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, Pieter Abbeel

***Abstract*—We present a new optimization-based approach for robotic motion planning among obstacles. Like CHOMP, 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 (i) A sequential convex optimization procedure, which penalizes collisions with a hinge loss and increases the penalty coefficients in an outer loop as necessary. (ii) 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 is freely available at http://rll.berkeley.edu/trajopt/ijrr.

I. 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 [21, 25] 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* [20] and discretization-based approaches [29, 28] 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 [1]. 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.

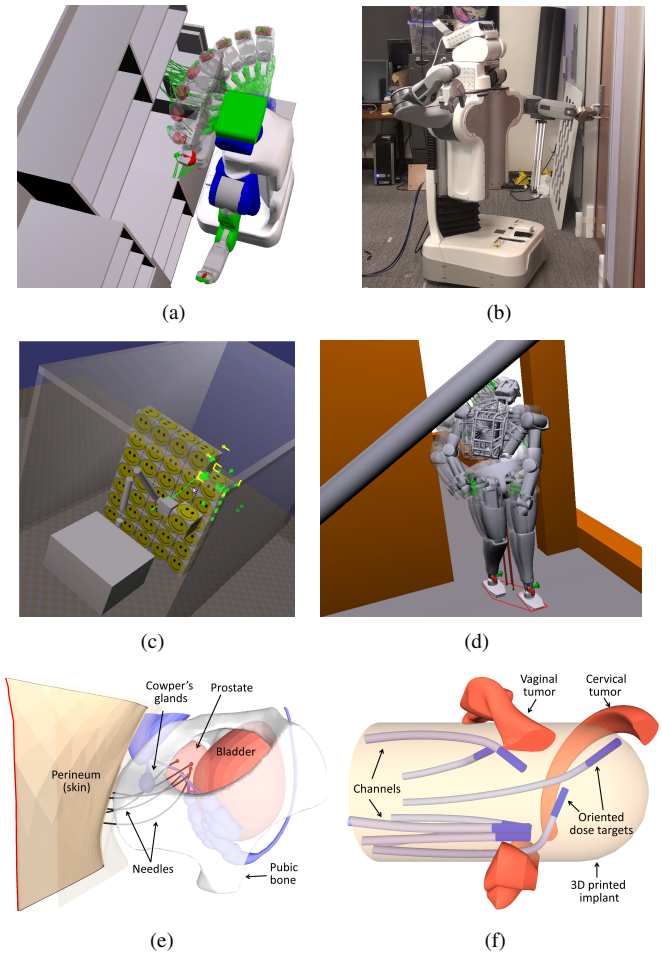


Fig. 1. TrajOpt applied to several motion planning scenarios: (a) planning an arm trajectory for the PR2 in simulation, (b) PR2 opening a door with a full-body motion, (c) industrial robot picking boxes, subject to an orientation constraint on the end effector, (d) humanoid robot model (DRC/Atlas) ducking 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 [14].

Even though trajectory optimization has been successfully used for optimal control in a number of domains, 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) [41, 61] 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: (1) formulation of trajectory costs that are invariant to the time parameterization of the trajectory, (2) using pre-computed signed distance fields for collision checking, and (3) using pre-conditioned gradient descent for numerical optimization.

Our approach uses optimization in the same spirit as CHOMP, with the following key differences: (1) the numerical optimization method used, and (2) 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, nonholonomic 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 [47], as well as a recent implementation of CHOMP [61]. Overall, our experimental results indicate that TrajOpt was computationally faster than the alternatives on the considered benchmark (around $100 - 200\,\mathrm{ms}$ 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 (Fig. 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 (Fig. 1(d)).

In this work, in addition to providing a revised and extended version of our work [43], (i) we describe an extension to the algorithm described in the RSS paper to plan trajectories in $SE(3)$, and (ii) we provide a discussion on cases where trajectory optimization fails to find a feasible solution. Regarding (i), 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 [2]. 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 [55, 42] (Fig. 1(e)). Our second application considers the problem of planning multiple, mutually collision-free, curvature-constrained channels within 3-D printed implants [14] for intracavitary brachytherapy (HDR-BT).

II. Related work

Trajectory optimization: Khatib proposed the use of potential fields for avoiding obstacles, including dynamic obstacles [22]. Warren [54] used a global potential field to push the robot away from configuration space obstacles, starting with a trajectory that was in collision. Quinlan and Khatib [40] 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. Brock and Khatib [3] 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. 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.

While the motivation for the presented work is very similar to the motivation behind CHOMP [41, 61, 8], which is most similar in terms of prior art, our algorithm differs fundamentally in the following two ways:

1) *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 [7] 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.

2) *Projected gradient descent versus SQP*: CHOMP uses (preconditioned) projected gradient descent, i.e., it takes steps $\mathbf{x} \leftarrow \mathrm{Proj}(\mathbf{x} - A^{-1}\nabla f(\mathbf{x}))$, 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 Sec. III. 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, nonholonomic systems.

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. [24] incorporate signed distances between polytopes as inequality constraints in an optimal control problem. Werner et al. [56] 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 [32, 26, 39, 51, 12]. 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.

Trajectory smoothing: Sampling-based motion planners can sometimes generate non-smooth trajectories that may contain unnecessary turns [25]. Many techniques have been proposed in the literature to generate smooth paths. Shortcut-based methods [17, 19, 36] 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. TrajOpt and other optimization approaches such as CHOMP [41, 61] can be used for trajectory smoothing in such cases.

3D curvature-constrained planning: This finds applications in a wide variety of domains, including motion planning for flexible, bevel-tip medical needles [55, 42], planning multiple curvature-constrained channels in 3D printed implants for brachytherapy dose delivery [14] or channels for cooling turbine blades [16], and path planning for unmanned aerial vehicles (UAVs) [60, 45]. Computing collision-free, curvature-constrained trajectories is challenging in 3D environments because it requires planning in the $SE(3)$ configuration space consisting of the 6D pose (position and orientation).

Duindam et al. proposed a fast, optimal planner based on inverse kinematics in 3D environments without obstacles [10]. Xu et al. [57, 58] used rapidly-exploring random trees (RRT) [25] which offers a probabilistically-complete, but computationally expensive, algorithm to search for collision-free trajectories. Duindam et al. [9] 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 et al. [37] proposed a fast RRT planner which uses duty-cycling spinning of the needle during insertion [11] to move the needle along bounded curvature trajectories, which can cause excessive tissue damage [11]. This approach was also used for designing bounded curvature channels within implants [14] but the issue of optimality was not addressed. Fast trajectory correction methods have been proposed to compensate for uncertainty during insertion [44, 38] but it is not clear if they can be used to plan trajectories from scratch.

Extensions to planning curvature-constrained trajectories in 3D have been proposed for unmanned aerial vehicles (UAVs) in environments without obstacles [48, 59, 45] and with obstacles [18, 60]. These methods are specialized for bounded curvature trajectory planning and have not considered planning of constant curvature trajectories in 3D environments.

While specialized planners have been proposed for underactuated, nonholonomic systems, TrajOpt can be generalized to this case (in our case, the $SE(3)$ configuration space) by considering optimization over increments.

III. 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:

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

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 t^{th} timestep. To encourage minimum-length paths, we use the sum of squared displacements,

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

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, nonholonomic motion planning problems, additional equality constraints are added to ensure that the kinematics are consistent. We will discuss some of these constraints in Sec. VII.

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: (1) a method for constraining the step to be small, so the solution vector remains within the region where the approximations are valid; (2) a strategy for turning the infeasible constraints into penalties, which eventually drives all of the constraint violations to zero. For (1), we use a trust region modeled as a box constraint around the current iterate. For (2) 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 [34], which is described in Algorithm 1.

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.

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.

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: $\tilde{f}, \tilde{g}, \tilde{h} = \text{ConvexifyProblem}(f, g, h)$
4: **for** TrustRegionIteration $= 1, 2, \ldots$ **do**
5: $\mathbf{x} \leftarrow \arg\min_{\mathbf{x}} \tilde{f}(\mathbf{x}) + \mu \sum_{i=1}^{n_{ineq}} |\tilde{g}_i(\mathbf{x})|^+ + \mu \sum_{i=1}^{n_{eq}} |\tilde{h}_i(\mathbf{x})|$
 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: $\mu \leftarrow k * \mu$

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 [34]. 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

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

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

A. Sequential Convex Optimization over $SE(3)$

The optimization method outlined above operates in vector spaces of the form $\mathbb{R}^n$. However, motion planning for underactuated, nonholonomic systems such as bevel-tipped flexible needles, or steerable needles, involves planning over manifolds. In this work, we consider the trajectory optimization problem defined over the special Euclidean group $SE(3)$, which is a 6D configuration space consisting of the robot pose (3D position and 3D orientation) i.e., at each time step $t \in \mathcal{T}$, the configuration $\mathbf{x}_t$ is parameterized as 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.

The Lie group $SE(3)$ is a smooth manifold. To perform local optimization over $SE(3)$, we will need to form a local coordinate parametrization of the manifold. This parametrization is provided by the Lie algebra $\mathfrak{se}(3)$, which is defined as the tangent vector space at the identity of $SE(3)$, and, informally, consists of infinitesimal rotations. The $SE(3)$ group and $\mathfrak{se}(3)$ algebra are related via the exponential and log maps, $\exp : \mathfrak{se}(3) \to SE(3)$ and $\log : SE(3) \to \mathfrak{se}(3)$, where exp and log correspond to the matrix exponential and log operations.

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 $^\wedge : \mathbb{R}^6 \to \mathfrak{se}(3)$ as

$$\bar{\mathbf{x}}^\wedge = \begin{bmatrix} [\bar{\mathbf{r}}] & \bar{\mathbf{p}} \\ \mathbf{0}_3^T & 0 \end{bmatrix}, \tag{5}$$

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

$$[\bar{\mathbf{r}}] = \begin{bmatrix} 0 & -\bar{\mathbf{r}}_z & \bar{\mathbf{r}}_y \\ \bar{\mathbf{r}}_z & 0 & -\bar{\mathbf{r}}_x \\ -\bar{\mathbf{r}}_y & \bar{\mathbf{r}}_x & 0 \end{bmatrix}. \tag{6}$$

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 $^\vee : \mathfrak{se}(3) \to \mathbb{R}^6$ 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

$$X = \hat{X} \cdot \exp(\bar{\mathbf{x}}^\wedge), \tag{7}$$

where $\exp(\bar{\mathbf{x}}^\wedge)$ can be explicitly computed as [33]:

$$\exp(\bar{\mathbf{x}}^\wedge) = \begin{bmatrix} I & \bar{\mathbf{p}} \\ \mathbf{0}_3^T & 1 \end{bmatrix}, \bar{\mathbf{r}} = \mathbf{0}_3 \quad \text{or} \quad \begin{bmatrix} e^{\bar{\mathbf{r}}} & A\bar{\mathbf{p}} \\ \mathbf{0}_3^T & 1 \end{bmatrix}, \bar{\mathbf{r}} \neq \mathbf{0}_3 \tag{8}$$

where

$$e^{\bar{\mathbf{r}}} = I + \frac{[\bar{\mathbf{r}}]}{\|\bar{\mathbf{r}}\|} \sin \|\bar{\mathbf{r}}\| + \frac{[\bar{\mathbf{r}}]^2}{\|\bar{\mathbf{r}}\|^2}(1 - \cos \|\bar{\mathbf{r}}\|), \tag{9}$$

$$A = I + \frac{[\bar{\mathbf{r}}]}{\|\bar{\mathbf{r}}\|^2}(1 - \cos \|\bar{\mathbf{r}}\|) + \frac{[\bar{\mathbf{r}}]^2}{\|\bar{\mathbf{r}}\|^3}(\|\bar{\mathbf{r}}\| - \sin \|\bar{\mathbf{r}}\|) \tag{10}$$

Note that an alternative approach would be to 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—for example, 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.

We now describe how to generalize sequential convex optimization to the case where the domain is a differentiable manifold rather than $\mathbb{R}^n$. There is an extra step in constructing each convex subproblem: we first form a local coordinate parametrization of the manifold around the current solution (a point on the manifold). Then we approximate the merit function $f_\mu(\boldsymbol{\theta})$ in terms of this parameterization.

In this work, at the i^{th} iteration of SQP our trajectory consists of a sequence of nominal poses $\hat{\mathcal{X}}^{(i)} = \{\hat{X}_0^{(i)}, \ldots, \hat{X}_T^{(i)}\}$. To construct the QP subproblem, we parametrize each pose in terms of increments to the previous solution: $\hat{\mathcal{X}}^{(i+1)} = \{\hat{X}_0^{(i)} \cdot \exp(\bar{\mathbf{x}}_0^{(i)\wedge}), \ldots, \hat{X}_T^{(i)} \cdot \exp(\bar{\mathbf{x}}_T^{(i)\wedge})\}$ where $\bar{\mathcal{X}}^{(i)} = \{\bar{\mathbf{x}}_0^{(i)}, \ldots, \bar{\mathbf{x}}_T^{(i)}\}$ is the sequence of incremental twists.

IV. 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.

A. 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 [13]. The distance between two sets $\mathcal{A}, \mathcal{B} \subset \mathbb{R}^3$, which is nonzero for non-intersecting sets, is defined as

$$\text{dist}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \ \big| \ (T + \mathcal{A}) \cap \mathcal{B} \neq \varnothing\} \tag{11}$$

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:

$$\text{penetration}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \ \big| \ (T + \mathcal{A}) \cap \mathcal{B} = \varnothing\} \tag{12}$$

The signed distance is defined as follows:

$$\text{sd}(\mathcal{A}, \mathcal{B}) = \text{dist}(\mathcal{A}, \mathcal{B}) - \text{penetration}(\mathcal{A}, \mathcal{B}) \tag{13}$$

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. [13].

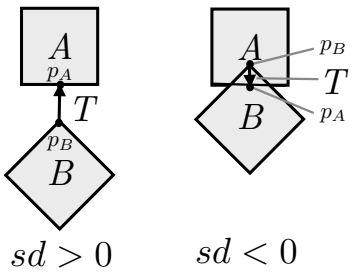


Fig. 2. Minimal translational distance and closest points.

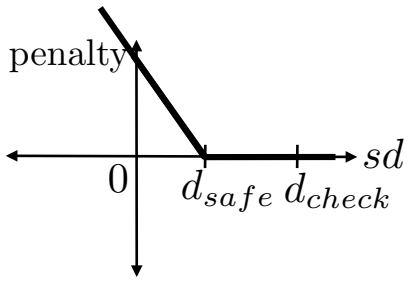


Fig. 3. Hinge penalty for collisions

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 [15], while the penetration depth is calculated by a different algorithm, the Expanding Polytope Algorithm (EPA) [52]. 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 $\mathbf{v}$:

$$s_A(\mathbf{v}) = \arg\max_{\mathbf{p}\in\mathcal{A}} \mathbf{v}\cdot\mathbf{p} \tag{14}$$

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

$$\begin{aligned} &\text{sd}(\mathcal{A}_i, \mathcal{O}_j) \geq d_{\text{safe}} \qquad \forall i \in \{1,2,\ldots,N_{\text{links}}\}, \\ &\qquad\qquad\qquad\qquad\qquad \forall j \in \{1,2,\ldots,N_{\text{obstacles}}\} \\ &\text{(obstacle collisions)} \\ &\text{sd}(\mathcal{A}_i, \mathcal{A}_j) \geq d_{\text{safe}} \qquad \forall i,j \in \{1,2,\ldots,N_{\text{links}}\} \\ &\text{(self collisions)} \end{aligned} \tag{15}$$

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

$$\begin{aligned} &\sum_{i=1}^{N_{\text{links}}}\sum_{j=1}^{N_{\text{obs}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+ \\ &+ \sum_{i=1}^{N_{\text{links}}}\sum_{j=1}^{N_{\text{links}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{B}_j)|^+ \end{aligned} \tag{16}$$

A single term of this penalty function $|d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+$ is illustrated in Fig. 3.

Note that in practice, we do not consider all pairs of objects for the collision penalty (Eq. (16)) 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 $\mathbf{p}^B \in \mathcal{B}^B$ 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:

$$\text{sd}(\{A, F_A^w\}, \{B, F_B^w\}) = \max_{\|\hat{\mathbf{n}}\|=1} \min_{\substack{\mathbf{p}_A\in A,\\ \mathbf{p}_B\in B}} \hat{\mathbf{n}}\cdot(F_A^w\mathbf{p}^A - F_B^w\mathbf{p}^B) \tag{17}$$

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 Eq. (17). Equivalently, the contact normal $\hat{\mathbf{n}}$ is the direction of the minimal translation T (as defined in Eqs. (11) and (12)), 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 (Fig. 2).

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 Eq. (17).

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

$$\text{sd}_{AB}(\mathbf{x}) \approx \hat{\mathbf{n}}\cdot(F_A^w(\mathbf{x})\mathbf{p}_A - F_B^w\mathbf{p}_B) \tag{18}$$

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

$$\begin{aligned} \nabla_{\mathbf{x}}\,\text{sd}_{AB}(\mathbf{x})\Big|_{\mathbf{x}_0} &\approx \hat{\mathbf{n}}^T J_{\mathbf{p}_A}(\mathbf{x}_0) \\ \text{sd}_{AB}(\mathbf{x}) &\approx \text{sd}_{AB}(\mathbf{x}_0) + \hat{\mathbf{n}}^T J_{\mathbf{p}_A}(\mathbf{x}_0)(\mathbf{x}-\mathbf{x}_0) \end{aligned} \tag{19}$$

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 Eq. (3).

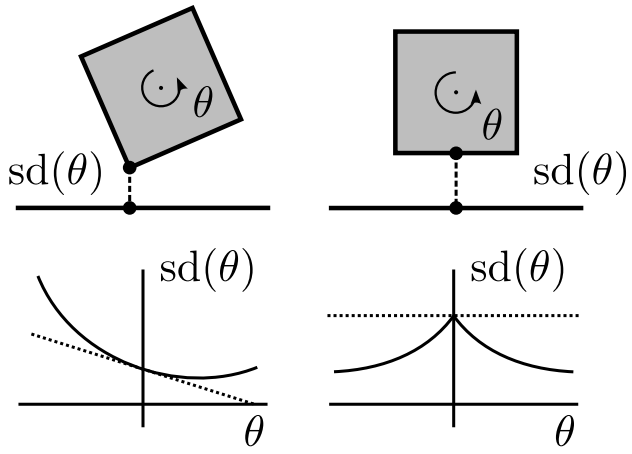


Fig. 4. Illustration of the non-differentiability of the signed distance function. Here, a square is rotated about its center by angle θ. The true function is shown by a solid line, and the linearization is shown by a dotted line. It is correct to first-order in non-degenerate situations, however, in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

Note that Eq. (19), 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. Fig. 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.

B. 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 Fig. 5.

We can modify the collision penalty from Section IV-A 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.

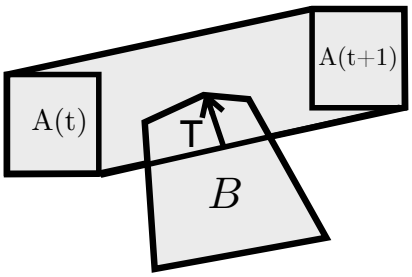


Fig. 5. Illustration of swept volume for use in our continuous collision cost.

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 rotations below.) Then the swept-out volume is the convex hull of the initial and final volumes [52]

$$\bigcup_{t\in[0,1]} \mathcal{A}(t) = \text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)) \tag{20}$$

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

$$\text{sd}(\text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{B}) \tag{21}$$

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 Sec. IV-A) the signed distance cost can be calculated using the support mappings. In particular, the support mapping is given by

$$s_{\text{convhull}(C,D)}(\mathbf{v}) = \begin{cases} s_C(\mathbf{v}) & \text{if } s_C(\mathbf{v})\cdot\mathbf{v} > s_D(\mathbf{v})\cdot\mathbf{v} \\ s_D(\mathbf{v}) & \text{otherwise} \end{cases} \tag{22}$$

Calculating the gradient of the swept-volume collision cost is slightly more involved than discrete case described in Eqs. (18) and (19). Let's consider the case where object A is moving and object B is stationary, as in Fig. 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: (1) all the vertices are from $A(t)$, (2) all of the vertices are from $A(t+1)$, and (3) otherwise. Cases (1) and (2) 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 (3), 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 (3) 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 $-\hat{\mathbf{n}}$.

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

$$\alpha = \frac{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\|}{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\| + \|\mathbf{p}_0 - \mathbf{p}_{\text{swept}}\|}, \tag{23}$$

where we make the approximation

$$\mathbf{p}_{\text{swept}}(\mathbf{x}) \approx \alpha\mathbf{p}_0 + (1-\alpha)\mathbf{p}_1 \tag{24}$$

3) Calculate the Jacobians of those points

$$J_{\mathbf{p}_0}(\mathbf{x}_0^t) = \frac{d}{d\mathbf{x}^t}\mathbf{p}_0, \quad J_{\mathbf{p}_1}(\mathbf{x}_0^{t+1}) = \frac{d}{d\mathbf{x}^{t+1}}\mathbf{p}_1 \tag{25}$$

4) Similarly to Eq. (19), linearize the signed distance around the trajectory variables at timesteps t and $t+1$

$$\begin{aligned}\mathrm{sd}_{AB}(\mathbf{x}^t, \mathbf{x}^{t+1}) \approx \mathrm{sd}_{AB}(\mathbf{x}_0^t, \mathbf{x}_0^{t+1}) + \alpha\hat{\mathbf{n}}^T J_{\mathbf{p}_0}(\mathbf{x}_0^t)(\mathbf{x}^t - \mathbf{x}_0^t) \\ + (1-\alpha)\hat{\mathbf{n}}^T J_{\mathbf{p}_1}(\mathbf{x}_0^{t+1})(\mathbf{x}^{t+1} - \mathbf{x}_0^{t+1})\end{aligned} \tag{26}$$

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 Fig. 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 $\mathrm{convhull}(A(t), A(t+1))$ by $d_{arc} = r\phi^2/8$, where r is the maximum distance from a point on A to the local rotation axis, then the swept-out volume is contained inside.

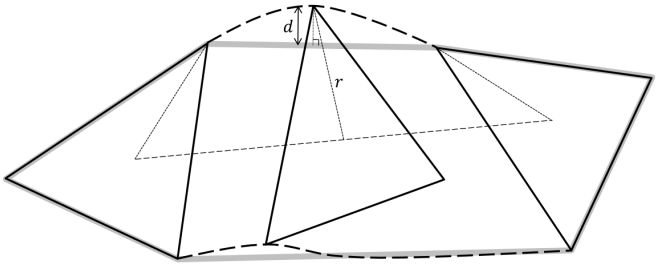


Fig. 6. Illustration of the difference between swept out shape and convex hull. The figure shows a triangle undergoing translation and uniform rotation. The swept-out area is enclosed by dotted lines, and the convex hull is shown by a thick gray line.

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

$$\mathrm{sd}(\mathrm{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{O}) > d_{\text{safe}} + d_{\text{arc}} \tag{27}$$

One could relax this constraint into a penalty as described in Sec. IV-A, by approximating $\phi(\mathbf{x}^t, \mathbf{x}^{t+1})$. In practice, we ignored the correction d_{arc}, since it was well under $1\,\mathrm{cm}$ 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.

V. 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 (Fig. 7). We ran all the experiments on a machine with an Intel i7 3.5 GHz CPU. 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.

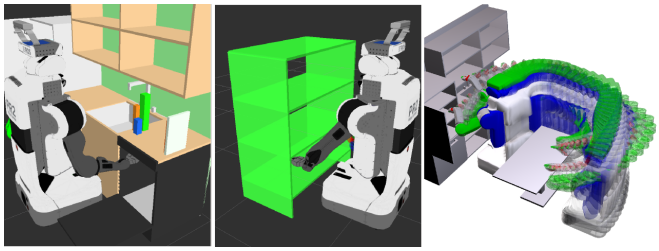


Fig. 7. Scenes in our benchmark tests. Left and center: two of the scenes used for the arm planning benchmark. Right: a third scene, showing the path found by our planner on an 18-DOF full-body planning problem.

We compared TrajOpt to open-source implementations of bi-directional RRT [23] and a variant of KPIECE [46] from OMPL/MoveIt! [4, 6], that is part of the ROS motion planning libraries. All algorithms were run using default parameters and post-processed by the default smoother used by MoveIt!. We also compared TrajOpt to a recent implementation of CHOMP [61] 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.

	OMPL-RRTConnect	OMPL-LBKPIECE	CHOMP-HMC	CHOMP-HMC-Multi	TrajOpt	TrajOpt-Multi
success fraction	0.85	0.76	0.65	0.83	0.82	0.96
avg. time (s)	0.62	1.30	4.91	9.27	0.19	0.30
avg. norm length	1.56	1.61	2.04	1.97	1.16	1.15

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

	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

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

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 Sec. IV-A. 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 (i) maximum of 40 iterations, (ii) minimum merit function improvement ratio of 10^{-4}, (iii) minimum trust region size 10^{-4}, and (iv) constant penalty scaling factor $k = 10$. We used the Bullet collision checker [7] 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 seconds on full-body planning problems.

Results: The results for arm planning are shown in Table I and for full-body planning are shown in Table II. We evaluated TrajOpt and compared it with other planners in terms of (1) average computation time for all successful planning runs computed over all problems, and (2) 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. Sec. IX presents a discussion of why multiple trajectory initializations are important.

VI. Physical Experiments

A. 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: (1) convex decomposition, and (2) meshes.

Convex decomposition: Convex decomposition seeks to represent a general 3D volume approximately as a union of convex bodies [27]. Hierarchical Approximate Convex Decomposition (HACD) [30] 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 [5].

B. Experiments

We performed several physical experiments involving a mobile robot (PR2) to explore two aspects of TrajOpt: (1) applying it to the "dirty" geometry data that we get from depth sensors such as the Kinect, and (2) 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).

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.

VII. Example Applications with Different Constraints

A. Humanoid walking: Static stability

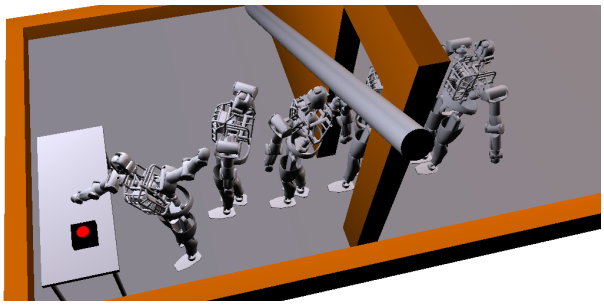


Fig. 8. The Atlas humanoid robot in simulation walking across the room while avoiding the door frame and other obstacles in the environment, and pushing a button. Each footstep was planned for separately using TrajOpt while maintaining static stability. Five time steps of the trajectory are shown.

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 (1) left foot planted, (2) both feet planted (3) right foot planted, and (4) 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 [53]. The convex support polygon is now represented as an intersection of k half-planes, yielding k inequality constraints:

$$a_i x_{\text{cm}}(\boldsymbol{\theta}) + b_i y_{\text{cm}}(\boldsymbol{\theta}) + c_i \leq 0, \;\; i \in \{1, 2, \ldots, k\}, \qquad (28)$$

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 Fig. 8. Each step is planned separately using the phases described above. 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.

B. Pose constraints

TrajOpt can readily incorporate kinematic constraints, for example, 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 six-dimensional error vector:

$$h(\mathbf{x}) = \log(F_{\text{targ}}^{-1} F_{\text{cur}}(\mathbf{x})) = (t_x, t_y, t_z, r_x, r_y, r_z) \qquad (29)$$

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 map, where $\log(X)$ can be explicitly computed for a matrix $X \in SE(3)$ as [33]:

$$\log(X) = \begin{bmatrix} I & \mathbf{p} \\ \mathbf{0}_3 & 0 \end{bmatrix}, R = I \quad \text{or} \quad \begin{bmatrix} \theta\hat{\omega} & A^{-1}\mathbf{p} \\ \mathbf{0}_3^T & 0 \end{bmatrix}, R \neq I \quad (30)$$

where

$$\theta = \arccos \frac{\text{trace}(R) - 1}{2}, \;\; \hat{\omega} = \frac{1}{2\sin\theta}(R - R^T) \qquad (31)$$

$$A^{-1} = I - \frac{1}{2}\theta\hat{\omega} + \frac{2\sin\theta - \theta(1 + \cos\theta)}{2\sin\theta}\hat{\omega}^2 \qquad (32)$$

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.

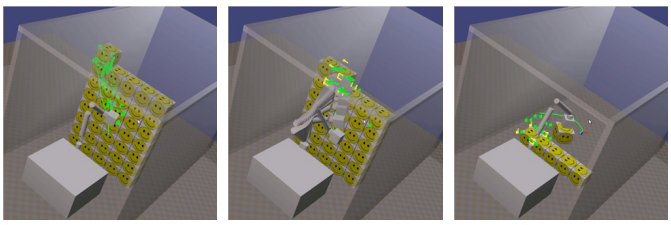


Fig. 9. Several stages of a box picking procedure, in which boxes are taken from the stack and moved to the side. The box, and hence the end effector of the robot arm, is subject to pose constraints.

Fig. 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.

VIII. Needle Steering and Channel Layout Planning

We formulate the curvature-constrained planning problem in 3D environments as a nonlinear, constrained optimization problem. Depending on the specific application, the trajectory may be required to have a constant curvature $\kappa_{\max}$ for all time steps, as required for planning trajectories for bevel-tip flexible needles, or a bounded curvature $0 \leq \kappa_t \leq \kappa_{\max}$ for each time step for given $\kappa_{\max}$, as required for planning curvature-constrained channels within 3D-printed implants. Although the following formulation is specific to needle steering and channel planning, it can be easily generalized to other curvature-constrained planning problems.

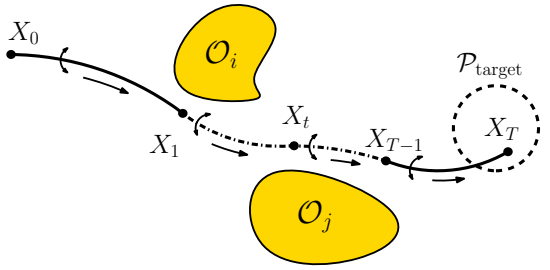


Fig. 10. A discretized curvature-constrained trajectory is parameterized as $\{X_0, \ldots, X_t, \ldots, X_T\}$, where $X_t \in SE(3)$ is the pose of the waypoint frame relative to a world coordinate frame at each time step t.

The state at each time step is denoted as $X_t = \begin{bmatrix} R_t & \mathbf{p}_t \\ \mathbf{0}_3^T & 1 \end{bmatrix} \in SE(3)$, where 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 frame relative to a world coordinate frame.

The trajectory optimization problem can then be stated as:

Objective: Given an entry zone $\mathcal{P}_{\text{entry}}$ and a target zone $\mathcal{P}_{\text{target}}$, determine a locally optimal and curvature-constrained trajectory $X_t : t \in \mathcal{T}$ which starts from the entry zone and reaches the target zone while avoiding obstacles.

Input: Obstacle definitions $\mathcal{O}_i \in \mathcal{O}$, an entry zone $\mathcal{P}_{\text{entry}}$, a target zone $\mathcal{P}_{\text{target}}$, the constant curvature or the maximum curvature $\kappa_{\max}$, and the discretization parameter T.

Output: Returns a feasible, collision-free trajectory $X_t : t \in \{1, \dots, T\}$ with $X_0 \in \mathcal{P}_{\text{entry}}$ and $X_T \in \mathcal{P}_{\text{target}}$, or reports that no feasible trajectory can be found.

Metrics: For applications like medical needle steering, it is important to address the issue of optimality of the computed trajectories since sub-optimal trajectories might cause excessive tissue damage during the procedure, leading to delayed recovery times. We consider the following metrics in our work to quantify plan optimality:

1) Minimizing the total needle insertion length: This metric is relevant to procedures in vital organs such as the brain where limiting tissue damage is important.

2) Minimizing the total twist: Unnecessary twisting of the needle causes tissue damage and also induces torsion in the needle shaft, which leads to errors while planning and controlling the motion of the needle [49].

3) Maximizing the minimum clearance from obstacles: Short trajectories often pass in close proximity to obstacles, thereby increasing the likelihood of collisions. Trajectories that have a greater minimum clearance from obstacles, on the other hand, are safer because they are less likely to collide with anatomical obstacles when deviations occur but tend to be longer. This metric could be useful when obstacle avoidance is critical but other tissue damage is manageable.

The optimization problem is formulated as follows:

$$\min_{\bar{\mathcal{X}}, \mathcal{U}} \ \alpha_\Delta \text{Cost}_\Delta + \alpha_\phi \text{Cost}_\phi + \alpha_\mathcal{O} \text{Cost}_\mathcal{O}, \tag{33a}$$

$$\text{s.t.} \quad \log((X_t \cdot \exp(\mathbf{w}_t^\wedge) \cdot \exp(\mathbf{v}_t^\wedge))^{-1} \cdot X_{t+1})^\vee = \mathbf{0}_6, \tag{33b}$$

$$\text{sd}(X_t, X_{t+1}, \mathcal{O}_i) \geq d_{\text{safe}} + d_{\text{arc}}, \tag{33c}$$

$$X_0 \in \mathcal{P}_{\text{entry}}, \ X_T \in \mathcal{P}_{\text{target}}, \tag{33d}$$

$$-\pi \leq \phi_t \leq \pi, \tag{33e}$$

$$\kappa_t = \kappa_{\max} \quad \text{or} \quad 0 \leq \kappa_t \leq \kappa_{\max}, \tag{33f}$$

$$\Delta \sum_{t=0}^{T-1} \kappa_t \leq c_{\max} \quad \text{for channel planning}, \tag{33g}$$

where $\mathcal{U}$ is the set of all control variables and $\bar{\mathcal{X}} = \{\bar{\mathbf{x}}_0, \dots, \bar{\mathbf{x}}_T\}$ is the sequence of incremental twists.

The constraints and costs are described in detail below.

1) *Kinematics constraint*: We use a "stop-and-turn" strategy [1] for the kinematics model. 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}. We require this distance to be the same for all time steps. See Fig. 11 for illustration. For a feasible trajectory, the poses at adjacent time steps X_t and X_{t+1} are related as:

$$X_t \cdot \exp(\mathbf{w}_t^\wedge) \cdot \exp(\mathbf{v}_t^\wedge) = X_{t+1}, \tag{34}$$

where $\mathbf{w}_t = \begin{bmatrix} 0 & 0 & 0 & 0 & \phi_t \end{bmatrix}^T$ and $\mathbf{v}_t = \begin{bmatrix} 0 & 0 & \Delta & \Delta\kappa_t & 0 & 0 \end{bmatrix}^T$. We transform this constraint in $SE(3)$ to a constraint in $\mathfrak{se}(3)$ using the log map to get Eq. (33b).

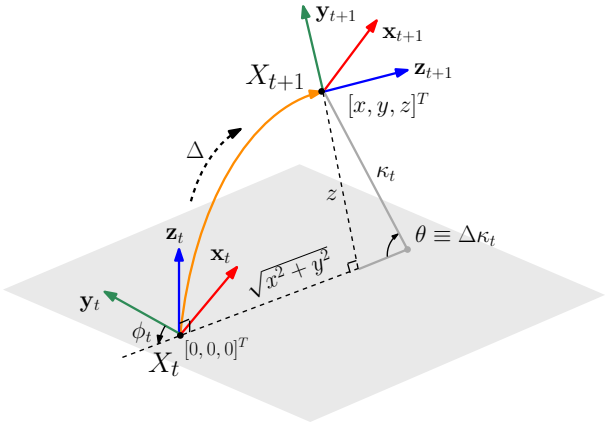


Fig. 11. Stop and turn strategy: Apply a rotation ϕ_t to the pose X_t at time step t and then propagate the frame by a distance Δ to arrive at X_{t+1}.

2) *No-collision constraint*: We impose constraints (Eq. (33c)) for the trajectory to be collision-free, where $\text{sd}(X_t, X_{t+1}, \mathcal{O}_i)$ is the signed distance between the trajectory segment in time interval $[t, t+1]$ and the i^{th} obstacle $\mathcal{O}_i$. 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 $\hat{\mathbf{n}}$. The signed distance linearization is given by

$$\text{sd}(X_t, \mathcal{O}_i) \approx \text{sd}(\hat{X}_t, \mathcal{O}_i) + \hat{\mathbf{n}}^T J_{\mathbf{p}_t}(\mathbf{0}_6) \bar{\mathbf{x}}, \tag{35}$$

where the Jacobian matrix $J_{\mathbf{p}_t}(\mathbf{0}_6) = \begin{bmatrix} \hat{R}_t & \mathbf{0}_{3\times 3} \end{bmatrix}$. The expression for the Jacobian follows from the fact that we get $\mathbf{p}_t = \hat{R}_t A \bar{\mathbf{p}}_t + \hat{\mathbf{p}}_t$ (Eqs. (7)–(10)), where A is defined in the exponential map with respect to $\mathbf{r}$ and is the identity matrix for $\bar{\mathbf{r}} = \mathbf{0}_3$. Hence

$$\frac{\partial \mathbf{p}_t}{\partial \bar{\mathbf{p}}_t}(\mathbf{0}_6) = \hat{R}_t, \quad \text{and} \quad \frac{\partial \mathbf{p}_t}{\partial \bar{\mathbf{r}}_t}(\mathbf{0}_6) = \mathbf{0}_{3\times 3}. \tag{36}$$

We include the continuous-time non-convex no-collisions constraint is included as a ℓ_1 penalty in the optimization, as described in Sec. IV-B.

[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.

3) *Total curvature constraint*: For channel planning, we impose a constraint (Eq. (33g)) on the total curvature of the trajectory. This is done to ensure that catheters carrying the radioactive source can be pushed through the channels [14]. We choose $c_{\max} = 1.57$ rad for all of our experiments.

4) *Costs*: To penalize tissue damage for needle steering and to make the trajectory more efficient for channel planning, we add costs on the total length of the trajectory and the twists to the objective at each time step:

$$\text{Cost}_\Delta = T\Delta \quad \text{and} \quad \text{Cost}_\phi = \sum_{t=0}^{T-1} \phi_t^2. \tag{37}$$

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

$$\text{Cost}_\mathcal{O} = - \min_{\substack{0 \le t \le T-1 \\ \mathcal{O}_i \in \mathcal{O}}} \text{sd}(X_t, X_{t+1}, \mathcal{O}_i). \tag{38}$$

Instead of directly optimizing over $\text{Cost}_\mathcal{O}$, we insert an auxiliary variable $d_{\min}$ and reformulate the cost term as:

$$\text{Cost}_\mathcal{O} = -d_{\min}, \qquad d_{\min} \le \text{sd}(X_t, X_{t+1}, \mathcal{O}_i). \tag{39}$$

The objective (Eq. (33a)) is a non-negative weighted sum of the costs above, where $\alpha_\Delta, \alpha_\phi, \alpha_\mathcal{O} \ge 0$ are user-defined 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.

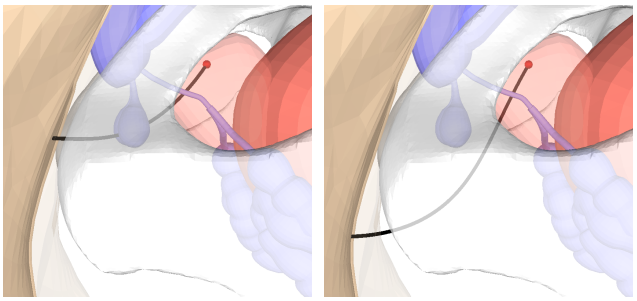
Shooting vs. Collocation: For trajectory optimization problems, especially ones involving differential constraints, there are two ways to construct locally convex approximations of the costs and constraints for setting up the QP subproblem. Shooting-based methods first integrate the current controls and then construct the convex approximation around the integrated trajectory, which is guaranteed to satisfy all differential constraints. Collocation-based methods, on the other hand, approximate the cost and constraints directly around the current solution, which might correspond to an infeasible trajectory that does not satisfy differential constraints.

In the context of optimization on $SE(3)$, shooting-based methods satisfy the curvature constraints, but the integrated trajectory might violate the constraints on the entry zone and target zone. It is easier to satisfy constraints on the start and target zones with collocation-based methods but the differential curvature constraint is difficult to satisfy. We refer the reader to [1] for details on these methods.

A. Simulation Experiments

We experimentally evaluated TrajOpt on two real-world applications involving medical needle steering and designing channel layouts for brachytherapy. We ran all the experiments on a machine with an Intel i7 3.5 GHz CPU.

1) *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 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 [50]. 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 1cm. For the objective function, we used $\alpha_\Delta = \alpha_\phi = 1$, and we compared the planned trajectory with different choices of the clearance coefficient $\alpha_\mathcal{O}$. Figs. 12(a) and 12(b) show examples of planned trajectories with different values of $\alpha_\mathcal{O} = 1$ and $\alpha_\mathcal{O} = 10$.



(a) (b)

Fig. 12. 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 $\alpha_\mathcal{O} = 10$.

We compared the performance of shooting and collocation-based methods for optimization. We also compared the performance of TrajOpt with a sampling-based rapidly-exploring random tree (RRT) planner [57]. The RRT planner was modified to plan backwards starting from target zones because it is easier to compute feasible constant curvature trajectories that reach a larger entry region.

Planning for a single needle: We analyzed the planned trajectory for single needle insertion using 400 sampled points in the prostate. For each task, we repeatedly ran the optimization initialized by a perturbed solution of the previous run, and we allowed up to 5 reruns. We evaluated the performance of collocation versus shooting in terms of the average running time and percentage of solved problems for the converged solutions. As shown in Table III, we observed that shooting outperforms collocation in terms of the fraction of problems solved and running times. Using a larger clearance coefficient results in trajectories farther away from obstacles, at the expense of slightly longer paths.

TrajOpt 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 a result of the randomized nature of the planning algorithm. Since the twist cost is directly correlated with tissue damage, the trajectories computed using TrajOpt are preferable over those computed by a randomized planner.

	RRT	TrajOpt			
		Collocation $\alpha_{\mathcal{O}} = 1$	Shooting $\alpha_{\mathcal{O}} = 1$	Collocation $\alpha_{\mathcal{O}} = 10$	Shooting $\alpha_{\mathcal{O}} = 10$
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	11.1 ± 1.5	11.3 ± 1.4	11.6 ± 1.7	11.9 ± 1.7	13.1 ± 2.3
twist cost	34.9 ± 10.0	1.4 ± 1.4	1.0 ± 1.0	1.6 ± 1.6	1.0 ± 1.0
clearance	0.5 ± 0.4	0.7 ± 0.5	0.5 ± 0.3	1.3 ± 0.4	1.2 ± 0.5

TABLE III
Single needle planning: Sampling-based RRT planner versus TrajOpt.

Simulation: To evaluate the feasibility and performance of TrajOpt under uncertainty, we ran 100 simulations for a specific target with increasing noise levels. System uncertainty was modeled by perturbing the incremental twists with additive Gaussian noise. We assumed that the pose of needle is measured accurately by a Kalman filter or sensing equipment, and we re-planned after every time step based on the estimated state. We considered a simulation to be successful if it was both collision free and if it reached the target zone. To ensure path safety, we chose $\alpha_{\mathcal{O}} = 10$ for all tasks. We examined the effect of increasing noise level on the success rate (Fig. 13).

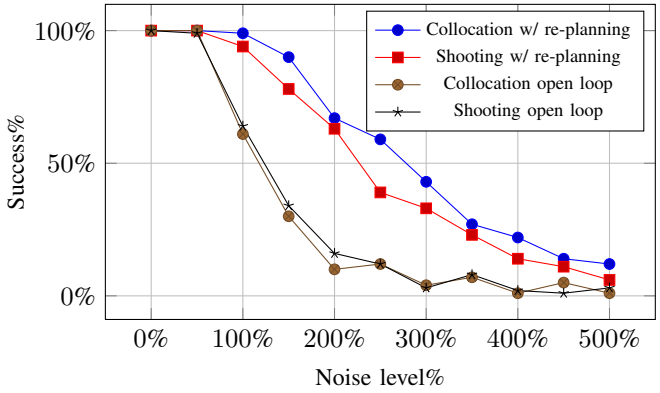


Fig. 13. Effect of noise level on the success rate.

Re-planning after each time step greatly increases the probability of success. Collocation consistently outperforms shooting in terms of success rate for all noise levels. Since shooting-based methods integrate the trajectory after each iteration of the optimization, the state at the last time step deviates from the target region, making it difficult to find a feasible solution. We also observed a significant difference in the total running time for the simulation, where collocation takes 2.3 seconds on average, and shooting takes 6.7 seconds on average. This suggests that it might be easier to perform local error corrections to an existing trajectory using collocation as compared to shooting based methods. In this regard, TrajOpt can be considered as a replacement for local trajectory correction approaches suggested by [44, 38]. We do not consider comparison with the RRT planning method since it computes trajectories working backwards from the target and hence is not suitable for re-planning in the forward direction.

Planning for multiple needles: We analyzed the performance of TrajOpt for planning for 5 needle trajectories using 1000 sampled points within the prostate (200 trials). We computed multiple collision-free trajectories by planning them sequentially such that the computed trajectories were mutually collision-free. We compared the result of collocation versus shooting-based strategies. For the multiple needle planning experiments, shooting offered an advantage over collocation in terms of computational time required to compute a feasible solution and the quality of trajectories computed. Fig. 1(e) shows planned trajectories for a single trial. Table IV summarizes our result, which shows that TrajOpt outperforms the RRT planner both in terms of computation time and the fraction of problems solved. The trajectories computed using the sampling-based RRT planner have a very high twist cost, which is also undesirable.

	RRT	TrajOpt	
		Collocation	Shooting
success fraction	0.48	0.75	0.79
time (s)	50.0 ± 19.0	18.0 ± 9.0	15.3 ± 15.2
path length	54.6 ± 3.1	53.9 ± 2.5	56.5 ± 3.4
twist cost	168.3 ± 28.4	3.8 ± 1.5	2.5 ± 1.8
clearance	0.1 ± 0.08	0.1 ± 0.03	0.1 ± 0.06

TABLE IV
Multiple needle planning: Sampling-based RRT planner versus TrajOpt.

2) *Channel layout planning*: We considered a scenario where 3D-printed implant is prepared for treatment of OB/GYN tumors (both vaginal and cervical), as shown in Fig. 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 2.5cm. The implant was designed based on dimensions reported by Garg et al. [14]. We set the entry region to be the base of the implant. We require that the curvature along the path is at most 1cm^{-1} and that the total curvature on the trajectory (Eq. 33g) is at most 1.57. This constraint is necessary 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 the shooting method, since the entry constraint is much easier to satisfy than the target constraint. Fig. 1(f) shows an optimized channel layout computed using our method.

The experiment results are summarized in Table V. We compared the performance of TrajOpt with a highly-optimized RRT-based planner [14]. Both the RRT-based approach and TrajOpt 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 account for the random nature. TrajOpt is able to compute a feasible solution in almost all cases, whereas the RRT planner fails more often to find a feasible solution and computes plans that have a higher cumulative path length and twist cost as compared to the solution computed using TrajOpt.

	RRT	TrajOpt
success fraction	0.74	0.98
time (s)	30.8 ± 17.9	27.7 ± 9.8
path length	41.3 ± 0.3	38.9 ± 0.1
twist cost	65.5 ± 8.4	4.1 ± 1.1

TABLE V
Channel layout planning: Sampling-based RRT planner versus TrajOpt.

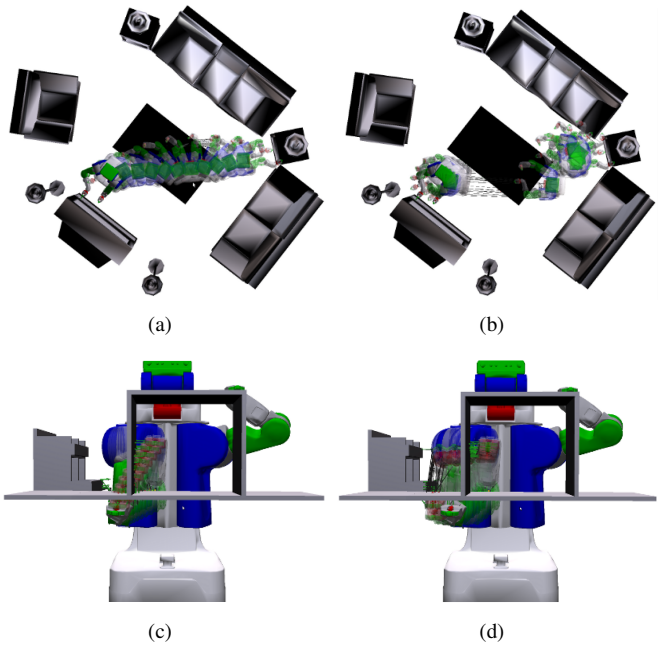


Fig. 14. Failure cases when using TrajOpt. (a) shows the initial path for full-body planning. (b) is the trajectory optimization outcome, which is stuck in an infeasible condition. (c) shows the initial path for the arm planning and the collision cannot be resolved in the final trajectory shown in (d).

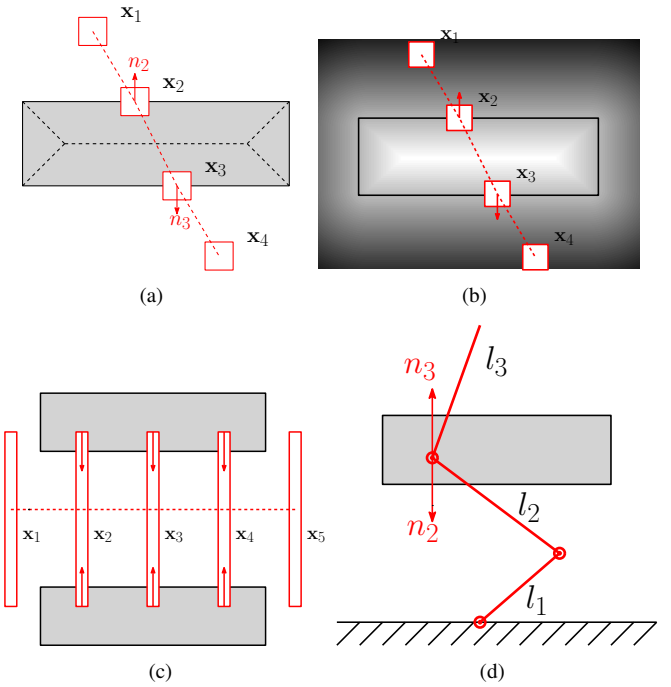


Fig. 15. Illustration of typical reasons for trajectory optimization to get stuck in local optima that are not collision-free. (a) The gradient based on penetration depth may push waypoints in in-consistent directions. (b) The gradient based on distance fields has the same problem. (c) When a robot collides 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.

IX. 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.

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 Fig. 14. Fig. 15 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 I, II).

X. Source Code and Reproducibility

All of our source code is available as a BSD-licensed open-source package called TrajOpt that is freely available here http://rll.berkeley.edu/trajopt. Optimization problems can be constructed and solved using the underlying C++ API or through Python bindings. Trajectory optimization problems 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 [4] 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: (1) Gurobi, a commercial solver, which is free for academic use [35]; and (2) BPMPD [31], 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 and arm planning with orientation constraints, are included with our software distribution.

XI. 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 also presented a discussion of the importance of trajectory initialization for optimization based approaches. The source code has been made available freely for the benefit of the research community.

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

References

[1] J.T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Society for Industrial & Applied Mathematics, 2010.

[2] J. Blanco. A tutorial on SE(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga, 2010.

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

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

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

[6] B. Cohen, I. Sucan, and S. Chitta. A generic infrastructure for benchmarking motion planners. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 589–595, 2012.

[7] E. Coumanns. Bullet physics library, 2012. www.bulletphysics.org.

[8] A.D. Dragan, N.D. Ratliff, and S.S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 4582–4588, 2011.

[9] V. Duindam, R. Alterovitz, S. Sastry, and K. Goldberg. Screw-based motion planning for bevel-tip flexible needles in 3D environments with obstacles. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 2483–2488, 2008.

[10] V. Duindam, J. Xu, R. Alterovitz, S. Sastry, and K. Goldberg. Three-dimensional motion planning algorithms for steerable needles using inverse kinematics. *Int. Journal of Robotics Research*, 29(7):789–800, 2010.

[11] J.A. Engh, D.S. Minhas, D. Kondziolka, and C.N. Riviere. Percutaneous intracerebral navigation by duty-cycled spinning of flexible bevel-tipped needles. *Neurosurgery*, 67(4):1117–1122, 2010.

[12] T. Erez and E. Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4914–4919, 2012.

[13] C. Ericson. *Real-time collision detection*. Morgan Kaufmann, 2004.

[14] A. Garg, S. Patil, T. Siauw, J. Cunha, I-C. Hsu, P. Abbeel, J. Pouliot, and K. Goldberg. An algorithm for computing customized 3d printed implants with curvature constrained channels for enhancing intracavitary brachytherapy radiation delivery. In *IEEE Int. Conf. on Automation Science and Engg. (CASE)*, pages 3306–3312, 2013.

[15] E. Gilbert, D. Johnson, and S. Keerthi. A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[16] J. Han, S Datta, and S Ekkad. *Gas turbine heat transfer and cooling technology*. CRC Press, 2013.

[17] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 2493–2498, 2010.

[18] M. Hwangbo, J. Kuffner, and T. Kanade. Efficient two-phase 3D motion planning for small fixed-wing UAVs. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 1035–1041, 2007.

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

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

[21] L. Kavraki, P. Svestka, J-C Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

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

[23] J. Kuffner and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, volume 2, pages 995–1001, 2000.

[24] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters. Trajectory planning for optimal robot catching in real-time. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 3719–3726, 2011.

[25] S. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[26] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar. Generation of whole-body optimal dynamic multi-contact motions. *Int. Journal of Robotics Research*, 2013, (to appear).

[27] J.M. Lien and N.M. Amato. Approximate convex decomposition of polyhedra. In *Proc. ACM symposium on Solid and physical modeling*, pages 121–131, 2007.

[28] M. Likhachev and A. Stentz. R* search. In *Proc. National Conference on Artificial Intelligence (AAAI)*, pages 344–350, 2008.

[29] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[30] K. Mamou and F. Ghorbel. A simple and efficient approach for 3D mesh approximate convex decomposition. In *IEEE Int. Conf. on Image Processing (ICIP)*, pages 3501–3504, 2009.

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

[32] I. Mordatch, E. Todorov, and Z. Popovic. Discovery of complex behaviors through contact-invariant optimization. *ACM SIG-*

GRAPH, 31(4):43, 2012.

[33] Richard M Murray and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[34] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer Verlag, 1999.

[35] Gurobi Optimization. Gurobi optimizer reference manual. *http://www.gurobi.com*, 2012.

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

[37] S. Patil and R. Alterovitz. Interactive motion planning for steerable needles in 3D environments with obstacles. In *Int. Conf. Biomedical Robotics and Biomechatronics (BioRob)*, pages 893–899, 2010.

[38] Q. Pham. Fast trajectory correction for nonholonomic mobile robots using affine transformations. *Robotics: Science and Systems (RSS)*, 2011.

[39] M. Posa and R. Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer, 2013.

[40] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 802–807, 1993.

[41] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 489–494, 2009.

[42] K. Reed, A. Majewicz, V. Kallem, R. Alterovitz, K. Goldberg, N. Cowan, and A. Okamura. Robot-assisted needle steering. *IEEE Robotics & Automation Magazine*, 18(4):35–46, 2011.

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

[44] K.M. Seiler, S.P.N. Singh, S. Sukkarieh, and H. Durrant-Whyte. Using Lie group symmetries for fast corrective motion planning. *Int. Journal of Robotics Research*, 31(2):151–166, 2012.

[45] M. Shanmugavel, A. Tsourdos, R. Zbikowski, and B. White. 3D path planning for multiple UAVs using Pythagorean hodograph curves. In *AIAA Guidance, Navigation, and Control Conference*, pages 20–23, 2007.

[46] I.A. Sucan and L.E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.

[47] I.A. Sucan, M. Moll, and L.E. Kavraki. The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19 (4):72–82, 2012.

[48] H.J. Sussmann. Shortest 3-dimensional paths with a prescribed curvature bound. In *IEEE Conf. on Decision and Control*, volume 4, pages 3306–3312, 1995.

[49] J. Swensen and N.J. Cowan. Torsional dynamics compensation enhances robotic control of tip-steerable needles. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 1601–1606, 2012.

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

[51] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4906–4913, 2012.

[52] G. van den Bergen. Proximity Queries and Penetration Depth Computation on 3D Game Objects. In *Game Developers Conference (GDC)*, 2001.

[53] M. Vukobratović and B. Borovac. Zero-moment pointthirty five years of its life. *Int. Journal of Humanoid Robotics*, 1(1):157–173, 2004.

[54] C.W. Warren. Global path planning using artificial potential fields. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 316–321, 1989.

[55] R. J. Webster III, J. S. Kim, Noah J. Cowan, G. S. Chirikjian, and Allison M. Okamura. Nonholonomic modeling of needle steering. *Int. Journal of Robotics Research*, 25(5-6):509–525, 2006.

[56] A. Werner, R. Lampariello, and C. Ott. Optimization-based generation and experimental validation of optimal walking trajectories for biped robots. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4373–4379, 2012.

[57] J. Xu, V. Duindam, R. Alterovitz, and K. Goldberg. Motion planning for steerable needles in 3D environments with obstacles using rapidly-exploring random trees and backchaining. In *IEEE Int. Conf. on Automation Science and Engg. (CASE)*, pages 41–46, 2008.

[58] J. Xu, V. Duindam, R. Alterovitz, J. Pouliot, J. A. Cunha, I. Hsu, and K. Goldberg. Planning fireworks trajectories for steerablemedical needles to reduce patient trauma. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4517–4522, 2009.

[59] G. Yang and V. Kapila. Optimal path planning for unmanned air vehicles with kinematic and tactical constraints. In *IEEE Conf. on Decision and Control (CDC)*, volume 2, pages 1301–1306, 2002.

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

[61] M. Zucker, N. Ratliff, A.D. Dragan, M. Pivtoraiko, M. Klingensmith, C.M. Dellin, J.A. Bagnell, and S.S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *Int. Journal of Robotics Research*, 2012.