STOCS: Simultaneous Trajectory Optimization and Contact Selection for Contact-Rich Manipulation

Mengchao Zhang[1], Devesh K. Jha[3], Arvind U. Raghunathan[3] and Kris Hauser[2]

***Abstract*— Contact-implicit trajectory optimization is an effective method to plan complex trajectories for various contact-rich systems including manipulation and locomotion. These methods formulate contact as complementarity constraints and require solving a mathematical program with complementarity constraints (MPCC). However, MPCC solve times increase steeply with the number of variables and complementarity constraints, which limits their applicability to problems with low geometric complexity. This paper introduces the simultaneous trajectory optimization and contact selection (STOCS) method that embeds the detection of salient contact points and contact times inside trajectory optimization. Because the number of active contact points is usually small, this approach minimize the number of MPCC variables and constraints, which makes solving manipulation trajectories for objects with complex, non-convex geometry computationally tractable. The proposed approach is validated on a pivoting problem in simulation and on a 6 DoF manipulator arm.**

I. Introduction

Trajectory optimization is a tool used throughout robotics to generate robot motion, but the representation of making and breaking contact remains a major research challenge. A hybrid trajectory optimization approach divides a trajectory into segments in which the set of contacts remains constant, but it requires the contact mode sequence to be known in advance [1] or explored by an auxiliary discrete search. Contact-invariant trajectory optimization (CITO) [2], [3], [4] is a more flexible approach that allows the optimizer to choose the sequence of contact by formulating a complementarity constraint, which ensures that the contact forces can be non-zero if and only if a point is in contact. Although this mathematical programming with complementarity constraint (MPCC) formulation is less restrictive than hybrid optimization, it still requires a set of predefined allowable contact points on the object. Moreover, running times rise sharply as the number of allowable points grows, which restricts its use to simple geometries.

In this paper, we propose simultaneous trajectory optimization and contact selection (STOCS), which is a novel CITO algorithm for manipulation that allows multiple changes of contact between the object and environment for a given manipulation mode. STOCS enables the application of MPCC on complex object and environment geometries by embedding the detection of salient contact points and contact times inside the trajectory optimization outer loop. Force variables are introduced for all of these contacts. Each instantiated MPCC iteration has relatively few constraints and is optimized for a handful of inner iterations before new contact points are identified. Force values are maintained from iteration to iteration to warm start the next MPCC.

This work is partially supported by NSF Grant #IIS-1911087.

[1] Mengchao Zhang is with the Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA 61801. mz17@illinois.edu

[2] Kris Hauser is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA 61801. kkhauser@illinois.edu

[3] Devesh K. Jha and Arvind U. Raghunathan are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA 02139 {jha,raghunathan}@merl.com

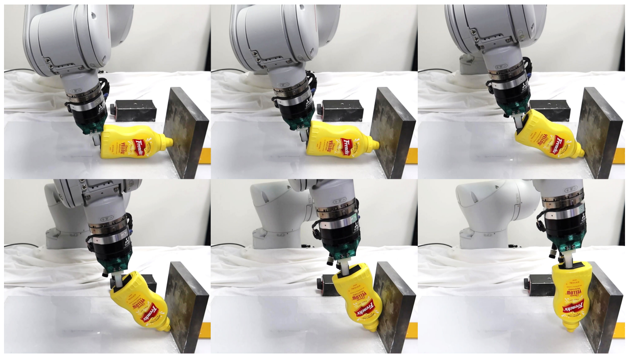


Fig. 1: A robot executes the optimized trajectory solved using STOCS for pivoting a non-convex object. [Best viewed in color.]

We apply STOCS to the problem of manipulating an object in contact with both the robot and the environment, e.g., by sliding and pivoting (Fig. 1). The planned manipulation could be used to reorient parts to allow grasping or assist in assembly by manipulating objects to a desired pose. We show that STOCS can successfully solve the manipulation problem while greatly improving computational efficiency beyond a naïve MPCC solver.

II. Simultaneous Trajectory Optimization and Contact Selection

In this section, we describe how STOCS solves contact-rich trajectory optimization problems, and a high-level illustration for the workflow of STOCS is shown in Fig. 2.

A. Semi-infinite programming (SIP), infinite programming (IP) for contact-rich trajectory optimization

To illustrate how STOCS works, we start from the SIP/IP problem that STOCS is designed to solve, and explain where the infinitely many optimization variables and constraints come from. An SIP problem is an optimization problem in finitely many variables $x \in \mathbb{R}^n$ on a feasible set described by infinitely many constraints:

$$\min_{q \in \mathbb{R}^n} \; f(q) \tag{1a}$$

$$\text{s.t. } g(q, y) \geq 0 \quad \forall y \in Y \tag{1b}$$

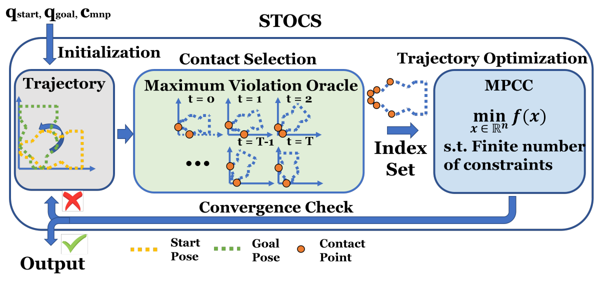


Fig. 2: Illustrating the workflow of STOCS. Given the start pose q_{start}, goal pose q_{goal}, and a manipulator contact state c_{mnp}, STOCS iterates between using the Maximum Violation Oracle to instantiate contact points, and solving a finite dimensional MPCC to decide a step direction until the convergence criteria is met. STOCS does not need a predefined contact set and can select the contact points simultaneously while solving trajectory optimization. [Best viewed in color.]

where $g(q, y) \in \mathbb{R}^m$ is the constraint function, y denotes the index parameter, and $Y \in \mathbb{R}^p$ is its domain.

For pose optimization with collision constraints, q describes the pose of the object, y is a point on the surface of the object $\mathcal{O}$, where $Y \equiv \partial\mathcal{O}$ denotes that surface of $\mathcal{O}$, and $g(\cdot,\cdot)$ is the distance from a point to the environment [5], [6]. For trajectory optimization, constraints need to be instantiated in space-time, which means $y = (t, p)$ includes both time t and contact point p on object's surface. However, to make the resulting optimization problem more stable, in this work, we assume that once a contact point is instantiated, it will be instantiated during the whole trajectory.

To include forces as part of the solution to ensure force and moment balance, we define $z : Y \rightarrow \mathbb{R}^r$ as an optimization variable [7]. Given an infinite number of contact points, infinitely many optimization variables will be instantiated, which makes the optimization become an IP problem in which the number of variables and the number of constraints are both possibly infinite [8].

To ensure that forces are only exerted at points where objects are in contact, z is required to satisfy a complementarity condition $0 \leq z(y) \perp g(q, y) \geq 0$, which ensures that the force is nonzero only if the distance between the geometries is zero. Meanwhile, there may be some other inequality constraints $h(q, y, z(y)) \geq 0$ that need to be satisfied pointwise, such as friction cone constraints. The constraint on the control input which makes sure the manipulator can only push the object rather than pull the object is denoted as $c(q, u) \geq 0$. Next, some additional constraints can be imposed on the integral of the field over the domain, such as force and torque balance.

To summarize, the constraints that need to be satisfied for each point on a trajectory are the following:

1. **Bound Constraint**:
$$q_t \in \mathcal{Q},\ u_t \in \mathcal{U},\ z_t(y) \in \mathcal{Z}\ \forall y \in Y \tag{2}$$
2. **Distance Complementarity Constraint**:
$$0 \leq z_t(y) \perp g(q_t, y) \geq 0 \quad \forall y \in Y \tag{3}$$
3. **Constraint on** $z_t(y)$:
$$h(q_t, y, z_t(y)) \geq 0 \quad \forall y \in Y \tag{4}$$
4. **Constraint on Control**:
$$c(q_t, u_t) \geq 0 \tag{5}$$
5. **Integral Constraint**:
$$\underbrace{s_{q,u}(q_t, u_t) + \int_{y \in Y} s_z(q_t, y, z_t(y))dy}_{=:s(q_t,u_t,z_t;Y)} = 0 \tag{6}$$

In order to extend this to the whole trajectory, additional constraints that could make sure the relative tangential velocity at a contact to be zero when the corresponding friction force is inside the friction cone need to be introduced ((7e) below). Hence, we formulate the following infinite programming with complementarity constraints trajectory optimization (IPCC-TO) problem denoted as $P(Y)$:

$$\min_{q,\dot{q},u,z} \ f(q, \dot{q}, u, z) \tag{7a}$$
$$\text{s.t. } q_0 = q_{start} \tag{7b}$$
$$(2), (3), (5), (6), \dot{q}_t \in \dot{\mathcal{Q}}\ \forall t \in \mathcal{T} \tag{7c}$$
$$q_t - q_{t+1} + dt\dot{q}_t = 0 \quad \forall t \in \mathcal{T} \tag{7d}$$
$$0 \leq v(q_t, \dot{q}_t, y) \perp h(q_t, y, z_t(y)) \geq 0 \quad \forall y \in Y,\ \forall t \in \mathcal{T} \tag{7e}$$

where $f(q, \dot{q}, u, z) := \sum_{t \in \mathcal{T}}[f_{q,\dot{q},u}(q_t, \dot{q}_t, u_t) + \int_{y \in Y} f_z(q_t, y, z_t(y))dy]$, dt is the time step duration, and $\mathcal{T} = \{0, \ldots, T-1\}$ with T the total number of time steps in the trajectory. For the sake of brevity, we use the notation $q = [q_0, \cdots, q_T]$, $\dot{q} = [\dot{q}_0, \cdots, \dot{q}_{T-1}]$, $u = [u_0, \cdots, u_{T-1}]$, $z = [z_0, \cdots, z_{T-1}]$. With a little abuse of notation, we use $z_t = [z_t(y)\ \forall y \in Y]$ where $z_t(\cdot)$ is the mapping and z_t is a concatenation of all the instantiated variable for all $y \in Y$.

B. Exchange method and oracle

The IPCC-TO problem not only has infinitely many constraints, but also introduces a continuous infinity of variables in z. To solve it using numerical methods, we hope that z only is non-zero at a finite number of points. We borrow this concept, which is used in the exchange method to solve SIP problems [9] to solve $P(Y)$.

The solving process can be viewed as a bi-level optimization. In the outer loop, index points are selected to be added to the index set $\tilde{Y}$, and then in the inner loop, the optimization $P(\tilde{Y})$ is solved. The outer loop will then decide how much should move toward the solution of $P(\tilde{Y})$.

Assume that there is an *Oracle* which gives us the active points $\tilde{Y} \subset Y$ when the object experiences a trajectory q, then we get a discretized version of the problem which only creates constraints and variables corresponding to a finite number of instantiated index points $\tilde{Y}$. Force variables z_t are instantiated for each index point at each of the time step. To replace integrals with sums, the true distribution $z(y)$ is represented by a set of Dirac impulses: $z_t(y) = \sum_i \delta(y - y_i)z_{t,i}$. Hence, we formulate the finite dimensional MPCC problem $P(\tilde{Y})$ in the following form:

$$\min_{q,\dot{q},u,z} \ \tilde{f}(q, \dot{q}, u, z) \tag{8a}$$
$$\text{s.t. } (2), (3), (5), \dot{q}_t \in \dot{\mathcal{Q}} \quad \forall t \in \mathcal{T} \tag{8b}$$
$$(7b), (7d), (7e) \tag{8c}$$

$$\tilde{s}(q_t, u_t, z_t; \tilde{Y}) = 0 \quad \forall t \in \mathcal{T} \tag{8d}$$

where $\tilde{f}(q, \dot{q}, u, z) := \sum_{t=0}^{T-1}[f_{q,\dot{q},u}(q_t, \dot{q}_t, u_t) + \sum_{y \in \tilde{Y}} f_z(q_t, y, z_t(y))]$, and $\tilde{s}(q_t, u_t, z_t; \tilde{Y}) = s_{q,u}(q_t, u_t) + \sum_{y \in \tilde{Y}} s_z(q_t, y, z_t(y)) = 0$.

To apply the exchange method to this IPCC-TO problem, we progressively instantiate index sets $\tilde{Y}$ and their corresponding finite-dimensional MPCCs whose solutions converge toward the true optimum [10]. Specifically, if we let $(\tilde{x}^* = [q^*, \dot{q}^*, u^*], \tilde{z}^*)$ be the optimal solution to $P(\tilde{Y})$, then as $\tilde{Y}$ grows denser, the iterates of $(\tilde{x}^*, \tilde{z}^*)$ will eventually approach an optimum of $P(Y)$.

C. Merit function for the outer iteration

After solving $P(\tilde{Y})$ for some iterations, we get a step direction from the current iterate $(\tilde{x}, \tilde{z})$ toward $(\tilde{x}^*, \tilde{z}^*)$. The IPCC-TO outer loop moves from $(\tilde{x}, \tilde{z})$ toward $(\tilde{x}^*, \tilde{z}^*)$. However, due to nonlinearity, the full step may lead to worse constraint violation. To avoid this problem, we perform a line search over the following merit function that balances reducing the objective and reducing the constraint error on the infinite dimensional problem $P(Y)$: $\phi(x, z; \mu) = f(x, z) + \mu\|b(x, z)\|_1$, where b denotes the vector of constraint violations of Problem (8). Also, in SIP for collision geometries, a serious problem is that using existing instantiated index parameters, a step may go too far into areas where the minimum of the inequality $g^*(q) \equiv \min_{y \in Y} g(q, y)$ violates the inequality, and the optimization loses reliability. So we add the max-violation $g^{*-}(x)$ to b, in which we denote the negative component of a term as $\cdot^- \equiv \min(\cdot, 0)$.

D. Convergence criteria

We denote the index set $\tilde{Y}$ instantiated at the k^{th} outer iteration as Y_k, the corresponding MPCC as $P_k = P(Y_k)$, and the solved solution as (x_k, z_k).

The convergence condition is defined as $\alpha\|[\Delta x, \Delta z]\| \leq \epsilon_x \cdot n_{xz}$ and $|z_k|^T|g(q_k, Y_k)| + |v(q_k, \dot{q}_k, Y_k)|^T|h(q_k, Y_k, z_k)| \leq \epsilon_{gap} \cdot n_{cc}$ and $|s(x_k, z_k, Y_k)| \leq \epsilon_s \cdot T$ and $\sum_t g_k^{-*}(x_{k,t}) < \epsilon_p \cdot T$, where n_{xz} is the dimension of the optimization variable and n_{cc} is the number of complementarity constraints, ϵ_x is the step size tolerance, ϵ_{gap} is the complementarity gap tolerance, ϵ_s is the balance tolerance, and ϵ_p is the penetration tolerance. With a little abuse of notation, $g(x_k, Y_k)$ is the concatenation of the function value of all the points in Y_k, and similar for $v(q_k, \dot{q}_k, Y_k)$ and $h(q_k, Y_k, z_k)$.

The overall STOCS algorithm is listed in Alg. 1.

III. Pivoting Problem Formulation

In this section, we define the pivoting problem studied in this paper, which is shown in Fig. 3. We make the following assumptions in this work: a) **Rigid body**: The object $\mathcal{O}$, environment $\mathcal{E}$, and manipulator are rigid, b) **Quasi-static**: The object is always in a quasi-static equilibrium, c) **Known geometry**: The geometry and the mass distribution of the object, and the frictional parameters between the object

Algorithm 1 STOCS

```
Require: $q_{start}$, $q_{goal}$, $c_{mnp}$
 1: $Y_0 = []$                                   ▷ Initialize empty constraint set
 2: $z_0 \leftarrow \emptyset$                    ▷ Initialize empty force vector
 3: $x_0 \leftarrow$ initialize trajectory($q_{start}$,$q_{goal}$,$c_{mnp}$)
 4: for $k = 1, \ldots, N^{max}$ do
 5:     ▷ Update constraint set and guessed forces $z_k$
 6:     Add point in $Y_{k-1}$ to $Y_k$, and initialize their forces in $z_k$
        with the corresponding values in $z_{k-1}$
 7:     Run the oracle to add more new points to $Y_k$, and initialize
        their corresponding forces in $z_k$ to 0
 8:     ▷ Solve for step direction
 9:     Set up inner optimization $P_k = P(Y_k)$
10:     Run $S$ steps of an NLP solver on $P_k$, starting from
        $x_{k-1}, z_{k-1}$
11:     if $P_k$ is infeasible then return INFEASIBLE
12:     else
13:         Set $x^*, z^*$ to its solution, and $\Delta x \leftarrow x^* - x_{k-1}$, $\Delta z \leftarrow$
            $z^* - z_{k-1}$
14:         Do backtracking line search with at most $N_{LS}^{max}$ steps
            to find optimal step size $\alpha$ such that $\phi(x_{k-1} + \alpha\Delta x, z_{k-1} +$
            $\alpha\Delta z; \mu) \leq \phi(x_{k-1}, z_{k-1}; \mu)$
15:     ▷ Update state and test for convergence
16:     $x_k \leftarrow x_{k-1} + \alpha\Delta x$, $z_k \leftarrow z_{k-1} + \alpha\Delta z$
17:     if Convergence condition is met then
            return $x_k$,$z_k$
    return NOT CONVERGED
```

and the environment μ_{env}, and between the object and the manipulator μ_{mnp}, are perfectly known, and d) **Known environment**: The geometry of the environment is perfectly known.

Geometry modeling: The object is represented as a densely sampled point cloud on its surface, and the environment is represented as a signed distance field (SDF) $\phi(x) : \mathbb{R}^2 \to \mathbb{R}$. We choose the index domain $Y = \partial\mathcal{O}$ be the surface point cloud and define $g(q, y) = \phi(T_q \cdot y)$ with T_q the object transform at configuration q.

Friction force modeling: The force variable of the i^{th} index point at time step t is $z_{t,i} = (z_{t,i}^N, z_{t,i}^+, z_{t,i}^-)$, which is divided into the normal component $z_{t,i}^N$ and frictional components $z_{t,i}^+$ and $z_{t,i}^-$ along the tangential direction of the contact surface [11]. Each of the components of $z_{t,i}$ is required to be non-negative.

Force and torque balance: We establish an integral equality constraint on the object's force and torque balance. Force balance requires that the force exerted by the manipulator and the environment on the object matches the gravity force of the object. Torque balance requires that the torque applied to the object with respect to its center of mass to be zero.

Maximum violation oracle: A maximum-violation or-

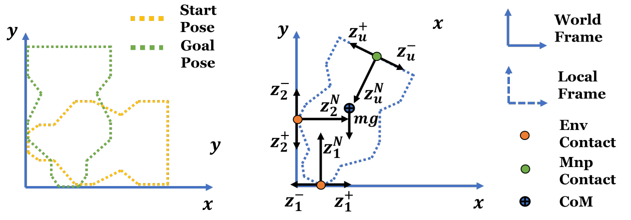
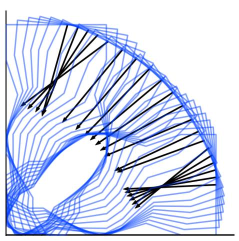


Fig. 3: Left: start and goal pose of a pivoting task. Right: free-body diagram of the object-robot-environment contact during the pivoting. [Best viewed in color.]


(a) Mustard bottle

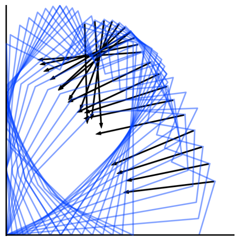

(b) Polygon

Fig. 4: Pivoting trajectories for 2 objects solved by STOCS. The object pose and the force trajectory of the contact between the object and the manipulator are plotted for each time step. [Best viewed in color.]

acle is implemented to help STOCS instantiate contact points that are part of an optimal solution. The implementation here chooses the union of the closest points $y = \arg\min_{y \in Y} g(q_t, y)$ between the object and the environment at each time step t on a trajectory $q = [q_0, \ldots, q_T]$ as the index set. So $Y_k = \cup_{t=0}^{T} \arg\min_{y \in Y} g(q_t, y)$.

Manipulation contact mode: The manipulation contact mode c_{mnp} is selected to be a sticking point contact with the object in this work. However, the formulation can be easily generalized to sliding contact mode.

IV. Experimental Results

STOCS is verified on a pivoting problem [12] with complex non-convex geometries, and the planned trajectories are executed on a physical robot.

A. Experiment Setup

We implement our method in Python using the PYROBOCOP framework [13], which uses the IPOPT solver for inner optimizations [14]. All experiments were run on a single core of a 3.6 GHz AMD Ryzen 7 processor with 64 GB RAM. Parameters used in the experiments include: a) **Physical properties**: Object mass $m = 0.1$ kg, environment friction coefficient $\mu_{env} = 0.3$, manipulator friction coefficient $\mu_{env} = 0.7$. b) **Algorithm parameters**: $N^{max} = 100$, $N^{max}_{LS} = 20$, $\epsilon_x = \epsilon_{gap} = \epsilon_s = \epsilon_p = 1e^{-4}$, $T = 20$, $dt = 0.1$ s, and $S = min(30 + 10 * k, 200)$.

B. Results of Numerical Optimization

We run STOCS on the pivoting task and all the runs successfully find an optimal solution. The experiment results are shown in Table. 1, and some solved trajectories are shown in Fig. 4.

Also, STOCS is compared with vanilla MPCC on the same tasks, and vanilla MPCC includes all points on the object's boundary as potential contacts. The comparison results are summarized in Table. 1. Note that some instantiated optimization problems' size are too large and result in an out of memory error for MPCC. We observe that STOCS can be around one to two orders of magnitude faster than MPCC, and can solve problems that MPCC cannot solve due to limitation of computational resources. STOCS selects only a small amount of points from the total number of points in the objects' representation on average, which greatly decreases the dimension of the instantiated optimization problem and reduces solve time.

Object	# Point	T	dt (s)	STOCS			MPCC
				Time (s)	Outer Iterations	Ave Active Contact Pts	Time (s)
Bolt	93			451.2	11	5.92	3002.0
Box	104			47.6	8	2.00	6672.8
Peg	104	20	0.1	223.5	13	3.23	8573.1
Mustard	247			402.2	13	4.85	OoM
Polygon	268			483.6	14	3.80	OoM

TABLE I: Numerical optimization results of STOCS and MPCC on the pivoting manipulation task. Number of points in the objects's representation (# Point), total time steps (T) and time step length (dt), solve time (Time), outer loop iteration number (Outer Iterations), and average active index points for each iteration are reported in the table. OoM means out of memory.

For simple objects such as the box, the low number of contact modes along the trajectory produces few active contact points, which makes solving fast. For complex objects such as the bolt and the mustard bottle, many changes of contacts along the trajectory results in many active contact points, which makes solving slower. For objects whose balance is hard to maintain, such as the polygon, the optimization takes more iterations to converge. (Note that pivoting the polygon with a single finger is even difficult for human.)

C. Hardware Experiments

For hardware experiments, we use a Mitsubishi Electric Assista industrial position-controller arm with a F/T sensor mounted at the wrist of the robot. We use the default stiffness controller of the robot and implement the computed force trajectory by converting desired force into relative position movement using the stiffness parameter of the controller. All the optimized trajectories were successfully executed on the robot, and one trajectory recorded during the robot experiment are shown in Fig 1.

V. Conclusion and Future Work

This paper presented STOCS, a contact-rich trajectory optimizer that embeds active contact detection into a contact-invariant trajectory optimization. We demonstrated that STOCS can solve pivoting trajectories for complex geometries both in simulation and on a real robot. Experiments show that STOCS is orders of magnitude faster than standard optimization approaches particularly when the object is represented by a large number of points.

Future work should investigate further speed gains by incorporating warm starting techniques for the inner MPCC solves. Currently, the STOCS solver uses the interior point method IPOPT [15], and warm starting interior point algorithms is generally challenging. Switching to sequential quadratic programming (e.g., [16]) will allow us to explore better warm start techniques. We also intend to investigate other strategies to initialize the trajectory optimizer, such as extracting an object's trajectory and active contact set from a demonstration video. Finally, we would like to implement time-active contact sets and active contact deletion, which would reduce the number of variables and complementarity constraints even further. This is conceptually compatible with our current implementation, but involves a great deal more bookkeeping.

References

[1] G. Schultz and K. Mombaur, “Modeling and optimal control of human-like running,” *IEEE/ASME Transactions on mechatronics*, vol. 15, no. 5, pp. 783–792, 2009.

[2] I. Mordatch, Z. Popović, and E. Todorov, “Contact-invariant optimization for hand manipulation,” in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, 2012, pp. 137–144.

[3] I. Mordatch, E. Todorov, and Z. Popović, “Discovery of complex behaviors through contact-invariant optimization,” *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[4] M. Posa, C. Cantu, and R. Tedrake, “A direct method for trajectory optimization of rigid bodies through contact,” *Int. J. Rob. Res.*, vol. 33, no. 1, pp. 69–81, 2014.

[5] M. Zhang and K. Hauser, “Non-penetration iterative closest points for single-view multi-object 6d pose estimation,” in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1520–1526.

[6] K. Hauser, “Semi-infinite programming for trajectory optimization with non-convex obstacles,” *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1106–1122, 2021.

[7] M. Zhang and K. Hauser, “Semi-infinite programming with complementarity constraints for pose optimization with pervasive contact,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6329–6335.

[8] E. J. Anderson and A. B. Philpott, *Infinite Programming: Proceedings of an International Symposium on Infinite Dimensional Linear Programming Churchill College, Cambridge, United Kingdom, September 7–10, 1984*. Springer Science & Business Media, 2012, vol. 259.

[9] M. López and G. Still, “Semi-infinite programming,” *European Journal of Operational Research*, vol. 180, no. 2, pp. 491–518, 2007.

[10] R. Reemtsen and S. Görner, “Numerical methods for semi-infinite programming: a survey,” in *Semi-Infinite Programming*. Springer, 1998, pp. 195–275.

[11] D. E. Stewart and J. C. Trinkle, “An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction,” *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.

[12] Y. Shirai, D. K. Jha, A. U. Raghunathan, and D. Romeres, “Robust pivoting: Exploiting frictional stability using bilevel optimization,” in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 992–998.

[13] A. U. Raghunathan, D. K. Jha, and D. Romeres, “Pyrobocop: Python-based robotic control & optimization package for manipulation and collision avoidance,” *arXiv preprint arXiv:2106.03220*, 2021.

[14] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[15] I. Dikin, “Iterative solution of problems of linear and quadratic programming,” in *Doklady Akademii Nauk*, vol. 174, no. 4. Russian Academy of Sciences, 1967, pp. 747–748.

[16] P. E. Gill, W. Murray, and M. A. Saunders, “Snopt: An sqp algorithm for large-scale constrained optimization,” *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.