Motion planning with sequential convex optimization and convex collision checking

John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg and Pieter Abbeel

The International Journal of Robotics Research published online 11 June 2014
DOI: 10.1177/0278364914528132

Article

The International Journal of Robotics Research
1–20

DOI: 10.1177/0278364914528132
ijr.sagepub.com

Motion planning with sequential convex optimization and convex collision checking

John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg and Pieter Abbeel

Abstract

We present a new optimization-based approach for robotic motion planning among obstacles. Like CHOMP (Covariant Hamiltonian Optimization for Motion Planning), our algorithm can be used to find collision-free trajectories from naïve, straight-line initializations that might be in collision. At the core of our approach are (a) a sequential convex optimization procedure, which penalizes collisions with a hinge loss and increases the penalty coefficients in an outer loop as necessary, and (b) an efficient formulation of the no-collisions constraint that directly considers continuous-time safety Our algorithm is implemented in a software package called TrajOpt.

We report results from a series of experiments comparing TrajOpt with CHOMP and randomized planners from OMPL, with regard to planning time and path quality. We consider motion planning for 7 DOF robot arms, 18 DOF full-body robots, statically stable walking motion for the 34 DOF Atlas humanoid robot, and physical experiments with the 18 DOF PR2. We also apply TrajOpt to plan curvature-constrained steerable needle trajectories in the SE(3) *configuration space and multiple non-intersecting curved channels within 3D-printed implants for intracavitary brachytherapy. Details, videos, and source code are freely available at: http://rll.berkeley.edu/trajopt/ijrr.*

Keywords

Motion planning, sequential convex optimization, convex collision checking, trajectory optimization

1. Introduction

The increasing complexity of robots and the environments that they operate in has spurred the need for high-dimensional motion planning. Consider, for instance, a PR2 personal robot operating in a cluttered household environment or an Atlas humanoid robot performing navigation and manipulation tasks in an unstructured environment. Efficient motion planning is important to enable these high DOF robots to perform tasks, subject to motion constraints while avoiding collisions with obstacles in the environment. Processing time is especially important where re-planning is necessary.

Sampling-based motion planners (Kavraki et al., 1996; LaValle, 2006) are very effective and offer probabilistic completeness guarantees. However, these planners often require a post-processing step to smooth and shorten the computed trajectories. Furthermore, considerable computational effort is expended in sampling and connecting samples in portions of the configuration space that might not be relevant to the task. Optimal planners such as RRT* (Karaman and Frazzoli, 2011) and discretization-based approaches (Likhachev et al., 2003; Likhachev and Stentz, 2008) are very promising but are currently computationally inefficient for solving high-dimensional motion planning problems.

Trajectory optimization is fundamental in optimal control where the objective is to solve for a trajectory encoded as a sequence of states and controls that optimizes a given objective subject to constraints (Betts, 2010). Optimization plays two important roles in robot motion planning. First, it can be used to smooth and shorten trajectories computed by other planning methods such as sampling-based planners. Second, it can be used to compute locally optimal, collision-free trajectories from scratch starting from naïve (straight-line) trajectory initializations that might collide with obstacles.

Even though trajectory optimization has been successfully used for optimal control in a number of domains,

Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, USA

Corresponding author:
Sachin Patil, University of Calfornia at Berkeley, Sutardja Dai Hall, 2594 Hearst Ave, CA 94709, USA.
Email: sachinpatil@berkeley.edu

it has traditionally not been used for robot motion planning because the presence of obstacles in the environment and other constraints requires solving a non-convex, constrained optimization problem. However, CHOMP (Covariant Hamiltonian Optimization for Motion Planning) (Ratliff et al., 2009; Zucker et al., 2012) revived interest in trajectory optimization methods by demonstrating the effectiveness on several robotic platforms including the HERB mobile manipulation platform, the LittleDog quadruped, and the PR2 robot. CHOMP has the following key features: (a) formulation of trajectory costs that are invariant to the time parameterization of the trajectory, (b) using precomputed signed distance fields for collision checking, and (c) using pre-conditioned gradient descent for numerical optimization.

Our approach uses optimization in the same spirit as CHOMP, with the following key differences: (a) the numerical optimization method used, and (b) the method of checking for collisions and penalizing them. We use sequential convex optimization, which involves solving a series of convex optimization problems that approximate the cost and constraints of the original problem. The ability to add new constraints and costs to the optimization problem allows our approach to tackle a larger range of motion planning problems, including planning for underactuated, non-holonomic systems. For collisions, we use signed distances using convex–convex collision detection, and safety of a trajectory between time steps, i.e. continuous-time safety, is taken into account by considering the swept-out volume of the robot between time steps. This formulation has little computational overhead in collision checking and allows us to use a sparsely sampled trajectory. Our method for handling collisions yields a polyhedral approximation of the free part of configuration space, which is directly incorporated into the convex optimization problem that is solved at each optimization iteration. This precludes the need for pre-computation of signed distance fields and is computationally efficient in practice.

We performed a quantitative comparison between TrajOpt and several implementations of motion planning algorithms, including sampling based planners from OMPL (Sucan et al., 2012), as well as a recent implementation of CHOMP (Zucker et al., 2012). Overall, our experimental results indicate that TrajOpt was computationally faster than the alternatives on the considered benchmark (around 100–200 ms on arm-planning problems and solves full body 18 DOF planning problems for the PR2 robot in under a second on an Intel i7 3.5 GHz CPU), and solved a larger fraction of the problems given a specified time limit. We also applied TrajOpt to high-DOF motion problems, including physical experiments with the PR2 robot where we simultaneously need to plan for two arms along with the base and torso (Figure 1(b)), and for planning foot placements with 28 DOF (+ 6 DOF pose) of the Atlas humanoid robot as it maintains static stability and avoids collisions (Figure 1(d)).

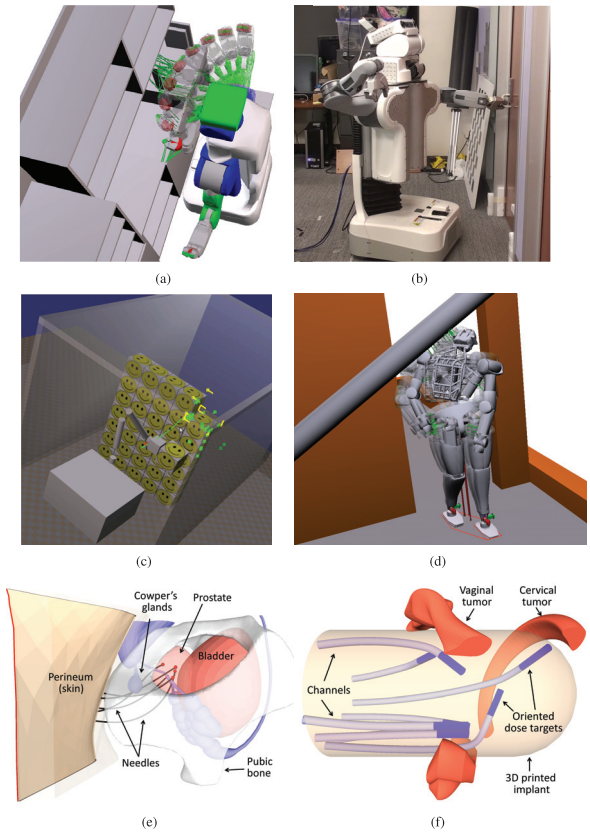


Fig. 1. TrajOpt applied to several motion planning scenarios: (a) planning an arm trajectory for the PR2 in simulation, (b) PR2 opening a door with a full-body motion, (c) industrial robot picking boxes, subject to an orientation constraint on the end effector, (d) humanoid robot model (DRC/Atlas) ducking underneath an obstacle while obeying static stability constraints, (e) multiple bevel-tip flexible needles inserted through the perineum to reach targets deep within the prostate following high-quality constant curvature trajectories, and (f) optimized layout for bounded curvature channels within 3D-printed vaginal implants for delivering radiation to OB/GYN tumors.

In this work, in addition to providing a revised and extended version of our work (Schulman et al., 2013), we (a) describe an extension to the algorithm described in the RSS paper to plan trajectories in $SE(3)$, and (b) provide a discussion on cases where trajectory optimization fails to find a feasible solution. Regarding (a), we consider the problem of planning curvature-constrained trajectories in 3D environments. This involves trajectory optimization over manifolds such as the $SE(3)$ Lie group, instead of just vector spaces of the form $\mathbb{R}^n$. We accomplish this by iteratively optimizing over increments to the trajectory, defined in terms of the corresponding Lie algebra—$\mathfrak{se}(3)$ in our case (Saccon et al., 2013). We applied this extension of TrajOpt to two real-world clinical applications. First, we considered the problem of planning collision-free, constant curvature trajectories that avoid obstacles in the environment and optimize clinically relevant metrics for flexible, bevel-tip medical needles (Webster et al., 2006; Reed

et al., 2011) (Figure 1(e)). Our second application considers the problem of planning multiple, mutually collision-free, curvature-constrained channels within 3-D printed implants (Garg et al., 2013) for intracavitary brachytherapy (HDR-BT).

2. Related work

Trajectory optimization with application to robotics has been extensively studied. Khatib proposed the use of potential fields for avoiding obstacles, including dynamic obstacles (Khatib, 1986). Warren used a global potential field to push the robot away from configuration space obstacles, starting with a trajectory that was in collision (Warren, 1989). Quinlan and Khatib locally approximated the free part of configuration space as a union of spheres around the current trajectory as part of a local optimization that tries to shorten the trajectory (Quinlan and Khatib, 1993). Brock and Khatib improved on this idea, enabling trajectory optimization for a robot in 3D, by using the Jacobian to map distances from task space into configuration space (Brock and Khatib, 2002). These approaches locally approximate the free space using a union of spheres, which is a overly conservative approximation and may not find feasible trajectories even if they exist. Lamiraux et al. used an iterative scheme to find collision free paths for non-holonomic robots, using a potential field based on the obstacles (Lamiraux et al., 2004).

While the motivation for the presented work is very similar to the motivation behind CHOMP (Ratliff et al., 2009; Dragan et al., 2011; Zucker et al., 2012), which is most similar in terms of prior art, our algorithm differs fundamentally in the following two ways: (a) we use a different approach for collision detection, and (b) we use a different numerical optimization scheme. We note that there are variants of CHOMP that use gradient-free, stochastic optimization, including STOMP (Stochastic Trajectory Optimization for Motion Planning) (Kalakrishnan et al., 2011) and ITOMP (Incremental Trajectory Optimization) for real-time replanning in dynamic environments (Park et al., 2012).

Other recent work in robotics uses sequential quadratic programming for trajectory optimization and incorporates collision avoidance as constraints, in a similar way to this work. Lampariello et al. (2011) incorporate signed distances between polytopes as inequality constraints in an optimal control problem. Werner et al. (2012) use sequential quadratic programming to optimize walking trajectories, also incorporating obstacle avoidance as hard constraints, along with stability constraints. However, these methods have not considered continuous-time collision checking or dealt with infeasible trajectory initializations that start deeply in collision. Finally, there recently has been considerable progress in trajectory optimization for dynamical systems (Erez and Todorov, 2012; Mordatch et al., 2012; Tassa et al., 2012; Lengagne et al., 2013; Posa and Tedrake, 2013). These approaches have obtained promising results but rely on a simplified, though conservative, representation of the robot geometry (e.g. union of spheres) to obtain solutions to planning problems. Fast algorithms have been developed that use sequential quadratic programming to compute solutions for trajectory optimization by relying on problem-specific code generation (Houska et al., 2011). However, these methods do not address the issue of avoiding collisions with obstacles in the environment.

Many techniques have been proposed in the literature to generate smooth trajectories from solutions obtained using sampling-based motion planners, as they can sometimes generate non-smooth trajectories that may contain unnecessary turns (LaValle, 2006). Shortcut-based methods (Kallmann et al., 2003; Hauser and Ng-Thow-Hing, 2010; Pan et al., 2012) replace non-smooth portions of a trajectory shorter linear or curved segments (e.g. parabolic arcs, Bézier curves). These methods tend to be fast and simple, and can produce high quality paths in many cases. However, they may not provide enough flexibility in terms of generating collision-free smooth trajectories in the presence of obstacles. Trajectory optimization approaches such as ours and CHOMP can be used for trajectory smoothing in such cases.

3. Background: Sequential convex optimization

Robotic motion planning problems can be formulated as non-convex optimization problems, i.e. minimize an objective subject to inequality and equality constraints:

$$\text{minimize } f(\mathbf{x}) \tag{1a}$$
$$\text{subject to} \tag{1b}$$
$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, n_{\text{ineq}} \tag{1c}$$
$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, n_{\text{eq}} \tag{1d}$$

where f, g_i, h_i, are scalar functions.

In kinematic motion planning problems, the optimization is done over a $T \times K$-dimensional vector, where T is the number of time-steps and K is the number of degrees of freedom. We denote the optimization variables as $\mathbf{x}_{1:T}$, where $\mathbf{x}_t$ describes the configuration at the tth timestep. To encourage minimum-length paths, we use the sum of squared displacements,

$$f(\mathbf{x}_{1:T}) = \sum_{t=1}^{T-1} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \tag{2}$$

Besides obstacle avoidance, common inequality constraints in motion planning problems include joint limits and joint angular speed limits. Common equality constraints include the end-effector pose (i.e. reach a target pose at the end of the trajectory) and orientation constraints (keep a held object upright). For underactuated, non-holonomic motion planning problems, additional equality constraints

Algorithm 1 ℓ_1 penalty method for sequential convex optimization.

Parameters:

μ_0: initial penalty coefficient
s_0: initial trust region size
c: step acceptance parameter
τ^+, τ^-: trust region expansion and shrinkage factors
k: penalty scaling factor
ftol, xtol: convergence thresholds for merit and x
ctol: constraint satisfaction threshold

Variables:

x: current solution vector
μ: penalty coefficient
s: trust region size

1: **for** PenaltyIteration $= 1, 2, \ldots$ **do**
2: **for** ConvexifyIteration $= 1, 2, \ldots$ **do**
3: $\tilde{f}, \tilde{g}, \tilde{h} =$ ConvexifyProblem(f, g, h)
4: **for** TrustRegionIteration $= 1, 2, \ldots$ **do**
5: $\mathbf{x} \leftarrow \arg\min_{\mathbf{x}} \tilde{f}(\mathbf{x}) + \mu \sum_{i=1}^{n_{\text{ineq}}} |\tilde{g}_i(\mathbf{x})|^+ + \mu \sum_{i=1}^{n_{\text{eq}}} |\tilde{h}_i(\mathbf{x})|$ subject to trust region and linear constraints
6: **if** TrueImprove / ModelImprove $> c$ **then**
7: $s \leftarrow \tau^+ * s$ ▷ Expand trust region
8: **break**
9: **else**
10: $s \leftarrow \tau^- * s$ ▷ Shrink trust region
11: **if** $s <$ xtol **then**
12: **goto** 15
13: **if** converged according to tolerances xtol or ftol **then**
14: **break**
15: **if** constraints satisfied to tolerance ctol **then**
16: **break**
17: **else**
18: $\mu \leftarrow k * \mu$

are added to ensure that the kinematics are consistent. We will discuss some of these constraints in Section 7.

Sequential convex optimization solves a non-convex optimization problem by repeatedly constructing a convex subproblem—an approximation to the problem around the current iterate x. The subproblem is used to generate a step Δx that makes progress on the original problem. Two key ingredients of a sequential convex optimization algorithm are: (a) a method for constraining the step to be small, so the solution vector remains within the region where the approximations are valid; (b) a strategy for turning the infeasible constraints into penalties, which eventually drives all of the constraint violations to zero. For (a), we use a trust region modeled as a box constraint around the current iterate. For (b) we use ℓ_1 penalties: each inequality constraint $g_i(\mathbf{x}) \leq 0$ becomes the penalty $|g_i(\mathbf{x})|^+$, where $|x|^+ = \max(x, 0)$; each equality constraint $h_i(\mathbf{x}) = 0$ becomes the absolute value penalty $|h_i(\mathbf{x})|$. In both cases, the penalty is multiplied by some coefficient μ, which is sequentially increased, usually by multiplying by a constant scaling factor at each step, during the optimization to drive constraint violations to zero. Note that ℓ_1 penalties are non-differentiable but convex, and convex optimization algorithms can efficiently minimize them. Our implementation uses a variant of the classic ℓ_1 penalty method (Nocedal and Wright, 1999), described in Algorithm 1.

The use of ℓ_1 penalties is called an exact penalty method, because if we multiply the penalty by a large coefficient (tending to infinity but the value is smaller in practice), then the minimizer of the penalized problem is exactly equal to the minimizer of the constrained problem. This is in contrast to the typical ℓ_2 penalty method that penalizes squared error, i.e. $g_i(\mathbf{x}) \leq 0 \rightarrow (|g_i(\mathbf{x})|^+)^2$ and $h_i(\mathbf{x}) = 0 \rightarrow h_i(\mathbf{x})^2$. ℓ_1 penalty methods give rise to numerically stable algorithms that drive the constraint violations to zero.

Note that the objective we are optimizing contains non-smooth terms like $|a \cdot x + b|$ and $|a \cdot x + b|^+$. However, the subproblems solved by our algorithm are quadratic programs—a quadratic objective subject to affine constraints. We accommodate these non-smooth terms while keeping the objective quadratic by adding auxilliary slack variables (Nocedal and Wright, 1999). To add $|a \cdot x + b|^+$, we add slack variable t and impose constraints

$$\begin{aligned} 0 &\leq t \\ a \cdot x + b &\leq t \end{aligned} \tag{3}$$

Note that at the optimal solution, $t = |a \cdot x + b|^+$. Similarly, to add the term $|a \cdot x + b|$, we add $s + t$ to the objective and impose constraints

$$\begin{aligned} 0 \leq s, \;\; 0 &\leq t \\ s - t &= a \cdot x + b \end{aligned} \tag{4}$$

At the optimal solution, $s = |a \cdot x + b|^+$, $t = |-a \cdot x - b|^+$, so $s + t = |a \cdot x + b|$.

In the outer loop (PenaltyIteration, line 1) we increase the penalty coefficient μ by a constant scaling factor ($k = 10$ in all our experiments) until all the constraints are satisfied, terminating when the coefficient exceeds some threshold. The next loop (ConvexifyIteration, line 2) is where we repeatedly construct a convex approximation to the problem and then optimize it. In particular, we approximate the objective and inequality constraint functions by convex functions that are compatible with a quadratic program (QP) solver, and we approximate the nonlinear equality constraint functions by affine functions. The nonlinear constraints are incorporated into the problem as penalties, while the linear constraints are directly imposed in the convex subproblems. The next loop (TrustRegionIteration, line 4) is like a line search; if the true improvement (TrueImprove) to the non-convex merit functions (objective plus constraint penalty) is a sufficiently large fraction of the improvement to our convex approximations (ModelImprove), then the step is accepted.

4. No-collisions constraint

This section describes how the no-collisions constraint can be efficiently formulated for a discretely sampled trajectory that ensures that a given robot configuration $\mathbf{x}$ is not in collision. We can use this constraint to encourage the robot to be collision-free at each time step. We later show how this can be extended to encourage continuous-time safety, i.e. the robot stays collision-free between time steps.

4.1. Discrete-time no-collisions constraint

Let A, B, O be labels for rigid objects, each of which is a link of the robot or an obstacle. The set of points occupied by these objects are denoted by calligraphic letters $\mathcal{A}, \mathcal{B}, \mathcal{O} \subset \mathbb{R}^3$. We sometimes use a superscript to indicate the coordinate system of a point or a set of points. $\mathcal{A}^w \subset \mathbb{R}^3$ denotes the set of points in world coordinates occupied by A, whereas $\mathcal{A}^A$ denotes the set of points in a coordinate system local to object A. The poses of the objects A, B are denoted as F_A^w, F_B^w, where F_A^w is a rigid transformation that maps from the local coordinate system to the global coordinate system.

Our method for penalizing collisions is based on the notion of minimum translation distance, common in collision detection (Ericson, 2004). The distance between two sets $\mathcal{A}, \mathcal{B} \subset \mathbb{R}^3$, which is nonzero for non-intersecting sets, is defined as

$$\text{dist}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \;\big|\; (T + \mathcal{A}) \cap \mathcal{B} \neq \varnothing\} \tag{5}$$

Informally, it's the length of the smallest translation T that puts the shapes in contact. The penetration depth, which is nonzero for overlapping shapes, is defined analogously as the minimum translation that takes two shapes out of contact:

$$\text{penetration}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \;\big|\; (T + \mathcal{A}) \cap \mathcal{B} = \varnothing\} \tag{6}$$

The signed distance is defined as follows:

$$\text{sd}(\mathcal{A}, \mathcal{B}) = \text{dist}(\mathcal{A}, \mathcal{B}) - \text{penetration}(\mathcal{A}, \mathcal{B}) \tag{7}$$

Note that these concepts can also be defined using the notion of a configuration space obstacle and the Minkowski difference between the shapes—see e.g. Ericson (2004).

The convex–convex signed distance computation can be performed efficiently. The distance between two shapes can be calculated by the Gilbert–Johnson–Keerthi (GJK) algorithm (Gilbert et al., 1988), while the penetration depth is calculated by a different algorithm, the Expanding Polytope algorithm (EPA) (Van den Bergen, 2001). One useful feature of these two algorithms, which makes them so generally applicable, is that they represent an object A by its *support mapping*, i.e. a function that maps vector $\mathbf{v}$ to the point in $\mathcal{A}$ that is furthest in direction $\mathbf{v}$:

$$s_A(\mathbf{v}) = \arg\max_{\mathbf{p} \in \mathcal{A}} \mathbf{v} \cdot \mathbf{p} \tag{8}$$

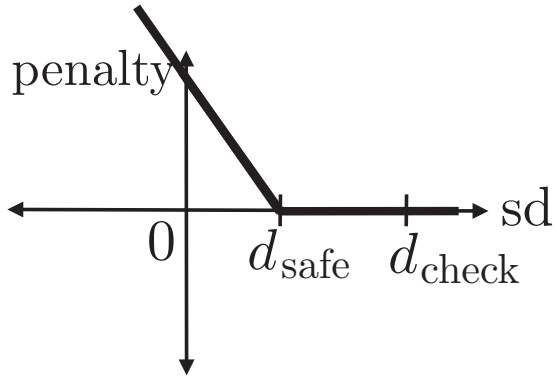


Fig. 2. Hinge penalty for collisions.

This representation makes it possible to describe convex shapes implicitly without considering explicit polyhedral representations of their surfaces. We will exploit this fact to efficiently check for collisions against swept-out volumes of the robot between time steps.

Two objects are non-colliding if the signed distance is positive. We will typically want to ensure that the robot has a safety margin d_{safe}. Thus, we want to enforce the following constraints at each timestep

$$\begin{aligned} \text{sd}(\mathcal{A}_i, \mathcal{O}_j) \geq d_{\text{safe}} \qquad & \forall i \in \{1, 2, \ldots, N_{\text{links}}\}, \\ & \forall j \in \{1, 2, \ldots, N_{\text{obstacles}}\} \\ & \text{(obstacle collisions)} \\ \text{sd}(\mathcal{A}_i, \mathcal{A}_j) \geq d_{\text{safe}} \qquad & \forall i, j \in \{1, 2, \ldots, N_{\text{links}}\} \\ & \text{(self collisions)} \end{aligned} \tag{9}$$

where $\{\mathcal{A}_i\}$ is the collection of links of the robot, and $\{\mathcal{O}_j\}$ is the set of obstacles. These constraints can be relaxed to the following ℓ_1 penalty

$$\begin{aligned} & \sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{obs}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+ \\ & + \sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{links}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{B}_j)|^+ \end{aligned} \tag{10}$$

A single term of this penalty function $|d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+$ is illustrated in Figure 2.

Note that in practice, we do not consider all pairs of objects for the collision penalty (Equation (10)) since the penalty corresponding to most pairs of faraway objects is zero. For computational efficiency, we query a collision checker for all pairs of nearby objects in the world with distance smaller than a user-defined distance d_{check} between them where $d_{\text{check}} > d_{\text{safe}}$, and formulate the collision penalty based on these pairs.

We can form a linear approximation to the signed distance using the robot Jacobian and the notion of closest points. Let $\mathcal{A}^A, \mathcal{B}^B \subset \mathbb{R}^3$ denote the space occupied by A and B in local coordinates, and let $\mathbf{p}^A \in \mathcal{A}^A$ and $\mathbf{p}^B \in \mathcal{B}^B$

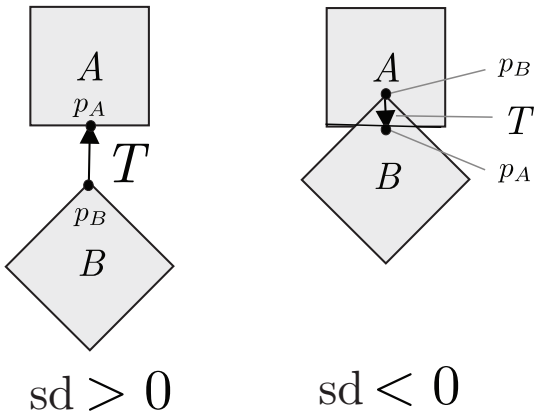


Fig. 3. Minimal translational distance and closest points.

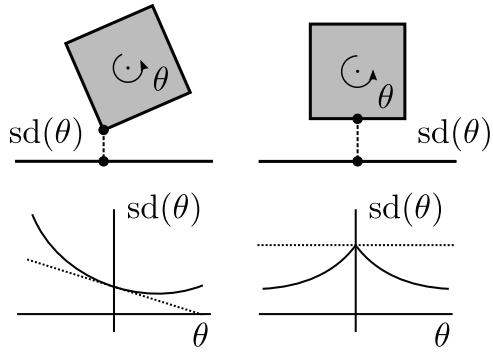


Fig. 4. Illustration of the non-differentiability of the signed distance function. Here, a square is rotated about its center by angle θ. The true function is shown by a solid line, and the linearization is shown by a dotted line. It is correct to first-order in non-degenerate situations, however, in degenerate situations where the signed distance is non-differentiable, it gives an erroneous gradient estimate. Empirically, the optimization works well despite this issue.

denote the local positions of contact points. F_A^w and F_B^w denote the objects' poses.

To define closest points and our derivative approximation, first note that the signed distance function is given by the following formula, which applies to both the overlapping and non-overlapping cases:

$$\mathrm{sd}(\{A, F_A^w\}, \{B, F_B^w\}) = \max_{\|\hat{\mathbf{n}}\|=1} \min_{\substack{\mathbf{p}_A \in A, \\ \mathbf{p}_B \in B}} \hat{\mathbf{n}} \cdot (F_A^w \mathbf{p}^A - F_B^w \mathbf{p}^B) \tag{11}$$

The closest points $\mathbf{p}^A, \mathbf{p}^B$ and normal $\hat{\mathbf{n}}$ are defined as a triple for which the signed distance is optimum, as described in Equation (11). Equivalently, the contact normal $\hat{\mathbf{n}}$ is the direction of the minimal translation T (as defined in Equations (5) and (6)), and $\mathbf{p}^A$ and $\mathbf{p}^B$ are a pair of points (expressed in local coordinates) that are touching when we translate A by T (Figure 3).

Let's assume that the pose of A is parameterized by the configuration vector $\mathbf{x}$ (e.g. the robot's joint angles), and B is stationary. (This calculation can be straightforwardly extended to the case where both objects vary with $\mathbf{x}$, which is necessary for dealing with self-collisions.) Then we can linearize the signed distance by assuming that the local positions $\mathbf{p}_A, \mathbf{p}_B$ are fixed, and that the normal $\mathbf{n}$ is also fixed, in Equation (11).

We first linearize the signed distance with respect to the positions of the closest points:

$$\mathrm{sd}_{AB}(\mathbf{x}) \approx \hat{\mathbf{n}} \cdot (F_A^w(\mathbf{x})\,\mathbf{p}_A - F_B^w \mathbf{p}_B) \tag{12}$$

By calculating the Jacobian of $\mathbf{p}_A$ with respect to $\mathbf{x}$, we can linearize this signed distance expression at $\mathbf{x}_0$:

$$\begin{aligned} \nabla_{\mathbf{x}}\, \mathrm{sd}_{AB}(\mathbf{x})\Big|_{\mathbf{x}_0} &\approx \hat{\mathbf{n}}^{\mathrm{T}} J_{\mathbf{p}_A}(\mathbf{x}_0) \\ \mathrm{sd}_{AB}(\mathbf{x}) &\approx \mathrm{sd}_{AB}(\mathbf{x}_0) + \hat{\mathbf{n}}^{\mathrm{T}} J_{\mathbf{p}_A}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \end{aligned} \tag{13}$$

The above expression allows us to form a local approximation of one collision cost term with respect to the robot's degrees of freedom. This approximation is used for every pair of nearby objects returned by the collision checker. After we linearize the signed distance, this cost can be incorporated into a quadratic program (or linear program) using Equation (3).

Note that Equation (13), which assumes that the normal $\hat{\mathbf{n}}$ and the closest points are fixed, is correct to first order in non-degenerate situations involving polyhedra. However, in degenerate cases involving face–face contacts, the signed distance is non-differentiable as a function of the poses of the objects, and the above formula deviates from correctness. Empirically, the optimization does not seem to get stuck at the points of non-differentiability. Figure 4 illustrates this phenomenon for two squares. An interesting avenue for future work would be to develop approximations to the the signed distance penalty that provide a better local approximation.

4.2. Continuous-time trajectory safety

The preceding discussion formulates the no-collisions constraint for a discretely sampled trajectory. However, when such a trajectory is converted to a continuous-time trajectory for execution, e.g. by linear interpolation or cubic splines, the resulting continuous-time trajectory might have collisions between time steps (see Figure 5).

We can modify the collision penalty from Section 4.1 to give a cost that enforces the continuous-time safety of the trajectory (though it makes a geometric approximation). It is only twice as computationally expensive than the discrete-time collision cost of the previous section since it involves twice as many narrow-phase collision queries.

Consider a moving object A and a static object B, for $0 \le t \le 1$. The motion is free of collision if the swept-out volume $\cup_t \mathcal{A}(t)$ does not intersect $\mathcal{B}$. First suppose that A undergoes only translation, not rotation. (We will consider

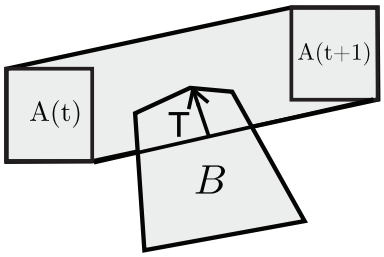


Fig. 5. Illustration of swept volume for use in our continuous collision cost.

rotations below.) Then the swept-out volume is the convex hull of the initial and final volumes (Van den Bergen, 2001)

$$\bigcup_{t\in[0,1]} \mathcal{A}(t) = \text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)) \tag{14}$$

Thus we can use the same sort of collision cost we described in Section 4.1, but now we calculate the signed distance between the swept-out volume of A and the obstacle B:

$$\text{sd}(\text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{B}) \tag{15}$$

We perform the necessary signed distance computation without having to calculate the convex hull of shapes $A(t), A(t+1)$, since (as noted in Section 4.1) the signed distance cost can be calculated using the support mappings. In particular, the support mapping is given by

$$s_{\text{convhull}(C,D)}(\mathbf{v}) = \begin{cases} s_C(\mathbf{v}) & \text{if } s_C(\mathbf{v})\cdot\mathbf{v} > s_D(\mathbf{v})\cdot\mathbf{v} \\ s_D(\mathbf{v}) & \text{otherwise} \end{cases} \tag{16}$$

Calculating the gradient of the swept-volume collision cost is slightly more involved than discrete case described in Equations (12) and (13). Let's consider the case where object A is moving and object B is stationary, as in Figure 5. Let's suppose that A and B are polyhedral. Then the closest point $\mathbf{p}_{\text{swept}} \in \text{convhull}(A(t), A(t+1))$ lies in one of the faces of this polytope. $\text{convhull}(A(t), A(t+1))$ has three types of faces: (a) all the vertices are from $A(t)$, (b) all of the vertices are from $A(t+1)$, and (c) otherwise. Cases (a) and (b) occur when the deepest contact in the interval $[t, t+1]$ occurs at one of the endpoints, and the gradient is given by the discrete-time formula. In case (c), we have to estimate how the closest point varies as a function of the poses of A at times t and $t+1$.

We use an approximation for case (c) that is computationally efficient and empirically gives accurate gradient estimates. It is correct to first order in non-degenerate 2D cases, but it is not guaranteed to be accurate in 3D. Let $\mathbf{p}_{\text{swept}}$, $\mathbf{p}_B$, denote the closest points and normals between $\text{convhull}(A(t), A(t+1))$, and B, respectively, and let $\hat{\mathbf{n}}$ be the normal pointing from B into A.

1. Find supporting vertices $\mathbf{p}_0 \in \mathcal{A}(t)$ and $\mathbf{p}_1 \in \mathcal{A}(t+1)$ by taking the support map of these sets along the normal $-\hat{\mathbf{n}}$.

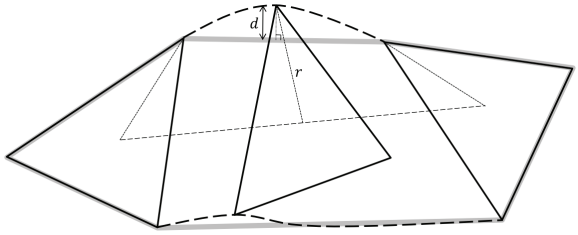


Fig. 6. Illustration of the difference between swept out shape and convex hull. The figure shows a triangle undergoing translation and uniform rotation. The swept-out area is enclosed by dotted lines, and the convex hull is shown by a thick gray line.

2. Our approximation assumes that the contact point $\mathbf{p}_{\text{swept}}$ is a fixed convex combination of $\mathbf{p}_0$ and $\mathbf{p}_1$. In some cases, $\mathbf{p}_0$, $\mathbf{p}_{\text{swept}}$, and $\mathbf{p}_1$ are collinear. To handle the other cases, we set

$$\alpha = \frac{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\|}{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\| + \|\mathbf{p}_0 - \mathbf{p}_{\text{swept}}\|} \tag{17}$$

where we make the approximation

$$\mathbf{p}_{\text{swept}}(\mathbf{x}) \approx \alpha\mathbf{p}_0 + (1-\alpha)\mathbf{p}_1 \tag{18}$$

3. Calculate the Jacobians of those points

$$J_{\mathbf{p}_0}(\mathbf{x}_0^t) = \frac{d}{d\mathbf{x}^t}\mathbf{p}_0, \quad J_{\mathbf{p}_1}(\mathbf{x}_0^{t+1}) = \frac{d}{d\mathbf{x}^{t+1}}\mathbf{p}_1 \tag{19}$$

4. Similarly to Equation (13), linearize the signed distance around the trajectory variables at timesteps t and $t+1$

$$\begin{aligned}\text{sd}_{AB}(\mathbf{x}^t, \mathbf{x}^{t+1}) \approx \text{sd}_{AB}(\mathbf{x}_0^t, \mathbf{x}_0^{t+1}) &+ \alpha\hat{\mathbf{n}}^{\mathrm{T}} J_{\mathbf{p}_0}(\mathbf{x}_0^t)(\mathbf{x}^t - \mathbf{x}_0^t) \\ &+ (1-\alpha)\hat{\mathbf{n}}^{\mathrm{T}} J_{\mathbf{p}_1}(\mathbf{x}_0^{t+1})(\mathbf{x}^{t+1} - \mathbf{x}_0^{t+1})\end{aligned} \tag{20}$$

The preceding discussion assumed that the shapes undergo translation only. However, the robot's links also undergo rotation, so the convex hull will underestimate the swept-out volume. This phenomenon is illustrated in Figure 6. We can calculate a simple upper-bound to the swept-out volume, based on the amount of rotation. Consider a shape A undergoing translation T and rotation angle ϕ around axis $\hat{k}$ in local coordinates. Let $A(t)$ and $A(t+1)$ be the occupied space at the initial and final times, respectively. One can show that if we expand the convex hull $\text{convhull}(A(t), A(t+1))$ by $d_{\text{arc}} = r\phi^2/8$, where r is the maximum distance from a point on A to the local rotation axis, then the swept-out volume is contained inside.

In summary, we can ensure continuous time safety by ensuring that for each time interval $[t, t+1]$

$$\text{sd}(\text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{O}) > d_{\text{safe}} + d_{\text{arc}} \tag{21}$$

One could relax this constraint into a penalty as described in Section 4.1, by approximating $\phi(\mathbf{x}^t, \mathbf{x}^{t+1})$. In practice, we

ignored the correction d_{arc}, since it was well under 1 cm in all of the problems we considered.

The no-collisions penalty for the continuous-time trajectory safety is only twice as expensive as the discrete no-collisions penalty since we have to calculate the support mapping of a convex shape with twice as many vertices. As a result, the narrow-phase collision detection takes about twice as long. The upshot is that the continuous collision cost solves problems with thin obstacles where the discrete-time cost fails to get the trajectory out of collision. An added benefit is that we can ensure continuous-time safety while parametrizing the trajectory with a small number of time steps, reducing the computational cost of the optimization.

5. Motion planning benchmark

Our evaluation is based on four test scenes included with the MoveIt! distribution—*bookshelves*, *countertop*, *industrial*, and *tunnel* scenes; and a living room scene imported from Google Sketchup. The set of planning problems was created as follows. For each scene we set up the robot in a number of diverse configurations. Each pair of configurations yields a planning problem. Our tests include 198 arm planning problems and 96 full-body problems (Figure 7). We ran all the experiments on a machine with an Intel i7 3.5 GHz CPU, and used Gurobi as the underlying Quadratic Program solver (Gurobi, 2012). The complete source code necessary to reproduce this set of experiments or evaluate a new planner is available at https://github.com/joschu/planning_benchmark.

We compared TrajOpt to open-source implementations of bi-directional RRT (Kuffner and LaValle, 2000) and a variant of KPIECE (Sucan and Kavraki, 2009) from OMPL/MoveIt! (Chitta et al., 2012; Cohen et al., 2012), that is part of the ROS motion planning libraries. All algorithms were run using default parameters and post-processed by the default smoother and shortcutting algorithm used by MoveIt!. We also compared TrajOpt to a recent implementation of CHOMP (Zucker et al., 2012) on the arm planning problems. We did not use CHOMP for the full-body planning problems because they were not supported in the available implementation.

Initialization: We tested both our algorithm and CHOMP under two conditions: single initialization and multiple initializations. For the single initialization, we used a straight line initialization in configuration space by linearly interpolating between start and goal configurations. For multiple initializations, we used the following methodology.

Arm planning problems: Prior to performing experiments, we manually selected four waypoints W_1, W_2, W_3, W_4 in joint space. These waypoints were fixed for all scenes and problems. Let S and G denote the start and goal states for a planning problem. Then we used the four initializations SW_1G, SW_2G, SW_3G, SW_4G, which linearly interpolate between S and W_i for the first $T/2$ time-steps, and then linearly interpolate between W_i and G for the next $T/2$ timesteps.

Full-body planning problems: We randomly sampled the environment for base positions (x, y, θ) with the arms tucked. After finding a collision-free configuration W of this sort, we initialized with the trajectory SWG as described above. We generated up to 5 initializations this way. Note that even though we initialize with tucked arms, the optimization typically untucks the arms to improve the cost.

Implementation details: Our current implementation of the continuous-time collision cost does not consider self-collisions, but we penalized self-collisions at discrete times as described in Section 4.1. For collision checking, we took the convex hull of the geometry of each link of the robot, where each link is made of one or more meshes. The termination conditions we used for the optimization were (a) a maximum of 40 iterations, (b) a minimum merit function improvement ratio of 10^{-4}, (c) a minimum trust region size 10^{-4}, and (d) a constant penalty scaling factor $k = 10$. We used the Bullet collision checker (Coumanns, 2012) for convex–convex collision queries. We used $T = 11$ timesteps for the arm and $T = 41$ timesteps for the full-body trajectories. The sampling-based planners were limited to 30 s on full-body planning problems.

Results: The results for arm planning are shown in Table 1 and for full-body planning are shown in Table 2. We evaluated TrajOpt and compared it with other planners in terms of (a) average computation time for all successful planning runs computed over all problems, and (b) average normalized trajectory length over all problems that is computed as the average of the trajectory lengths normalized by dividing by the shortest trajectory length for that problem across all planners (value of 1 for a planner indicates that the shortest trajectory was found by the planner for all problem instances). TrajOpt solves a higher percentage of problems on this benchmark, is computationally more efficient, and computes shorter trajectories on average. TrajOpt with multiple initializations outperformed the other approaches in both sets of problems. Multiple trajectory initializations are important to guide the optimization out of local minima and improves the success rate for both TrajOpt and CHOMP. Section 9 presents a discussion of why multiple trajectory initializations are important.

The bottom three rows of Table 1 indicate the reasons for failure of the different algorithms on the arm planning problems; the numbers indicate the fraction of problems with each failure case. The sampling-based planners (OMPL-RRTConnect and OMPL-LBKPIECE) failed when the search algorithm found a path but the subsequent path verification step found that it was in collision. This type of failure is possible because the search algorithm uses a fast collision checking method that is not perfectly accurate. In the CHOMP failures, the optimizer returned a path that was in collision or had joint limit violations. In the TrajOpt failures, the optimizer was not able to find a collision-free path after all of the initializations.

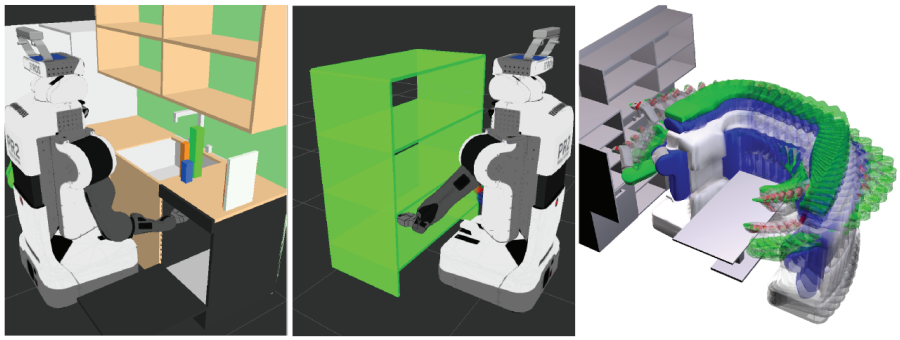


Fig. 7. Scenes in our benchmark tests. (*Left and center*) Two of the scenes used for the arm planning benchmark. (*Right*) A third scene, showing the path found by our planner on an 18-DOF full-body planning problem.

Table 1. Results on 198 arm planning problems for a PR2, involving 7 degrees of freedom.

	OMPL-RRTConnect	OMPL-LBKPIECE	CHOMP	CHOMP-Multi	TrajOpt	TrajOpt-Multi
Success fraction	0.838	0.833	0.677	0.833	0.843	0.990
Avg. time (s)	0.566	1.33	3.16	6.24	0.206	0.307
Avg. norm length	1.55	1.63	1.32	1.33	1.15	1.14
Failure: collision	0.162	0.167	0.278	0.116	0.157	0.010
Failure: joint limit	0	0	0.040	0.045	0	0
Failure: other	0	0	0.005	0.005	0	0

Table 2. Results on 96 full-body planning problems for a PR2, involving 18 degrees of freedom (two arms, torso, and base).

	OMPL-RRTConnect	OMPL-LBKPIECE	TrajOpt	TrajOpt-multi
Success fraction	0.41	0.51	0.73	0.88
Avg. time (s)	20.3	18.7	2.2	6.1
Avg. norm length	1.54	1.51	1.06	1.05

6. Physical experiments

6.1. Environment preprocessing

One of the main challenges in porting motion planning from simulation to reality is creating a useful representation of the environment's geometry. Depending on the scenario, the geometry data might be live data from a Kinect or laser range finder, or it might be a mesh produced by an offline mapping procedure. We used our algorithm with two different representations of environment geometry: (a) convex decomposition and (b) meshes.

Convex decomposition: Convex decomposition seeks to represent a general 3D volume approximately as a union of convex bodies (Lien and Amato, 2007). Hierarchical Approximate Convex Decomposition (HACD) (Mamou and Ghorbel, 2009) is a leading method for solving this problem, and it is similar to agglomerative clustering algorithms. It starts out with each triangle of a surface mesh as its own cluster, and it repeatedly merges pairs of clusters, where the choice of which clusters to merge is based on an objective function. The algorithm is terminated once a sufficiently small number of clusters is obtained. We used Khaled Mammou's implementation of HACD, which, in our experience, robustly produced good decompositions, even on the open meshes we generated from single depth images. Example code for generating meshes and convex decompositions from Kinect data, and then planning using our software package TrajOpt, is provided in a tutorial at http://rll.berkeley.edu/trajopt.

Meshes: Our algorithm also can be used directly with mesh data. The mesh is viewed as a soup of triangles (which are convex shapes), and we penalize collision between each triangle and the robot's links. For best performance, the mesh should first be simplified to contain as few triangles as possible while faithfully representing the geometry, e.g. see Cignoni et al. (1998).

6.2. Experiments

We performed several physical experiments involving a mobile robot (PR2) to explore two aspects of TrajOpt: (a) applying it to the "dirty" geometry data that we get from depth sensors such as the Kinect, and (b) validating if the full-body trajectories can be executed in practice. Our end-to-end system handled three full-body planning problems:

1. Grasp a piece of trash on a table and place it in a garbage bin under a table (one arm + base).

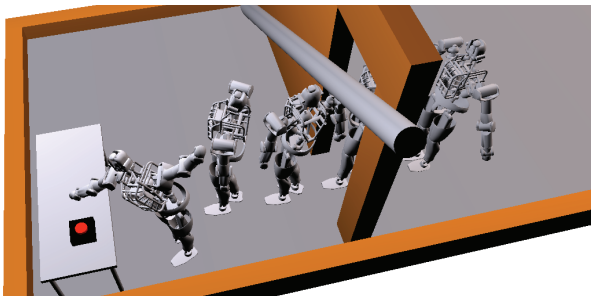


Fig. 8. The Atlas humanoid robot in simulation walking across the room while avoiding the door frame and other obstacles in the environment, and pushing a button. Each footstep was planned for separately using TrajOpt while maintaining static stability. Five time steps of the trajectory are shown.

2. Open a door, by following the appropriate pose trajectory to open the handle and push (two arms + torso + base).
3. Drive through an obstacle course, where the PR2 must adjust its torso height and arm position to fit through overhanging obstacles (two arms + torso + base).

The point clouds we used were obtained by mapping out the environment using SLAM and then preprocessing the map to obtain a convex decomposition. Videos of these experiments are available at http://rll.berkeley.edu/trajopt/ijrr.

7. Example applications with different constraints

7.1. Humanoid walking: Static stability

We used TrajOpt to planning a statically stable walking motion for the Atlas humanoid robot model. The degrees of freedom include all 28 joints and the 6 DOF pose, where we used the axis-angle (exp map) representation for the orientation. The walking motion is divided into four phases (a) left foot planted, (b) both feet planted, (c) right foot planted, and (d) both feet planted. We impose the constraint that the center of mass constantly lies above the convex hull of the planted foot or feet, corresponding to the zero-moment point stability criterion (Vukobratović and Borovac, 2004). The convex support polygon is now represented as an intersection of k half-planes, yielding k inequality constraints:

$$a_i x_{\text{cm}}(\boldsymbol{\theta}) + b_i y_{\text{cm}}(\boldsymbol{\theta}) + c_i \leq 0,\ i \in \{1, 2, \ldots, k\} \qquad (22)$$

where the ground-projection of the center of mass $(x_{\text{cm}}, y_{\text{cm}})$ is a nonlinear function of the robot configuration.

Using this approach, we use TrajOpt to plan a sequence of steps across a room, as shown in Figure 8. Each step is planned separately using the phases described above. The optimization is initialized with a stationary trajectory that remains at the initial configuration for T timesteps, where $T = 10$. The robot is able to satisfy these stability and footstep placement constraints while ducking under an obstacle and performing the desired task of pushing a button.

7.2. Pose constraints

TrajOpt can readily incorporate kinematic constraints, e.g. the constraint that a redundant robot's end effector is at a certain pose at the end of the trajectory. A pose constraint can be formulated as follows. Let $F_{\text{targ}} = \begin{bmatrix} R_{\text{targ}} & \mathbf{p}_{\text{targ}} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \in SE(3)$ denote the target pose of the gripper, and let $F_{\text{cur}}(\mathbf{x})$ be the current pose. Then $F_{\text{targ}}^{-1} F_{\text{cur}}(\mathbf{x})$ gives the pose error, measured in the frame of the target pose. This pose error can be represented as the 6D error vector:

$$h(\mathbf{x}) = \log(F_{\text{targ}}^{-1} F_{\text{cur}}(\mathbf{x})) = (t_x, t_y, t_z, r_x, r_y, r_z) \qquad (23)$$

where (t_x, t_y, t_z) is the translation part, and (r_x, r_y, r_z) is the axis-angle representation of the rotation part obtained using the log operator. We refer the reader to the appendix for additional details on the log operator.

One can also impose partial orientation constraints. For example, consider the constraint that the robot is holding a box that must remain upright. The orientation constraint is an equality constraint, namely that an error vector $(v_x^w, v_y^w)(\mathbf{x})$ vanishes. Here, $\mathbf{v}$ is a vector that is fixed in the box frame and should point upwards in the world frame.

Figure 9 shows our algorithm planning a series of motions that pick boxes from a stack. Our algorithm typically plans each motion in 30–50 ms.

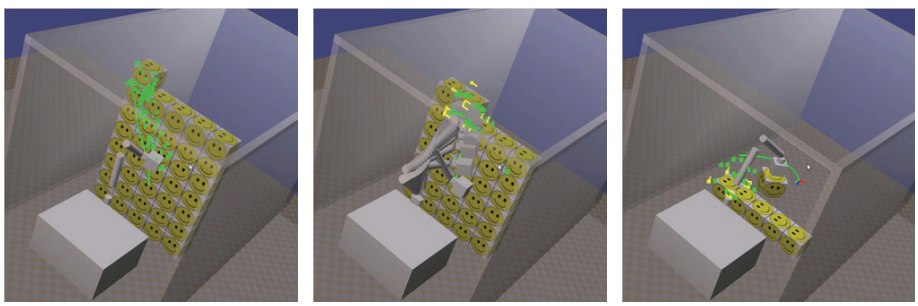


Fig. 9. Several stages of a box picking procedure, in which boxes are taken from the stack and moved to the side. The box, and hence the end effector of the robot arm, is subject to pose constraints.

8. Needle steering and channel layout planning

The need to plan curvature-constrained trajectories in 3D environments arises in a wide variety of domains. For instance, a new class of highly flexible, bevel-tip needles are being developed that enable the needle to move along constant curvature trajectories within tissue when a forward pushing force is applied and the direction of motion can be changed by reorienting the bevel tip through twisting of the needle at its base (Webster et al., 2006). They facilitate access to previously inaccessible clinical targets while avoiding obstacles such as sensitive anatomical tissues (e.g. vital organs and vessels) and impenetrable structures (e.g. bones), as shown in Figure 1(e). Another important application is the design of multiple bounded curvature channels in intracavitary 3D printed implants through which a radioactive source is guided for delivering radiation doses for high dose rate brachytherapy (HDR-BT) (Figure 1(f)) (Garg et al., 2013). The need for designing such channels also arises in applications such as turbine blade design for delivering coolant through the blades to cool them during operation (Han et al., 2013), and planning bounded curvature trajectories for unmanned aerial vehicles (UAVs) (Yang and Sukkarieh, 2010).

Computing collision-free, curvature-constrained trajectories in 3D environments with obstacles is challenging because it requires planning in the $SE(3)$ configuration space consisting of the 6D pose (position and orientation). We formulate this as a constrained, non-convex trajectory optimization problem defined over manifolds such as the $SE(3)$ Lie group instead of vector spaces of the form $\mathbb{R}^n$. We accomplish this by iteratively optimizing over increments to the trajectory, defined in terms of the corresponding Lie algebra ($\mathfrak{se}(3)$ in our case) (Saccon et al., 2013). Second, we consider the problem of planning multiple trajectories that are mutually collision-free, which arises in planning trajectories for multiple needles for medical procedures (Xu et al., 2009), multiple channels in intracavitary implants (Garg et al., 2013), or simultaneously planning for multiple UAVs (Shanmugavel et al., 2007).

Although the following formulation is specific to needle steering and channel planning, it can be easily generalized to other curvature-constrained planning problems.

8.1. Related work

Planning a curvature-constrained shortest path in a 2D plane between two configurations for a Dubins car robot has been extensively studied (Dubins, 1957; Reeds and Shepp, 1990). Webster et al. (2006) experimentally showed that bevel-tipped steerable needles follow paths of constant curvature when inserted into tissue. Planning constant curvature trajectories for such needles in a plane has also been explored (Alterovitz et al., 2007; Bernardes et al., 2013).

Computing collision-free, curvature-constrained trajectories in 3D environments requires planning in the 6D configuration space consisting of both position and orientation. Existing optimal motion planning approaches that rely on discretizing the configuration space (Pivtoraiko, 2012) or sampling-based planners like RRT* (Karaman and Frazzoli, 2011) require solving a two-point boundary value problem (BVP) for connecting two states in $SE(3)$, closed-form solutions for which are not known (Belta and Kumar, 2002). Duindam et al. (2010) proposed a fast, optimal planner based on inverse kinematics, but this approach does not consider obstacle avoidance. Xu et al. (2008, 2009) used rapidly exploring random trees (RRT) (LaValle, 2006) which offers a probabilistically complete, but computationally intensive, algorithm to search for collision-free trajectories. Duindam et al. (2008) formulated planning for steerable needles as a non-convex optimization problem, which computes collision-free solutions in a few seconds but collision avoidance is treated as a cost and not as a hard constraint. Patil and Alterovitz (2010); Patil et al. (2014) proposed a RRT planner which plans bounded curvature trajectories for a needle by relying on duty-cycled spinning of the needle during insertion (Minhas et al., 2007; Majewicz et al., 2014). However, this can cause excessive tissue damage (Engh et al., 2010). This approach was also used for designing bounded curvature channels within implants (Garg et al., 2013) but the issue of optimality of channel layout was not addressed. In recent years, extensions to planning curvature-constrained trajectories in 3D have been proposed for unmanned aerial vehicles (UAVs) in environments without obstacles (Shanmugavel et al., 2007), and with obstacles (Hwangbo et al., 2007; Yang and Sukkarieh, 2010). These methods do not consider the problem of planning constant curvature trajectories in 3D.

Prior work on trajectory optimization on Lie groups has proposed Newton-like optimization methods (Absil et al., 2009), direct (collocation) methods for trajectory optimization for continuous time optimal control problems (Saccon et al., 2013), and primitive-based motion planning (Frazzoli et al., 2005). However, these approaches do not address the issue of avoiding collisions with obstacles in the environment.

8.2. Problem definition and formulation

We assume that a trajectory is discretized into time intervals $\mathcal{T} = \{0, 1, \ldots, T\}$. At each time step $t \in \mathcal{T}$, a trajectory waypoint is parameterized by a pose $X_t = \begin{bmatrix} R_t & \mathbf{p}_t \\ \mathbf{0}_3^T & 1 \end{bmatrix} \in SE(3)$, where $\mathbf{p}_t \in \mathbb{R}^3$ is the position and $R_t \in SO(3)$ is the rotation matrix that encodes the orientation of the waypoint frame relative to a world coordinate frame (Figure 10).

The planning objective can then be stated as:

Input: Set of obstacles $\mathcal{O}$, an entry zone $\mathcal{P}_{\text{entry}}$, a target zone $\mathcal{P}_{\text{target}}$, the maximum curvature $\kappa_{\max}$, and the discretization parameter T.

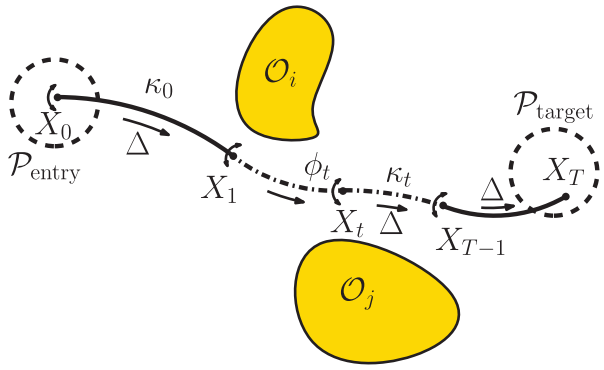


Fig. 10. A discretized curvature-constrained trajectory is parameterized as $\{X_0, \ldots, X_t, \ldots, X_T\}$, where $X_t \in SE(3)$ is the pose of the waypoint frame relative to a world coordinate frame at each time step t.

Output: Given an entry zone $\mathcal{P}_{\text{entry}}$ and a target zone $\mathcal{P}_{\text{target}}$, determine a locally optimal, collision-free, and curvature-constrained trajectory $\{X_t : t \in \mathcal{T}\}$ with $X_0 \in \mathcal{P}_{\text{entry}}$ and $X_T \in \mathcal{P}_{\text{target}}$, or report that no feasible trajectory can be found.

We first describe the curvature-constrained kinematic model used in this work and then formulate the planning objective as a constrained, non-convex optimization problem.

Curvature-constrained kinematic model: In this work, we assume that the trajectory is composed of a sequence of $(T-1)$ circular arcs, each connecting a pose X_t to the subsequent pose X_{t+1} and of curvature κ_t. Depending on the application, the trajectory may be required to have a constant curvature $\kappa_t = \kappa_{\max}$ for all time steps, or a bounded curvature $0 \leq \kappa_t \leq \kappa_{\max}$ at each time step.

We make two design choices in formulating the curvature-constrained kinematics. First, we constrain the length of each circular arc Δ to be the same for all time steps. One can just as easily have a separate length parameter Δ_t for each time step. However, in our experiments, we observed that some of these Δ_t values shrink to 0 as a result of the optimization, producing large gaps between time steps which is not suitable for collision checking with obstacles in the environment.

Second, we use a "stop-and-turn" strategy for the kinematics, i.e. at each time step $t : 0 \leq t \leq T-1$, we apply a rotation ϕ_t to the pose X_t and then propagate the frame by a distance Δ to arrive at X_{t+1}. This is a natural choice for needle steering, since it corresponds to first twisting the base of the needle, and then pushing it forward, which induces less damage than constantly twisting the needle tip while pushing it. This strategy also results in channels that are easier for catheters to go through. See Figure 10 for an illustration. Without loss of generality, we assume that the object (either the needle tip or a small trajectory segment for the channels) is oriented along the positive z-axis. Hence, the poses at adjacent time steps X_t and X_{t+1} are related as:

$$X_{t+1} = \exp(\mathbf{v}_t^{\wedge}) \cdot \exp(\mathbf{w}_t^{\wedge}) \cdot X_t \tag{24}$$

where $\mathbf{w}_t = [\,0\;0\;0\;0\;0\;\phi_t\,]^{\mathrm{T}}$ and $\mathbf{v}_t = [\,0\;0\;\Delta\;\Delta\kappa_t\;0\;0\,]^{\mathrm{T}}$ are the twist vectors corresponding to the rotation ϕ_t and propagating the frame by distance Δ, respectively. We refer the reader to the appendix and to the excellent treatise on the $SE(3)$ Lie group by Murray and Shankar (1994) for details on the $^{\wedge} : \mathbb{R}^6 \to \mathfrak{se}(3)$ and $\exp : \mathfrak{se}(3) \to SE(3)$ operators.

Optimization Formulation: For notational convenience, we concatenate the states from all time steps as $\mathcal{X} = \{X_t : t \in \mathcal{T}\}$ and the control variables as $\mathcal{U} = \{\phi_t, \kappa_t : t \in \mathcal{T}, \Delta\}$. The planning objective is transcribed as a constrained, non-convex trajectory optimization problem as given below:

$$\min_{\mathcal{X},\mathcal{U}} \; \alpha_{\Delta}\text{Cost}_{\Delta} + \alpha_{\phi}\text{Cost}_{\phi} + \alpha_{\mathcal{O}}\text{Cost}_{\mathcal{O}} \tag{25a}$$

subject to (25b)

$$\log(X_{t+1} \cdot (\exp(\mathbf{v}_t^{\wedge}) \cdot \exp(\mathbf{w}_t^{\wedge}) \cdot X_t)^{-1})^{\vee} = \mathbf{0}_6 \tag{25c}$$

$$X_0 \in \mathcal{P}_{\text{entry}},\; X_T \in \mathcal{P}_{\text{target}} \tag{25d}$$

$$\text{sd}(X_t, X_{t+1}, \mathcal{O}) \geq d_{\text{safe}} + d_{\text{arc}} \tag{25e}$$

$$-\pi \leq \phi_t \leq \pi \tag{25f}$$

$$\kappa_t = \kappa_{\max} \quad \text{or} \quad 0 \leq \kappa_t \leq \kappa_{\max} \tag{25g}$$

$$\Delta \sum_{t=0}^{T-1} \kappa_t \leq c_{\max} \quad \text{for channel planning} \tag{25h}$$

The constraints and costs are described in detail below.

8.2.1. Kinematics constraint (Equation (25c)). We transform the kinematic constraint from Equation (24) to a standard non-convex equality constraint form by using the log map and relying on the identity $\log(\mathbf{I}_{4\times4}) = \mathbf{0}_6$. We refer the reader to the appendix for more details.

8.2.2. Collision constraint (Equation (25e)). We impose constraints to ensure that the trajectory avoids collisions, where $\text{sd}(X_t, X_{t+1}, \mathcal{O})$ is the signed distance between the trajectory segment in time interval $[t, t+1]$ and the set of obstacles $\mathcal{O}$. The signed distance corresponds to the minimum translation distance required to either put two geometric shapes in contact or separate them if they are overlapping. Two objects are non-colliding if the signed distance is positive, and we want to ensure that the trajectory has a user-defined safety margin d_{safe}. The distance between two convex shapes can be calculated by the GJK algorithm (Gilbert et al., 1988) and the penetration depth is calculated by the EPA (Van den Bergen, 2001). We approximate the segment by the convex hull of the object (the needle tip or a small segment on the channel) between time t and $t+1$, and we account for the approximation error in rotation by adding an error correction term d_{arc}. Instead of numerically computing the gradient, we linearize the signed distance using the contact normal $\hat{\mathbf{n}}$. We include the continuous-time non-convex no-collisions constraint is included as a ℓ_1 penalty in the optimization (Section 4.2).

8.2.3. Total curvature constraint (Equation (25h)). For channel planning, we constrain the total curvature of the

trajectory to ensure that catheters carrying the radioactive source can be pushed through the channels without buckling (Garg et al., 2013).

8.2.4. Costs (Equation (25a)). To penalize tissue damage for needle steering and to optimize channel lengths for minimum radiation exposure, the objective imposes costs on the total length of the trajectory and the twists at each time step:

$$\text{Cost}_\Delta = T\Delta \quad \text{and} \quad \text{Cost}_\phi = \sum_{t=0}^{T-1} \phi_t^2 \tag{26}$$

For needle steering, we add an extra term to favor large minimum clearance from obstacles to deal with expected needle deflections during execution:

$$\text{Cost}_\mathcal{O} = - \min_{\substack{0\le t\le T-1 \\ \mathcal{O}_i\in\mathcal{O}}} \text{sd}(X_t, X_{t+1}, \mathcal{O}_i) \tag{27}$$

Instead of directly including the non-convex cost term $\text{Cost}_\mathcal{O}$ in the objective, we include an auxiliary variable $d_{\min}$ in the optimization and reformulate the cost as

$$\text{Cost}_\mathcal{O} = -d_{\min}, \qquad d_{\min} \le \text{sd}(X_t, X_{t+1}, \mathcal{O}_i) \tag{28}$$

The objective (25a) is a weighted sum of the costs above, where $\alpha_\Delta, \alpha_\phi, \alpha_\mathcal{O} \ge 0$ are user-defined, non-negative coefficients to leverage different costs. A relatively large $\alpha_\mathcal{O}$, for instance, may result in trajectory with larger clearance from obstacles, at the expense of a longer trajectory.

8.3. Trajectory optimization over $SE(3)$

The optimization problem outlined in Equation (25) is, however, described directly over the set of poses $\mathcal{X}$. One could use a global parameterization of the rotation group, such as axis-angle coordinates or Euler angles. The drawback of those parameterizations is that they distort the geometry—e.g. consider how a map of the world is distorted around the poles. This distortion can severely slow down an optimization algorithm, by reducing the neighborhood where local (first- and second-order) approximations are good.

In this work, we generalize sequential convex optimization to the case where the domain is a differentiable manifold such as the $SE(3)$ Lie group rather than $\mathbb{R}^n$ by considering a local coordinate parameterization of the manifold (Saccon et al., 2013). This parameterization is given by the Lie algebra $\mathfrak{se}(3)$, which is defined as the tangent vector space at the identity of $SE(3)$. We refer the reader to the appendix for additional details.

In this work, we construct and solve each convex subproblem in terms of the increments to the previous solution. At the ith iteration of SQP, let $\bar{\mathcal{X}}^{(i)} = \{\bar{\mathbf{x}}_0^{(i)}, \ldots, \bar{\mathbf{x}}_T^{(i)}\}$ be the sequence of incremental twists (step) computed by solving the convex subproblem. Given a trajectory consisting of a sequence of nominal poses $\hat{\mathcal{X}}^{(i)} = \{\hat{X}_0^{(i)}, \ldots, \hat{X}_T^{(i)}\}$, the subsequent sequence of poses is obtained by applying $\bar{\mathcal{X}}^{(i)}$ as $\hat{\mathcal{X}}^{(i+1)} = \{\exp(\bar{\mathbf{x}}_0^{(i)\wedge}) \cdot \hat{X}_0^{(i)}, \ldots, \exp(\bar{\mathbf{x}}_T^{(i)\wedge}) \cdot \hat{X}_T^{(i)}\}$.

Convexification: For trajectory optimization problems, there are two ways to construct locally convex approximations of the costs and constraints for setting up the convex subproblem. One can either convexify the costs and constraints directly around the current solution $\hat{\mathcal{X}}^{(i)}$, which might correspond to an infeasible trajectory that does not satisfy the kinematic constraints (Equation (24)). Alternatively, we can forward integrate the computed controls and then construct the convex approximation around the integrated trajectory, which is guaranteed to satisfy all kinematic constraints, but the trajectory might violate the constraints on the entry zone and target zone. It is easier to satisfy constraints on the start and target zones without forward integration but the differential curvature constraint is difficult to satisfy. We present a detailed comparison of these two methods below.

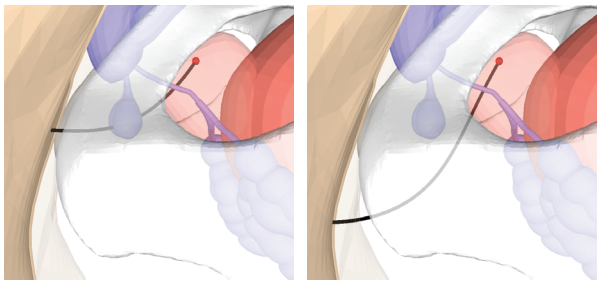
Multi-trajectory optimization: In this work, we also consider the problem of computing multiple curvature-constrained trajectories that are mutually collision-free. The complexity of solving n_{traj} trajectories simultaneously while avoiding collisions between trajectories increases rapidly as a function of n_{traj}. Although the size of the optimization vector grows linearly, the number of collision constraints between trajectories grows quadratically in n_{traj}. In addition, the chances of getting stuck in an infeasible local optima becomes much higher as n_{traj} increases. A natural extension is to solve for each trajectory sequentially in a predefined order while avoiding collisions with previously computed trajectories. However this approach may result in conflicts where trajectories that are computed first may collide with the target zone of trajectories that need to be solved for later.

Instead, we repeatedly compute each trajectory individually, where the optimization is initialized by a perturbed version of the previous solution. The previously computed trajectories are added as static obstacles to the environment since the objective is to compute trajectories that are mutually collision-free. Randomly perturbing the solution from previous optimization runs also has the desirable side effect of perturbing the optimization to potentially finding better local optima.

8.4. Simulation experiments

We experimentally evaluated our approach in two real-world applications involving medical needle steering and designing channel layouts for intracavitary brachytherapy. We implemented our algorithm in C++ and ran all the experiments on a machine with a Intel i7 3.5 GHz CPU and used Gurobi as the underlying Quadratic Program solver.

Medical needle steering: We used an anatomical model of the human male pelvic region to simulate needle insertion in tissue for delivering radioactive doses to targets



(a) (b)

Fig. 11. Changing the value of the parameter $\alpha_{\mathcal{O}}$ influences the clearance of the trajectory from obstacles in the environment. Zoomed in view of the male prostate region (target inside prostate shown in red). (a) Smaller clearance from obstacles (Cowper's glands) with $\alpha_{\mathcal{O}} = 1$ resulting in a potentially unsafe trajectory. (b) Larger clearance from obstacles with $\alpha_{\mathcal{O}} = 10$.

within the prostate. We considered randomly sampled targets within the prostate for our experiments. We set the entry zone to be a 0.1 cm $\times$ 5 cm $\times$ 2.5 cm region on the perineum (skin) through which needles are typically inserted for needle-based prostate procedures. The target zones were modeled as spheres around the target points with radius 0.25 cm, within the range of average placement errors ($\approx$ 0.63 cm) encountered during procedures performed by experienced clinicians (Taschereau et al., 2000). The average distance between the entry zone and the target zone is 10 cm and and we set $\kappa_{\max} = 0.125\,\text{cm}^{-1}$. We used $T = 10$ time steps for our experiments, such that the step length was roughly 1 cm. For the objective function, we used $\alpha_{\Delta} = \alpha_{\phi} = 1$, and we compared the planned trajectory with different choices of the clearance coefficient $\alpha_{\mathcal{O}}$.

We compared the effect forward integration on the entire trajectory for constructing the underlying convex subproblems. We also compared the performance of our optimization-based approach with a sampling-based RRT planner (Xu et al., 2008) for computing constant curvature trajectories for the needle. The planner was modified to plan backwards starting from target zones because it is easier to compute feasible constant curvature trajectories.

Planning for a single needle: We first analyzed the planned trajectory for single needle insertion using 400 sampled points in the prostate. In addition to the setup above, we require that the needle insertion axis is at a deviation of at most 5° from the horizontal, which is a restriction usually imposed by needle steering hardware that constrains the needle to be horizontal. We do not constrain the orientation of the needle tip at the target. We enforced a safety distance $d_{\text{safe}} = 0.25$ cm between the trajectory and obstacles. The error correction term for rotations (Section 4.2) is computed to be $d_{\text{arc}} = 0.001$ cm, which is ignored considering the scale of the environment we are planning in (of the order of cm). We compared the planned trajectory with $\alpha_{\mathcal{O}} = 1$ or $\alpha_{\mathcal{O}} = 10$, examples of which are shown in Figures 11(a) and 11(b). Using a larger clearance coefficient results in trajectories farther away from obstacles, at the expense of slightly longer paths.

For each task, we repeatedly ran the optimization initialized by a perturbed solution of the previous run, and we allowed up to five reruns. We evaluated the performance of no forward integration versus forward integration in terms of the average running time, percentage of solved problems, and quality metrics for the converged solutions. From the statistics listed in Table 3, we can see that forward integration outperforms no forward integration in terms of percentage of solved problems and running times. It is worth noting that the optimization solves a larger percentage of problems with $\alpha_{\mathcal{O}} = 10$ as compared to using $\alpha_{\mathcal{O}} = 1$ because in the latter case, the optimization finds it difficult to simultaneously satisfy both the kinematics constraint (Equation (25c)) and the collision avoidance constraint (Equation (25e)) when the trajectory is closer to obstacles and has less free space in the environment for improvement.

Our approach outperforms the RRT planner in terms of the number of problems solved. Here, the RRT planner was allotted 10 s to find a solution, pending which it reported that a solution could not be found. The trajectories computed using the RRT planner also have a very high twist cost, which is a result of the randomized nature of the planning algorithm. Since the twist cost is directly correlated with tissue damage, the trajectories computed using our approach are preferable over those computed by a randomized planner.

Planning for multiple needles: We analyzed the performance of our algorithm planning for five needle trajectories using 1000 sampled points within the prostate (200 trials). We compared the result of no forward integration vs forward integration, applying our proposed multi-trajectory planning algorithm. Using forward integration offers an advantage over not using it in terms of computational time required to compute a feasible solution and the quality of trajectories computed. Figure 1(e) shows planned trajectories for a single trial. Table 4 summarizes our result, which shows the advantage of our proposed approach. Our approach outperforms the RRT planner in terms of the number of problems solved. The trajectories computed using the RRT planner have a very high twist cost, which is also undesirable. We also tested planning for multiple trajectories simultaneously, but the running time was too long and the algorithm failed to find a solution for three needles or more.

Channel layout design: We set up a simplified scene for designing the channel layout. We consider a scenario where a 3D printed implant is prepared for treatment of OB/GYN tumors (both vaginal and cervical), as shown in Figure 1(f).

The implant was modeled as a cylinder of height 7 cm and radius 2.5 cm, with a hemisphere on top with radius 2.5 cm. The dimensions of the implant was designed based on dimensions reported by Garg et al. (2013). We placed three tumors and picked eight (oriented) target poses inside

Table 3. Single needle planning: Sampling-based RRT planner versus TrajOpt.

	RRT	TrajOpt			
		No forward integration $\alpha_{\mathcal{O}} = 1$	Forward integration $\alpha_{\mathcal{O}} = 10$	No forward integration $\alpha_{\mathcal{O}} = 10$	Forward integration $\alpha_{\mathcal{O}} = 10$
Success fraction	0.67	0.76	0.80	0.79	0.89
Time (s)	9.8 ± 8.1	1.8 ± 1.2	1.6 ± 1.7	1.9 ± 1.3	1.8 ± 1.7
Path length: cm	11.1 ± 1.5	11.3 ± 1.4	11.6 ± 1.7	11.9 ± 1.7	13.1 ± 2.3
Twist cost: radians	34.9 ± 10.0	1.4 ± 1.4	1.0 ± 1.0	1.6 ± 1.6	1.0 ± 1.0
Clearance: cm	0.5 ± 0.4	0.7 ± 0.5	0.5 ± 0.3	1.3 ± 0.4	1.2 ± 0.5

Table 4. Multiple needle planning: Sampling-based RRT planner versus TrajOpt.

	RRT	TrajOpt	
		No forward integration	Forward integration
Success fraction	0.48	0.75	0.79
time (s)	50.0 ± 19.0	18.0 ± 9.0	15.3 ± 15.2
Path length: cm	54.6 ± 3.1	53.9 ± 2.5	56.5 ± 3.4
Twist cost: radians	168.3 ± 28.4	3.8 ± 1.5	2.5 ± 1.8
Clearance: cm	0.1 ± 0.08	0.1 ± 0.03	0.1 ± 0.06

Table 5. Channel layout planning: Sampling-based RRT planner versus TrajOpt.

Success fraction	0.74	0.98
Time (s)	30.8 ± 17.9	27.7 ± 9.8
Path length: cm	41.3 ± 0.3	38.9 ± 0.1
Twist cost: radians	65.5 ± 8.4	4.1 ± 1.1

the implant. We set the entry region to be the base of the implant, with a deviation angle at most 10° to the perpendicular direction. We require that the curvature along the path is at most $1\,\mathrm{cm}^{-1}$ and that the total curvature on the trajectory (Equation (25h)) is at most 1.57. This constraint is important to ensure that catheters carrying the radioactive seed can be pushed through the channels. Instead of planning forward from the entry to the target, we planned backwards from the target to the entry zone using collocation with backward integration, since the entry constraint is much easier to satisfy than the target constraint. Figure 1(f) shows a channel layout computed using our method.

We compared the performance of our approach with a highly optimized RRT-based planner (Garg et al., 2013) proposed for this specific application (Table 5). Both the RRT-based approach and our approach have a randomization aspect associated with them—while the RRT uses random sampling, our multi-trajectory planning procedure uses random perturbations to initialize the optimization. We solved the same problem 100 times to investigate the randomized aspect of both approaches. Our approach is able to compute a feasible solution in almost all cases, whereas the RRT algorithm fails more often to find a feasible solution. The RRT planner also computed plans that have a higher cumulative path length and twist cost as compared to the solution computed using our approach, which is undesirable.

9. Discussion

In this section, we compare our approach vis-à-vis CHOMP (Ratliff et al., 2009; Zucker et al., 2012) and sampling-based motion planners (LaValle, 2006), and discuss the importance of trajectory initialization for trajectory optimization methods.

9.1. Comparison with CHOMP

Our approach uses optimization in the same spirit as CHOMP, with the following key differences: (a) the numerical optimization method used, and (b) the method of checking for collisions and penalizing them.

a. *Distance fields versus convex–convex collision checking*: CHOMP uses the Euclidean distance transform—a precomputed function on a voxel grid that specifies the distance to the nearest obstacle, or the distance out of an obstacle. Typically each link of the robot is approximated as a union of spheres, since the distance between a sphere and an obstacle can be bounded based on the distance field. The advantage of distance fields is that checking a link for collision against the environment requires constant time and does not depend on the complexity of the environment. On the other hand, spheres and distance fields are arguably not very well suited to situations where one needs to accurately model geometry, which is why collision-detection methods based on meshes and convex primitives are more prevalent in applications like real-time physics simulation (Coumanns, 2012) for speed and accuracy. Whereas convex–convex collision detection takes two colliding shapes and computes the minimal translation to get them out of collision, the distance field (and its gradient) merely computes how to get each robot point (or sphere)

out of collision; however, two points may disagree on which way to go. Thus convex–convex collision detection arguably provides a better local approximation of configuration space, allowing us to formulate a better shaped objective.

The CHOMP objective is designed to be invariant to reparametrization of the trajectory. This invariance property makes the objective better shaped, helping the gradient pull the trajectory out of an obstacle instead of encouraging it to jump through the obstacle faster. Our method of collision checking against the swept-out robot shape achieves this result in a completely different way.

b. *Projected gradient descent versus SQP*: CHOMP uses (preconditioned) projected gradient descent, i.e. it takes steps $\mathbf{x} \leftarrow \text{Proj}(\mathbf{x} - A^{-1}\nabla f(\mathbf{x}))$, whereas our method uses sequential quadratic programming (SQP), which constructs a locally quadratic approximation of the objective and locally linearizes constraints. Taking a projected gradient step is cheaper than solving a QP. However, an advantage of sequential quadratic programming is that it can handle infeasible initializations and other constraints on the motion using penalties and merit functions, as described in Section 3. We note that popular non-convex optimization solvers such as KNITRO and SNOPT also use an SQP variant. Another advantage of using SQP is that there is additional flexibility in adding other cost terms to the objective and constraints, which allows TrajOpt to tackle a larger range of planning problems, including planning for underactuated, non-holonomic systems.

9.2. Comparison with sampling-based planners

It is important to note that our approach is not a replacement for sampling-based motion planning methods such as RRTs (LaValle, 2006). It is not expected to find solutions to difficult planning problems (e.g. bug trap or maze path finding) and is not guaranteed to find a solution if one exists, i.e. it does not offer probabilistic completeness guarantees. However, our experiments indicate that our approach can still efficiently compute locally optimal, collision-free trajectories from scratch using infeasible trajectory initializations as opposed to smoothing a previously computed collision-free trajectory. In contrast to other trajectory smoothing methods, our approach does not necessarily require a collision-free trajectory initialization to begin with.

9.3. Importance of trajectory initialization

Trajectory optimization for motion planning is a challenging non-convex constrained optimization problem. Given an initial trajectory that may contain collisions and violate constraints, trajectory optimization methods such as TrajOpt and CHOMP can often quickly converge to a high-quality, locally optimal solution. However, these methods suffer from a critical limitation: their performance heavily depends on the provided trajectory initialization and they are not guaranteed to find a collision-free solution as the no-collisions constraints in the optimization are non-convex.

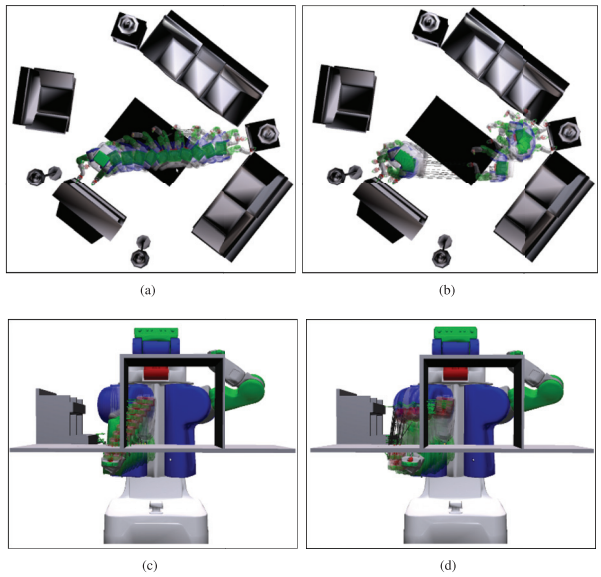


Fig. 12. Failure cases when using TrajOpt. (a) Initial path for full-body planning. (b) The trajectory optimization outcome, which is stuck in an infeasible condition. (c) The initial path for the arm planning and the collision cannot be resolved in the final trajectory (d).

For instance, certain initializations passing through obstacles in unfavorable ways may get stuck in infeasible solutions and cannot resolve all the collisions in the final outcome, as illustrated in Figure 12. Figure 13 shows some scenarios illustrating how trajectory optimization tends to get stuck in local optima that are not collision-free. It is important whether the signed distance normal is consistent between adjacent links or adjacent waypoints in an initial trajectory, else a bad initialization tends to have adjacent waypoints which push the optimization in opposing directions. As a consequence, these methods typically require multiple initializations. This explains why the use of multiple trajectory initializations performs better for challenging planning problems (Tables 1 and 2).

Sampling-based motion planning methods such as RRTs or PRMs could be used to compute a feasible initialization that could be used to seed our optimization approach. This could potentially improve the success rate of our approach at the cost of additional computation.

10. Source code and reproducibility

All of our source code is available as a BSD-licensed open-source package called TrajOpt that is freely available at http://rll.berkeley.edu/trajopt. Optimization problems can be constructed and solved using the underlying C++ API or through Python bindings. Trajectory optimization problems

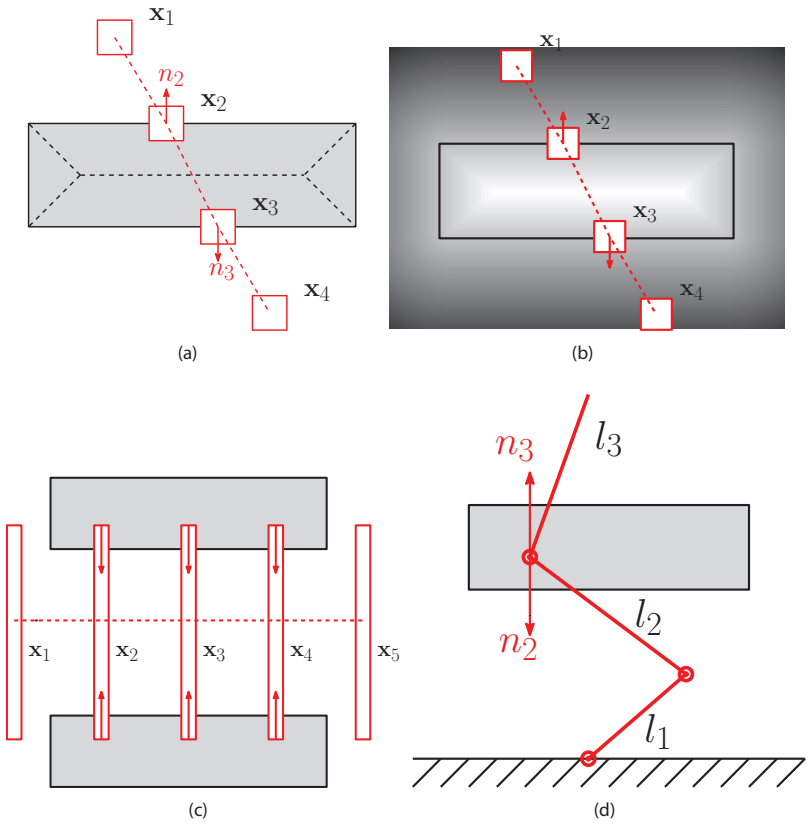


Fig. 13. Illustration of typical reasons for trajectory optimization to get stuck in local optima that are not collision-free. (a) The gradient based on penetration depth may push waypoints in in-consistent directions. (b) The gradient based on distance fields has the same problem. (c) When a robot collides simultaneously with multiple obstacles, the robot may get stuck in an infeasible local optimum as different obstacles push the robot in different directions. (d) For a robot with multiple links, the gradient may result in inconsistent directions for different links. $\mathbf{x}_i$ in these figures denote configurations at different time steps along the trajectory.

can be specified in JSON string that specifies the costs, constraints, degrees of freedom, and number of timesteps. We are also working on a MoveIt plugin (Chitta et al., 2012) so our software can be used along with ROS tools.

For robot and environment representation, we use OpenRAVE, and for collision checking we use Bullet, because of the high-performance GJK-EPA implementation and collision detection pipeline. Two different backends can be used for solving the convex subproblems: (a) Gurobi, a commercial solver, which is free for academic use (Gurobi, 2012); and (b) BPMPD (Mészáros, 1999), a free solver included in our software distribution.

The benchmark results presented in this paper can be reproduced by running scripts provided at http://rll.berkeley.edu/trajopt/ijrr. Various examples, including humanoid walking, arm planning with orientation constraints, and curvature-constrained trajectory planning for medical needle steering and designing channel layouts, are included with our software distribution.

11. Conclusion

We presented TrajOpt, a trajectory optimization approach for solving robot motion planning problems. At the core of our approach is the use of sequential convex optimization with ℓ_1 penalty terms for satisfying constraints, an efficient formulation of the no-collision constraint in terms of the signed distance, which can be computed efficiently for convex objects, and the use of support mapping representation to efficiently formulate the continuous-time no-collision constraints.

We benchmarked TrajOpt against sampling-based planners from OMPL and CHOMP. Our experiments indicate that TrajOpt offers considerable promise for solving a wide variety of high-dimensional motion planning problems. We presented a discussion of the importance of trajectory initialization for optimization based approaches. We also presented an extension of our trajectory optimization approach to planning curvature-constrained trajectories in 3D environments with obstacles. The source code for all the reported experiments and the associated benchmark has been made available freely for the benefit of the research community.

Acknowledgements

We thank Jeff Trinkle, Dmitry Berenson, Nikita Kitaev, and anonymous reviewers for insightful discussions and comments on the

paper. We thank Kurt Konolige and Ethan Rublee from Industrial Perception Inc. for supporting this work and providing valuable feedback. We thank Ioan Sucan and Sachin Chitta for help with MoveIt!, and we thank Anca Dragan, Chris Dellin, and Siddhartha Srinivasa for help with CHOMP.

Funding

This research was supported in part by the National Science Foundation (NSF) (grant number IIS-1227536: Multilateral Manipulation by Human-Robot Collaborative Systems), by the Air Force Office of Scientific Research (AFOSR) (Young Investigator Program (YIP) grant number FA9550-12-1-0345), by a Sloan Fellowship, and by the Intel Science and Technology Center on Embedded Computing.

References

Absil P, Mahony R and Sepulchre R (2009) *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press.

Alterovitz R, Siméon T and Goldberg K (2007) The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In: *Proceedings of Robotics: Science and systems (RSS)*, Atlanta, GA, USA, 27–30 June 2007.

Belta C and Kumar V (2002) Euclidean metrics for motion generation on SE(3). *Journal of Mechanical Engineering Science* 216(1): 47–60.

Bernardes MC, Adorno BV, Poignet P, et al. (2013) Robot-assisted automatic insertion of steerable needles with closed-loop imaging feedback and intraoperative trajectory replanning. *Mechatronics* 23(6): 630–645.

Betts JT (2010) *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, Vol. 19. Philadelphia, PA: Society for Industrial & Applied Mathematics.

Brock O and Khatib O (2002) Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research* 21(12): 1031–1052.

Chitta S, Sucan I and Cousins S (2012) Moveit![ROS topics]. *IEEE Robotics and Automation Magazine* 19(1): 18–19.

Cignoni P, Montani C and Scopigno R (1998) A comparison of mesh simplification algorithms. *Computers and Graphics* 22(1): 37–54.

Cohen B, Sucan I and Chitta S (2012) A generic infrastructure for benchmarking motion planners. In: *Proceedings of the International conference on Intelligent robots and systems (IROS)*, Algarve, Portugal, 7–12 October 2012, pp. 589–595.

Coumanns E (2012) Bullet physics library. Available at: http://www.bulletphysics.org.

Dragan AD, Ratliff ND and Srinivasa SS (2011) Manipulation planning with goal sets using constrained trajectory optimization. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 4582–4588.

Dubins LE (1957) On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79(3): 497–516.

Duindam V, Alterovitz R, Sastry S, et al. (2008) Screw-based motion planning for bevel-tip flexible needles in 3D environments with obstacles. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Pasadena, CA, USA, 19–23 May 2008, pp. 2483–2488.

Duindam V, Xu J, Alterovitz R, et al. (2010) Three-dimensional motion planning algorithms for steerable needles using inverse kinematics. *International Journal of Robotics Research* 29(7): 789–800.

Engh JA, Minhas DS, Kondziolka D, et al. (2010) Percutaneous intracerebral navigation by duty-cycled spinning of flexible bevel-tipped needles. *Neurosurgery* 67(4): 1117–1122.

Erez T and Todorov E (2012) Trajectory optimization for domains with contacts using inverse dynamics. In: *Proceedings of the International conference on intelligent robots and systems (IROS)*, Algarve, Portugal, 7–12 October 2012, pp. 4914–4919.

Ericson C (2004) *Real-time Collision Detection*. Amsterdam; London: Morgan Kaufmann.

Frazzoli E, Dahleh MA and Feron E (2005) Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics* 21(6): 1077–1091.

Garg A, Patil S, Siauw T, et al. (2013) An algorithm for computing customized 3D printed implants with curvature constrained channels for enhancing intracavitary brachytherapy radiation delivery. In: *Proceedings of the IEEE International conference on automation science and engineering (CASE)* Madison, WI, USA, 17–21 August 2013, pp. 3306–3312.

Gilbert E, Johnson D and Keerthi S (1988) A Fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation* 4(2): 193–203.

Gurobi (2012) Gurobi optimizer reference manual. Available at: http://www.gurobi.com.

Han J, Datta S and Ekkad S (2013) *Gas Turbine Heat Transfer and Cooling Technology*. Boca Raton, FL: CRC Press.

Hauser K and Ng-Thow-Hing V (2010) Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Anchorage, AK, USA 3–8 May 2011, pp. 2493–2498.

Houska B, Ferreau HJ and Diehl M (2011) An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica* 47(10): 2279–2285.

Hwangbo M, Kuffner J and Kanade T (2007) Efficient two-phase 3D motion planning for small fixed-wing UAVs. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Roma, Italy, 10–14 April 2007, pp. 1035–1041.

Kalakrishnan M, Chitta S, Theodorou E, et al. (2011) STOMP: Stochastic trajectory optimization for motion planning. In: *Proceedings of the international conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 4569–4574.

Kallmann M, Aubel A, Abaci T, et al. (2003) Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer Graphics Forum* 22(3): 313–322.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7): 846–894.

Kavraki L, Svestka P, Latombe JC, et al. (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

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

Kuffner J and LaValle S (2000) RRT-connect: An efficient approach to single-query path planning. In: *Proceedings of the International conference on robotics and automation (ICRA)*, San Francisco, CA, USA, 24–28 April 2000, Vol. 2, pp. 995–1001.

Lamiraux F, Bonnafous D and Lefebvre O (2004) Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics* 20(6): 967–977.

Lampariello R, Nguyen-Tuong D, Castellini C, et al. (2011) Trajectory planning for optimal robot catching in real-time. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 3719–3726.

LaValle S (2006) *Planning Algorithms*. Cambridge; New York: Cambridge University Press.

Lengagne S, Vaillant J, Yoshida E, et al. (2013) Generation of whole-body optimal dynamic multi-contact motions. *International Journal of Robotics Research* 32(10): 1104–1119.

Lien JM and Amato NM (2007) Approximate convex decomposition of polyhedra. In: *Proceedings of the ACM symposium on solid and physical modeling*, Beijing, China, 4–6 June 2007, pp. 121–131.

Likhachev M, Gordon G and Thrun S (2003) ARA*: Anytime A* with provable bounds on sub-optimality. In: *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, BC, Canada, 9–11 December 2003, pp. 1–8.

Likhachev M and Stentz A (2008) R* search. In: *Proceedings of the national conference on artificial intelligence (AAAI)*, Chicago, IL, USA, 13–17 July 2008, pp. 344–350.

Majewicz A, Siegel J and Okamura A (2014) Design and evaluation of duty-cycling steering algorithms for robotically-driven steerable needles. In: *Proceedings of the International conference on robotics and automation (ICRA)* (in press).

Mamou K and Ghorbel F (2009) A simple and efficient approach for 3D mesh approximate convex decomposition. In: *IEEE International conference on image processing (ICIP)*, Cairo, Egypt, 7–9 November 2009, pp. 3501–3504.

Mészáros C (1999) The BPMPD interior point solver for convex quadratic problems. *Optimization Methods and Software* 11(1–4): 431–449.

Minhas DS, Engh JA, Fenske MM, et al. (2007) Modeling of needle steering via duty-cycled spinning. In: *Proceedings of the international conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, Lyon, France, 23–26 August 2007, pp. 2756–2759.

Mordatch I, Todorov E and Popovic Z (2012) Discovery of complex behaviors through contact-invariant optimization. *ACM SIGGRAPH* 31(4): 43.

Murray RM and Shankar SS (1994) *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press.

Nocedal J and Wright S (1999) *Numerical Optimization*. New York: Springer Verlag.

Pan J, Zhang L and Manocha D (2012) Collision-free and smooth trajectory computation in cluttered environments. *International Journal of Robotics Research* 31(10): 1155–1175.

Park C, Pan J and Manocha D (2012) ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In: *Proceedings of the International conference on automated planning and scheduling (ICAPS)*, Sao Paulo, Brazil, 25–29 June 2012, pp. 207–215.

Patil S and Alterovitz R (2010) Interactive motion planning for steerable needles in 3D environments with obstacles. In: *Proceedings of the International conference on biomedical robotics and biomechatronics (BioRob)*, Tokyo, Japan, 26–29 September 2010, pp. 893–899.

Patil S, Burgner J, Webster RJ III, et al. (2014) Needle steering in 3D via rapid replanning. *IEEE Transactions on Robotics* (in press).

Pivtoraiko M (2012) Differentially constrained motion planning with state lattice motion primitives. Technical Report CMU-RI-TR-12-07, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Posa M and Tedrake R (2013) Direct trajectory optimization of rigid body dynamical systems through contact. In: *Algorithmic Foundations of Robotics X*. Berlin; New York: Springer, pp. 527–542.

Quinlan S and Khatib O (1993) Elastic bands: Connecting path planning and control. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Atlanta, GA, USA, May 1993, pp. 802–807.

Ratliff N, Zucker M, Bagnell J, et al. (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Kobe, Japan, 12–17 May 2009, pp. 489–494.

Reed K, Majewicz A, Kallem V, et al. (2011) Robot-assisted needle steering. *IEEE Robotics and Automation Magazine* 18(4): 35–46.

Reeds J and Shepp L (1990) Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145(2): 367–393.

Saccon A, Hauser J and Aguiar AP (2013) Optimal control on Lie groups: The projection operator approach. *IEEE Transactions on Automatic Control* 58(9): 2230–2245.

Schulman J, Ho J, Lee A, et al. (2013) Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Proceedings of Robotics: Science and systems (RSS)*, Berlin, Germany, 24–28 June 2013.

Shanmugavel M, Tsourdos A, Zbikowski R, et al. (2007) 3D path planning for multiple UAVs using Pythagorean hodograph curves. In: *Proceedings of the AIAA guidance, navigation, and control conference*, Hilton Head, SC, USA, 20–23 August 2007, pp. 20–23.

Sucan IA and Kavraki LE (2009) Kinodynamic motion planning by interior–exterior cell exploration. In: *Algorithmic Foundation of Robotics VIII*. Berlin; Heidelberg, Germany: Springer, pp. 449–464.

Sucan IA, Moll M and Kavraki LE (2012) The open motion planning library. *IEEE Robotics and Automation Magazine* 19(4): 72–82.

Taschereau R, Pouliot J, Roy J, et al. (2000) Seed misplacement and stabilizing needles in transperineal permanent prostate implants. *Radiotherapy and Oncology* 55(1): 59–63.

Tassa Y, Erez T and Todorov E (2012) Synthesis and stabilization of complex behaviors through online trajectory optimization. In: *Proceedings of the International conference on intelligent robots and systems (IROS)*, Algarve, Portugal, 7–12 October 2012, pp. 4906–4913.

Van den Bergen G (2001) Proximity queries and penetration depth computation on 3D game objects. In: *Proceedings of the game developers conference (GDC)*, Hilton Head, SC, USA, 20–23 August 2007.

Vukobratović M and Borovac B (2004) Zero-moment point—Thirty five years of its life. *International Journal of Humanoid Robotics* 1(1): 157–173.

Warren CW (1989) Global path planning using artificial potential fields. In: *Proceedings of the International conference on robotics and automation (ICRA)*, Scottsdale, AZ, USA 14–19 May 1989, pp. 316–321.

Webster RJ III, Kim JS, Cowan NJ, et al. (2006) Nonholonomic modeling of needle steering. *International Journal of Robotics Research* 25(5–6): 509–525.

Werner A, Lampariello R and Ott C (2012) Optimization-based generation and experimental validation of optimal walking trajectories for biped robots. In: *Proceedings of the International conference on Intelligent robots and systems (IROS)*, Algarve, Portugal, 7–12 October 2012, pp. 4373–4379.

Xu J, Duindam V, Alterovitz R, et al. (2008) Motion planning for steerable needles in 3D environments with obstacles using rapidly-exploring random trees and backchaining. In: *IEEE International conference on automation science and engineering (CASE)*, Washington DC, USA, 23–26 August 2008, pp. 41–46.

Xu J, Duindam V, Alterovitz R, et al. (2009) Planning fireworks trajectories for steerable medical needles to reduce patient trauma. In: *Proceedings of the International conference on intelligent robots and systems (IROS)*, St Louis, MO, USA 11–15 October 2009, pp. 4517–4522.

Yang K and Sukkarieh S (2010) An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics* 26(3): 561–568.

Zucker M, Ratliff N, Dragan AD, et al. (2012) CHOMP: Covariant Hamiltonian optimization for motion planning. *International Journal of Robotics Research* 32(9–10): 1164–1193.

Appendix: Background on *SE*(3)

The special Euclidean group $SE(3)$ is a 6D configuration space consisting of the pose (3D position and 3D orientation). The Lie algebra $\mathfrak{se}(3)$ is defined as the tangent vector space at the identity of $SE(3)$. The $SE(3)$ group and $\mathfrak{se}(3)$ algebra are related via the exponential and log maps, $\exp : \mathfrak{se}(3) \rightarrow SE(3)$ and $\log : SE(3) \rightarrow \mathfrak{se}(3)$, where exp and log correspond to the matrix exponential and log operations. In this particular case, closed-form expressions exist for the exp and log operators (Appendix A in Murray and Shankar (1994)).

Given a vector $\bar{\mathbf{x}} = \begin{bmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{r}} \end{bmatrix} \in \mathbb{R}^6$ that represents the incremental twist, the corresponding Lie algebra element is given by the mapping $^\wedge : \mathbb{R}^6 \rightarrow \mathfrak{se}(3)$ as

$$\bar{\mathbf{x}}^\wedge = \begin{bmatrix} [\bar{\mathbf{r}}] & \bar{\mathbf{p}} \\ \mathbf{0}_3^T & 0 \end{bmatrix}$$

where the notation $[\bar{\mathbf{r}}]$ for the vector $\bar{\mathbf{r}} = [\bar{\mathbf{r}}_x \; \bar{\mathbf{r}}_y \; \bar{\mathbf{r}}_z]^{\mathrm{T}} \in \mathbb{R}^3$ is the 3×3 skew-symmetric matrix given by

$$[\bar{\mathbf{r}}] = \begin{bmatrix} 0 & -\bar{\mathbf{r}}_z & \bar{\mathbf{r}}_y \\ \bar{\mathbf{r}}_z & 0 & -\bar{\mathbf{r}}_x \\ -\bar{\mathbf{r}}_y & \bar{\mathbf{r}}_x & 0 \end{bmatrix}$$

Intuitively, $\bar{\mathbf{r}}$ represents the incremental rotation and $\bar{\mathbf{p}}$ represents the incremental translation to be applied to a nominal pose. The inverse is defined by the operator $^\vee : \mathfrak{se}(3) \rightarrow \mathbb{R}^6$ to recover $\bar{\mathbf{x}}$ given a Lie algebra element, i.e. $\begin{bmatrix} [\bar{\mathbf{r}}] & \bar{\mathbf{p}} \\ \mathbf{0}_3^T & 0 \end{bmatrix}^\vee = \bar{\mathbf{x}}$. The local neighborhood X of a nominal pose $\hat{X} \in SE(3)$ is defined in terms of $\bar{\mathbf{x}} \in \mathbb{R}^6$ as

$$X = \exp(\bar{\mathbf{x}}^\wedge) \cdot \hat{X}$$

Conversely, given a nominal pose $\hat{X}$ and a pose X, the corresponding twist $\bar{\mathbf{x}} \in \mathbb{R}^6$ can be recovered as:

$$\bar{\mathbf{x}} = \log(X \cdot \hat{X}^{-1})^\vee$$