Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization

John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow and Pieter Abbeel

***Abstract*—We present a novel approach for incorporating collision avoidance into trajectory optimization as a method of solving robotic motion planning problems. 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 and enables the algorithm to reliably solve motion planning problems, including problems involving thin and complex obstacles.**

We benchmarked our algorithm against several other motion planning algorithms, solving a suite of 7-degree-of-freedom (DOF) arm-planning problems and 18-DOF full-body planning problems. We compared against sampling-based planners from OMPL, and we also compared to CHOMP, a leading approach for trajectory optimization. Our algorithm was faster than the alternatives, solved more problems, and yielded higher quality paths.

Experimental evaluation on the following additional problem types also confirmed the speed and effectiveness of our approach: (i) Planning foot placements with 34 degrees of freedom (28 joints + 6 DOF pose) of the Atlas humanoid robot as it maintains static stability and has to negotiate environmental constraints. (ii) Industrial box picking. (iii) Real-world motion planning for the PR2 that requires considering all degrees of freedom at the same time.

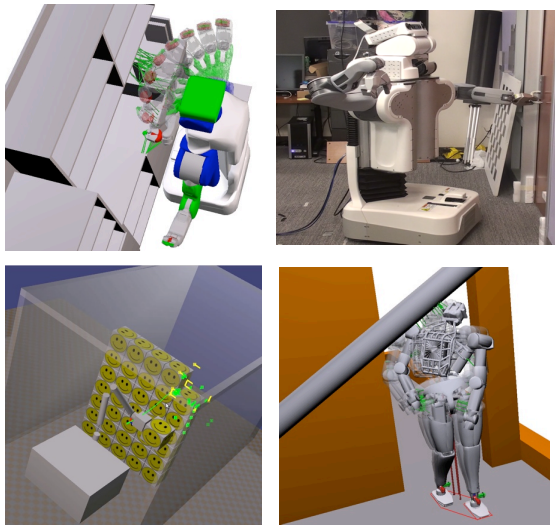

Fig. 1. Several problem settings were we have used our algorithm for motion planning. Top left: planning an arm trajectory for the PR2 in simulation, in a benchmark problem. Top right: PR2 opening a door with a full-body motion. Bottom left: industrial robot picking boxes, obeying an orientation constraint on the end effector. Bottom right: humanoid robot model (DRC/Atlas) ducking underneath an obstacle while obeying static stability constraints.

I. Introduction

Trajectory optimization algorithms have two roles in robotic motion planning. First, they can be used to smooth and shorten trajectories generated by some other method. Second, they can be used to plan from scratch: one initializes with a trajectory that contains collisions and perhaps violates constraints, and one hopes that the optimization converges to a high-quality trajectory satisfying constraints. Using an optimization algorithm to plan from scratch is an especially attractive option in problems with many degrees of freedom (DOF), since the computation time scales favorably with the number of DOF.

Two of the key ingredients in trajectory optimization for motion planning are (1) the numerical optimization method, and (2) the method of checking for collisions and penalizing them. For numerical optimization, we use sequential convex optimization, with ℓ_1 penalties for equality and inequality constraints. This approach involves solving a series of convex optimization problems that approximate the cost and constraints of the true problem, which is non-convex. For collisions, we compute signed distances using convex-convex collision detection, and we ensure the continuous-time safety of a trajectory by considering the swept-out volume. These two aspects of our approach are complementary, since our collision checking method yields a polyhedral approximation of the free part of configuration space, which can be directly incorporated into the convex optimization problem that is solved at each iteration of the optimization.

The first advantage of our approach is speed. Our implementation solves typical arm planning problems in around $100 - 200\,\mathrm{ms}$ and solves problems involving many more degrees of freedom in under a second. This is largely enabled by our novel formulation of the the collision penalty, which guarantees safety in continuous time by considering swept-out volumes. This cost formulation has little computational overhead in collision checking and allows us to use a sparsely sampled trajectory. The second advantage of our approach is its reliability—it solves a surprisingly large fraction of planning problems. In our experiments, our algorithm solved a larger fraction of problems than any of the sampling-based planners, which were given a ten second time limit. The third advantage of our approach regards path quality: once the trajectory is free of collisions, our approach will treat collision avoidance as a hard constraint (i.e., keep a certain safe distance from obstacles.) Our algorithm will converge to a locally optimal solution subject to this constraint, without compromising the other objective criteria. The fourth advantage of our approach is flexibility: new constraints and cost

terms can easily be added to the problem since the underlying numerical optimization method is numerically robust, and it can deal with initializations that are deeply infeasible.

We performed a quantitative comparison between our algorithm and several open-source implementations of motion planning algorithms, including sampling based planners from OMPL [**?**], as well as a recent implementation of CHOMP. Overall, our algorithm was not only faster than the alternatives, but it solved a larger fraction of the problems. (All planners were given a ten second time limit.)

We have successfully used our algorithm on high-DOF problems in the real world involving a mobile robot (PR2) and sensor data. We solve problems where we simultaneously need to plan for two arms along with the base and torso.

We also validated our approach on a very high-DOF problem: planning foot placements with 28 degrees of freedom (+ 6DOF pose) of the Atlas humanoid robot as it maintains static stability and avoid collisions.

Videos corresponding to the results described here are available at the website [2].

II. Related work

Optimizing over trajectories is one of the fundamental ideas of optimal control, especially the direct methods, which solve for a sequence of states and controls [3]. In the domain of robotics, the set of collision-free configurations is highly non-convex, and collision checking is usually computationally expensive, making trajectory optimization challenging. Khatib proposed the use of potential fields for avoiding obstacles, including moving ones [12]. Warren [27] suggested using a global potential field to push the robot away from configuration-space obstacles, starting with a trajectory that contained collisions. Quinlan and Khatib [21] suggested locally approximating 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 [4] improved on this idea, enabling trajectory optimization on paths of a robot in 3D, by using the Jacobian to map distances from task space into configuration space.

CHOMP (Covariant Hamiltonian Optimization for Motion Planning) is a set of ideas for how to formulate the objective in robotic motion planning problems and how to perform the numerical optimization [22, 28**?**]. The most notable features of their approach are (1) using trajectory costs that are invariant to time parameterization of the trajectory, (2) using pre-computed signed distance fields for collision checking, (3) using preconditioned gradient descent for numerical optimization, with projections to enforce constraints. While the motivation for the presented work is very similar to the motivation behind CHOMP (and, indeed, CHOMP is the most closely related prior art), our algorithm differs fundamentally in many ways, and we will discuss the relative merits between our approach and CHOMP in the Discussion section.

Other recent work on robot trajectory optimization for (typically) kinematic motion planning includes STOMP (Stochastic Trajectory Optimization for Motion Planning), which uses a gradient-free, stochastic scheme for optimization [11]; and ITOMP (Incremental trajectory optimization for real-time replanning in dynamic environments), which deals with dynamic obstacles and real-time replanning [**?**].

Some 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. [14] incorporate signed distances between polytopes as inequality constraints in an optimal control problem. Unlike this work, they don't consider continuous-time collision checking or deal with trajectories that start deeply in collision (for which we resort to a penalty method.) Werner et al. use sequential quadratic programming to optimize walking trajectories, also incorporating obstacle avoidance as hard constraints, along with stability constraints [**?**]. Finally, there recently has been considerable progress in trajectory optimization for (complicated) dynamical systems, some of the most notable results include the ones described by Mordatch et al. [19], Tassa et al. [25], and Erez and Todorov [8].

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:

$$\text{minimize } f(\mathbf{x}) \tag{1}$$

$$\text{subject to} \tag{2}$$

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, n_{ineq} \tag{3}$$

$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, n_{eq} \tag{4}$$

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. Henceforth, we will denote the optimization variables as $\boldsymbol{\theta}_{1:T}$, where $\boldsymbol{\theta}_t$ describes the configuration at the tth timestep. To encourage minimum-length paths, we use the sum of squared displacements,

$$f(\boldsymbol{\theta}_{1:T}) = \sum_{t=1}^{T-1} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2. \tag{5}$$

In problems with dynamics, the optimization might also include joint torques and contact forces. This paper just considers kinematic problems, but the methods developed can be straightforwardly extended to find collision-free paths in problems with dynamic constraints. Besides obstacle avoidance, common inequality constraints in motion planning problems include joint limits (which are simply bound constraints on the variables), speed limits (in Cartesian space or joint space), and static stability constraints. 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). We will discuss some of these constraints in Section 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 as follows: (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, but eventually ensuring that all of the constraint violations are driven to zero. For (1), we use a trust region (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 adjusted during the optimization to ensure that the constraint violation is driven 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 [20], which is described in Algorithm 1.

In the outer loop (PenaltyIteration) we increase the penalty coefficient μ until all the constraints are satisfied, terminating when the coefficient exceeds some threshold. The next loop (ConvexifyIteration) 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) 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. (Usually we only need one iteration.)

The approach of using ℓ_1 penalties is called an exact penalty method, because if we multiply the penalty by a high enough coefficient, then the minimizer of the penalized problem is exactly equal to the minimizer of the constrained problem. This is in contrast with 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 error to zero. Note that this procedure updates the penalty coefficients in a generic way, so one does not need to tune them when setting up a new problem.

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. Using a well-known trick, we accommodate these non-smooth terms while keeping the objective quadratic by adding auxilliary (slack) variables. To add term term $|a \cdot x + b|^+$, we add slack variable

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

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

t and impose constraints

$$\begin{aligned} 0 &\leq t \\ a \cdot x + b &\leq t \end{aligned} \tag{6}$$

Clearly, 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

$$\begin{aligned} 0 \leq s, \quad 0 \leq t \\ s - t = a \cdot x + b \end{aligned} \tag{7}$$

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

IV. Discrete-time no-collisions constraint

This paragraph introduces some notation. A, B, O are 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 [9]. The distance between two sets $\mathcal{A}, \mathcal{B} \subset \mathbb{R}^3$, which is nonzero for non-intersecting sets, is defined as

$$\text{dist}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \mid (T + \mathcal{A}) \cap \mathcal{B} \neq \varnothing\} \tag{8}$$

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:

$$\text{penetration}(\mathcal{A}, \mathcal{B}) = \inf\{\|T\| \mid (T + \mathcal{A}) \cap \mathcal{B} = \varnothing\} \tag{9}$$

The signed distance is defined as follows:

$$\text{sd}(\mathcal{A}, \mathcal{B}) = \text{dist}(\mathcal{A}, \mathcal{B}) - \text{penetration}(\mathcal{A}, \mathcal{B}) \tag{10}$$

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. [9].

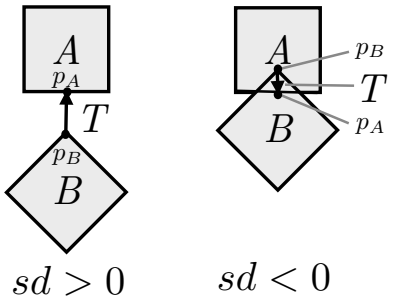

Fig. 2. Minimal translational distance and closest points.

The distance between two shapes can be calculated by the Gilbert-Johnson-Keerthi (GJK) algorithm [10]. The penetration depth is calculated by a different algorithm, the Expanding Polytope Algorithm (EPA) [26]. 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 $\mathbf{v}$:

$$s_A(\mathbf{v}) = \arg\max_{\mathbf{x} \in \mathcal{A}} \mathbf{v} \cdot \mathbf{x} \tag{11}$$

This representation makes it possible to describe shapes implicitly without constructing polyhedra or explicit representations of their surfaces. We will exploit this fact to efficiently check for collisions against swept-out volumes.

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

$$\begin{aligned} &\text{sd}(\mathcal{A}_i, \mathcal{O}_j) \geq d_{\text{safe}} \qquad \forall i \in \{1, 2, \ldots, N_{\text{links}}\}, \\ &\qquad\qquad\qquad\qquad\qquad \forall j \in \{1, 2, \ldots, N_{\text{obstacles}}\} \\ &\text{(obstacle collisions)} \\ &\text{sd}(\mathcal{A}_i, \mathcal{A}_j) \geq d_{\text{safe}} \qquad \forall i, j \in \{1, 2, \ldots, N_{\text{links}}\} \\ &\text{(self collisions)} \end{aligned} \tag{12}$$

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

$$\begin{aligned} &\sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{obs}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+ \\ &+ \sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{links}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{B}_j)|^+ \end{aligned} \tag{13}$$

A single term of this penalty function is illustrated in Figure 3. After we linearize the signed distance (described below), this cost can be incorporated into a quadratic program (or linear program) using the trick from Equation 6.

The collision penalty 13 looks prohibitively expensive to evaluate because of the double sum. However, most of the terms are zero and correspond to pairs of faraway objects, and our optimization does not explicitly represent these terms. The collision penalty (or equivalently, the constraint violations in Equation 12) can be computed by querying a collision checker for all pairs of nearby objects in the world. To locally approximate this cost around the current iterate, we query the collision checker for all pairs of objects with distance smaller than d_{check} between them, where $d_{\text{check}} > d_{\text{safe}}$. It is important for convergence that d_{check} is strictly greater than d_{safe}, so that our local approximation to the cost function includes terms for pairs of objects that are currently safely out of collision (with zero penalty). This way, the local approximation is aware of this pair of nearby objects when generating a step, i.e. solving the QP subproblem.

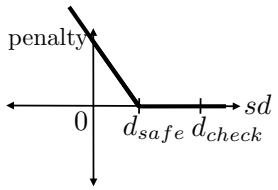

Fig. 3. Hinge penalty for collisions

We can form a linear approximation to the signed distance using the robot Jacobian and the notion of closest points. A similar calculation is performed in dynamics simulations to resolve contact constraints. 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:

$$\mathrm{sd}(\{A, F_A^w\}, \{B, F_B^w\}) = \max_{\|\hat{\mathbf{n}}\|=1} \min_{\substack{\mathbf{p}_A \in A, \\ \mathbf{p}_B \in B}} \hat{\mathbf{n}} \cdot (F_A^w \mathbf{p}^A - F_B^w \mathbf{p}^B) \tag{14}$$

The closest points $\mathbf{p}^A, \mathbf{p}^B$ and normal $\hat{\mathbf{n}}$ are defined as a triple that achieve the optimum described in (14). Equivalently, the contact normal $\hat{\mathbf{n}}$ is the direction of the minimal translation T (as defined in Equations (8) and (9)), 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. See Figure 2 for illustration.

Let's assume that the pose of A is parameterized by vector $\boldsymbol{\theta}$ (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 $\boldsymbol{\theta}$, which is necessary for dealing with self-collisions.) Then we can linearize the signed distance by assuming that the local positions $\mathbf{p}_A, \mathbf{p}_B$ are fixed, and that the normal $\mathbf{n}$ is also fixed, in Equation (14).

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

$$\mathrm{sd}_{AB}(\boldsymbol{\theta}) \approx \hat{\mathbf{n}} \cdot (F_A^w(\boldsymbol{\theta}) \mathbf{p}_A - F_B^w \mathbf{p}_B) \tag{15}$$

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

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathrm{sd}_{AB}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}_0} &\approx \hat{\mathbf{n}}^T J_{\mathbf{p}_A}(\boldsymbol{\theta}_0) \\ \mathrm{sd}_{AB}(\boldsymbol{\theta}) &\approx \mathrm{sd}_{AB}(\boldsymbol{\theta}_0) + \hat{\mathbf{n}}^T J_{\mathbf{p}_A}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \end{aligned} \tag{16}$$

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.

Note that this formula, 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.

V. Ensuring Continuous-Time Safety

The preceding section describes how to formulate a collision constraint or penalty that ensures that a given robot configuration $\boldsymbol{\theta}$ is not in collision. We can use this constraint or penalty to ensure that the robot is collision-free at each waypoint of a discretely-sampled trajectory. These waypoints will need to be converted to a continuous-time trajectory, e.g. by linear interpolation or cubic splines. However, the resulting continuous-time trajectory might have collisions between the waypoints—see Figure 4.

We can modify the collision penalty from Section IV to give a cost that enforces the continuous-time safety of the trajectory (though it makes a geometric approximation). It is only moderately more computationally expensive than the discrete-time collision cost of the previous section.

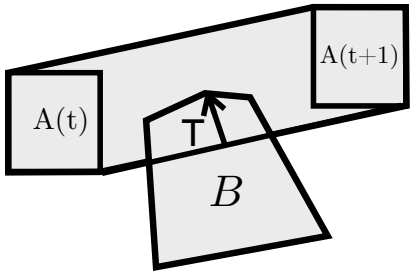

Fig. 4. Illustration of swept volume, which we use in our continuous collision cost.

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 [26]

$$\bigcup_{t \in [0,1]} \mathcal{A}(t) = \mathrm{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)) \tag{17}$$

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

$$\mathrm{sd}(\mathrm{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{B}) \tag{18}$$

It turns out that we don't have to calculate the convex hull of shapes $A(t), A(t+1)$ to perform the necessary signed distance computation, since (as noted in Section IV) the signed distance cost can be calculated using the support mappings. In particular, the support mapping is given by

$$s_{\mathrm{convhull}(C,D)}(\mathbf{v}) = \begin{cases} s_C(\mathbf{v}) & \text{if } s_C(\mathbf{v}) \cdot \mathbf{v} > s_D(\mathbf{v}) \cdot \mathbf{v} \\ s_D(\mathbf{v}) & \text{otherwise} \end{cases} \tag{19}$$

Calculating the gradient of the swept-volume collision cost is slightly more involved than discrete case described in Equations (15) and (16). Let's consider the case where object A is moving and object B is stationary, as in Figure 4. Let's suppose that A and B are polyhedral. Then the closest point $\mathbf{p}_{\mathrm{swept}} \in \mathrm{convhull}(A(t), A(t+1))$ lies in one of the faces of this polytope. $\mathrm{convhull}(A(t), A(t+1))$ 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 in the normal direction $-\hat{\mathbf{n}}$.
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
$$\alpha = \frac{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\|}{\|\mathbf{p}_1 - \mathbf{p}_{\text{swept}}\| + \|\mathbf{p}_0 - \mathbf{p}_{\text{swept}}\|} \tag{20}$$
We make the approximation
$$\mathbf{p}_{\text{swept}}(\boldsymbol{\theta}) \approx \alpha \mathbf{p}_0 + (1-\alpha)\mathbf{p}_1 \tag{21}$$
3) Calculate the Jacobians of those points
$$J_{\mathbf{p}_0}(\boldsymbol{\theta}_0^t) = \frac{d}{d\boldsymbol{\theta}^t}\mathbf{p}_0, \quad J_{\mathbf{p}_1}(\boldsymbol{\theta}_0^{t+1}) = \frac{d}{d\boldsymbol{\theta}^{t+1}}\mathbf{p}_1 \tag{22}$$
4) Similarly to Equation 16, linearize the signed distance around the trajectory variables at timesteps t and $t+1$
$$\begin{aligned} \text{sd}_{AB}(\boldsymbol{\theta}^t, \boldsymbol{\theta}^{t+1}) \approx \text{sd}_{AB}(\boldsymbol{\theta}_0^t, \boldsymbol{\theta}_0^{t+1}) \\ +\alpha \hat{\mathbf{n}}^T J_{\mathbf{p}_0}(\boldsymbol{\theta}_0^t)(\boldsymbol{\theta}^t - \boldsymbol{\theta}_0^t) \\ +(1-\alpha)\hat{\mathbf{n}}^T J_{\mathbf{p}_1}(\boldsymbol{\theta}_0^{t+1})(\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}_0^{t+1}) \end{aligned} \tag{23}$$

The preceding discussion assumed that the shapes undergo translation only. However, the robot's links also undergo rotation, so the convex hull will underestimate the swept-out volume. This phenomenon is illustrated in Figure 5. We can calculate a simple upper-bound to the swept-out volume, based on the amount of rotation. Consider a shape A undergoing translation T and rotation angle ϕ around axis $\hat{k}$ in local coordinates. Let $A(t)$ and $A(t+1)$ be the occupied space at the initial and final times, respectively. One can show that if we expand the convex hull $\text{convhull}(A(t), A(t+1))$ by $d_{arc} = r\phi^2/8$, where r is the maximum distance from a point on A to the local rotation axis, then the swept-out volume is contained inside.

In summary, we can ensure continuous time safety by ensuring that for each time interval $[t, t+1]$
$$\text{sd}(\text{convhull}(\mathcal{A}(t), \mathcal{A}(t+1)), \mathcal{O}) > d_{\text{safe}} + d_{\text{arc}} \tag{24}$$
One could relax this constraint into a penalty as described in Section IV, by approximating $\phi(\boldsymbol{\theta}^t, \boldsymbol{\theta}^{t+1})$. In practice, we ignored the correction d_{arc}, since it was well under $1\,\text{cm}$ in all of the problems we considered.

The method described in this section for continuous-time collision detection only has a modest performance penalty versus the discrete-time collision detection, where the slow-down is because 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 waypoints, reducing the computational cost of the optimization.

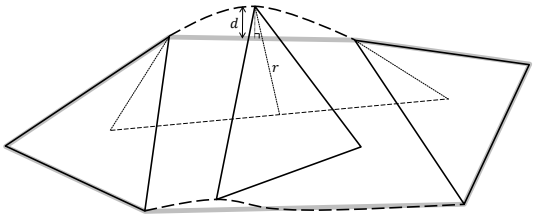

Fig. 5. Illustration of the difference between swept out shape and convex hull. The figure shows a triangle undergoing translation and uniform rotation. The swept-out area is enclosed by dotted lines, and the convex hull is shown by a thick gray line.

VI. Motion planning benchmark

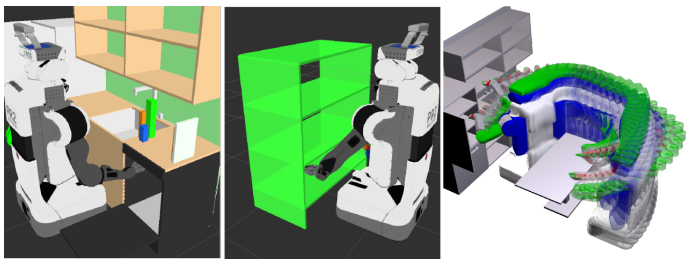

Fig. 6. Scenes in our benchmark tests. Left and center: two of the scenes used for the arm planning benchmark. Right: a third scene, showing the path found by our planner on an 18-DOF full-body planning problem.

We compared our algorithm to several other motion planning algorithms on a collection of problems in simulated environments. Our evaluation is based on four test scenes included with the MoveIt! distribution that is part of the ROS motion planning libraries [5**?**]. We used the *bookshelves*, *countertop*, *industrial*, and *tunnel* scenes for the evaluation because they were the most complex. 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. We assume that the end configuration is fixed, as opposed to some other constraint like the gripper pose.

Our tests include 198 arm planning problems and 96 full-body problems. We compared to the top-performing planning algorithms from OMPL / MoveIt. They include a bi-directional RRT [13] and a variant of KPIECE [24]. All of these algorithms were run using default parameters and post-processed by the default smoother used by MoveIt. We also compared to the latest implementation of CHOMP on the arm planning problems. This version is not yet publicly available at the time of publication, but it was made available to us by the authors [28]. We did not use CHOMP for the full-body

planning problems because we did not have the documentation or data files we would need to run these experiments on the PR2.

We tested both our algorithm and CHOMP under two conditions: single initialization and multiple initializations. For the single initialization, we used a straight line in configuration space from the start to the goal. For multiple initializations, we used the following methodology.

- For the arm planning problems, prior to performing these 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.
- For the 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.

A few more implementation details for our algorithm are given below:

- 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 IV.
- For collision checking, we took the convex hull of every mesh of the robot. Each link is made of one or more meshes. We used the Bullet collision checker [7].
- 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}. Conditions (ii) and (iii) occurred in the vast majority of these cases, indicating good convergence.
- The arm trajectories have 11 timesteps, and the full-body trajectories have 41 timesteps.

CHOMP was run for five seconds for each initialization, with Hamiltonian Monte Carlo enabled. We chose this duration because the success rate on this benchmark sharply decreased when it was run for less than 3 seconds per initialization. OMPL was limited to 30 seconds on full-body scenes.

The results for arm planning are shown in Table I. The results for full-body planning are shown in Table II. Our algorithm with multiple initializations substantially outperformed the other approaches in both sets of problems. The path lengths were normalized by dividing by the shortest path length for that problem (across all planners).

VII. Other Applications

Trajectory optimization is widely applicable to problems involving a variety of interesting constraints, including non-holonomic and dynamic constraints.

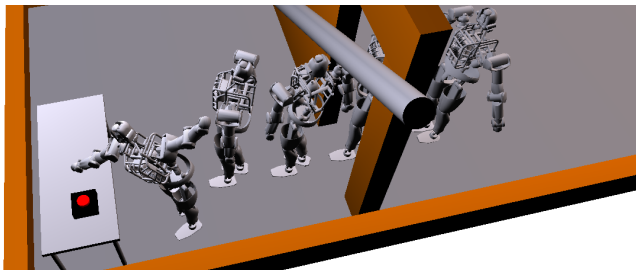

Fig. 7. Atlas robot in simulation walking across the room and pressing a button. Each footstep was planned separately. Five states out of a long motion are shown.

A. Humanoid walking: static stability

We have validated that our approach scales to a high-DOF situation; 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 (exponential-map) representation for the orientation. Walking is divided into four phases (1) left foot planted, (2) both feet planted (3) right foot planted, (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. That is, the convex support polygon is represented as an intersection of k half-planes, yielding k inequality constraints

$$a_i x_{\mathrm{cm}}(\boldsymbol{\theta}) + b_i y_{\mathrm{cm}}(\boldsymbol{\theta}) + c_i \leq 0, \ i \in \{1, 2, \ldots, k\} \qquad (25)$$

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

Using this approach, we plan a sequence of steps across a room, as shown in figure 7. Each step is planned separately using the phases described above. The robot is able to obey these stability and footstep placement constraints while ducking under an obstacle.

B. Pose constraints

Our approach can readily handle kinematic constraints, for example, the constraint that a redundant robot's gripper is at a certain pose at the end of the trajectory. A pose constraint can be formulated as follows. Let F_{targ} denote the target pose of the gripper, and let $F_{\mathrm{cur}}(\boldsymbol{\theta})$ be the current pose. Then $F_{\mathrm{targ}}^{-1} F_{\mathrm{cur}}(\boldsymbol{\theta})$ gives the pose error, measured in the frame of the target pose. This pose error can be represented as the six-dimensional error vector

$$h(\boldsymbol{\theta}) = (t_x, t_y, t_z, r_x, r_y, r_z) \qquad (26)$$

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.

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 $(v_x^w, v_y^w)(\boldsymbol{\theta})$

	Trajopt	Trajopt-Multi	ompl-RRTConnect	ompl-LBKPIECE	CHOMP-HMC	CHOMP-HMC-Multi
success fraction	0.818	0.955	0.854	0.758	0.652	0.833
average time (s)	0.191	0.3	0.615	1.3	4.91	9.27
avg normed length	1.16	1.15	1.56	1.61	2.04	1.97

TABLE I
Results on 198 arm planning problems for a PR2, involving 7 degrees of freedom. Trajopt refers to our algorithm.

	Trajopt	Trajopt-multi	OMPL-RRTConnect	OMPL-LBKPIECE
success fraction	0.729	0.875	0.406	0.51
average time (s)	2.2	6.1	20.3	18.7
avg normed length	1.06	1.05	1.54	1.51

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

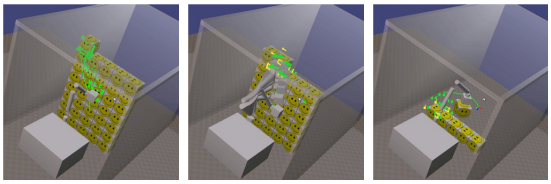

Fig. 8. Several stages of a box picking procedure, in which boxes are taken from the stack and moved to the side. The box is subject to orientation constraints.

vanishes. Here, $\mathbf{v}$ is a vector that is fixed in the box frame and should point upwards in the world frame.

Figure 8 shows our algorithm planning a series of motions that pick boxes from a stack. Our algorithm typically plans each motion in $30 - 50\,\mathrm{ms}$.

VIII. Real-world Experiments

A. Environment preprocessing

One of the main challenges in taking 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 have successfully used our algorithm with two different representations of environment geometry: (1) convex decomposition, and (2) meshes.

The process of going from a surface mesh to a union of convex shapes is called approximate convex decomposition [16]. Convex decomposition is a popular approach for simplifying geometric models for collision checking and simulation, e.g. for video games [7]. We used the HACD software of Khaled Mamou [17], which, in our experience, robustly produced good decompositions, even on the open meshes we generated from single depth images.

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 [6].

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 [1].

B. Real-world experiments

We performed several real-world experiments involving a mobile robot (PR2) to explore and validate two aspects of our approach: (1) applying it to the "dirty" geometry data that we get from 3D sensors, and (2) seeing if the full-body trajectories can be executed, in practice. Our end-to-end system successfully 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 the website for this paper [2].

IX. Discussion

While the motivation of this work is similar to CHOMP, our approach differs from CHOMP in several important dimensions, most notably that (1) we use a different approach for collision detection, and (2) we use a different numerical optimization scheme.

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 doesn't 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, which require speed and accuracy.

One important consideration for trajectory optimization is how "well-shaped" the objective is. That is, given a trajectory that contains collisions, how reliably does following the gradient get the trajectory out of collisions, rather than getting stuck in a bad local optima? 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 of the objective makes it much 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 shape achieves this result in a completely different way. We did not try scaling our collision penalty by speed as in the CHOMP objective, but that would be interesting.

2) SQP versus projected gradient descent: 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). Taking a projected gradient step is cheaper than solving a QP. However, these projected gradient steps don't incorporate much second-derivative information, and projected gradient descent has linear, rather than quadratic convergence. That said, although we use a second-order method, we don't necessarily get quadratic convergence because we don't calculate the full Hessian of all terms in the objective. Another advantage of sequential quadratic programming is that it can handle deeply infeasible initializations using penalties and merit functions, as described in Section III. We'll note that about half of the modern software for generic non-convex optimization uses an SQP variant (e.g. KNITRO, SNOPT). The remainder uses interior point methods and augmented Lagrangian methods [15].

So in both numerical optimization and collision detection, our approach requires more computation per iteration but generates a subproblem that is a better approximation of the true problem. Thus our algorithm takes a small number of iterations to converge (usually 15 or 20), but each iteration is more expensive; whereas CHOMP has the opposite attributes. (Though in practice, CHOMP iterations are somewhat expensive because of the larger number of timesteps required to make it work well.) One could potentially use a different combination of approaches: gradient descent with convex collision detection, or SQP with distance fields. However, the choices are not orthogonal, since if you choose one slow method and one fast method, you may end up with the worst of both worlds: expensive iterations and slow convergence.

X. Source code and reproducibility

All of our source code is available as a BSD-licensed open-source package called Trajopt [1]. 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 [5] 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; and (2) BPMPD [18], a free solver, which is included in our software distribution.

The benchmark results presented in this paper can be reproduced by running scripts provided at the paper's webpage [2]. Various examples, including humanoid walking and arm planning with orientation constraints, are included with our software distribution [1].

XI. Conclusion

We presented a novel algorithm that uses trajectory optimization for robotic motion planning. We benchmarked our algorithm against sampling-based planners from OMPL and CHOMP. Our algorithm was faster than the alternatives, solved a larger fraction of problems, and produced better paths. Aside from the benchmark, we validated our approach on planning stepping motions the Atlas humanoid robot, industrial box picking, and the real PR2 and its sensor data.

XII. Acknowledgements

We thank Jeff Trinkle, Sachin Patil, and Dmitry Berenson, and Nikita Kitaev 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 Sidd Srinivasa for help with CHOMP. This research has been funded in part by the Intel Science and Technology Center on Embedded Computing, by an AFOSR YIP grant, and by a Sloan Fellowship.

References

[1] Webpage for Trajopt software package. URL http://rll.berkeley.edu/trajopt.

[2] Webpage for this paper. URL http://rll.berkeley.edu/trajopt/rss.

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

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

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

[6] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

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

[8] T. Erez and E. Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *Proc. IROS*, 2012.

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

[10] E. G. Gilberg, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 1988.

[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

[12] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[13] James J Kuffner Jr and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

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

[15] S. Leyffer and A. Mahajan. Nonlinear constrained optimization: methods and software. *Argonee National Laboratory, Argonne, Illinois*, 60439, 2010.

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

[17] K Mamou and F Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *16th IEEE International Conference on Image Processing (ICIP V9)*, pages 3501–3504, 2009.

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

[19] I. Mordatch, E. Todorov, and Z. Popovic. Discovery of complex behaviors through contact-invariant optimization. In *ACM SIGGRAPH*, 2012.

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

[21] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 802–807. IEEE, 1993.

[22] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[23] I.A. Sucan, M. Kalakrishnan, and S. Chitta. Combining planning techniques for manipulation using realtime perception. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2895–2901. IEEE, 2010.

[24] Ioan A Şucan and Lydia E Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.

[25] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proc. IROS*, 2012.

[26] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[27] C.W. Warren. Global path planning using artificial potential fields. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 316–321. IEEE, 1989.

[28] 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. *International Journal of Robotics Research*, 2012.