Journal Title
XX(X):1–16
©The Author(s) 2020
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Semi-Infinite Programming for Trajectory Optimization with Nonconvex Obstacles

Kris Hauser

Abstract

This paper presents a novel optimization method that handles collision constraints with complex, non-convex 3D geometries. The optimization problem is cast as a semi-infinite program in which each collision constraint is implicitly treated as an infinite number of numeric constraints. The approach progressively generates some of these constraints for inclusion in a finite nonlinear program. Constraint generation uses an oracle to detect points of deepest penetration, and this oracle is implemented efficiently via signed distance field (SDF) versus point cloud collision detection. This approach is applied to pose optimization and trajectory optimization for both free-flying rigid bodies and articulated robots. Experiments demonstrate performance improvements compared to optimizers that handle only convex polyhedra, and demonstrate efficient collision avoidance between nonconvex CAD models and point clouds in a variety of pose and trajectory optimization settings.

Keywords

Optimization, numerical methods, motion planning, collision avoidance

Introduction

Optimization in robotics has long been complicated by the difficulty of encoding collision constraints between complex geometries, even though complex geometries are routinely handled by planners that produce feasible (non-optimal) paths. Sampling-based motion planners (Karaman and Frazzoli 2011) have been successful in part because they leverage the extensive body of work on fast collision detection between non-convex bodies (Gottschalk et al. 1996). However, collision queries are binary computations that cannot be easily incorporated into numerical optimizers, which usually require real-valued, differentiable constraints. A naïve numerical encoding for objects composed of M and N geometric primitives (e.g., triangles) would require $O(MN)$ pairwise constraints. Hence, past optimization approaches require robot links and environments to be represented using simple convex geometries, such as ellipsoids (Saramago and Junior 2000), capsules (Zucker et al. 2013), polyhedra (Richards et al. 2002; Schulman et al. 2014), and superquadrics (Chakraborty et al. 2008) to speed up computation. This imposes severe limitations on allowable shapes of objects and robot geometries.

This paper introduces a novel constraint encoding of non-convex collision constraints that enable optimization to be completed in a computationally-efficient manner. This approach represents each collision constraint as an *infinite set* of simpler constraints in a semi-infinite programming (SIP) framework. Although it is not possible to optimize an infinite number of constraints directly, a finite subset of constraints, suitably chosen, is sufficient to define an optimum. An *exchange method* is used that instantiates a series of finite optimization problems, each of which progressively adds some number of constraints. Using a judicious constraint selection procedure, called an *oracle*, the series of problems converges toward one that contains a true optimum.

For collision avoidance between a pair of objects, we establish constraints that every point on the surface of one object is required to be outside of or on the surface of the other object. We call the first object the privileged object. The oracle detects the point in the privileged geometry that penetrates most deeply into the other. To make this procedure fast, the privileged geometry is represented as a point cloud, and the other is represented as a signed distance field (SDF) which supports $O(1)$ depth lookup and $O(1)$ gradient

Corresponding author:
Kris Hauser, Department of Computer Science, Department of Electrical and Computer Engineering University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA.
Email: kkhauser@illinois.edu

estimation at a point. Bounding volume hierarchies speed up the oracle query. For trajectories, dynamic collision detection techniques are able to find the deepest point in both space and time. A benefit of this technique is that raw point clouds can be directly used in optimization, while precomputed SDFs can be employed for robot links. Similar work by Kuntz et al has also considered trajectory optimization with point cloud collision constraints Kuntz et al. (2017), and like this work, our approach avoids the expense of computing geometric data structures on-the-fly as environmental point clouds are acquired from sensors. Unlike Ref. Kuntz et al. (2017), however, our semi-infinite programming formulation has the ability to extract solutions from penetration, instantiates fewer constraints and avoids the use of heuristic methods for discontinuity detection.

The technique is implemented for free-flying rigid bodies as well as articulated robots, both to optimize static poses and trajectories. The technique converges quickly to local optima even with highly complex geometries. Poses can be optimized in tens of milliseconds and trajectories in a few seconds on standard PC hardware. Surprisingly, performance is comparable to and sometimes better than optimizers specialized for convex polyhedral geometries, because on the geometric level, SIP operates with smoothly differentiable constraints whereas the min-distance function is non-differentiable. On the temporal level, it may instantiate fewer constraints than direct collocation methods. The method is also suitable as a postprocessor for a sampling-based motion planner to generate high-quality, consistent paths.

This paper is an expanded version of a conference paper Hauser (2018). It adds a novel technique for improved penetration depth estimation (Section *Performance improvements with inner spheres*); a new method that integrates the optimizer with feasible motion planning (Section *Optimal Trajectory Planning*); and additional details, references, and experiments (Section *Experiments*).

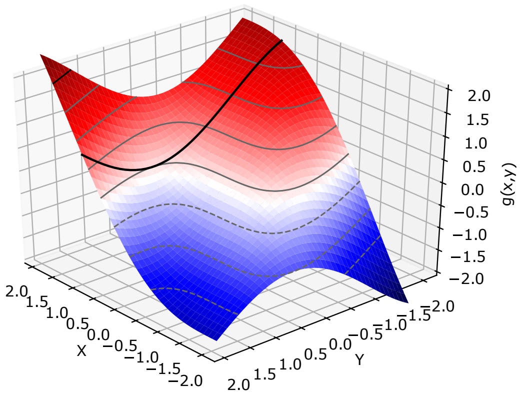


Figure 1. An example of a simple semi-infinite optimization problem with $f(x) = x$, one constraint $g_1(x, y) = x - \sin(x + y) \geq 0$, and domain $y \in [-2, 2]$. The minimum is at $x = 1$, with the supporting value of $y = \pi/2 - 1$. The dark curve plots the constraint value while x is fixed at the optimum.

Semi-infinite programming with collision constraints

Semi-infinite programming

A semi-infinite programming (SIP) problem over the state variable $x \in \mathbb{R}^n$ is defined as follows

$$\begin{gathered}\min_x f(x) \\ \text{s.t.} \\ g_1(x, y_1) \geq 0 \quad \forall y_1 \in Y_1 \\ \vdots \\ g_M(x, y_M) \geq 0 \quad \forall y_M \in Y_M.\end{gathered} \tag{1}$$

where $g_i(x, y) \in \mathbb{R}^{m_i}$ are the constraint functions, y_i denotes the i'th *index parameter*, and $Y_i \subseteq \mathbb{R}^{p_i}$ is its domain. Each of the functions f is assumed to be twice differentiable, $g_1, \ldots, g_M$ are assumed to be differentiable, and inequalities are interpreted element-wise.

The constraints in (1) define an infinite set of parameters for which the constraint must be satisfied. An example of such a problem is shown in Fig. 1. SIP has also been applied to robot trajectory optimization problems under state and control constraints, in which the 1D time variable of the spline is the index parameter (Vaz et al. 2004). It has also been applied to robust optimization problems in which the disturbance is the index parameter (Zeng and Zhao 2013).

Of course, existing optimization solvers cannot directly consider a continuous infinity of constraints, which has motivated a rich set of approaches in the optimization literature. The "scenario approach" used in the field of robust optimal control samples from a continuous set of disturbances, and solves an optimization of finite dimension over the instantiated constraints (Campi et al. 2009). However, this approach only ensures feasibility for a fraction of the index parameter domain. A minimax approach (Reemtsen and Görner 1998) eliminates the y terms by replacing constraints with inner minimizations of the form:

$$\tilde{g}_i(x) \equiv \min_{y_i \in Y_i} g_i(x, y_i) \geq 0. \tag{2}$$

For example, the constraint in Fig. 1 could be replaced with

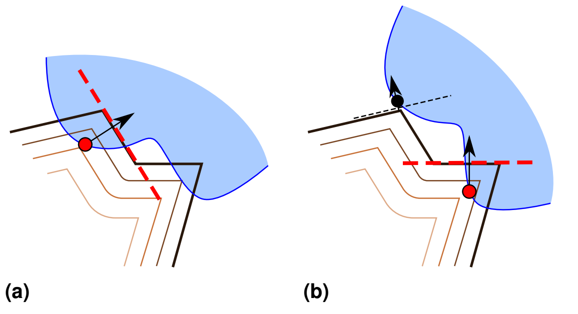
$$\tilde{g}(x) = \begin{cases} x - 1 & \text{if } \pi/2 \in [x-2, x+2], \\ x - \max(\sin(x-2), \sin(x+2)) & \text{otherwise.} \end{cases} \tag{3}$$

This approach allows (1) to be encoded as a standard nonlinear program, but a nondifferentiable one. Prior work in trajectory optimization for humanoids has applied this approach to ensure feasibility of the dynamic constraints over a time interval (Lengagne et al. 2013). For collision constraints, a common approach is to employ the distance function (Bobrow 1988; Shiller and Dubowsky 1989), which only depends on the configuration x. The distance function approach has been applied to collision avoidance for humanoid robots, for example (Lee et al. 2012). But in practice, convergence difficulties arise due to nondifferentiability, which occurs when the minimizing parameter jumps discontinuously. For example, consider a rectangle slightly angled so that vertex a penetrates a plane and defines the penetration depth. On the next iteration, another vertex, say vertex b, may penetrate and define the active constraint. This process may then oscillate between a and b because both vertices cannot be considered simultaneously active.

Optimization with instantiated constraints

Suppose we are given a finite number N of *instantiated* constraint indices $i^{(1)}, \ldots, i^{(N)}$ and corresponding index parameters $y^{(1)}, \ldots, y^{(N)}$. Here, $i^{(j)} \in \{1, \ldots, M\}$ and $y^{(j)} \in Y_{i^{(j)}}$, for $j = 1, \ldots, N$. We can then define an instantiation of the structured problem corresponding to these parameters as follows:

$$\begin{gathered} \min_x f(x) \\ \text{s.t.} \\ g_{i^{(1)}}\left(x, y^{(1)}\right) \geq 0 \\ \vdots \\ g_{i^{(N)}}\left(x, y^{(N)}\right) \geq 0. \end{gathered} \tag{4}$$



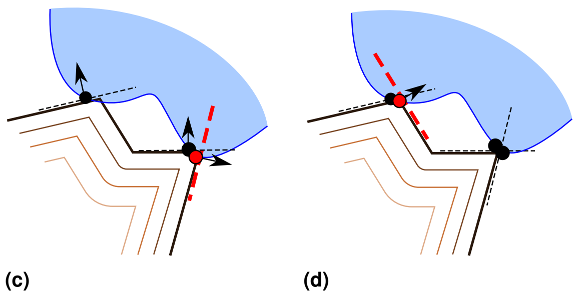


Figure 2. Illustrating the SIP exchange method to enforce separation of two objects. A contour plot of the fixed object's signed distance field is illustrated. After each iteration, the oracle adds the deepest penetrating point (red circle) to the list of constraint points (black circles).

This is a finite-dimensional nonlinear program (NLP) with $O(N \max(m_1, \ldots, m_M))$ constraints, and can be solved (locally) using standard methods like sequential quadratic programming (SQP) or interior point methods.

The idea of the *exchange method* is to progressively instantiate constraints and parameters $(i^{(1)}, y^{(1)}), (i^{(2)}, y^{(2)}), \ldots$ giving rise to a sequence of instantiated NLPs whose solutions converge toward the true optimum (Reemtsen and Görner 1998). Specifically, define P_k as the instantiation corresponding to the first k elements of the constraint sequence, and let $x_k^\star$ its solution. A naïve approach would sample points incrementally from each domain (e.g., randomly or on a grid), and with a sufficiently dense set of points the iterates $x_1^\star, x_2^\star, \ldots$ will eventually approach an optimum. But this approach is inefficient as most samples will not affect the iterated solutions. It is also possible to delete constraints from the constraint set when they are not deemed necessary (the "exchange"), which saves time in later NLP solve steps. For example, one simple strategy is to delete all instantiated constraints whose value at the current iterate exceeds a threshold γ.

Constraint generation oracle The key question for constraint generation is *which* new constraint $g_{i^{(k)}}(x, y^{(k)}) \geq 0$ to add to yield fast convergence. We rely on an *oracle*, a subroutine that performs this selection process. As an example, a *maximum-violation oracle* identifies a parameter value that has a large effect on the next iterated solution. Specifically, on iteration k this strategy calculates the most violating parameter of each constraint, keeping x fixed at x_{k-1}:

$$y_i^{min} \leftarrow \arg\min_{y \in Y_i} \min g_i(x_{k-1}, y). \tag{5}$$

in which the second minimization finds the smallest element in the g_i vector. Then, the constraint with minimum value is computed as

$$\begin{aligned} i^{(k)} &\leftarrow \arg\min_{i=1}^{M} \min g_i\left(x_{k-1}, y_i^{(k)}\right) \\ y^{(k)} &\leftarrow y_{i^{(k)}}^{min}. \end{aligned} \tag{6}$$

The constraint generation process using this oracle is illustrated in Fig. 2.

This approach does, however, beg the question about how to perform the minimization (5) over each Y_i efficiently. Best results are obtained by obtaining a global minimum, and to do this quickly it is often necessary to resort to implementation-specific procedures such as branch-and-bound. In the following discussion we shall assume that such a minimizer is available, and postpone discussion of its implementation until later sections.

Pseudocode The basic meta-algorithm is listed in Algorithm 1. It takes as input the problem, an initial guess x_0, and an iteration count N. It also uses the *Oracle* subroutine. Return status may include *infeasible*, which indicates an infeasible local minimum, *converged*, which indicates a feasible local or global minimum, and *not converged*, which means the iteration count is exhausted.

Algorithm 1 Basic SIP solver pseudocode

```
1:  procedure SIP($f, g_1, \ldots, g_M, Y_1, \ldots, Y_M, x_0, N, S, \gamma$)    ▷ $x_0$ is an initial guess
2:      $I \leftarrow \{\}$    ▷ Instantiated constraints
3:      for $k = 1, 2, \ldots, N$ do
4:          Generate $i_k, y_k$ via $Oracle(x_{k-1})$
5:          Add $(i_k, y_k)$ to $I$
6:          Remove from $I$ any $(i, y)$ where $g_i(x_{k-1}, y) \geq \gamma$
7:          Let $P_k$ be (4) instantiated with $I$.
8:          Run $S$ steps of an NLP solver on $P_k$, starting from $x_{k-1}$
9:          If $P_k$ is infeasible, return "infeasible"
10:         Otherwise, set $x_k$ to its solution
11:         if $x_k$ is unchanged, return $x_k$, "converged"
12:     end for
13:     return $x_N$, "not converged"
14: end procedure
```

Besides the choice of N and the oracle, there are a number of performance characteristics to tune.

Constraint instantiation strategy. The strategy used in Steps 4 and 5 is an important component of performance, with most-violating constraint at one end of a spectrum. There is a tradeoff when choosing how many constraints to instantiate, since adding more constraints helps the method converge in fewer iterations, but increases the cost of solving the optimization problem at each iteration. We have experimented with the approach of instantiating M constraints in a single iteration, one for each index i and most-violating parameter y_i^{min}. Another approach, if a global optimization technique is used, might instantiate constraints corresponding to all *local* minima of (5).

It is important to detect and ignore duplicated or near-duplicated index parameters in Step 5, because the most-violating parameter may stay unchanged between subsequent steps. This avoids adding multiple identical constraints that may cause numerical difficulties during NLP solving.

Inner optimization strategy. The method employed to solve for the new optimization variable in Step 8 is also important. Coherence between problems P_{k-1} and P_k may suggest the use of warm-starting to speed up solve times. The number of steps S may be set low to avoid spending too much time on problems whose constraint sets are not sufficiently populated. There is a tradeoff between effort expended on inner optimization and convergence rate, because effort may be wasted on early problems that omit crucial constraints. On the other hand, solve times are slower on later outer iterations because more constraints are instantiated. The extreme $S = 1$ version of this approach is to take a single QP step, which is most favorable when the oracle is fast and f is approximated well by its quadratic Taylor expansion.

Each quadratic programming (QP) call finds a step Δx s.t.

$$\begin{gathered} \min_{\Delta x} \frac{1}{2} \Delta x^T \nabla^2 f(x_k) \Delta x + \nabla f(x_k) \Delta x \\ \text{s.t.} \\ \nabla_x g_{i^{(1)}}\left(x_k, y^{(1)}\right) \Delta x + g_{i^{(1)}}\left(x_k, y^{(1)}\right) \geq 0 \\ \vdots \\ \nabla_x g_{i^{(N)}}\left(x_k, y^{(N)}\right) \Delta x + g_{i^{(N)}}\left(x_k, y^{(N)}\right) \geq 0 \end{gathered} \tag{7}$$

and then takes a step $x_{k+1} \leftarrow x_k + \alpha_k \Delta x$. Here α_k is a parameter in $[0, 1]$ determined via line search or a trust region method. One problem that occurs is that after the move to x_k, we may discover that it deeply violates constraints that were not previously instantiated. This may lead to oscillatory behavior or large jumps into basins of attraction of infeasible local minima. To avoid this problem, we implement a trust

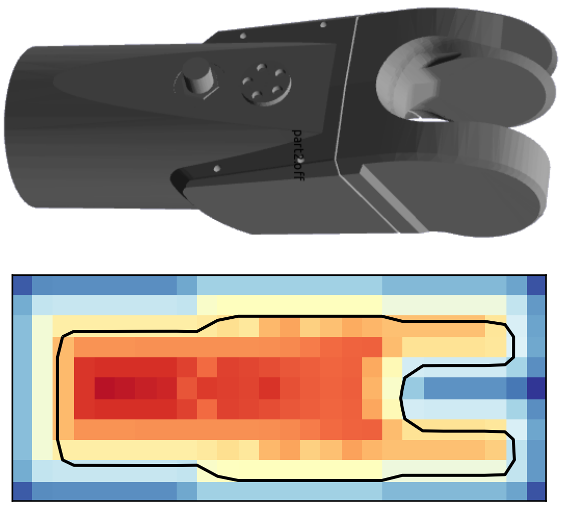


Figure 3. Illustrating a slice of a signed distance field (SDF) for a non-convex link of an industrial robot.

region approach that limits the step size to a box $-h_k \leq \Delta x \leq h_k$. If a new deeper point is discovered after a candidate step, the step is not taken ($\alpha_k = 0$) and the trust region is shrunk. The step is also rejected if a merit function is increased. Otherwise, the full step is taken and the trust region is grown. We implement trust region shrinking with $h_{k+1} \leftarrow h_k \cdot 0.5$ and growing with $h_{k+1} \leftarrow h_k \cdot 2.5$

Another issue to be addressed is that the QP (7) may not be feasible. In this case, we formulate a relaxed QP that introduces slack variables into the constraints. We then minimize a weighted sum of the standard objective function and the sum of squares of slack variables to determine the desired step. The relative weight of the objective function decreases with increasing iteration count, which gives an increasing preference to obtaining feasibility vs optimality as the algorithm progresses.

Collision-free constraint formulation

Let q denote a configuration of the system and A and B two objects. A collision constraint dictates that the workspace occupied by the geometries of the objects $G_A \subset \mathbb{R}^3$ and $G_B \subset \mathbb{R}^3$ do not intersect. Let us assume that G_A and G_B are open sets, so the constraint requires that $G_A(q) \cap G_B(q) = \emptyset$.

We assume each object is a rigid body, and hence the configuration only affects the object's rigid transform $T_A(q)$ relative to the world frame. Hence, we can express $G_A(q) = T_A(q)G_A^0$, where G_A^0 is the object's geometry in its reference frame. Let us assume that each reference geometry has a surface representation ∂G_A^0 and a *signed distance function* (SDF) $D_A(y)$. The SDF is defined so that $|D_A(y)|$ gives the distance from $y \in \mathbb{R}^3$ to ∂G_A^0, and $sign(D_A(y)) = -1$ if $y \in G_A^0$, $sign(D_A(y)) = 0$ if $y \in \partial G_A^0$, and $sign(D_A(y)) = 1$ if y is strictly outside.

Barring the possibility of B being completely enclosed by A, an estimate of distance between A and B is given by

$$d_{AB}(q) \equiv \min_{y \in \partial G_A^0} D_B\left(T_A^{-1}(q)T_B(q)y\right). \tag{8}$$

This value is exact when A and B are disjoint, and is an approximation of the negated penetration depth when A and B intersect (provided that $B \not\subseteq A$.)

The benefit of this approach is that penetration depth lookup for a single point is performed in $O(1)$ time. SDF gradient calculation is also $O(1)$ using finite differencing. Given a closed polygonal mesh, the SDF is calculated using the Fast Marching Method (FMM) applied on a voxel grid. Values off of the grid vertices are approximated via trilinear interpolation. If y is outside of the grid entirely, the distance is approximated by determining the closest point y' in the grid, and assigning $D_A(y) = \|y - y'\| + D_A(y')$.

To represent the distance between two objects, we define

$$g_{AB}(q, y) \equiv D_B\left(T_A^{-1}(q)T_B(q)y\right) \tag{9}$$

to establish a semi-infinite constraint over the domain $y \in G_A^0$.

It is a straightforward matter to provide constraint Jacobian information, which is used by most optimization algorithms like SQP use to determine linearized approximations of constraints at each inner iteration. The Jacobian of the constraint function is

$$\nabla_q g_{AB}(q, y) = \nabla D_B\left(T_B^{-1}(q)T_A(q)y\right) \cdot J_A^B(q, y) \tag{10}$$

where ∇D_B is the distance field gradient, calculated by finite differences, and $J_A^B(q, y)$ is $\nabla_q(T_B^{-1}(q)T_A(q)y)$. Specifically this is the Jacobian of the coordinates of point y relative to B's coordinate frame, where y is given in A's local coordinates. This matrix is calculated via forward kinematics depending on whether q encodes an articulated robot or object pose.

Performance notes

It is important to note that the constraint is non-symmetric, as object A is represented as a surface model while B is the SDF. We call object A the *privileged object*, and the choice of the privileged object in a pair can affect performance in two ways. First, the more complex B is, the more likely optimization will fall into local minima in its SDF. Second, the amount of precomputation needed to build an SDF is non-negligible, and SDF construction from a mesh may suffer from artifacts if the mesh is non-watertight. As a result, it is typically better to represent robot links as SDFs because they do not change over time, and this gives developers a chance to manually inspect the SDF for a high-quality representation. Note that to enforce self-collision constraints, robot links will also need to store a surface representation. Environment geometries are better suited for surface representations, since these can be constructed dynamically from sensor data with minimal amounts of precomputation.

It should be noted that SDFs may contain non-differentiable points along the medial axis of the object, which may slow convergence of optimization. But each optimization step drives points away from these poorly behaved areas, and furthermore the smoothing effect of finite differences mitigates the potential performance degradation.

Finally, significant speed gains can be obtained by implementing *branch-and-bound techniques* in the most-violating constraint oracle. Here, an upper bound $\bar{g}$ on the most-violating constraint value is maintained, initialized with the minimum value of g_i over all previous instantiated constraints. Then, during each minimization of (5), $\bar{g}$ is used to discard subsets of the domain that have no chance of yielding a smaller value. $\bar{g}$ is then updated as smaller constraint values are found.

Most-violating parameter calculation

When ∂G_A^0 is approximated via a point cloud of p points, the most-violating parameter of (9), e.g., the closest / deepest penetrating point, can be determined in $O(p)$ time using brute force computation. Brute force computation is trivially parallelizable and suitable for implementation on a GPU. However, for large values of p a bounding volume hierarchy (BVH) approach may be significantly faster.

A BVH is a hierarchical geometric data structure of progressively smaller bounding geometries, where leaf nodes contain one or more primitive points (Gottschalk et al. 1996). The significance is that proximity queries between two BVHs can be answered quickly using branch-and-bound techniques. Although BVH proximity queries are more complex and BVH construction incurs some precomputation cost, the speed gains across multiple penetration depth may ultimately make it worthwhile. The BVH approach also allows for branch-and-bound techniques to apply a common upper bound across multiple constraints, which speeds up most-violating parameter determination.

Our approach builds a sphere-based BVH of the point cloud. Specifically, an octree data structure is built containing the point cloud, subdividing until each cell contains no more than 10 points. For each leaf octree node, an axis-aligned bounding box is fit to the points it contains, and for each non-leaf node, a bounding box is fit to the bounding boxes of its children. Bounding spheres of leaf nodes are fit to the points inside the node, with the sphere center set to the centroid of the points contained within. Bounding spheres of non-leaf nodes are fit to the the bounding spheres of non-empty child nodes. Specifically, the center c is set to a weighted centroid of child bounding sphere centers c_i, with a weight equal to $\epsilon + r_i$, where ϵ is a small constant and r_i is the radius of the bounding sphere of child i. The radius r is then fit to contain all child bounding spheres:

$$r \leftarrow \max_{i \in \text{children}} (\|c - c_i\| + r_i). \tag{11}$$

If $2r$ is greater than the distance from the lower and upper ranges of the bounding box, p_{min} and p_{max}, respectively, then we set $c \leftarrow (p_{min} + p_{max})/2$ and $r = |p_{min} - p_{max}\|/2$.

To query for the closest / deepest penetrating point against an SDF geometry d_B, we use a branch and bound method that relies on a fast lower bound query between a node N in the BVH and the SDF. Given N's bounding sphere with center c and radius r, the minimum value of the SDF within this sphere is lower bounded by $l_N = d_B(T_B^{-1} T_A c) - r$. This is also a lower bound on the SDF value at the points contained with N.

The nodes of the BVH are traversed, starting from the root in order of non-decreasing l_N. A lowest encountered distance value d_{min} is maintained, which is initialized to an arbitrary point in ∂G_A^0. If traversal encounters any node N with $l_N \geq d_{min}$, it is pruned. Otherwise, if N is a leaf node then all of its points are checked for being the deepest point. Otherwise, its children $C_1, \ldots, C_k$ are added to the traversal queue with priority values $l_{C_1}, \ldots l_{C_k}$.

Performance improvements with inner spheres

A drawback of the surface point cloud method is that penetration depths are not estimated accurately when the non-privileged object is deeply penetrating the point cloud. For example, if a robot's forearm were penetrating a wall more than halfway, the separation direction for the deepest penetrating point would have the forearm penetrate deeper past the wall's surface. This has the opposite effect of the desired behavior. To improve the accuracy of penetration depth estimation, a point cloud can be augmented with *inner spheres* that represent parts of the internal volume of the privileged object. This is inspired by the inner sphere tree (Weller and Zachmann 2009) approach for calculating penetration volumes for interactive haptic simulation. In our case the inner spheres are allowed to be overlapping which arguably leads to better estimates of penetration depth.

In this method, we add some interior points $p_1, \ldots, p_M$ to the surface representation of A along with their radii $r_1, \ldots, r_M$. The radii are the distance from the point to the nearest point on the surface (Fig. 4.a). If an SDF is available, these points are added by random sampling, keeping interior points, and evaluating the radius as the negative value of the SDF.

The new signed distance function between two objects, is the same for non-overlapping configurations, but can be more accurate during penetrations, as illustrated in Figs. 4.b and c. Eq. 8 is rewritten as follows:

$$\begin{aligned} d_{AB}(q) \equiv \min\Bigg(&\min_{y \in \partial G_A^0} D_B\left(T_A^{-1}(q) T_B(q) y\right), \\ &\min_{i=1,\ldots,M} \left(D_B(T_A^{-1}(q) T_B(q) p_i) - r_i\right)\Bigg). \end{aligned} \tag{12}$$

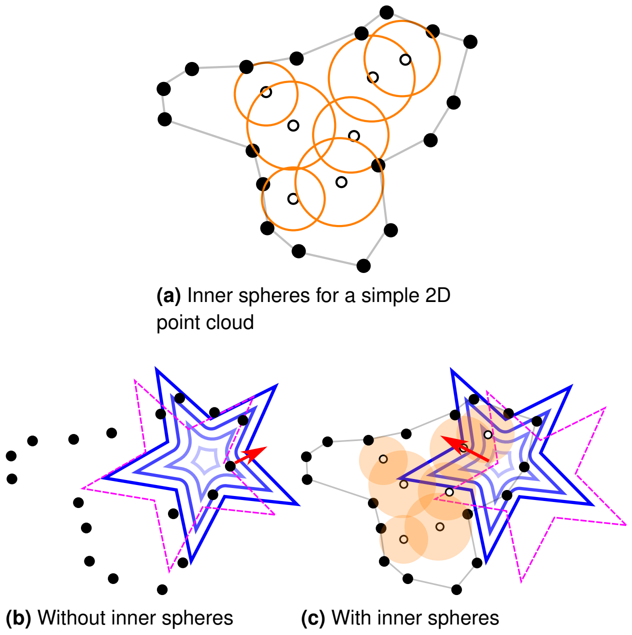
In practice, this is implemented by augmenting the point cloud with a radius attribute, where all surface points have radius 0. The discrete minimization in the second part of (12) is then used. The index parameter y of the semi-infinite constraint is now a 4-D vector containing both the point coordinates p as well as the point's radius r. To make the most-violating parameter calculation fast, the BVH approach described in the prior section is modified so that each bounding volume contains both surface points and the inner spheres.

It should be noted that this is best suited for watertight geometries for which the inner depth can be estimated accurately. For sensed point clouds, however, it can be difficult to determine large interior regions that are occluded from the camera's point of view. On the other hand, it is reasonable to add inner spheres in occluded regions to prevent a robot from making inadvertent contact.

Semi-Infinite Formulation for Trajectory Optimization

Trajectory collision constraints

We also take a semi-infinite approach to formulate collision constraints along an entire trajectory. This is in contrast to a classical collocation approach, which instantiates static constraints at a discrete set of points in the time domain. The two disadvantages of collocation are that collisions may be missed between collocation points, and that a large number of collocation points leads to a large number of constraints and hence slow optimization times.


(a) Inner spheres for a simple 2D point cloud

(b) Without inner spheres

(c) With inner spheres

Figure 4. Inner spheres allow for better penetration depth estimation with deeply overlapping shapes. Without inner spheres, moving the star in the negative estimated separation direction (red) moves it inward, increasing penetration. With inner spheres, moving in the negative estimated separation direction moves the star outward, as desired.

Instead, we consider time to be a parameter in a semi-infinite constraint that enforces collision constraints across the entire continuous trajectory. Let the optimization variable x define the configuration trajectory $q(t)$, e.g., for splines x is a stacked set of control points. To make the dependence of the trajectory on x clear, we shall say $q(t; x)$.

Now, we modify (9) to include time as an additional variable as follows:

$$\begin{aligned} h_{AB}(x, p, t) &\equiv g_{AB}(q(t; x), p) \\ &= D_B(T_A^{-1}(q(t; x)) T_B(q(t; x)) p) \end{aligned} \tag{13}$$

which is treated as a semi-infinite constraint over the index parameter $y = (p, t)$ with domain $y \in G_A^0 \times [0, T]$. The Jacobian with respect to x is computed via the chain rule, and for spline representations the Jacobian is sparse.

The most-violating constraint of (13) can be determined efficiently via a branch-and-bound technique. Determine a Lipschitz constraint K_k bounding the magnitude of (9) with respect to the kth entry of q. Then, for any q', the change of (9) from its value at a different configuration q is bounded in magnitude by

$$|g_{AB}(q', y) - g_{AB}(q, y)| \leq K(q - q') \tag{14}$$

where

$$K(\Delta q) \equiv \sum_{k=1}^{n} K_k |\Delta q_k|. \tag{15}$$

Over any time interval $[t^a, t^b]$, from the spline representation we can determine a Lipschitz bound K_Δ on the trajectory derivative. This establishes a parallelipiped in state-time space that is guaranteed to contain the trajectory segment:

$$\begin{aligned} |q(t) - q(t^a)| &\le |t - t^a| \cdot K_\Delta \\ |q(t) - q(t^b)| &\le |t - t^b| \cdot K_\Delta \end{aligned} \tag{16}$$

We may then use this in conjunction with (14) to bound the value of $h_{AB}(x,p,t)$ over all p and $t \in [t^a, t^b]$. If we have calculated $g^a = g_{AB}(q(t^a), p^a)$ and $g^b = g_{AB}(q(t^b), p^b)$ along with most-violating points p^a and p^b at the endpoints, then we can obtain the bounds

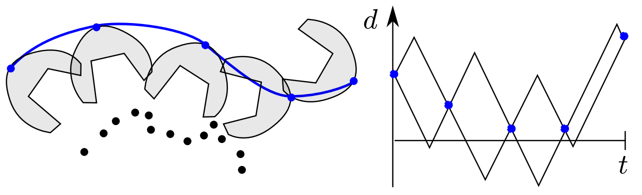
$$\begin{aligned} |h_{AB}(x,p,t) - g^a| &\le |t - t^a| \cdot K(K_\Delta) \\ |h_{AB}(x,p,t) - g^b| &\le |t - t^b| \cdot K(K_\Delta) \\ &\forall\, p \in \partial G_A^0 \text{ and } t \in [t^a, t^b]. \end{aligned} \tag{17}$$

Hence, a lower bound on the possible values of $h_{AB}(x,p,t)$ over this domain is obtained at the value of t such that $g^a - (t - t^a)K(K_\Delta) = g^b - (t^b - t)K(K_\Delta)$. This value is $t = \frac{1}{2K(K_\Delta)}(g^a - g^b) + \frac{t^b + t^a}{2}$ giving

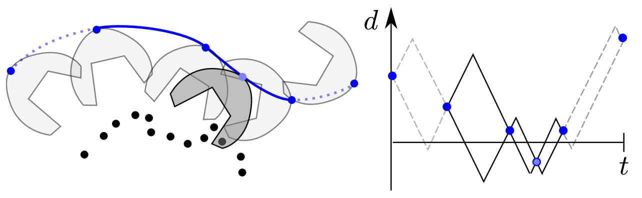
$$h_{AB}(x,p,t) \ge \frac{1}{2}(g^a + g^b) - \frac{1}{2}(t_b - t_a)K(K_\Delta). \tag{18}$$

If this lower bound is greater than the least currently established upper bound on the distance $\bar{h}$, then the interval $[t^a, t^b]$ can be pruned from consideration.

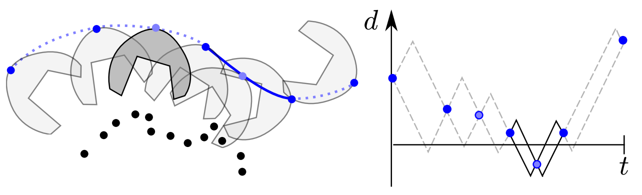
Overall the approach uses recursive subdivision to find the most-violating continuous time and point on A, up to a given resolution ϵ (which is far finer than would be reasonable for collocation methods, e.g. 10^{-5}). First, we evaluate statically the penetration depth at states $q(t_0), \ldots, q(t_S)$ where $t_0, \ldots, t_S$ are the spline knot points. An upper bound $\bar{h}$ on the most violating point and time is initialized to the minimizer of g_{AB} across knot points. For each of the intermediate ranges (t_k, t_{k+1}) we recursively subdivide while pruning any segment for which the r.h.s. of (18) exceeds $\bar{h}$. At the midpoint $(t_k + t_{k+1})/2$, the static penetration depth is evaluated, and the process recurses. Furthermore, to lower $\bar{h}$ as quickly as possible, candidate segments are stored in a priority queue sorted by increasing lower bound. This process is illustrated in Fig. 5. The process quickly narrows down to a small range of possible most-violating times, and empirically we have observed that the number of static penetration depth computations is approximately $O(\log \epsilon)$.



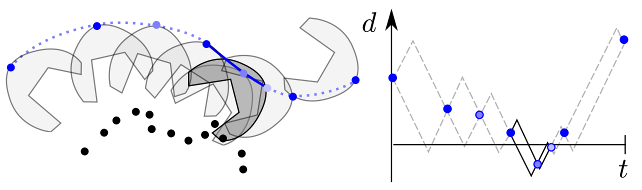
(a) Initial trajectory milestones



(b) First subdivision



(c) Second subdivision



(d) Third subdivision

Figure 5. Illustrating the branch-and-bound method for finding the minimum distance along a trajectory. Left column shows evaluated milestones, and the milestone whose distance is evaluated in each subdivision is highlighted. Right column shows the distances at evaluated milestones and Lipschitz bounds. Pruned trajectory segments are drawn as dashed lines.

To further improve performance, it is useful to perform branch-and-bound over many constraints at once. This is especially true when considering a large number of potential collision pairs. To do so we unify the segment priority queues and maintain a common upper bound $\bar{h}$. This quickly eliminates the need for dynamic collision checking for pairs that have no chance of defining the most violating constraint.

Optimal Trajectory Planning

It should be noted that our method, like other optimization methods, performs much better when initialized with a collision-free trajectory, rather than having to extricate the robot from a deeply-penetrating pose. This is because the separation direction is poorly approximated when the objects penetrate deeply, and furthermore the interaction between geometric penetration and robot kinematics is highly nonlinear and complex. For example, if an initial configuration has the robot's hand behind a two-sided wall, the back-face of the wall would suggest that the forearm should move forward through the wall in order to resolve the collision. As a result, optimization performs better when used as a postprocessor for a feasible motion planner, e.g., PRM or RRT, rather than a replacement.

We propose using a feasible, sampling-based planner to generate a seed trajectory for the optimizer, similar to the interleaved sampling / optimization approaches introduced in Refs. Kim and Yoon (2020); Kuntz et al. (2017). However, the sampled solution may be unlikely to lie in the basin of attraction of an optimum. To have a higher chance of finding an optimal path, we employ a random restart approach, where after optimization, the process is begun again, and if a better path is found, it is kept. This is an any-time approach that can be terminated at any point and returns the best path found so far (Luo and Hauser 2014). The algorithm is listed in Alg. 2.

Algorithm 2 Any-time planning with trajectory optimization

```
1:  $q_{best} \leftarrow nil$
2:  $f_{best} \leftarrow \infty$
3:  while $t < t_{max}$ do
4:      $q_{seed} \leftarrow$Plan($t_{plan}$)
5:      if $q_{seed} \neq nil$ then                    ▷ Optimize new plan
6:          $q \leftarrow$Optimize($q_{seed}, N_{opt}/2$)
7:          if $f(q) < f_{best}$ then
8:              $f_{best} \leftarrow f(q)$
9:              $q_{best} \leftarrow q$
10:         end if
11:     end if
12:     if $q_{best} \neq nil$ then                   ▷ Improve current best
13:         $q \leftarrow$Optimize($q_{best}, N_{opt}/2$)
14:         if $f(q) < f_{best}$ then
15:             $f_{best} \leftarrow f(q)$
16:             $q_{best} \leftarrow q$
17:         end if
18:     end if
19: end while
20: return $q_{best}$
```

There are three major parameters in this algorithm: t_{max} is the maximum time spent planning and optimizing, t_{plan} is the maximum amount of time for a single planner iteration, and N_{opt} is the maximum number of optimization iterations per outer iteration. The application itself should determine the total time t_{max}. The more important tradeoff is between the planning time t_{plan} and the number of optimization iterations N_{opt}. If it is likely for the optimizer to get stuck in local minima, then it is better to give the planner another chance to find a better optimum, and hence N_{opt} should be set low so that the planner has more chances to run. On the other hand, if there are few optima, then better results are obtained by letting the optimizer run longer.

If the planned path consists of only a few milestones, it is better to subdivide the planned path before optimization in order to obtain better minima. Since highly subdivided paths lead to longer running times, a few dozen milestones are usually adequate. In our experiments, if the initial path has fewer than 30 milestones, we double the number of milestones, allocating new milestones proportionally to the length of the path segment.

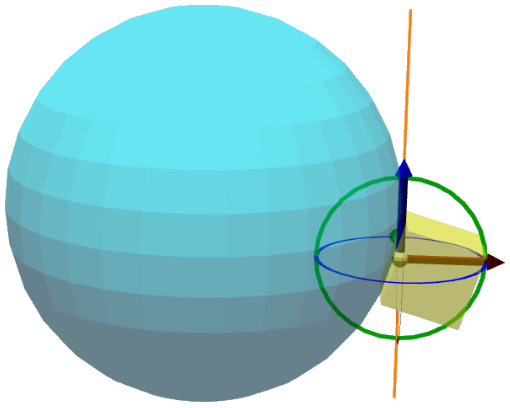
If path length is the objective function, it has been found that "shortcutting" is a fast and effective method for improving the quality of planned paths (Luna et al. 2013). This method repeatedly picks two random points along the path, and replaces the intervening path with a straight line segment if the segment is feasible. The approach is indeed fast and asymptotically optimal with an any-time implementation, but is somewhat more prone to local minima than genuine trajectory optimization. In this approach, after a first plan is generated, the shortcutting process is performed until time t_{plan} is elapsed. Shortcutting tends to generate paths with extremely nonuniform spacing between milestones, typically being "bunched up" around curves.

Experiments

The SIP algorithm is implemented with front end in the Python programming language, with Python bindings to the OSQP QP solver (Stellato et al. 2017), which is implemented in C, and custom C++ collision detection software. All experiments were run on a single core of a 2.6 GHz Intel i7 processor. Parameters were chosen as follows. The maximum iteration count was set to $N = 50$ to avoid outliers that spend many iterations near convergence. The inner optimization step count was set to $S = 1$ because the minimum distance oracle is several times faster than the QP solver. The constraint deletion threshold was set to $\gamma = 0.1$, but this was not tuned carefully; it appears that performance is not terribly sensitive to this parameter. The convergence tolerance is set to $\|\Delta x\| \leq 10^{-4}\sqrt{n}$.

Performance characteristics

This set of experiments characterize the performance of the algorithm in static pose optimization. First, we consider the cube-sphere collision scenario of Fig. 6. The cube has dimension 0.5 m on each axis and the sphere has radius 1 m. The cube is represented as a signed distance field with 2 cm resolution and the sphere is discretized into a point set of



Method	Time (ms)	Obj (m)	Pen (mm)	% Pen
SIP	35.7	0.102	**0**	**0**
MP	**30.5**	**0.0998**	4.0	18
NLP-1k	1,281	0.119	0.52	4
SIP-100k	65.8	0.112	**0**	**0**

Figure 6. The sphere-cube test scenario. Targets are chosen along a straight line trajectory, and for each run the optimization minimizes the distance between the cube center and the target. Test results include average computation time (Time), optimized objective function value (Obj), penetration depth (Pen), and fraction penetrating (% Pen).

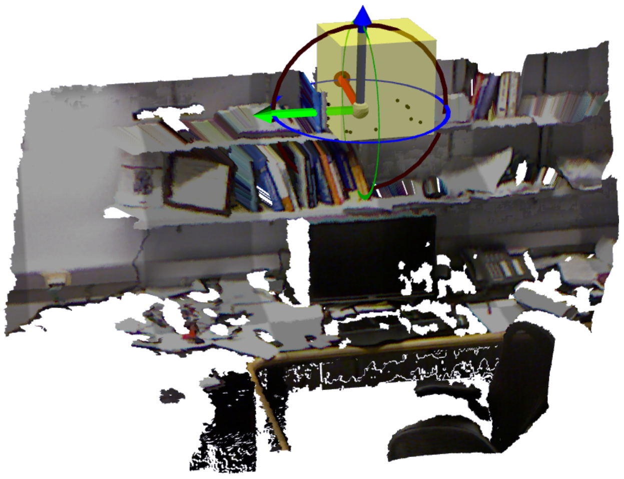


Figure 7. Optimizing a geometry to be as close as possible to the given widget while avoiding contact with the point cloud. The 640×480 point cloud is obtained from the Cornell RGB-D scene dataset and contains 228,352 valid points.

10,298 points with 5 cm resolution. No inner spheres are added for these tests. For each run, 50 targets along the given trajectory are sampled on a uniform grid and the cube is initialized with its center at that position. At all positions, the cube penetrates the sphere. The optimization is used to minimize the distance between the object's center and the target while avoiding collision.

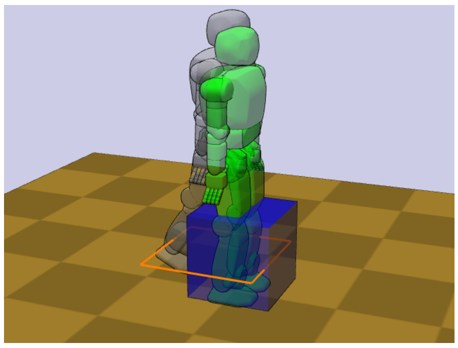
We compare SIP to a standard NLP formulation with max-penetration constraints (2) (MP). Since these objects are convex, we can also formulate them as convex polytopes. We use the exact penetration depth computation of the GJK algorithm (Gilbert et al. 1988) to implement the MP constraint (using the libccd library, written in C). We also compare an NLP that instantiates no-penetration constraints between the SDF and 1,000 points sampled from the sphere (NLP-1k). Results show that MP has the lowest computation time, but SIP is not far behind. MP and SIP obtain similar objective function values. NLP-1k is 40x slower due to the large number of instantiated constraints. Certainly, if all 10,000+ points were included, a standard NLP would be even more expensive.

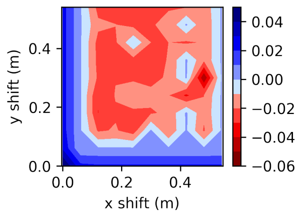
The most severe weakness of MP is that it often fails to satisfy constraints. In fact, 18% of its runs terminated with some penetration, with an average of 4.0 mm penetration. This is due to oscillation between penetrating and non-penetrating conditions, since MP only considers the effect of one support point at each iteration. NLP-1k does somewhat better, but also fails to detect some collisions due to the limited number of instantiated points.

We also tested a version of SIP where the sphere was discretized even more finely to obtain 108,200 points (SIP-100k). Despite the number of points increasing by an order of magnitude, the added resolution only adds about 40% more computation time. This is because collision detection costs scale sub-linearly, and the number of constraints instantiated still remains small, with SIP-100k instantiating 7.3 constraints and SIP instantiating 2.3 constraints per problem, on average.

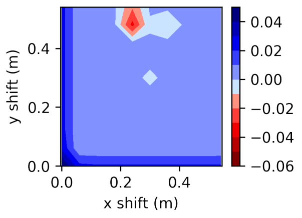
To illustrate the capabilities of our algorithm in handling complex geometries, Fig. 7 shows the same cube in a point cloud scan of an office environment from the Cornell RGB-D Scene Understanding dataset (Koppula et al. 2011). The 640×480 RGB-D image contains 307,200 points, 228,352 of which are valid. Precomputation of the point cloud into an octree took 198 ms, although this could be sped up further if the structured nature of the point cloud were taken into account. A similar test to the test above was run, with the target location moving horizontally across the bookshelf, with some penetration at each location. On average, SIP instantiates 7.8 constraints and terminates in 77.5 ms.

To study the performance of the inner sphere method, we tested a set of scenarios with varying initial penetrations. Fig. 8.a illustrates the problem setup with the Hubo-II+ robot model and a cube obstacle. The center of the cube is varied over a 10×10 grid in the indicated square. For each cube position, the pose of the robot is optimized to obtain a collision-free configuration while minimizing deviation from the initial pose. The optimizer is run to at most 50


(a) Test scenario



(b) Surface points only



(c) With inner spheres

Figure 8. Evaluating the inner sphere method in deeply-penetrating scenarios with a model of a humanoid robot. The cube location is varied across the square indicated in (a), and the configuration is optimized to minimize the distance to the vertical standing configuration (green). The solution is drawn in grey. Distances of the resulting optimized solutions without and with inner spheres are plotted in (b) and (c), respectively (negative numbers indicate penetration).

iterations. Fig. 8.b shows the resulting constraint residual using only surface points to represent to cube. Very few of the deeply-penetrating initial scenarios are successfully solved to a feasible point. Fig. 8.c shows that the use of inner spheres allow most of the scenarios to be solved to completion. It should be noted that inner spheres give the best performance benefits for fat objects, while thin objects do not benefit much, if at all.

Settling objects into a pile

An application of the semi-infinite optimization approach is to generate piles of objects in stable configurations. There are several reasons for doing so, such as pose estimation in cluttered scenes (Mitash et al. 2017), packing (Wang and Hauser 2019), and generation of cluttered scenes for training data for robot learning (Sui et al. 2017). A common approach to generating such scenes is physics simulation of objects falling from a collision-free initial guess, but this has two drawbacks. First, most off-the-shelf rigid body simulators are slow and not robust when handling contact between non-convex objects. Second, physics simulators act on object velocity, so to obtain a static pile configuration it is necessary to either wait for a long time, or to artificially dampen the simulation.

Table 1. Running times for settling object piles

Problem	# Objects	SIP (s)	Sim (s)	Sim all (s)
APC+YCBa	8	4.0	79.0	380.6
APC+YCBb	8	3.8	115.3	473.7
APC+YCBc	8	6.4	>500s	>500s
APC+YCBd	12	9.8	112	443.2
PSB Chess	20	6.5	54.7	>500s

In comparison, our method can be used to generate (relatively) stable piles quickly. It handles non-convex objects and is faster than physics simulation because it iterates the pile configuration in quasi-static fashion. We minimize the height of each object, one-by-one, subject to semi-infinite collision constraints between the environment and all previous objects. We note that these piles are not truly stable under gravity because contact forces are not simulated. But, it should be noted that Lagrange multipliers calculated inside the SQP solvers are directly analogous to the contact forces for frictionless contacts.

In these experiments we use a maximum of 50 optimization iterations. The Table 1 describes the experimental results, when objects are asked to be placed into a 40 cm $\times$ 40 cm $\times$ 20 cm box. In the APC+YCB rows, the object sets are drawn at random from the YCB (Calli et al. 2017) and APC (Rennie et al. 2016) object datasets. Figs. 9 and 10 illustrate two of these examples. The meshes in these examples contain 22,000-28,000 triangles on average. In the PSB Chess row, the items are drawn from chess pieces from the Princeton Shape Benchmark (PSB), and have 1,500 triangles on average. This scenario is illustrated in Fig. 11. The SIP column gives computation times for the method described above, using a maximum of 50 optimization iterations per item. No inner spheres are added for these tests, since most meshes are not watertight, making it difficult to determine inside and outside points. For comparison, we use the Klamp't physics simulator, which is able to handle non-convex objects. The Sim column gives running times when the Klamp't settle subroutine is used, which performs physics simulations on one object at a time, dropped from a starting point immediately above the prior highest object, while imposing a 20% viscous drag on the simulated object.

Trajectory optimization

Experiments illustrated in Figs. 12 and 13 test the ability of SIP to handle non-convex geometries in robot trajectory optimization. In each case, the start and end of the trajectory were fixed while 10 intermediate milestones were optimized. The robot model here is the 6DOF Staübli TX90L industrial

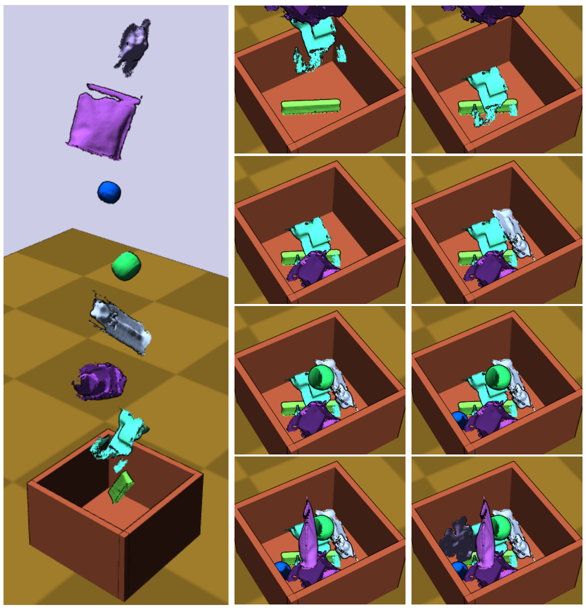


Figure 9. Left: settling 8 objects in the APC+YCBa problem. Right: each object is settled using a call to the semi-infinite programming solver.

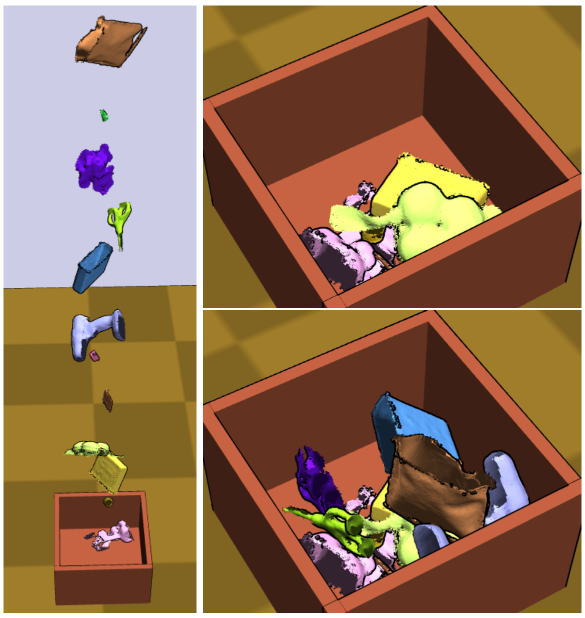
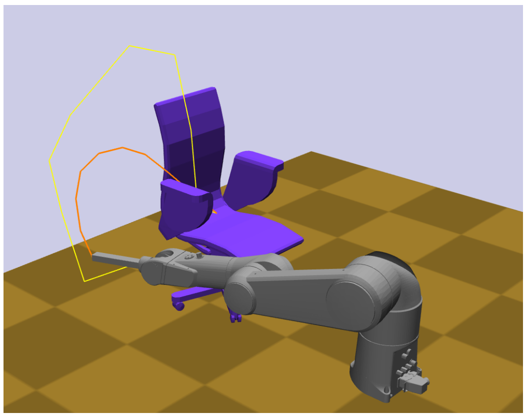
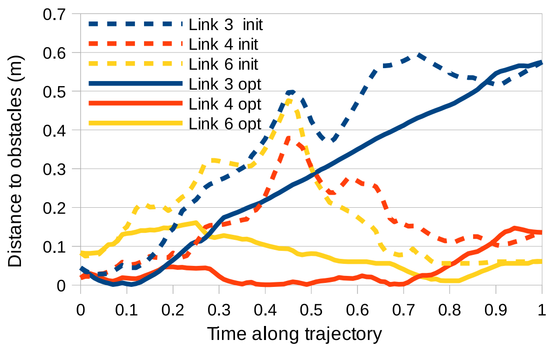


Figure 10. Left: settling 12 objects in the APC+YCBd problem. Right: two frames from the settling sequence.

Figure 11. 20 objects settled in the PSB Chess example.



(a) Initial end effector trajectory (yellow) and optimized trajectory (orange)

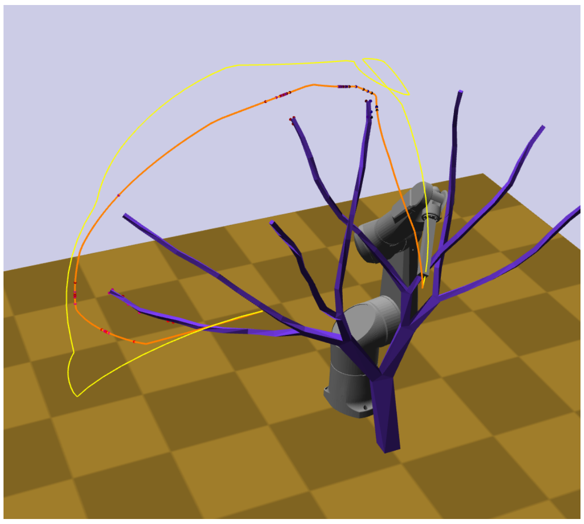


(b) Link-environment distances

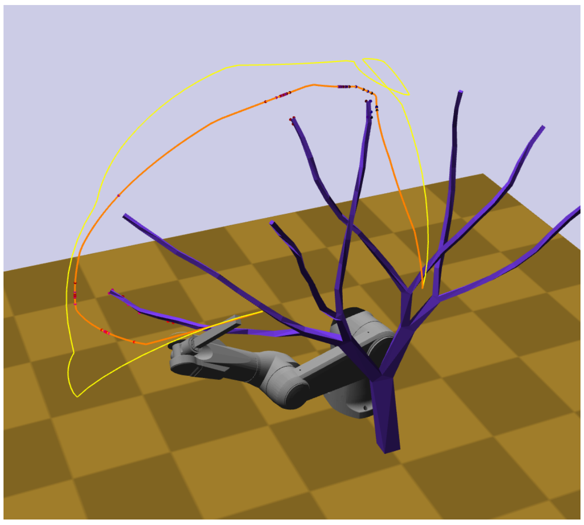
Figure 12. (a) Optimizing a robot trajectory in close proximity to an office chair obstacle. (b) Distances to the environment for the 3 links supporting the optimized trajectory.

manipulator, with a pipe attached to its terminal link. In both cases, objects from the Princeton Shape Benchmark (Shilane et al. 2004) are instantiated near the robot, and the optimizer minimizes the sum of squared distances between milestones. The robot link geometries are converted to SDFs with 2 cm resolution, and the obstacle is represented as a surface point cloud with 2 cm resolution. This supersampling is performed by adaptively subdividing the longest edge of the obstacle mesh until the maximum edge length is less than 2 cm. A suboptimal collision-free trajectory is given as the initial seed.

Fig. 12 shows the optimized path for the chair obstacle, in which the end effector rises from under the seat and slides between the chair back and the armrest. The plot shows robot-obstacle distances along the original and optimized path for each of the limiting links. This demonstrates that the trajectory is first limited by link 3 (the “elbow”) which passes very close to the underside of the seat. Next, link 4 (the “forearm”) passes around the armrest. Finally, link 6 (the end effector) slides along the seat back and chair. This path was computed in 7.26 s, and it should be noted that running time is heavily dependent on the desired level of convergence.



(a) Start configuration

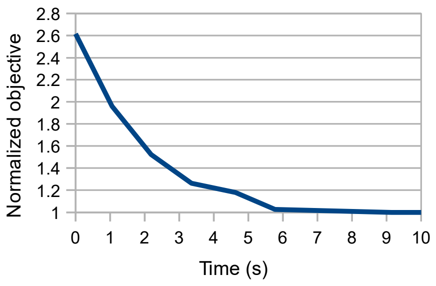


(b) Goal configuration

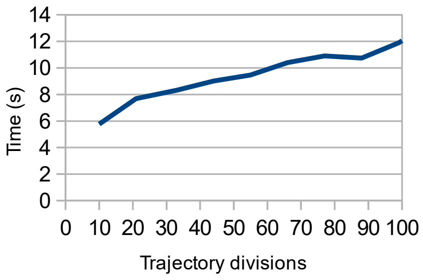
Figure 13. Optimizing a robot trajectory in close proximity to a tree-shaped obstacle. The yellow curve is the initial end effector trajectory, and the orange one is the optimized trajectory. Dots along the trajectory and on the obstacle indicate the instantiated constraint points.

This example exhausts the maximum of 50 iterations, with the bulk of iterations performing "fine-tuning" with step magnitude < 0.01. Terminating at 20 iterations would have terminated in 2.23 s but sacrifices only 10% of optimality.

Fig. 13 shows the optimized path for the tree obstacle. The robot must extract itself between two branches, and then pass underneath the lower-left branch. Here, collision with the end effector is the primary limiting constraint. Total computation time is 5.77 s, and the objective function value is still somewhat improving after 50 iterations. Fig. 14 plots computation time against the objective function extending to 100 iterations, showing convergence at around 7 s. Another issue to examine is the number of milestones used in the path representation. As the number of milestones increases, the path becomes slightly more optimized. Running times are



(a) Convergence of trajectory optimization



(b) Scalability of trajectory optimization

Figure 14. Numerical experiments on the example of Fig. 13. (a) Optimization converges within a few seconds to a near-optimal solution. (b) Running times are roughly linear in the number of milestones in the trajectory representation, and the linear coefficient is small.

shown in Fig. 14, showing a roughly linear and relatively shallow relationship between milestone count and running time.

Optimal Trajectory Planning

To evaluate SIP as a postprocessor for sampling-based motion planning, we compare 8 different asymptotically-optimal planner variants:

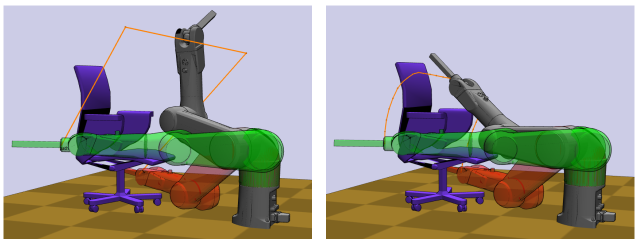
1. RRT*: The RRT* algorithm of Karaman and Frazzoli (2011).
2. Lazy-RRG*: A "lazy" version of RRT* that delays edge collision checks (Hauser 2015).
3. rSBL: Repeated restarts of the SBL planner (Sánchez and Latombe 2002), keeping the best path found.
4. rsSBL: Repeated restarts of SBL, followed by shortcutting Luna et al. (2013).
5. X+Opt: Hybrid of sampling-based planner X, followed by optimization via SIP.

In each variant, path length is used as the objective function. For the X+Opt variants the parameters $t_{plan} = 3\,\text{s}$ and $N_{opt} = 25$.

Fig. 15 illustrates example results on the environment of Fig. 12. (Note that the start and goal configurations are different from those illustrated in Fig. 12.) Because RRT* and Lazy-RRG* are asymptotically-optimal planners, the

Plan() step in RRT*+Opt and Lazy-RRG*+Opt continues refining the prior roadmap rather than starting from scratch. In rSBL+Opt the planning phase is terminated after the first SBL path found, while in rsSBL+Opt the full t_{plan} is devoted to SBL followed by shortcutting. After $t_{max} = 30\,\text{s}$ of computation, the results from our method are clearly closer to optimal compared to RRT*. Further details are plotted in Fig. 16. Here, 10 trials of each algorithm were conducted with different random seeds, and the cost of the best path found at each point in time is recorded. These results show that RRT*, Lazy-RRG*, and rSBL lead to quite suboptimal paths. This observation is consistent with prior studies that show that purely sampling-based methods do not converge quickly in high-dimensional spaces. Using hybrid planning makes the cost converge far more quickly and consistently. Lazy-RRG* and rSBL produce good initial paths for the optimizer, resulting in extremely low variance after optimization. rsSBL is a fairly good baseline planner that outperforms the three other sampling-based methods, but the use of shortcutting still does not converge as quickly to an optimum as rsSBL+Opt. Note that rsSBL+Opt seems to converge somewhat slower than Lazy-RRG*+Opt and rSBL+Opt; this is likely due to the large number of waypoints introduced by shortcutting leading to optimization problems of higher dimension, which slows down optimization. Qualitatively, it is apparent that the variance of the paths planned using optimization is much lower than other methods, as illustrated in Fig. 17.

It should be noted that the added expense of local optimization may not be worthwhile for problems with many local optima until it is reasonably certain that repeated restarts of a planner have found a seed path in an appropriate basin of attraction. This is because any effort spent optimizing paths that end up as local optima is wasted. Fig. 18 illustrates such a problem with the UR5 robot in close proximity to a human model. Not only are local minima prevalent, but many locally optimal paths share similar cost. Here, it is more fruitful to run a planner many times before beginning optimization on the best path found. In the rsSBL+Opt 2 case (Fig. 18.c), rsSBL is run for 30 s with $t_{plan} = 3$ (10 restarts), and then the optimizer is run afterwards on the shortest path. We find that this approach produces more consistent paths than standard rsSBL+Opt.



(a) RRT* **(b)** Hybrid sampling / trajectory optimization (rsSBL+Opt)

Figure 15. Planning collision-free paths for an industrial robot between two configurations, shown in green and red. Paths shown are the best paths found after 30 s of computation.

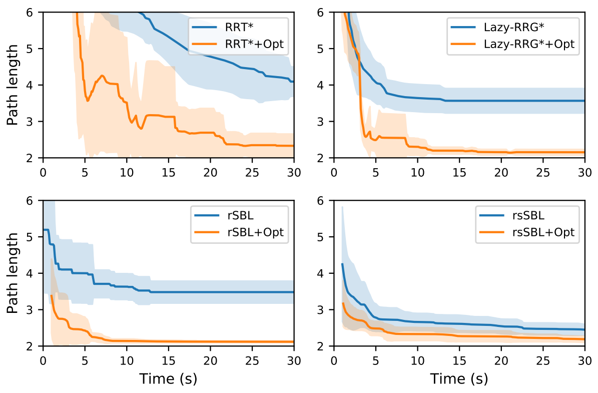
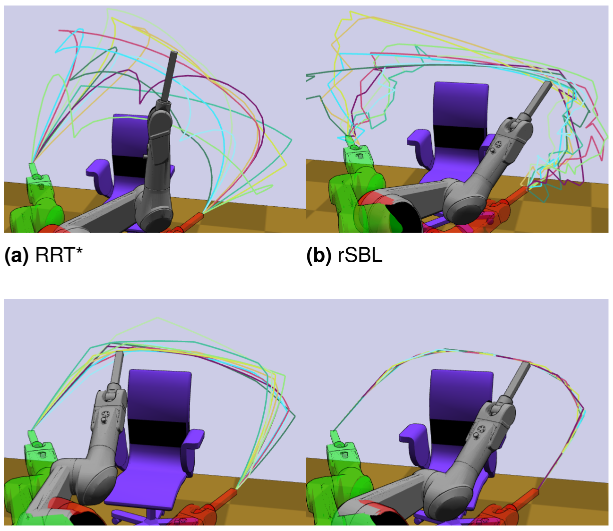


Figure 16. Comparison of any-time path planning with various techniques. Solid lines indicate mean path length over 10 runs; transparent regions indicate standard deviation. (Best viewed in color)



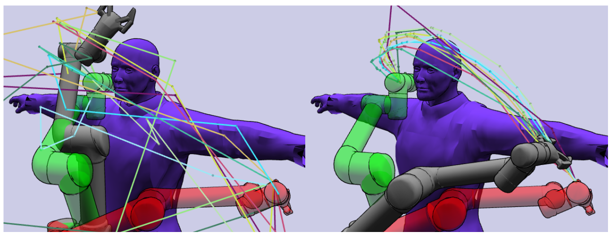
(a) RRT* **(b)** rSBL

(c) rsSBL **(d)** rsSBL+Opt

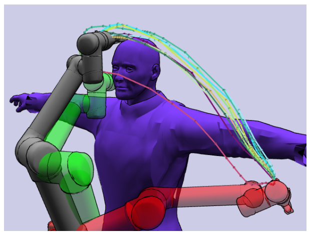
Figure 17. Close up generated paths after 30 s of computation from selected planners in Fig. 16. 10 paths from each planner are overlaid and drawn in a distinct color. Using SIP as an optimizer leads to substantially more consistent paths.

Conclusion

This paper presents a novel optimization method to handle nonpenetration constraints with highly non-convex and geometric complex objects. A semi-infinite programming approach combined with an efficient deepest-penetration oracle allows it to be applied to robot pose and trajectory optimization with models composed of hundreds of



(a) RRT* **(b)** rsSBL



(c) rsSBL+Opt

Figure 18. Overlay of 10 paths after 60 s of computation from selected planners on an example of a robot in close proximity to a human-like mannequin. The start and goal configurations are denoted in green and red, respectively.

thousands of primitives. Experiments demonstrate that the approach rapidly converges in challenging scenarios. Code for the algorithms can be found at https://github.com/krishauser/SemiInfiniteOptimization.

Future research should investigate several issues that may further improve the performance of this framework. Especially in deeply-penetrating cases, better techniques for penetration depth and direction estimation should help avoid the problem of local minima. For example, mesh models provided for many robots are not watertight and/or contain irrelevant internal structures, which leads to poor quality signed distance fields (SDFs) as calculated by the fast marching method. Better results may be obtained using approximate methods that explicitly solve for a smooth implicit surface that interpolates the input mesh (Calakli and Taubin 2011), but in practice it appears that manual inspection of SDF quality is still currently necessary to verify that the reconstruction is artifact-free.

Other directions of interest include extending the basic SIP exchange method framework. Performance improvements may be obtained by moving an instantiated constraint continuously, such as moving the time at which a collision constraint is met along a path, rather than adding new discrete constraints all having similar values. We are also interested in the possibility of dynamic instantiation of optimization variables, such as refining the path discretization during optimization. It may also be possible to instantiate contact forces at each instantiated index parameter, which could be related to the system dynamics via constraints.

Acknowledgment

This work is partially supported by NSF grants #1253553, #1527826, and #1911087.

References

Bobrow JE (1988) Optimal robot plant planning using the minimum-time criterion. *IEEE Journal on Robotics and Automation* 4(4): 443–450.

Calakli F and Taubin G (2011) Ssd: Smooth signed distance surface reconstruction. In: *Computer Graphics Forum*, volume 30. Wiley Online Library, pp. 1993–2002.

Calli B, Singh A, Bruce J, Walsman A, Konolige K, Srinivasa S, Abbeel P and Dollar AM (2017) Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research* 36(3): 261 —- 268.

Campi MC, Garatti S and Prandini M (2009) The scenario approach for systems and control design. *Annual Reviews in Control* 33(2): 149–157.

Chakraborty N, Peng J, Akella S and Mitchell JE (2008) Proximity queries between convex objects: An interior point approach for implicit surfaces. *IEEE Transactions on Robotics* 24(1): 211–220.

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

Gottschalk S, Lin MC and Manocha D (1996) Obbtree: A hierarchical structure for rapid interference detection. In: *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, pp. 171–180.

Hauser K (2015) Lazy collision checking in asymptotically-optimal motion planning. In: *IEEE International Conference on Robotics and Automation*.

Hauser K (2018) Semi-infinite programming for trajectory optimization with nonconvex obstacles. In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

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

Kim D and Yoon SE (2020) Simultaneous planning of sampling and optimization: study on lazy evaluation and configuration free space approximation for optimal motion planning algorithm. *Autonomous Robots* 44(2): 165–181.

Koppula HS, Anand A, Joachims T and Saxena A (2011) Semantic labeling of 3d point clouds for indoor scenes. In: *Neural Information Processing Systems*.

Kuntz A, Bowen C and Alterovitz R (2017) Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization. In: *International Symposium on Robotics Research*.

Lee Y, Lengagne S, Kheddar A and Kim YJ (2012) Accurate evaluation of a distance function for optimization-based motion planning. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1513–1518.

Lengagne S, Vaillant J, Yoshida E and Kheddar A (2013) Generation of whole-body optimal dynamic multi-contact motions. *The International Journal of Robotics Research* 32(9-10): 1104–1119.

Luna R, Sucan I, Moll M and Kavraki L (2013) Anytime solution optimization for sampling-based motion planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*. pp. 5068—-5074.

Luo J and Hauser K (2014) An empirical study of optimal motion planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Mitash C, Bekris KE and Boularias A (2017) A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 545–551.

Reemtsen R and Görner S (1998) Numerical methods for semi-infinite programming: a survey. In: *Semi-infinite programming*. Springer, pp. 195–275.

Rennie C, Shome R, Bekris KE and De Souza AF (2016) A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters* 1(2): 1179–1185.

Richards A, Schouwenaars T, How JP and Feron E (2002) Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics* 25(4): 755–764.

Sánchez G and Latombe JC (2002) On delaying collision checking in PRM planning: Application to multi-robot coordination. *The International Journal of Robotics Research* 21(1): 5–26.

Saramago SF and Junior VS (2000) Optimal trajectory planning of robot manipulators in the presence of moving obstacles. *Mechanism and Machine Theory* 35(8): 1079–1094.

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.

Shilane P, Min P, Kazhdan M and Funkhouser T (2004) The princeton shape benchmark. In: *Shape Modeling International*. Genova, Italy.

Shiller Z and Dubowsky S (1989) Robot path planning with obstacles, actuator, gripper, and payload constraints. *The International Journal of Robotics Research* 8(6): 3–18.

Stellato B, Banjac G, Goulart P, Bemporad A and Boyd S (2017) OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints* .

Sui Z, Xiang L, Jenkins OC and Desingh K (2017) Goal-directed robot manipulation through axiomatic scene estimation. *The International Journal of Robotics Research* 36(1): 86–104.

Vaz AIF, Fernandes EM and Gomes MPS (2004) Robot trajectory planning with semi-infinite programming. *European Journal of Operational Research* 153(3): 607–617.

Wang F and Hauser K (2019) Stable bin packing of non-convex 3d objects with a robot manipulator. In: *IEEE International Conference on Robotics and Automation*.

Weller R and Zachmann G (2009) A unified approach for physically-based simulations and haptic rendering. In: *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*. pp. 151–159.

Zeng B and Zhao L (2013) Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters* 41(5): 457–461.

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013) Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9-10): 1164–1193.