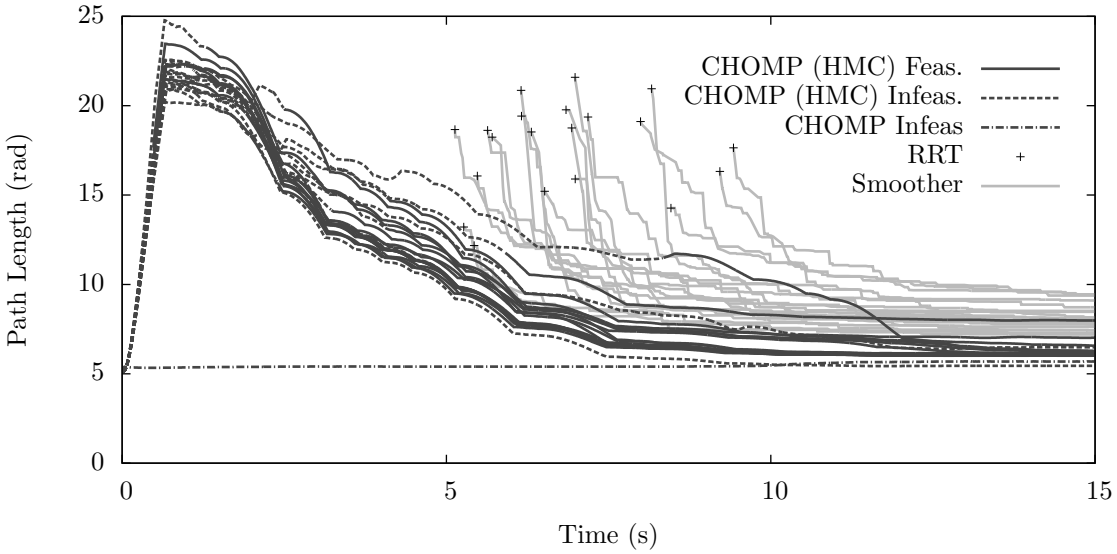
Figure 14: The performance of CHOMP and RRT on the testing set of clutter problems for different time budgets. Infeasible (CHOMP) or unfound (RRT) results are assigned an infinite length, partitioning the data as shown in the table. For short budgets (a), the planners find short paths for some problems, but fail for most. As the budget is increased (b), CHOMP can solve more problems, also yielding significantly shorter paths. Long budgets (c) favor the RRT, with a better success rate and improved path lengths.



(a) Summary of HMC results on clutter problems.



(b) Example problem requiring HMC.



(c) Comparison of CHOMP with and without HMC to RRT over a 20 s time budget.

Figure 15: In cases where the deterministic CHOMP algorithm is unable to find a feasible solution due to local minima, the addition of Hamiltonian Monte Carlo sampling is often able to improve performance. Restricted to the 33 solvable problems on which deterministic CHOMP failed (Fig 15(a)), HMC resulted in a feasible path 56% of the time. Fig 15(b) shows an example problem for which HMC was necessary; the problem is difficult because the initial trajectory passes directly through the table. Fig 15(c) shows the time performance of the RRT and CHOMP (with and without HMC) on the problem from Fig 15(b). Deterministic CHOMP falls immediately into an infeasible local minimum. Driven by significant initial momenta, each run of CHOMP with HMC begins by initially exploring different parts of the space of trajectories before settling down. In this problem, 18 of the 20 runs resulted in feasible trajectories. The performance of the RRT is also shown for comparison.

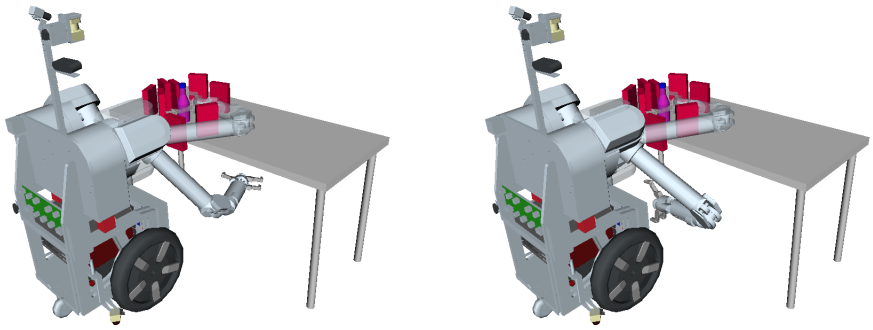


Figure 16: An easy (left) and a difficult (right) starting configuration. Although they might seem very similar, the local gradient information leads CHOMP to vastly different results: while it can solve 91% of problems using the first configuration as the start, it can only solve 55% of those using the second one. The RRT is less affected by this, with a difference in success rate of 15% instead of CHOMP's 36%.

average of 6.34 s, and produced an average path length of 11.64 rad. On none of our testing problems was it able to improve its first path within the 20 s time budget (although we did verify that for other problems, this does happen with a long enough time budget).

7.1.8 Problem Type Effects on Performance

Our problems varied in the starting configuration, target object location, and clutter amount. We also chose a goal configuration for each problem from a goal set of feasible grasps of the target object. For this final stage of our analysis, we are interested in capturing the effects of such environment features on the performance of CHOMP. We used these features as factors in an ANOVA with the amount of time taken by CHOMP to produce a feasible trajectory as the response, and found that the fit was rather poor ($R^2 = 26.24\%$). This indicates that *the complexity of the problem is far more subtle than, for example, a general notion of how much clutter is found around the goal, or how far the starting configuration is.* The fit was better when using path length as a response ($R^2 = 51.9\%$), and all factors had significant effects. The result is intuitive: the length of the final path produced by seeding deterministic CHOMP with a straight line trajectory is in large determined by the start and goal configurations.

We also analyzed the effects of these features on the success rate of CHOMP. We found that the starting configuration has a large effect on this measure, with an easy starting configuration leading to a 91% success rate, and a difficult one to merely 55%. What is more, the difference is not intuitive to the naked eye (see Fig. 16): covariant gradients in high-dimensional spaces can have unintuitive effects.

7.2 Beyond Grasping in Clutter

Our 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, we investigated the algorithm's performance on the problem setup depicted in Fig. 17: reaching to the back of a narrow microwave placed in a corner, with little free space for the arm to move through. We ran CHOMP and BiRRT-Connect for 8 different scenarios (with different start and goal IK configurations). 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 (requiring a speed up of 60x to make the RRT first-feasible time equal to that of CHOMP).

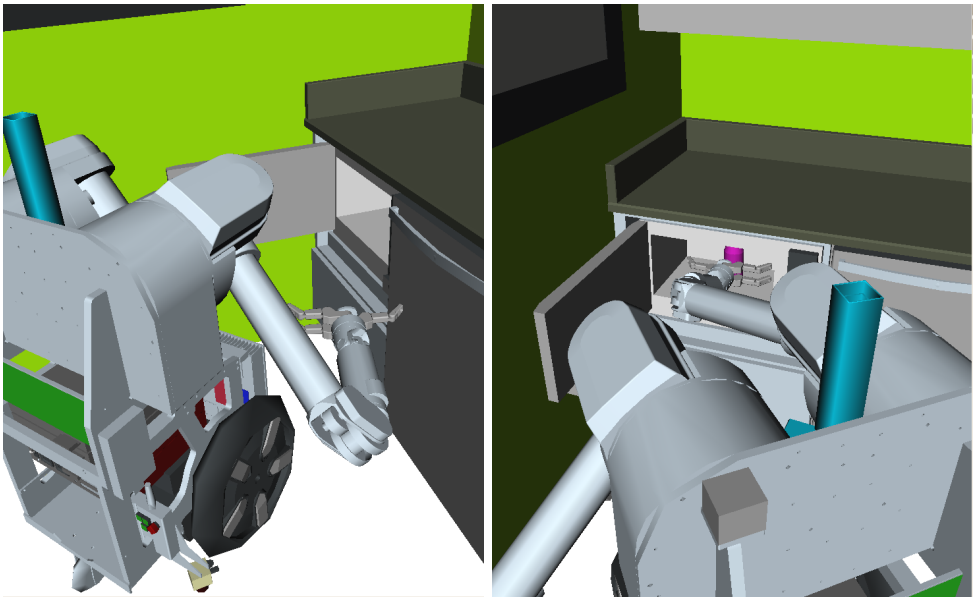


Figure 17: The start and the goal for a complex problem of reaching into the back of a narrow microwave. The robot is close to the corner of the room, which makes the problem particularly challenging because it gives the arm very little space to move through. The goal configuration is also very different from the start, requiring an "elbow flip". Two starts were used, one with a flipped turret (e.g. J1 and J3 offset by π, and J2 negated), leading to very different straight-line paths.

7.3 Goal Set Constraints

Sec. 6 derived the CHOMP algorithm under trajectory-wide constraints, and analyzed the particular case of constraints that affect only the endpoint of the trajectory. This type of constraint enables relaxing the constant goal assumption made in Sec. 3 and allows the optimizer the flexibility of a set of goal configurations.

7.3.1 Experimental Setup

Hypothesis: *Taking advantage of the goal set describing manipulation tasks during optimization results in final trajectories with significantly lower cost.*

We use a 7-DOF Barrett WAM mounted atop of a Segway base for most of this section of our experiments. To ensure a fair comparison, we use the same parameter set for both algorithms. We also restrict ourselves to deterministic optimization: these experiments run CHOMP from a straight line trajectory and do not use HMC.

We focus on day-to-day manipulation tasks, and define four types of tasks: grasping in cluttered scenes with both an easy and a difficult starting pose of the arm, handing off an object, or placing it in the recycle bin – see Fig. 18. We set up various scenarios that represent different obstacle configurations and, in the case of hand-offs and recycling, different initial poses of the robot. Each scenario is associated with a continuous goal set. e.g. the circle of grasps around a bottle, or the rectangular prism in workspace that ensures the object will fall into the bin. We compare the algorithms starting from straight line trajectories to each goal in a discretized version of this set. This reduces the variance and avoids biasing the comparison towards one algorithm or the other, by selecting a particularly good goal or a particularly bad one. For each scenario and initial goal, we measure the cost of the final trajectory produced by each algorithm.

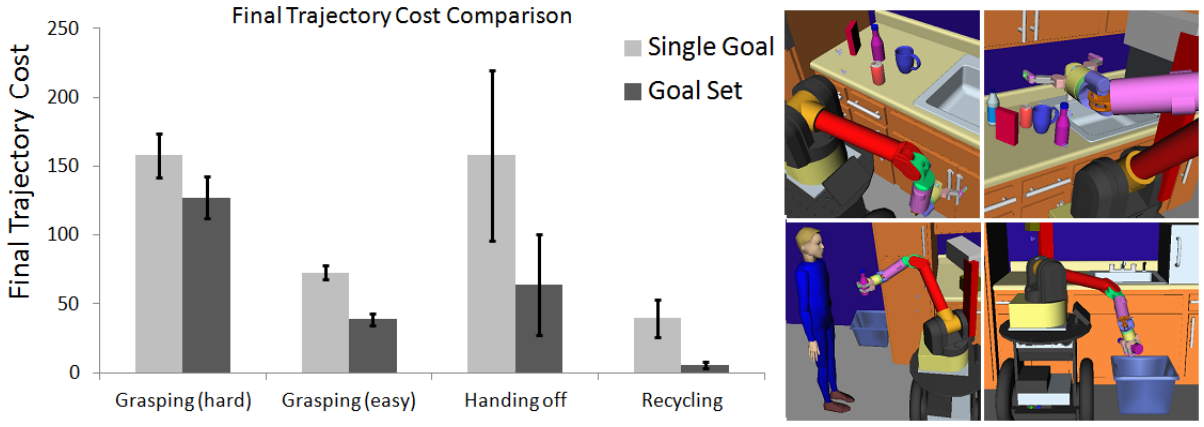


Figure 18: A cost comparison of the single goal with the goal set variant of CHOMP on problems from four different environment types: grasping in clutter from a difficult, and from an easy starting configuration, handing off an object, and placing it in the recycle bin.

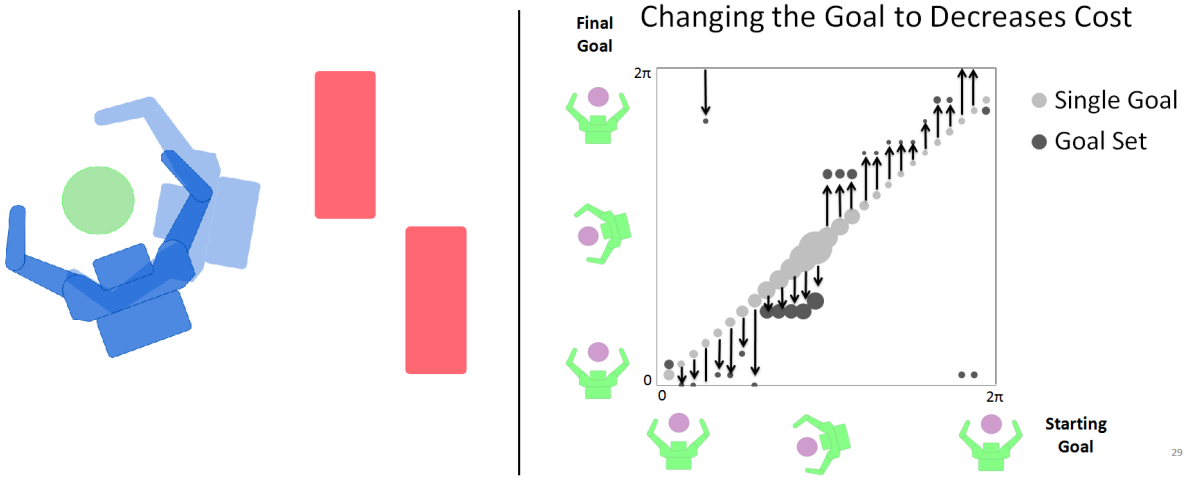


Figure 19: The goal set algorithm modifies the trajectory's goal in order to reduce its final cost. The figure plots the initial vs. the final goals obtained by the single goal and the goal set algorithm on a grasping in clutter problem. The area of each bubble is proportional to the cost of the final trajectory.

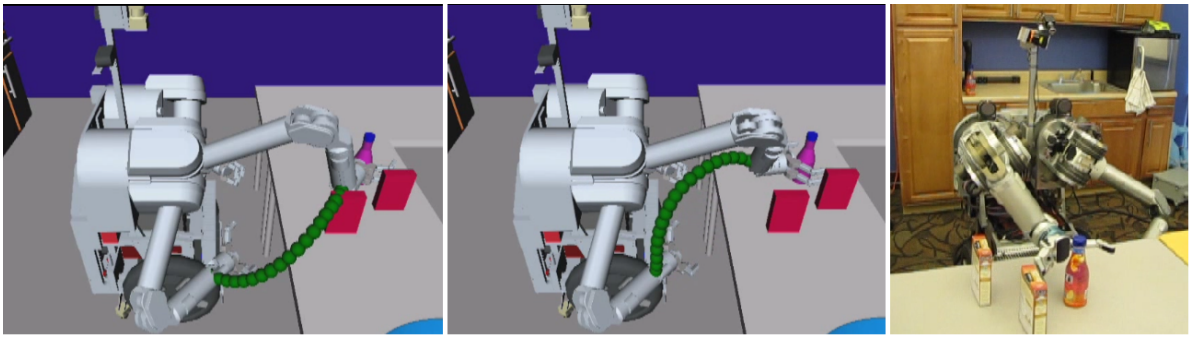


Figure 20: The end effector trajectory before and after optimization with Goal Set CHOMP. The initial (straight line in configuration space) trajectory ends at a feasible goal configuration, but collides with the clutter along the way. The final trajectory avoids the clutter by reaching from a different direction.

7.3.2 Results and Analysis

We ran CHOMP and Goal Set CHOMP for various scenarios and goals, leading to approximately 1300 runs of each algorithm. Fig. 18 shows the results on each task type: the Goal Set Algorithm does achieve lower costs. We used a two-tailed paired t-test on each task to compare the performances of the two algorithms, and found significant differences in three out of the four: on all task but grasping in clutter from a hard starting configuration, taking advantage of goal sets led to significantly better trajectories ($p < 0.05$). Across all tasks, we found a 43% improvement in cost in favor of Goal Set CHOMP, and the difference was indeed significant ($p < 0.001$), confirming our hypothesis.

We did find scenarios in which the average performance of Goal Set CHOMP was in fact worse than that of CHOMP. This can be theoretically explained by the fact that both these algorithms are local methods, and the goal set one could make a locally optimal decision which converges to a shallower local minima. At the same time, we do expect that the average performance improves by allowing goal sets. A further analysis of these scenarios suggested a different explanation: although on most cases the goal set version was better, there were a few runs when it did not converge to a "goal-feasible" trajectory (and therefore reported a very high cost of the last feasible trajectory, which was close to the initial one). We noticed that this is mainly related to the projection being impeded by joint limits. Formalizing joint limits as trajectory constraints and projecting onto both constraint sets at the same time would help mediate this problem.

Fig. 19 shows one of the successful scenarios. Here, the target object is rotationally symmetric and can be grasped from any direction. The figure depicts how Goal Set CHOMP changed the grasp direction and obtained lower costs (as indicated by the size of the bubbles). The next figure, Fig. 20, shows a similar setup for a different mobile manipulator. Although the initial trajectory ends at a collision-free goal, it intersects with the clutter. Goal Set CHOMP converges to a trajectory ending at a goal in free space, which is much easier to reach.

7.4 Trajectory-Wide Constraints

Our experience with trajectory-wide constraints on CHOMP has been mixed. CHOMP does successfully find collision-free trajectories that abide by such constraints, as the theory shows. For example, we solved the task of bimanually moving a box by enforcing a fixed relative transform between the robot's hands, and the task of keeping an object upright while extracting it from the microwave (Fig. 21). However, this is computationally expensive when the constraint affects all points along the trajectory. Every iteration requires the inversion of a new matrix $CA^{-1}C^T$, an $O((nd)^{2.376})$ operation (where n is the number of trajectory points and d is the dimensionality of the constraint at each point). For example, for the task in Fig. 21, d is 2 and CHOMP solves the problem in 17.02 seconds.

Furthermore, handling joint limits separately, as CHOMP usually does, can sometimes oppose the constraint projection: joint limits need to also be handled as hard constraints, and the unconstrained

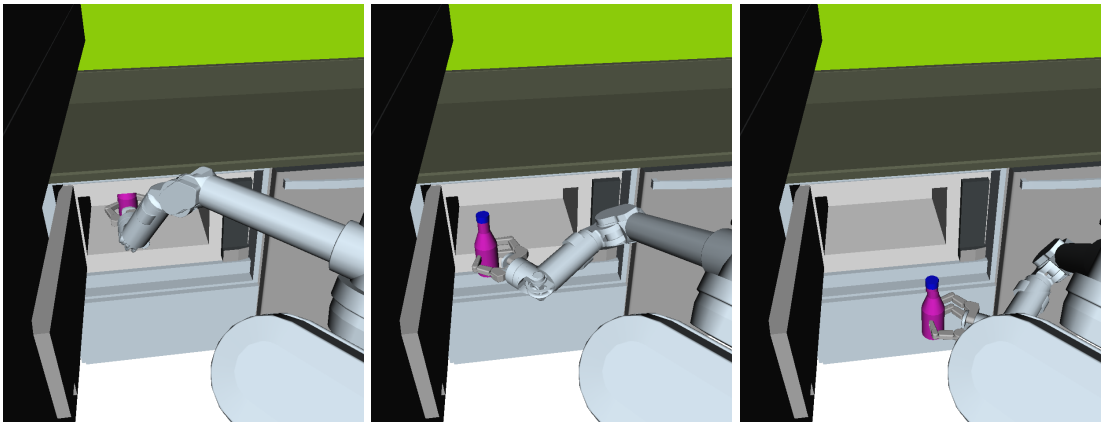


Figure 21: The trajectory obtained by CHOMP for extracting the bottle from the microwave while keeping it upright (a trajectory-wide constraint).

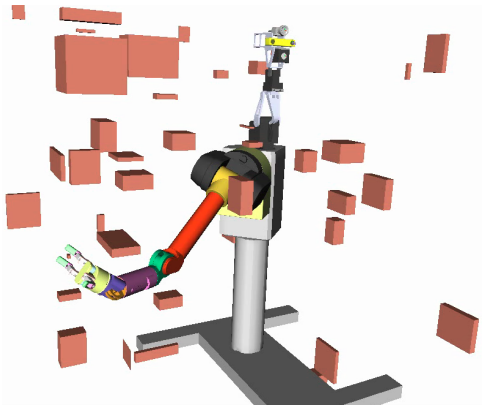


Figure 22: Comparison with a standard configuration of CHOMP by Ratliff et al. [65] was performed in randomized simulated clutter scenes similar to this one.

step needs to be projected on both constraints at once.

7.5 Fast Replanning Configuration

Simulated and real-robot experimental results are presented in the context of efficient replanning due to unforeseen changes in the environment or moving obstacles. To this end, we configure the CHOMP algorithm with pre-computed compositional distance fields (Section 4.1.2) and weight scheduling (Section 3.6). This configuration, particularly well suited for replanning, is henceforth referred to as CHOMP-R. We first demonstrate that this configuration performs well with respect to the original formulation of CHOMP described in [65]. Second, we apply it to replanning on a physical robot and demonstrate fast real-time replan rates in realistic scenarios.

In comparing to the original CHOMP, we utilized 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, as illustrated in Figure 22. 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, i.e. *baseline*. 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 in Section 4.1.2.

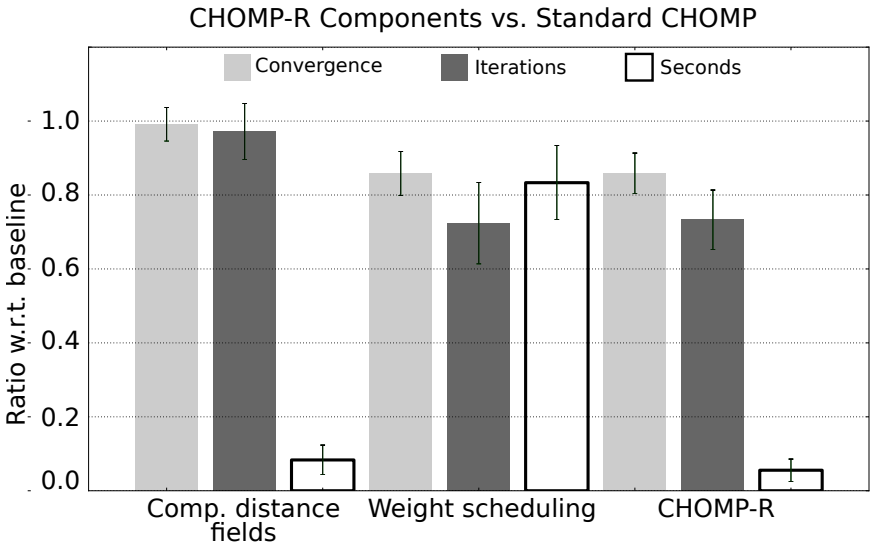


Figure 23: A replanning configuration of CHOMP, referred to as CHOMP-R, is compared to the original configuration of the algorithm described in [65], here considered the *baseline*. Two major components of CHOMP-R – compositional distance fields (left) and weight scheduling (center) – are compared to the baseline individually, along with their combination (right). The *convergence* category (leftmost bar in each set) is the reciprocal of the ratio of successful planning queries, such that *lower* bar values represent more attractive performance throughout the plot.

- The cost function weights were scheduled as described in Section 3.6.
- CHOMP-R: both object distance primitives and weight scheduling were combined.

Figure 23 demonstrates how the three planner configurations compare to baseline. It uses three measures for the comparison: the (reciprocal of) ratio of the planning queries that were solved by a planner (failure means exceeding 500 iterations), the number of iterations a planner took and the amount of time (in seconds) that was required to find a solution. These measures are color-coded in the figure as light gray, dark gray and white, respectively, and presented as ratios: each of the quantities was measured with each of the variants above and divided by the respective quantity measured using baseline.

In particular, the three bars on the left of the plot demonstrate that replacing the standard distance field with the compositional one does not affect performance, except by requiring far less runtime (in seconds) for each planning query. A reduction of over an order of magnitude is attained by virtue of eliminating the need to re-compute the distance field on-line during planning.

The advantage of the cost function weight scheduling is demonstrated in the center of the plot. The improvement is two-fold: both in the greater ratio of queries solved and in reducing the number of iterations – with an advantage of about 20% in each category. Runtime in seconds is also lower as a consequence of the reduction in iterations.

Lastly, the three bars on the right of the figure compare CHOMP-R, the combination of the two features compares thus far, to baseline. This instance of the design is the one that we apply to the real robot replanning experiments that follow.

The ARM-S manipulator robot, featuring a 7-DOF Barrett arm, was used for physical experimentation. The arm repeatedly executes a pre-defined trajectory, which becomes invalidated due to an unexpected obstacle. Once the perception system localizes the obstacle, the replanner reacts to it by modifying the trajectory to avoid the obstacle. 100 trials of the experiment were performed in two configurations: CHOMP-R with and without the end-effector workspace pose constraint, implemented via (22), compared to standard CHOMP. As Figure 24 shows, CHOMP-R resulted in notable replanning speedup on this platform.

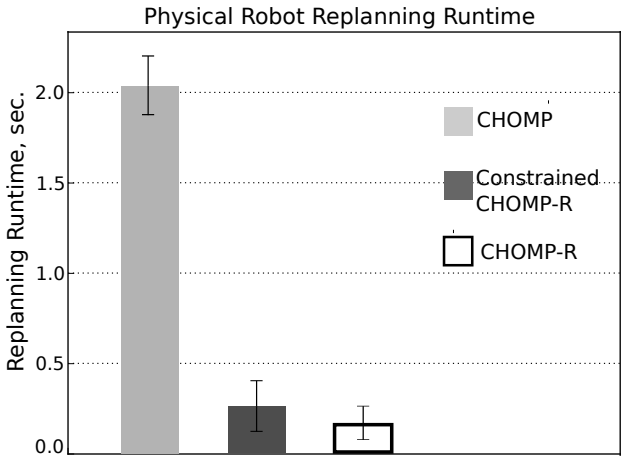


Figure 24: CHOMP and CHOMP-R are compared on the ARM-S robot platform. The figure shows that the constraint claimed greater runtime in seconds. The average CHOMP replan runtime was 2.03 sec, while the average CHOMP-R runtime was 0.28 and 0.17 sec. with and without the end-effector constraint implemented via (22), respectively. This result is significant in demonstrating that CHOMP-R is sufficiently fast to be responsible for the robot's reactive behavior. In addition, it demonstrates that satisfying workspace pose constraint maintains sub-second replan rate.

Figure 25: Boston Dynamics Inc. LittleDog quadruped robot, shown crossing several types of rough terrain. *Images reproduced from* [84].

8 Implementation on a Quadruped Robot

In Section 7, we describe a number of experiments conducted to compare CHOMP with other planning approaches, and to investigate the effectiveness of the various extensions to the algorithm. In this section, however, we describe how CHOMP was successfully integrated into a larger system to achieve high performance quadruped locomotion over rough terrain.

In 2007-2010, the Robotics Institute fielded one of six teams participating in the DARPA Learning Locomotion project, a competitive program focusing on developing strategies for quadruped locomotion on rough terrain [84]. Each team developed software to guide the LittleDog robot, designed and built by Boston Dynamics Inc., over standardized terrains quickly and robustly. With legs fully extended, LittleDog has approximately 12 cm clearance off of the ground. As shown in figure 26, some of the standardized terrains require stepping over and onto obstacles in excess of 7 cm.

Our approach to robotic legged locomotion decomposes the problem into a *footstep planner* which informs the robot where to place its feet as it traverses the terrain [12], and a *footstep controller* which generates full-body trajectories to realize the planned footsteps. In the final two years of the project, our team came to rely on CHOMP as a critical component of our 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.

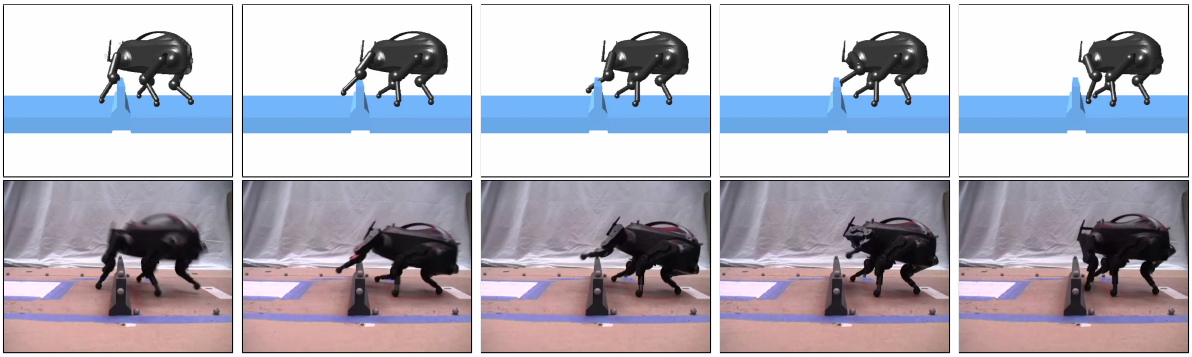


Figure 26: Using CHOMP with the Boston Dynamics LittleDog quadruped robot. *Top row:* the initial, heuristically determined input trajectory has significant collisions between the barrier and the robot's knee, shin, and foot during the leg swing. *Bottom row:* CHOMP pitches the trunk down and tips forward to bring the leg out of collision.

Additionally, during the swing phase, the three degrees of freedom for the swing leg may be controlled. For a given footstep, we run CHOMP 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 [36], 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 terrain (see right hand side of Figure 27).

To construct the signed distance field (SDF) representation of the terrain, we begin by scan-converting triangle mesh models of the terrain into a discrete grid representation. To determine whether a grid sample lies inside the terrain, we shoot a ray through the terrain and use the even/odd rule. Typical terrains are on the order of 1.8 m $\times$ 0.6 m $\times$ 0.3 m. We set the grid resolution for the SDF to 5 mm. The resulting SDFs usually require about 10-20 megabytes of RAM to store. The scan-conversion and computation of the SDF is created as a preprocessing step before optimization, and usually takes under 5 seconds on commodity hardware.

When running CHOMP with LittleDog, we exploit domain knowledge by adding a prior to the workspace potential function $c(x)$. The prior is 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 is to add a small gradient term that sends colliding points of the robot upwards regardless of the true gradient of the SDF.

For the trunk trajectory, in addition to the workspace obstacle potential, the objective function includes 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 are also multiplied through A^{-1} when applying the gradient, just as the workspace potential is.[8]

Although we typically represent the orientation of the trunk as a unit quaternion, we represent it to CHOMP as an exponential map vector corresponding to a differential rotation with respect to the "representative 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 [18]. The representative orientation is computed as the orientation halfway between the initial and final orientation of the trunk for the footstep. Because the total amount of body rotation over a footstep is generally quite small (well under 30°), the error between the inner product on exponential map vectors and the true quaternion distance metric is negligible.

[8]Since the ZMP is a dynamical constraint, it would be best handled using the theory defined in Section 6; however, the quadruped implementation predates that work. Instead, we exploit the fact that smoothly moving the robot center of

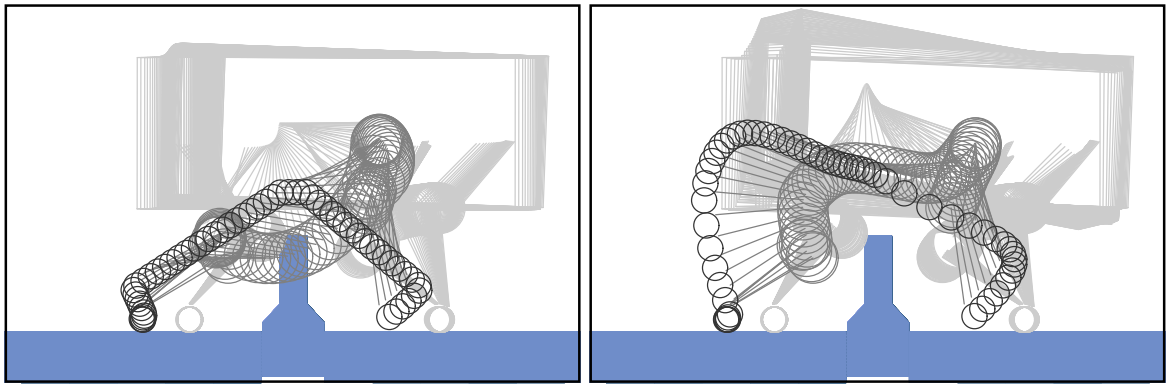


Figure 27: Another illustration of CHOMP optimizing a step over a barrier. *Left:* Initial trajectory. *Right:* After optimization. Note that although the initial trajectory results in severe collisions between the knee and the barrier, the optimized trajectory keeps the swing leg free of collisions. *Images reproduced from* [84].

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. We discretize the trajectories at the LittleDog host computer control cycle frequency, which is 100 Hz. Currently, trajectories are pre-generated 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 converges after less than 0.5s.[9]

As shown in figures 26 and 27, 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. CHOMP was developed as part of Phase II of our approach during the three phases of the Learning Locomotion program, and refined during Phase III. It aided substantially in raising our mean walking speed across standardized terrains from 2.6 cm/s in Phase I, to 5.6 cm/s in Phase II, and finally to 9.0 cm/s in Phase III (the program's metric speeds to beat in each phase were 1.2 cm/s, 4.2 cm/s, and 7.2 cm/s, respectively). The CHOMP algorithm was also adopted by another team participating in the program [37].

9 Limitations

Our work is limited in many ways, from the optimizer itself to our experimental evaluation.

CHOMP inherits the problems of optimization in high-dimensional spaces on non-convex cost functions. 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 (Hamiltonian Monte Carlo, Sec. 5, which makes CHOMP probabilistically complete), or through learning from previous experiences to initialize the optimizer in a good basin of attraction [16, 20].

Finding the right cost function to optimize is also a challenge. A limitation of CHOMP is that the current cost does not always align with our (or a user's) actual preferences. 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 [21]. 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

mass using A^{-1} has a very similar effect to treating the ZMP correctly.

[9] Here, using the speedups discussed in Section 7.5 would have presumably helped achieve better real-time performance, but again, the quadruped implementation predates that work.

observers have different expectations [22]. Learning an appropriate cost function and customizing it for the user is an active area of research.

CHOMP also depends on several parameters, which require tuning, like the trade-off between obstacle and smoothness cost or the step size schedule.

An additional limitation relates to handling constraints. While CHOMP 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) can lead to poor solutions when the penalty and the cost have opposing gradients.

Finally, our work does not lead to a clear understanding of what problems are easier and what problems are harder for CHOMP. Further work is needed to investigate what makes day-to-day, structured problems too complex to solve in a practically suitable amount of time.

Our experimental evaluation also has several limitations. We compared CHOMP with RRT-based algorithms, but other algorithms would also constitute good basis for comparisons (e.g. PRMs [41]). Although our goal was to provide the fairest comparison, every decision comes with alternatives that could shift the results. In our comparison, we used the standard implementations from OMPL of RRT and RRT* (coming from the author of the algorithm) and the standard collision checker from the OpenRAVE simulator. This collision checker, albeit standard, is not state-of-the art. Recent advances, e.g. FCL [58], would drastically improve the RRT performance. CHOMP, on the other hand, uses a different type of collision checking, via the Signed Distance Field. This is much faster, but also approximate – CHOMP has a built-in tolerance for this approximation via the obstacle cost function. Nonetheless, a similar collision checking method could be used for the RRT. We chose standard libraries instead in order to provide a comparison with the typical performance of RRTs on our scenes.

Our scene selection is limited as well. Firstly, it is mostly restricted to grasping in clutter (outside of a few additional examples), and fails to explore a wide range of real-world situations. This is a common issue in motion planning: the community still lacks real-world scene generators that would enable thorough testing of planners. Secondly, the clutter is generated in an artificial manner, by placing the same object in random positions around the target. It fails to capture the complexity of the real world. Thirdly, the amount of clutter still allows for feasible grasping configurations. If we imagine reaching into a real cluttered fridge, however, the clutter would not be distributed in a way that enables this – the robot would have to adopt more complex strategies than planning from a start to a goal (or goal set), e.g. pushing objects aside to make room [19].

Acknowledgments

We gratefully acknowledge support via the DARPA Learning Locomotion project, the DARPA Autonomous Robotic Manipulation-Software project, the Quality of Life Technologies NSF ERC, and Intel Pittsburgh. We thank Gil Jones and Willow Garage for their support, particularly in performing ROS experiments.

References

[1] Shun-Ichi Amari and Hiroshi Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.

[2] J. Andrew Bagnell. *Learning Decisions: Robustness, Uncertainty, and Approximation*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.

[3] Jérôme 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.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, Alvaro Collet Romea, and James Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[6] James E. Bobrow. A direct minimization approach for obtaining the distance between convex polyhedra. *International Journal of Robotics Research*, 8(3):65–76, 1989.

[7] Michael S. Branicky, Steven M. LaValle, Kari Olson, and Libo Yang. Quasi-randomized path planning. *Proc. of the International Conference on Robotics and Automation*, 2001.

[8] Oliver Brock and Oussama Khatib. Elastic Strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052, 12 2002.

[9] Arancha Casal. *Reconfiguration planning for modular self-reconfigurable robots*. PhD thesis, Aeronautics and Astronautics Dept., Stanford U., 2001.

[10] M. Chaichian and A. Demichev. *Path Integrals in Physics Volume 2: Quantum Field Theory, Statistical Physics & Other Modern Applications*. Taylor & Francis, 2001.

[11] Pang C. Chen and Yong K. Hwang. SANDROS: A dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14(3):390403, 1998.

[12] Joel Chestnutt. *Navigation Planning for Legged Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2007.

[13] 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.

[14] S.D. Conte and Carl de Boor. *Elementary Numerical Analysis*. McGraw-Hill, New York, 1972.

[15] 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.

[16] Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J. Andrew Bagnell. Contextual sequence prediction with application to control library optimization. In *R:SS*, July 2012.

[17] Rosen Diankov. *Automated Construction of Robotics Manipulation Programs*. PhD thesis, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, 10 2010.

[18] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. Technical report, Stanford University, 2006.

[19] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[20] A. Dragan, G. Gordon, and S. Srinivasa. Learning from experience in manipulation planning: Setting the right goals. In *ISRR*, 2011.

[21] Anca Dragan, Nathan Ratliff, and Siddhartha Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, May 2011.

[22] Anca Dragan, Kenton Lee, and Siddhartha Srinivasa. Legibility and predictability of robot motion. In *Human-Robot Interaction*, 2013.

[23] Elmar Eisemann and Xavier Decoret. Single-pass GPU solid voxelization for real-time applications. In *Proceedings of the Graphics Interface Conference*, 2008.

[24] B. Faverjon. Hierarchical object models for efficient anti-collision algorithms. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 333–340, 1989. doi: 10.1109/ROBOT.1989.100010.

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

[26] E. Gelenbe, Ricardo Lent, and Zhiguang Xu. Design and analysis of cognitive packet networks. *Performance Evaluation*, pages 155–76, 2001.

[27] I. M. Gelfand and S. V. Fomin. *Calculus of Variations*. Prentice-Hall, Inc., 1963.

[28] Roland Geraerts and Mark H. Overmars. Creating high-quality roadmaps for motion planning in virtual environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, page 43554361, 2006.

[29] 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.

[30] Sadri Hassani. *Mathematical Physics: A modern introduction to its foundations*. Springer-Verlag New York, Inc., 1999.

[31] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. In *In COLT*, pages 499–513, 2006.

[32] David Hsu. *Randomized single-query motion planning in expansive spaces*. PhD thesis, Computer Science Dept., Stanford University, 2000.

[33] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proc. Conf. IEEE Int Robotics and Automation*, volume 3, pages 2719–2726, 1997. doi: 10.1109/ROBOT.1997.619371.

[34] C. Igel, M. Toussaint, and W. Weishui. Rprop using the natural gradient. *Trends and Applications in Constructive Approximation*, pages 259–272, 2005.

[35] N. Jetchev and M. Toussaint. Trajectory prediction in cluttered voxel environments. In *IEEE International Conference on Robotics and Automation*, pages 2523–2528. IEEE, 2010.

[36] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, 2003.

[37] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Learning, planning, and control for quadruped locomotion over challenging terrain. *international journal of robotics research*, 30(2):236–258, 2011.

[38] 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.

[39] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

[40] Lydia E. Kavraki and Jean-Claude Latombe. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, chapter Probabilistic roadmaps for robot path planning. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[41] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. ISSN 1042-296X. doi: 10.1109/70.508439.

[42] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2001.

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

[44] Robert Kindel. *Motion planning for free-flying robots in dynamic and uncertain environments*. PhD thesis, Aeronaut. & Astr. Dept., Stanford University, 2001.

[45] Hagen Kleinert. *Path Integrals in Quantum Mechanics, Statistics, Polymer Physics, and Financial Markets*. World Scientific Publishing Company, fifth edition, 2009.

[46] James J. Kuffner. *Autonomous agents for real-time animation*. PhD thesis, Computer Science Dept., Stanford University, 1999.

[47] James J. Kuffner and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '00*, volume 2, pages 995–1001, 2000. doi: 10.1109/ROBOT.2000.844730.

[48] C. Lauterbach, Q. Mo, and D. Manocha. gProximity: Hierarchical GPU-based operations for collision and distance queries. *Computer Graphics Forum*, 29(2):419–428, 2010.

[49] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[50] Steven M. LaValle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[51] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23 (7/8):673–692, 2004.

[52] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime a* with provable bounds on sub-optimality. In *Workshop on Neural Information Processing Systems*, 2004.

[53] Maxim Likhachev and Anthony Stentz. R* search. In *AAAI*, 2008.

[54] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1008–1014, 1991. doi: 10.1109/ROBOT.1991.131723.

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

[56] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical report, University of Toronto, 1993.

[57] Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2010.

[58] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for proximity and collision queries. In *ICRA*, 2012.

[59] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012.

[60] Sean Quinlan. *Real-time modification of collision-free paths*. PhD thesis, Stanford University, 12 1994.

[61] 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.

[62] N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *Proc. of the International Conference on Machine Learning*, 2006.

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

[64] Nathan Ratliff, J. Andrew Bagnell, and Martin Zinkevich. (Online) Subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*, March 2007.

[65] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2009.

[66] Monica Reggiani, Mirko Mazzoli, and Stefano Caselli. An experimental evaluation of collision detection packages for robot motion planning, 2002.

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

[68] Bernhard Schlkopf and Alexander J. Smola. *Learning with Kernels*. Massachusetts Institute of Technology, 2001.

[69] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *European Control Conference*, 2001.

[70] 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.

[71] Christian Sigg, Ronald Peikert, and Markus Gross. Signed distance transform using graphics hardware. In *Proceedings of the IEEE Visualization Conference*, 2003.

[72] Siddhartha Srinivasa, David Ferguson, Casey Helfrich, Dmitry Berenson, Alvaro Collet Romea, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and J Michael Vandeweghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, January 2010.

[73] Avneesh Sud, Naga Govindaraju, Russell Gayle, and Dinesh Manocha. Interactive 3D distance field computation using linear factorization. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2006.

[74] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 11:31373181, 11 2010.

[75] 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.

[76] Emanuel Todorov. Linearly-solvable Markov decision problems. In *Proceedings of Neural Information Processing Systems*, pages 1369–1376. MIT Press, 2006.

[77] M. Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1049–1056. ACM, 2009.

[78] 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.

[79] Miloš Žefran, Jaydev P. Desai, and Vijay Kumar. Continuous motion plans for robotic systems with changing dynamic behavior. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics*, 1996.

[80] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Machine Learning Department, Carnegie Mellon University, Dec 2010.

[81] Brian D. Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J. Andrew Bagnell, Martial Hebert, Anind Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *Proc. IROS 2009*, October 2009.

[82] Brian D. Ziebart, J. Andrew Bagnell, and Anind Dey. Modeling interaction via the principle of maximum causal entropy. In *International Conference on Machine Learning*, June 2010.

[83] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 928–936, 2003.

[84] Matt Zucker, Nathan Ratliff, Martin Stolle, Joel Chestnutt, J. Andrew Bagnell, Christopher G. Atkeson, and James J. Kuffner. Optimization and learning for rough terrain legged locomotion. *International Journal of Robotics Research*, 30(2):175–191, February 2011.