Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

A Biconvex Method for Minimum-Time Motion Planning Through Sequences of Convex Sets

Tobia Marcucci, Mathew Halm, William Yang, Dongchan Lee, and Andrew D. Marchese
Amazon Robotics
{tobmar,mshalm,yangwilm,ldc,andymar}@amazon.com

***Abstract*—We consider the problem of designing a smooth trajectory that traverses a sequence of convex sets in minimum time, while satisfying given velocity and acceleration constraints. This problem is naturally formulated as a nonconvex program. To solve it, we propose a biconvex method that quickly produces an initial trajectory and iteratively refines it by solving two convex subproblems in alternation. This method is guaranteed to converge, returns a feasible trajectory even if stopped early, and does not require the selection of any line-search or trust-region parameter. Exhaustive experiments show that our method finds high-quality trajectories in a fraction of the time of state-of-the-art solvers for nonconvex optimization. In addition, it achieves runtimes comparable to industry-standard waypoint-based motion planners, while consistently designing lower-duration trajectories than existing optimization-based planners.**

I. Introduction

Selecting the most effective motion-planning algorithm for a robotic system often requires balancing three competing objectives: reliability, computational efficiency, and trajectory quality. Consider Sparrow, the robot arm in Fig. 1 that sorts individual products into bins before they get packaged in the Amazon warehouses. The algorithms that move Sparrow must be extremely reliable, as these robots handle millions of diverse products every day, and each failure requires expensive interventions. They must be efficient, since every millisecond spent planning is taken away from other crucial computations, and limits the robot reactivity to sensor observations. Finally, they should generate trajectories that push the robot to its physical limits, so that the work-cell throughput is maximized and the hardware is fully utilized. Unfortunately, general-purpose methods for motion planning do not excel in all of these areas at once.

Sampling-based methods like PRM [18], RRT [19], and their asymptotically optimal versions [17] can be fast enough for real-time applications. They are highly parallelizable [38] and can run on a GPU [2, 31]. They are also reliable in low-dimensional spaces, where dense sampling is computationally feasible. However, they become significantly less effective as the space dimension grows. Additionally, although their kinodynamic variants support differential constraints [20, 16, 22], sampling-based methods remain considerably less practical for designing smooth continuous trajectories than producing polygonal paths.

Trajectory-optimization methods based on nonconvex programming [1, 33] scale well to high-dimensional spaces and explicitly factor in the robot kinematics and dynamics.

Fig. 1. Sparrow robot sorting products into bins in the Amazon warehouses.

Over the years, these techniques have become significantly faster [39, 13] and, with the advent of specialized GPU implementations [35], they are now even viable for real-time motion planning. Despite these advances, the main limitation of trajectory optimization remains its reliance on local solvers, which require extensive parameter tuning, handcrafted warm starts, may suffer from inconsistent runtimes, and can even fail to find a solution. While various strategies have been proposed to address these issues [37, 15, 49, 14], trajectory optimization remains often too brittle for industrial deployment.

Recently, a new family of motion planners that combine sampling-based and trajectory-optimization methods has stemmed from [8]. First, the collision-free space is decomposed into safe convex sets. This decomposition can be computed using region-inflation algorithms [7, 6, 45, 42, 47, 46] and tailored sampling strategies [44]. For UAVs, also safe flight corridors are widely used [4, 24, 47]. Then, the continuous trajectory is optimized in conjunction with the discrete sequence of sets to be traversed. The work in [27] has shown that, for a limited class of costs and constraints, this discrete-continuous problem is solvable through a single convex program, using the framework called Graphs of Convex Sets (GCS) [29, 26]. The extensions of GCS in [30, 5] have enabled the solution of larger problems in a fraction of the time. The motion planner in [28] tackles a similar problem, but first selects a discrete sequence of safe sets using a heuristic, and later optimizes a continuous trajectory within these fixed sets. This split sacrifices optimality but preserves completeness, and

enables support for a broader range of costs and constraints.

This paper focuses on a problem similar to the one in the second phase of [28]: we seek a trajectory that traverses a sequence of convex sets in minimum time, and satisfies convex velocity and acceleration constraints. This is a purely continuous problem, but is nonconvex due to the joint optimization of the trajectory shape and timing. Our contribution is a biconvex method, which we call Sequence of Convex Sets (SCS), that solves this problem effectively. SCS starts by quickly producing a feasible trajectory. Then, it alternates between two convex subproblems. The first is obtained from the original nonconvex problem by fixing the points where the trajectory transitions from one safe set to the next. The second is derived similarly, by fixing the transition velocities.

As most multi-convex methods [34], SCS is heuristic: it typically finds high-quality trajectories quickly, but might not converge to the problem optimum, or within a given distance of it. On the other hand, SCS is complete (i.e., guaranteed to find a feasible solution). Its main algorithmic advantage is that the two convex subproblems are conservative approximations of the original nonconvex problem. This allows us to take whole steps in the direction their optima without using a line search or a trust region, as done in [28] and other trajectory-optimization methods [33, 14]. This makes the convergence of SCS fast and monotone, and eliminates any parameter tuning. Furthermore, it makes our algorithm anytime (it returns a feasible trajectory even if stopped early).

We show that SCS consistently finds high-quality trajectories in a fraction of the time of the state-of-the-art solvers `SNOPT` [11] and `IPOPT` [41]. We also demonstrate SCS on the task of transferring packages between bins using two Sparrow robots. In this task, SCS designs lower-cost trajectories than the trust-region method from [28], and achieves runtimes comparable to waypoint-based methods that are commonly used in industry.

A. Outline

This paper is organized as follows. In §II, we state our motion-planning problem and, in §III, we give a high-level overview of SCS. The details on the two convex subproblems and the initialization step are illustrated in §IV, §V, and §VI. Up to this point, we work only with infinite-dimensional trajectories. In §VII, we show how our method can be efficiently implemented on a computer by using piecewise Bézier curves as a finite-dimensional trajectory parameterization. The strengths and limitations of SCS are discussed in §VIII and §IX. In §X, we demonstrate the effectiveness of SCS through a variety of numerical experiments.

B. Notation and convexity background

In this paper, the variable i is always understood to be a positive integer. Thus, when saying for all $i \leq I$ we mean for all $i \in \{1, \dots, I\}$. Conversely, the variable k is always nonnegative, and $k \leq K$ is shorthand for $k \in \{0, \dots, K\}$.

We use calligraphic letters to represent sets, and bold letters for vectors, vector-valued functions, and matrices. We use the notation $\lambda\mathcal{S} = \{\lambda \boldsymbol{x} : \boldsymbol{x} \in \mathcal{S}\}$ to denote the product of a scalar $\lambda \in \mathbb{R}$ and a set $\mathcal{S} \subseteq \mathbb{R}^n$. We will use multiple times the fact that if a set $\mathcal{S}$ is convex then also the set $\{(\boldsymbol{x}, \lambda) : \lambda \geq 0\ \boldsymbol{x} \in \lambda\mathcal{S}\}$ is convex [3, §2.3.3]. The latter set is easily computed in practice: for instance, if $\mathcal{S}$ is a polytope of the form $\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, then the condition $\boldsymbol{x} \in \lambda\mathcal{S}$ is equivalent to $\boldsymbol{A}\boldsymbol{x} \leq \lambda\boldsymbol{b}$. A similar formula can be used for any set $\mathcal{S}$ described in standard conic form [29, Ex. 4.3].

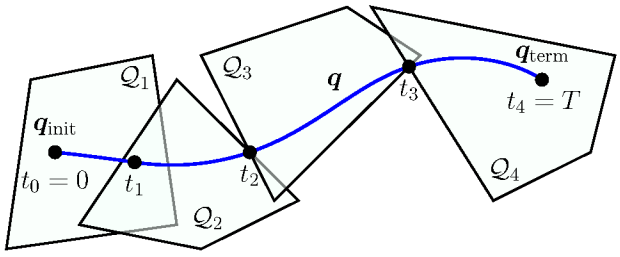

Fig. 2. Example of the motion-planning problem. The safe convex sets to be traversed are $\mathcal{Q}_1, \dots, \mathcal{Q}_4$. The trajectory $\boldsymbol{q} : [0, T] \to \mathbb{R}^2$ is shown in blue. The initial and terminal points are $\boldsymbol{q}_{\text{init}}$ and $\boldsymbol{q}_{\text{term}}$. The times $t_0, \dots, t_4$ determine the trajectory piece assigned to each safe set.

II. Problem Statement

We seek a trajectory that traverses a sequence of convex sets in minimum time, subject to boundary conditions and convex velocity and acceleration constraints. Fig. 2 shows a simple instance of this problem, and illustrates our notation.

The sequence of safe convex sets is denoted as

$$\mathcal{Q}_1, \dots, \mathcal{Q}_I \subseteq \mathbb{R}^n,$$

where I is the number of sets and n is the space dimension. Each set is assumed to be closed and intersect with the next:

$$\mathcal{Q}_i \cap \mathcal{Q}_{i+1} \neq \emptyset, \qquad i \leq I - 1.$$

The trajectory is represented by the function $\boldsymbol{q} : [0, T] \to \mathbb{R}^n$, with time duration $T > 0$. The initial $\boldsymbol{q}(0)$ and terminal point $\boldsymbol{q}(T)$ are fixed to $\boldsymbol{q}_{\text{init}} \in \mathcal{Q}_1$ and $\boldsymbol{q}_{\text{term}} \in \mathcal{Q}_I$, respectively.

The safe sets must be traversed in the given order, and no set can be skipped. We denote with $t_1 \leq \ldots \leq t_{I-1}$ the *transition times* at which the trajectory moves from one set to the next. For simplicity of notation, we also let $t_0 = 0$ and $t_I = T$. We then require that the trajectory $\boldsymbol{q}(t)$ lie in the set $\mathcal{Q}_i$ for all $t \in [t_{i-1}, t_i]$ and $i \leq I$.

The trajectory velocity and the acceleration are denoted as $\dot{\boldsymbol{q}}(t)$ and $\ddot{\boldsymbol{q}}(t)$, respectively. The first is assumed to be continuous, while the second is allowed to have discontinuities (i.e., $\boldsymbol{q}$ is continuously differentiable). The initial $\dot{\boldsymbol{q}}(0)$ and terminal $\dot{\boldsymbol{q}}(T)$ velocities are fixed to zero. The trajectory derivatives must satisfy the constraints

$$\dot{\boldsymbol{q}}(t) \in \mathcal{V}, \qquad \ddot{\boldsymbol{q}}(t) \in \mathcal{A},$$

at all times $t \in [0, T]$. The sets $\mathcal{V}$ and $\mathcal{A}$ are closed and convex, and contain the origin in their interior:

$$\boldsymbol{0} \in \operatorname{int}(\mathcal{V}), \quad \boldsymbol{0} \in \operatorname{int}(\mathcal{A}).$$

Among the trajectories that verify the constraints above, we seek one of minimum time duration T. This leads us to the following optimization problem:

$$\begin{aligned}
\text{minimize} \quad & T && && \text{(1a)}\\
\text{subject to} \quad & \boldsymbol{q}(0) = \boldsymbol{q}_{\text{init}}, && && \text{(1b)}\\
& \boldsymbol{q}(T) = \boldsymbol{q}_{\text{term}}, && && \text{(1c)}\\
& \dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}(T) = \boldsymbol{0}, && && \text{(1d)}\\
& \boldsymbol{q}(t) \in \mathcal{Q}_i, && t \in [t_{i-1}, t_i],\ i \le I, && \text{(1e)}\\
& \dot{\boldsymbol{q}}(t) \in \mathcal{V}, && t \in [0, T], && \text{(1f)}\\
& \ddot{\boldsymbol{q}}(t) \in \mathcal{A}, && t \in [0, T], && \text{(1g)}\\
& t_i \le t_{i+1}, && i \le I - 1, && \text{(1h)}\\
& t_0 = 0,\ t_I = T. && && \text{(1i)}
\end{aligned}$$

The variables are the trajectory $\boldsymbol{q}$, the duration T, and the times $t_0, \dots, t_I$. The first makes the problem infinite dimensional. The *problem data* are the endpoints $\boldsymbol{q}_{\text{init}}$ and $\boldsymbol{q}_{\text{term}}$, the safe sets $\mathcal{Q}_1, \dots, \mathcal{Q}_I$, and the constraint sets $\mathcal{V}$ and $\mathcal{A}$. The differentiability constraint on the function $\boldsymbol{q}$ is implicit here.

A. Feasibility

With the next proposition, we establish the feasibility of problem (1). As in [28, §II-C], we do so by constructing a polygonal (i.e., piecewise linear) trajectory that satisfies all the problem constraints. A similar construction will be used to initialize our method.

Proposition 1. *If the problem data satisfy the assumptions listed above, then problem* (1) *is feasible.*

Proof: We construct a trajectory $\boldsymbol{q}$ that starts at $\boldsymbol{q}(t_0) = \boldsymbol{q}_{\text{init}}$, terminates at $\boldsymbol{q}(t_I) = \boldsymbol{q}_{\text{term}}$, and interpolates any *transition points* $\boldsymbol{q}(t_i) \in \mathcal{Q}_i \cap \mathcal{Q}_{i+1}$ for $i \le I-1$. For $i \le I$, the trajectory piece within the set $\mathcal{Q}_i$ connects $\boldsymbol{q}(t_{i-1})$ and $\boldsymbol{q}(t_i)$ through a straight line. Thus, the overall trajectory stays within the union of the safe sets and traverses them in the desired order. We let the times $t_0, \dots, t_I$ be well spaced, so that the velocity $\dot{\boldsymbol{q}}$ and the acceleration $\ddot{\boldsymbol{q}}$ can be small enough to lie in the constraint sets $\mathcal{V}$ and $\mathcal{A}$ at all times (recall that these sets contain the origin in their interior). Finally, we require that the velocity be zero at each time $t_0, \dots, t_I$. This makes the velocity continuous, even though the trajectory is polygonal. The resulting trajectory is feasible for problem (1). ■

B. Positive traversal times

According to constraint (1h), the *traversal time* $T_i = t_i - t_{i-1}$ of a safe set $\mathcal{Q}_i$ can be zero. This can be optimal if, e.g., a safe set is lower dimensional or our trajectory touches it only at an extreme point. However, our biconvex method will assume that the traversal times T_i are strictly positive, for all $i \le I$. The following is a simple sufficient condition on the problem data that ensures this. It forces our trajectory to cover a nonzero distance within each safe set.

Assumption 1. The boundary points and the safe sets are such that $\boldsymbol{q}_{\text{init}} \notin \mathcal{Q}_2$, $\boldsymbol{q}_{\text{term}} \notin \mathcal{Q}_{I-1}$, and

$$\mathcal{Q}_i \cap \mathcal{Q}_{i+1} \cap \mathcal{Q}_{i+2} = \emptyset, \qquad i \le I - 2.$$

We will let this assumption hold throughout the paper, so that zero traversal times will always be infeasible in our optimization problems. Alternatively, our algorithm can be easily modified to incorporate a small lower bound on the traversal times. For most practical problems, this modification has a negligible effect on the optimal trajectories.

III. Biconvex Method

We give a high-level overview of our biconvex method here, deferring the details to later sections.

The observation at the core of SCS is that problem (1) reduces to a convex program if we fix either the *transition points* or the *transition velocities*:

$$\boldsymbol{q}(t_1), \dots, \boldsymbol{q}(t_{I-1}), \qquad \dot{\boldsymbol{q}}(t_1), \dots, \dot{\boldsymbol{q}}(t_{I-1}).$$

(More precisely, this is true modulo a small conservative approximation of the acceleration constraint (1g), which relies on an estimate of the traversal times.) In these convex programs, the transition points or velocities are fixed, but the rest of the trajectory is optimized.

This observation motivates the method illustrated in Fig. 3 for solving problem (1):

- *Initialization (1st panel).* We compute a polygonal trajectory that connects the initial $\boldsymbol{q}_{\text{init}}$ and terminal point $\boldsymbol{q}_{\text{term}}$, and has short time duration. This is designed through a small number of convex programs.
- *Fixed transition points (2nd panel).* We fix the transition points $\boldsymbol{q}(t_1), \dots, \boldsymbol{q}(t_{I-1})$ of the polygonal trajectory, and use its traversal times $T_1, \dots, T_I$ to approximate the acceleration constraints. This leads to a convex subproblem that improves the polygonal trajectory.
- *Fixed transition velocities (3rd panel).* We fix the transition velocities $\dot{\boldsymbol{q}}(t_1), \dots, \dot{\boldsymbol{q}}(t_{I-1})$ of the improved trajectory, and use its traversal times $T_1, \dots, T_I$ to approximate the acceleration constraints. This leads to another convex subproblem that further improves our solution.
- *Iterations (4th panel).* We keep refining our trajectory by solving the two convex subproblems in alternation.
- *Termination (5th panel).* We terminate when the relative objective decrease of an iteration is smaller than a fixed tolerance $\varepsilon \in (0, 1]$. The objective decrease is measured between any two consecutive subproblems of the same kind (fixed transition points or velocities).

The following sections detail our algorithm. We first illustrate the subproblem with fixed transition velocities, then the one with fixed transition points, and lastly the initialization step. This order simplifies the exposition, although it is the opposite order of how these steps appear in our algorithm.

IV. Subproblem with Fixed Transition Velocities

This section illustrates the convex subproblem with fixed transition velocities $\dot{\boldsymbol{q}}(t_1), \dots, \dot{\boldsymbol{q}}(t_{I-1})$. First, we formulate problem (1) as a more tractable nonconvex program. Then, we convexify this program by fixing the transition velocities and approximating the acceleration constraint (1g).

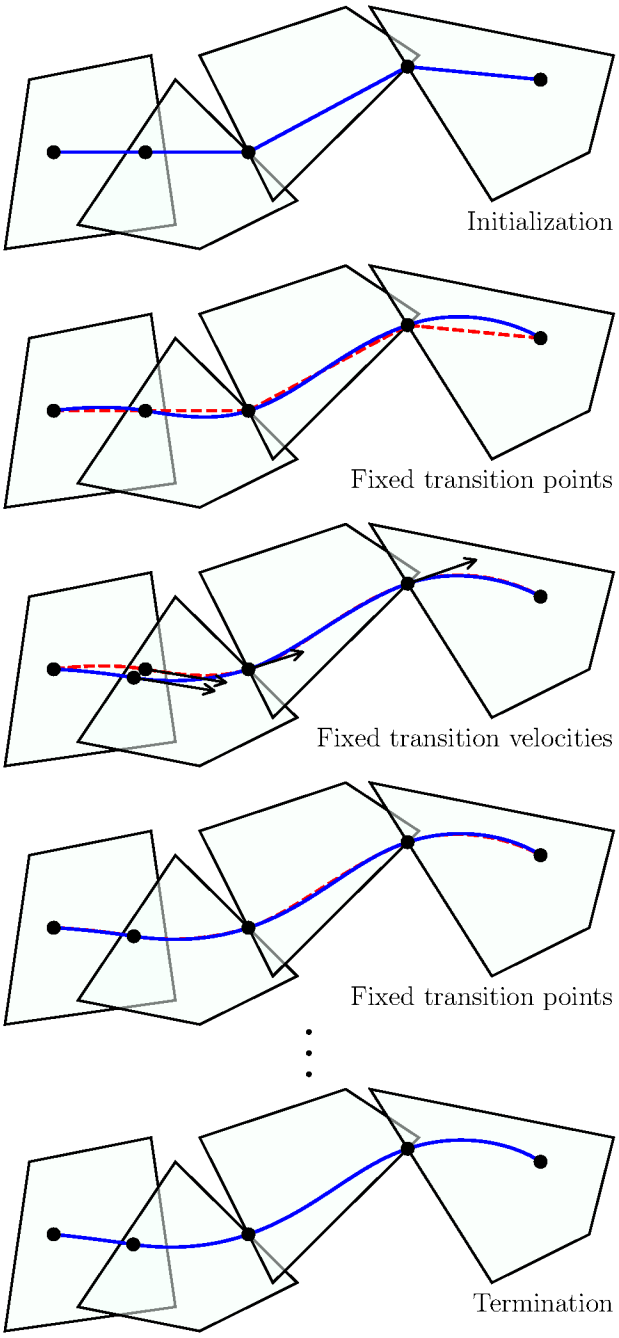

Fig. 3. Steps of SCS during the solution of the problem in Fig. 2. In the initialization (1st panel), we quickly compute a feasible polygonal trajectory. Then, we alternate between a convex subproblem with fixed transition points and one with fixed transition velocities (2nd to 4th panels). We terminate when the cost decrease of an iteration is small enough (5th panel). The trajectory computed at each iteration (solid blue) is overlaid on the trajectory from the previous iteration (dashed red).

A. Change of variables

We parameterize the trajectory within each safe set $\mathcal{Q}_i$ using a function $\boldsymbol{q}_i : [0,1] \to \mathbb{R}^n$ and a scalar $T_i > 0$. These decide the trajectory shape and traversal time, respectively. We also define the function $h_i(t) = (t - t_{i-1})/T_i$ that maps the interval of time $[t_{i-1}, t_i]$ assigned to the set $\mathcal{Q}_i$ to the unit interval $[0,1]$. This allows us to reconstruct our trajectory as

$$\boldsymbol{q}(t) = \boldsymbol{q}_i(h_i(t)),$$

for all $t \in [t_{i-1}, t_i]$ and $i \leq I$. By differentiating the last equality, we obtain the following expressions for the trajectory velocity and acceleration:

$$\dot{\boldsymbol{q}}(t) = \frac{\dot{\boldsymbol{q}}_i(h_i(t))}{T_i}, \qquad \ddot{\boldsymbol{q}}(t) = \frac{\ddot{\boldsymbol{q}}_i(h_i(t))}{T_i^2},$$

which hold for all $t \in [t_{i-1}, t_i]$ and $i \leq I$.

B. Nonconvex formulation

We express problem (1) in terms of the new variables. The objective function (1a) simply becomes

$$\sum_{i=1}^{I} T_i. \tag{2}$$

The boundary conditions in (1b) to (1d) become

$$\boldsymbol{q}_1(0) = \boldsymbol{q}_{\text{init}}, \quad \boldsymbol{q}_I(1) = \boldsymbol{q}_{\text{term}}, \quad \dot{\boldsymbol{q}}_1(0) = \dot{\boldsymbol{q}}_I(1) = \boldsymbol{0}, \tag{3}$$

where in the last constraint we canceled the traversal times T_1 and T_I since the right-hand side is zero. The next conditions ensure that the trajectory and its derivative are continuous:

$$\boldsymbol{q}_i(1) = \boldsymbol{q}_{i+1}(0), \qquad i \leq I-1, \tag{4a}$$

$$\frac{\dot{\boldsymbol{q}}_i(1)}{T_i} = \frac{\dot{\boldsymbol{q}}_{i+1}(0)}{T_{i+1}}, \qquad i \leq I-1. \tag{4b}$$

The constraints in (1e), (1f), and (1g) become

$$\boldsymbol{q}_i(s) \in \mathcal{Q}_i, \qquad s \in [0,1],\ i \leq I, \tag{5a}$$

$$\dot{\boldsymbol{q}}_i(s) \in T_i \mathcal{V}, \qquad s \in [0,1],\ i \leq I, \tag{5b}$$

$$\ddot{\boldsymbol{q}}_i(s) \in T_i^2 \mathcal{A}, \qquad s \in [0,1],\ i \leq I, \tag{5c}$$

where we multiplied both sides of the velocity and the acceleration constraints by T_i and T_i^2, respectively. Finally, constraint (1h) results in

$$T_i > 0, \qquad i \leq I, \tag{6}$$

where zero traversal times are excluded because of Assumption 1.

Overall, problem (1) is reformulated as

$$\begin{aligned} &\text{minimize} && (2) \\ &\text{subject to} && (3) \text{ to } (6), \end{aligned} \tag{7}$$

with variables $\boldsymbol{q}_i$ and T_i for $i \leq I$. The objective and most of the constraints of this problem are linear. The position constraint (5a) is convex. As mentioned in §I-B, also the velocity constraint (5b) is convex. On the other hand, the velocity continuity (4b) and the acceleration constraint (5c) are nonconvex. Therefore, the overall problem is nonconvex.

C. Convex restriction

The next step is to construct a *convex restriction* (informally, a convex inner approximation [9, §2.1]) of the nonconvex constraints of problem (7). To do so, we assume that the transition velocities have fixed value,

$$\dot{\boldsymbol{q}}(t_i) = \boldsymbol{v}_i, \qquad\qquad i \leq I-1,$$

and that we are given nominal values $\bar{T}_i > 0$ for the traversal times T_i, for all $i \leq I$.

With the transition velocities fixed, the velocity-continuity constraints (4b) become linear:

$$\dot{\boldsymbol{q}}_i(1) = \boldsymbol{v}_i T_i, \;\; \dot{\boldsymbol{q}}_{i+1}(0) = \boldsymbol{v}_i T_{i+1}, \qquad i \leq I-1. \tag{8}$$

To approximate the acceleration constraint (5c), we underestimate the convex function T_i^2 with its linearization around the nominal value $\bar{T}_i$:

$$T_i^2 \geq \bar{T}_i(2T_i - \bar{T}_i). \tag{9}$$

We then replace (5c) with

$$\begin{aligned} &\ddot{\boldsymbol{q}}_i(s) \in \bar{T}_i(2T_i - \bar{T}_i)\mathcal{A}, && s \in [0,1], \; i \leq I, && \text{(10a)}\\ &2T_i \geq \bar{T}_i, && i \leq I. && \text{(10b)} \end{aligned}$$

These constraints are convex (see again the discussion in §I-B). Moreover, they imply (5c) because of the inequality (9) and the assumption that the constraint set $\mathcal{A}$ contains the origin.

Collecting all the pieces, we have the following convex restriction of problem (7):

$$\begin{aligned} &\text{minimize} && \text{(2)} \\ &\text{subject to} && \text{(3) to (5) except (4b) and (5c),} \\ &&& \text{(8) and (10).} \end{aligned} \tag{11}$$

Constraint (6) is omitted here since it is implied by (10b). Given a feasible trajectory with transition velocities $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{I-1}$ and traversal times $\bar{T}_1, \ldots, \bar{T}_I$, this problem yields another feasible trajectory with lower or equal cost.

V. Subproblem with Fixed Transition Points

We now illustrate the convex subproblem with fixed transition points $\boldsymbol{q}(t_1), \ldots, \boldsymbol{q}(t_{I-1})$. In the previous section, we parameterized the trajectory at the "position level" using the functions $\boldsymbol{q}_i$ for $i \leq I$. The velocity and acceleration were $\dot{\boldsymbol{q}}_i/T_i$ and $\ddot{\boldsymbol{q}}_i/T_i^2$, respectively. This choice made all the position constraints convex, and gave us some nonconvex velocity and acceleration constraints. Here, we parameterize the trajectory at the "velocity level" using the functions $\dot{\boldsymbol{r}}_i = \dot{\boldsymbol{q}}_i/T_i$. We recover the position as $T_i \boldsymbol{r}_i$ and the acceleration as $\ddot{\boldsymbol{r}}_i/T_i$. Furthermore, we work with the reciprocals $S_i = 1/T_i$ of the traversal times. This yields a problem equivalent to (7) where all the velocity constraints are convex, and the nonconvexities are only at the position and acceleration levels:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{I} \frac{1}{S_i} && && \text{(12a)}\\ \text{subject to} \quad & \boldsymbol{r}_1(0) = S_1 \boldsymbol{q}_{\text{init}}, && && \text{(12b)}\\ & \boldsymbol{r}_I(1) = S_I \boldsymbol{q}_{\text{term}}, && && \text{(12c)}\\ & \dot{\boldsymbol{r}}_1(0) = \dot{\boldsymbol{r}}_I(1) = \boldsymbol{0}, && && \text{(12d)}\\ & \frac{\boldsymbol{r}_i(1)}{S_i} = \frac{\boldsymbol{r}_{i+1}(0)}{S_{i+1}}, && i \leq I-1, && \text{(12e)}\\ & \dot{\boldsymbol{r}}_i(1) = \dot{\boldsymbol{r}}_{i+1}(0), && i \leq I-1, && \text{(12f)}\\ & \boldsymbol{r}_i(s) \in S_i \mathcal{Q}_i, && s \in [0,1], \; i \leq I, && \text{(12g)}\\ & \dot{\boldsymbol{r}}_i(s) \in \mathcal{V}, && s \in [0,1], \; i \leq I, && \text{(12h)}\\ & \ddot{\boldsymbol{r}}_i(s) \in (1/S_i)\mathcal{A}, && s \in [0,1], \; i \leq I, && \text{(12i)}\\ & S_i > 0, && i \leq I. && \text{(12j)} \end{aligned}$$

Observe that the objective of this problem is still convex, even though we work with the traversal-time reciprocals. The only nonconvex constraints are the position continuity (12e) and the acceleration constraint (12i), which have the same structure as the constraints (4b) and (5c), respectively.

We proceed as in the previous section to construct a convex restriction of problem (12). This time we assume that the transition points are fixed:

$$\boldsymbol{q}(t_i) = \boldsymbol{p}_i, \qquad\qquad i \leq I-1.$$

This makes the position continuity (12e) linear:

$$\boldsymbol{r}_i(1) = \boldsymbol{p}_i S_i, \;\; \boldsymbol{r}_{i+1}(0) = \boldsymbol{p}_i S_{i+1}, \qquad i \leq I-1. \tag{13}$$

To approximate the acceleration constraint (12i), we assume again that we are given nominal values $\bar{T}_i > 0$ of the traversal times. We underestimate the convex function $1/S_i$ with its linearization around the nominal point:

$$\frac{1}{S_i} \geq \bar{T}_i(2 - \bar{T}_i S_i).$$

This gives us the following convex restriction of the acceleration constraint:

$$\begin{aligned} &\ddot{\boldsymbol{r}}_i(s) \in \bar{T}_i(2 - \bar{T}_i S_i)\mathcal{A}, && s \in [0,1], \; i \leq I, && \text{(14a)}\\ &2 \geq \bar{T}_i S_i, && i \leq I. && \text{(14b)} \end{aligned}$$

Overall, the subproblem with fixed transition points is

$$\begin{aligned} &\text{minimize} && \text{(12a)} \\ &\text{subject to} && \text{(12b) to (12j) except (12e) and (12i),} \\ &&& \text{(13) and (14).} \end{aligned} \tag{15}$$

This convex subproblem allows us to improve a given feasible trajectory with transition points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{I-1}$ and traversal times $\bar{T}_1, \ldots, \bar{T}_I$.

VI. Initialization with Polygonal Trajectory

In the initialization of SCS we quickly identify a low-cost feasible trajectory for problem (1). As in the proof of Proposition 1, a natural candidate for this role is a polygonal trajectory that comes to a full stop at each "kink."

The shape of our polygonal trajectory is computed through the following convex program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{I-1} \|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|_2 && \text{(16a)}\\
\text{subject to} \quad & \boldsymbol{p}_0 = \boldsymbol{q}_{\text{init}}, && \text{(16b)}\\
& \boldsymbol{p}_I = \boldsymbol{q}_{\text{term}}, && \text{(16c)}\\
& \boldsymbol{p}_i \in \mathcal{Q}_i \cap \mathcal{Q}_{i+1}, \qquad i \leq I-1. && \text{(16d)}
\end{aligned}$$

Here the decision variables are the points $\boldsymbol{p}_0, \dots, \boldsymbol{p}_I$ that the trajectory interpolates through straight lines (black dots in the first panel of Fig. 3). The objective minimizes the total Euclidean length of the trajectory.

Next, we select the *vertices* of the polygonal trajectory, i.e., the points $\boldsymbol{p}_i$ that do not lie on the line connecting $\boldsymbol{p}_{i-1}$ to $\boldsymbol{p}_{i+1}$. (This condition can be efficiently checked using the triangle inequality.) For ease of notation, we also include $\boldsymbol{p}_0$ and $\boldsymbol{p}_I$ in the list of vertices. As an example, in the top panel of Fig. 3, the only point that is not a vertex is $\boldsymbol{p}_1$ (the second).

The initialization is completed by connecting each pair of consecutive vertices through a minimum-time trajectory segment, with zero velocity at the endpoints. While these vertex-to-vertex problems could be solved in closed form when working with infinite-dimensional trajectories, in practice, we use a finite-dimensional trajectory parameterization, and it is convenient to formulate them as convex programs. To this end, let us assume that we are connecting two vertices that are consecutive points $\boldsymbol{p}_{i-1}$ and $\boldsymbol{p}_i$. (If not, we can proceed as follows and, afterwards, split the designed trajectory into pieces.) The vertex-to-vertex problem can be formulated as a convex program similar to the nonconvex problem (12):

$$\begin{aligned}
\text{minimize} \quad & T_i && \text{(17a)}\\
\text{subject to} \quad & \boldsymbol{r}_i(0) = S_i \boldsymbol{p}_{i-1}, && \text{(17b)}\\
& \boldsymbol{r}_i(1) = S_i \boldsymbol{p}_i, && \text{(17c)}\\
& \dot{\boldsymbol{r}}_i(0) = \dot{\boldsymbol{r}}_i(1) = \boldsymbol{0}, && \text{(17d)}\\
& \dot{\boldsymbol{r}}_i(s) \in \mathcal{V}, \qquad s \in [0,1], && \text{(17e)}\\
& \ddot{\boldsymbol{r}}_i(s) \in T_i \mathcal{A}, \qquad s \in [0,1], && \text{(17f)}\\
& T_i \geq 1/S_i, \ S_i > 0. && \text{(17g)}
\end{aligned}$$

The variables are the traversal time T_i, its reciprocal S_i, and the function $\boldsymbol{r}_i : [0,1] \to \mathbb{R}^n$ (which represents $\boldsymbol{q}_i/T_i$). The last constraint relaxes the nonconvex equality $T_i = 1/S_i$ to a convex inequality. However, this relaxation is lossless: in fact, given any feasible solution $\bar{T}_i$, $\bar{S}_i$, and $\bar{\boldsymbol{r}}_i$, the solution $T_i = \bar{T}_i$, $S_i = 1/\bar{T}_i$, and $\boldsymbol{r}_i = \bar{\boldsymbol{r}}_i/(\bar{T}_i \bar{S}_i)$ is also feasible, has equal cost, and satisfies $T_i = 1/S_i$. In practice, we solve problem (17) as a one-dimensional problem, leveraging the fact that its optimal trajectories are straight lines. This accelerates our algorithm when working in high-dimensional spaces.

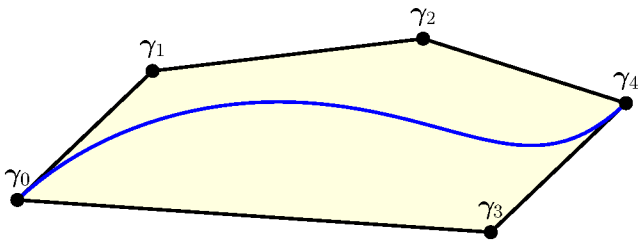

Fig. 4. A two-dimensional Bézier curve with control points $\gamma_0, \dots, \gamma_4$. The area shaded in yellow is the convex hull of the control points.

VII. Numerical Implementation

The numerical implementation of our method requires a finite-dimensional trajectory parameterization. In some special cases, it is possible to use a parameterization that captures the infinite-dimensional optimum of problem (1). However, in general, optimal trajectories can be quite complex, and some approximation error is unavoidable.

Bézier curves have been widely used in motion planning, and enjoy several properties that make them particularly well suited for our problems. In this section, we first collect some basic definitions and properties of Bézier curves, following [28, §V-A]. Then we show how the infinite-dimensional problems in the previous sections can be translated into efficient finite-dimensional programs.

A. Bézier curves

Bézier curves are constructed using Bernstein polynomials. The *Bernstein polynomials* of degree K are defined over the interval $[0,1] \subset \mathbb{R}$ as follows:

$$\beta_k(s) = \binom{K}{k} s^k (1-s)^{K-k}, \quad k \leq K. \qquad (18)$$

(Recall that in this paper k is nonnegative and $k \leq K$ is shorthand for $k \in \{0, \dots, K\}$.) The Bernstein polynomials are nonnegative and, by the binomial theorem, sum up to one. Therefore, the scalars $\beta_0(s), \dots, \beta_K(s)$ represent the coefficients of a convex combination for all $s \in [0,1]$. We use these coefficients to combine a given set of *control points* $\boldsymbol{\gamma}_0, \dots, \boldsymbol{\gamma}_K \in \mathbb{R}^n$, and obtain a *Bézier curve*:

$$\boldsymbol{\gamma}(s) = \sum_{k=0}^{K} \beta_k(s) \boldsymbol{\gamma}_k. \qquad (19)$$

The function $\boldsymbol{\gamma} : [0,1] \to \mathbb{R}^n$ is a (vector-valued) polynomial of degree K. Fig. 4 shows a Bézier curve of degree $K = 4$ in $n = 2$ dimensions.

The following are a few selected properties of Bézier curves. We refer to [10] for a more comprehensive list.

Property 1 (Derivative)**.** The derivative $\dot{\gamma}$ of the Bézier curve γ is a Bézier curve of degree $K-1$. Its control points are computed via the linear difference equation

$$\dot{\boldsymbol{\gamma}}_k = K(\boldsymbol{\gamma}_{k+1} - \boldsymbol{\gamma}_k), \quad k \leq K-1.$$

Property 2 (Endpoint)**.** The Bézier curve γ starts at its first control point and ends at its last control point:

$$\gamma(0) = \gamma_0, \qquad \gamma(1) = \gamma_K.$$

Property 3 (Convex hull)**.** The Bézier curve γ is contained in the convex hull of its control points at all times:

$$\gamma(s) \in \text{conv}(\{\gamma_0, \ldots, \gamma_K\}), \quad s \in [0, 1].$$

This convex hull is shaded in yellow in Fig. 4.

B. Finite-dimensional trajectory parameterization

When solving the programs (11), (15) and (17) numerically, we restrict our trajectory segments ($\boldsymbol{q}_i$ or $\boldsymbol{r}_i$) to Bézier curves of degree K, and enforce all the necessary constraints leveraging the properties above. Property 1 tells us that the trajectory velocity and acceleration are also piecewise Bézier curves, of degree $K-1$ and $K-2$, respectively. Using Property 2, we can then easily enforce any boundary or continuity condition by constraining the first and last control points of our Bézier curves. The containment of a trajectory segment (or its derivatives) in a convex set can be enforced using Property 3: if all the control points of a Bézier curve lie in a convex set, then so does the whole curve. In the initialization step, we might also have to split a trajectory segment, obtained by solving problem (17), into multiple pieces. This is easily done by using De Casteljau's algorithm [10, §2.4].

For completeness, in §A, we report the finite-dimensional versions of the convex programs (11), (15), and (17). We also report the finite-dimensional version of the nonconvex program (7), which will serve as a baseline in the experiments below.

VIII. Strengths

This section illustrates the main strengths of SCS.

A. Convergence and completeness

Under our assumptions on the problem data, SCS is guaranteed to converge monotonically. In fact, the initialization step must succeed, since problems (16) and (17) are feasible and admit an optimal solution. Then, the convex subproblems (11) and (15) are guaranteed to produce trajectories that are not worse than the ones they are initialized with. This makes our algorithm *complete* (guaranteed to find a solution) and *anytime* (returns a feasible solution even if stopped early).

These results extend to the finite-dimensional implementation of SCS from §VII, provided that our Bézier curves have degree $K \geq 3$. This minimum degree is sufficient for our trajectory segments to represent straight lines with zero endpoint velocity, and ensures the success of the initialization step. After that, the biconvex alternation can only improve our finite-dimensional trajectory. Notably, our piecewise Bézier trajectories satisfy the constraints of problem (1) at all continuous times, rather than at a finite set of times, as is common for sampling-based and trajectory-optimization methods.

B. Optimality

SCS is *heuristic*: it is not guaranteed to find an optimal solution (global or local), or to converge within a fixed distance from one. However, it typically finds high-quality trajectories in a fraction of the time of state-of-the-art solvers (see the experiments in §X-B). The trajectory parameterization using Bézier curves can also affect the optimality of our trajectories. In this direction, we remark that a Bézier curve is as expressive as any polynomial of equal degree [10, §1.3]. Another source of suboptimality are the conservative convex constraints obtained using Property 3. However, these constraints get arbitrarily accurate as the degree K increases. Potentially, we could also use exact containment conditions like sums of squares [32, 8], but this would make our programs much more expensive to solve.

C. Computational efficiency

The runtime of an iteration of SCS is polynomial in all the relevant problem data, and linear in the number I of safe sets and the degree K of the Bézier curves. In fact, the subproblems (11) and (15) have banded structure, and are solvable in a time that is linear in I and K (see, e.g., [43]). Problems (16) and (17) are also banded, and solvable in a time that is linear in I and K, respectively. In addition, the latter problem is solved at most I times.

The overall time complexity of SCS is harder to quantify. However, in practice, we observed that the number of iterations necessary for convergence is often insensitive to I and K (see the experiments in §X-B). This leads to overall runtimes that are often linear in I and K.

D. Limited parameter tuning

SCS does not require the tuning of any step-size or trust-region parameter. The only numerical values set by the user are the degree K of the Bézier curves and the convergence tolerance ε. The first should be at least three to ensure convergence, and can be increased to improve the solution quality. For the second, we have found that $\varepsilon = 0.01$ is sufficiently small for most problems.

E. Advantages over existing methods

As discussed in §I, SCS addresses a problem similar to the one in [28, §V]. Compared to that approach, SCS applies to a narrower set of motion-planning problems, but converges much faster (see experiments in §X-C). This is because its subproblems are convex restrictions of the original nonconvex program, and at every iteration we can take a full step towards their optima.

The GCS motion planner from [27] requires a convex trajectory parameterization within each safe set. However, as also seen in this paper, this is very challenging when we optimize both the trajectory shape and timing, and imposes strict limitations on the types of costs and constraints that GCS can handle. For instance, the method in [27] can only enforce coarse approximations of the acceleration constraints (1g). The recent work [48] proposes a semidefinite relaxation for

these time-scaling problems, broadening the list of costs and constraints that GCS can accommodate but sacrificing the algorithm completeness. Overall, GCS and SCS can be viewed as complementary methods, and can be combined in hybrid approaches where GCS provides an approximate solution to the high-level discrete-continuous problem and SCS refines the trajectory within a fixed sequence of safe sets.

Optimization problems similar to the one considered in this paper are also faced by UAV motion planners based on safe flight corridors [4, 24, 47]. However, these planners typically bypass the problem nonconvexity by fixing the corridor traversal times using heuristics, while here we optimize these times explicitly.

The main advantage of SCS over general-purpose methods for trajectory optimization is its reliability and completeness. Furthermore, SCS can generate high-quality trajectories for complex planning problems within a few milliseconds (see §X-C). In contrast, trajectory-optimization methods require a GPU to achieve comparable runtimes [35]. Finally, most common trajectory-optimization methods do not take full advantage of the structure of minimum-time problems.

The minimum-distance problem (16), solved to initialize SCS, is similar to the problem addressed by common sampling-based methods. This step is straightforward for us since we assume that the free space is represented as a sequence of convex sets. Contrarily, sampling-based methods rely solely on a collision checker, which makes finding a minimum-distance curve significantly more challenging. The work [46] explores a combined approach, where a sampling-based method is used to find a polygonal curve that is later inflated into a sequence of safe sets for SCS to plan through.

Finally, various convex relaxations and reformulations of time-optimal control and trajectory-tracking problems have been proposed over the years (see, e.g., [40, 23, 21, 25]). However, none of these methods applies directly to the problem of designing trajectories through sequences of convex sets.

IX. Limitations

Our method has a few worth-noting limitations. First of all, SCS is restricted to minimum-time problems. However, a similar approach can be applied to problems with fixed final time and cost function that penalizes the magnitude of the trajectory velocity and acceleration.

SCS requires that the robot free space is described as a sequence of convex sets. This description can be challenging to compute for high-dimensional problems and cluttered environments. However, as mentioned in §I, many practical methods for decomposing complex spaces into convex sets are now available, and also GPU-based algorithms have been recently developed [46].

The trajectories generated by SCS may have acceleration jumps, which can make them difficult to track on real hardware. A simple workaround is to add a smoothing step. Alternatively, we can ensure that the trajectory acceleration (as well as any higher-order derivative) is continuous by setting it to zero at the transition times. This is easily seen to be a linear constraint. A similar limitation is that SCS can only handle constraints on the velocity and acceleration but not, for example, on the trajectory jerk.

We have seen that SCS cannot handle problems where an optimal traversal time T_i is zero (in which case the corresponding variable S_i in the subproblem with fixed transition points (15) is infinity). Although Assumption 1 is sufficient to rule out this scenario, some practically relevant problems do not meet this assumption. In these cases, we can enforce an artificial lower bound on the time spent in each safe set.

X. Numerical Experiments

We demonstrate SCS on three numerical experiments. First, we conclude the simple running example in Fig. 2 and 3 by reporting its solution statistics. Second, we analyze the performance of SCS as a function of multiple problem data, and we compare it with state-of-the-art solvers for nonconvex optimization. Finally, we demonstrate SCS on a minimum-time package-transfer problem with two Sparrow robots, and we benchmark it against other motion-planning methods.

The Python implementation of SCS used in the experiments below is available at

`github.com/TobiaMarcucci/scsplanning`.

It is based on `Drake` [36], and uses the open-source solver `Clarabel` [12] for the convex programs. All the experiments are run on a laptop with Apple M2 Pro processor and 16 GB of RAM. The solvers `SNOPT` [11] and `IPOPT` [41] are also called through `Drake`'s Python interface (and are warm started with the same polygonal trajectory as SCS).

A. Running example

We provide here the details of the running example illustrated in Fig. 2. The initial and terminal points are $\boldsymbol{q}_{\text{init}} = (0, 0)$ and $\boldsymbol{q}_{\text{term}} = (10, 1.5)$, respectively. The geometry of the safe sets can be deduced from the figure. The constraint sets $\mathcal{V}$ and $\mathcal{A}$ are circles centered at the origin of radius 10 and 1, respectively. The trajectory in Fig. 2 has time duration $T = 7.45$, and is designed by SCS with degree $K = 5$ and termination tolerance $\varepsilon = 0.01$.

The curves in Fig. 3 represent the actual iterations of SCS. The initial polygonal trajectory has time duration $T = 12.49$ (1st panel). This value decreases to 8.82 in the first subproblem with fixed transition points (2nd panel), then to 8.06 and 7.51 in the subsequent subproblems (3rd and 4th panels). SCS converges after solving only five subproblems.

As a baseline for SCS, we solve the finite-dimensional version of the nonconvex program (7) with `SNOPT` and `IPOPT`. This problem is stated in §A, see (20), and uses the same trajectory parameterization as SCS. Both solvers yield the trajectory duration $T = 7.40$, which is only 0.7% shorter than ours. Our simple Python implementation of SCS takes 10 ms to converge, while `SNOPT` takes 21 ms and `IPOPT` needs 261 ms. Note, however, that these solvers use smaller termination tolerances than SCS. Increasing the optimality tolerances of the nonconvex solvers does not reduce their

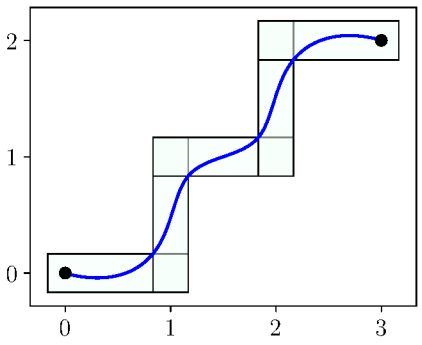

Fig. 5. Benchmark problem with $I = 5$ safe sets in $n = 2$ dimensions, with $m = 4$ facets each. The optimal trajectory is shown in blue.

runtimes significantly. Conversely, if we decrease the SCS tolerance to, e.g., $\varepsilon = 10^{-4}$, the objective gap between SCS and the nonconvex solvers decreases to 0.1%, but the runtime of SCS increases to 50 ms. This is typical for multi-convex methods: they find high-quality solutions quickly, but can be slow if we seek very accurate solutions [34].

B. Runtime analysis and comparison with nonconvex solvers

We analyze the runtimes of SCS, `SNOPT`, and `IPOPT` as functions of several problem parameters: the number I of safe sets, the number m of facets of each safe set, the space dimension n, and the trajectory degree K. We show that, across a wide range of problem instances, SCS finds low-cost trajectories more quickly and reliably than the two state-of-the-art solvers.

We construct an instance of problem (1) where each safe set $\mathcal{Q}_i$ represents one link of an n-dimensional staircase. The safe sets are polytopes that approximate ellipsoids with increasing accuracy as their number m of facets grows. Fig. 5 shows an instance of this problem with the corresponding optimal trajectory. In this instance, we have $I = 5$ safe sets in $n = 2$ dimensions, and each set has $m = 4$ facets (rectangular safe sets). More details on the construction of these problems are reported in §B.

We consider a first batch of instances where we let the number I of safe sets grow from 3 to 3000, while we fix the space dimension to $n = 3$, the number of facets to $m = 6$, and the trajectory degree to $K = 3$. The top panel of Fig. 6 shows the runtimes of SCS, `SNOPT`, and `IPOPT`. The two nonconvex solvers return trajectories with equal cost, when `SNOPT` does not fail or reach our time limit of 1 h (missing markers in the figure). SCS designs trajectories that have slightly higher cost (1.2% in the worst case). SCS is faster in almost all instances: `SNOPT` and `IPOPT` have comparable runtimes only on the smallest and largest problems, respectively. The runtimes of SCS increase a little more than linearly: as the number of safe sets grows by a factor of 1000, its runtimes increase by 3060. The number of subproblems necessary for SCS to converge with tolerance $\varepsilon = 0.01$ ranges between 5 and 8.

The second panel in Fig. 6 shows the effects of increasing the number m of facets of the safe sets from 3 to 3000, while keeping $I = 20$, $n = 2$, and $K = 5$. In this case, SCS and the nonconvex solvers find identical trajectories (despite the larger termination tolerance of SCS). SCS solves each problem much faster than `SNOPT` and `IPOPT`, and its runtimes increase sublinearly with m (as the number of facets grows by 1000, the

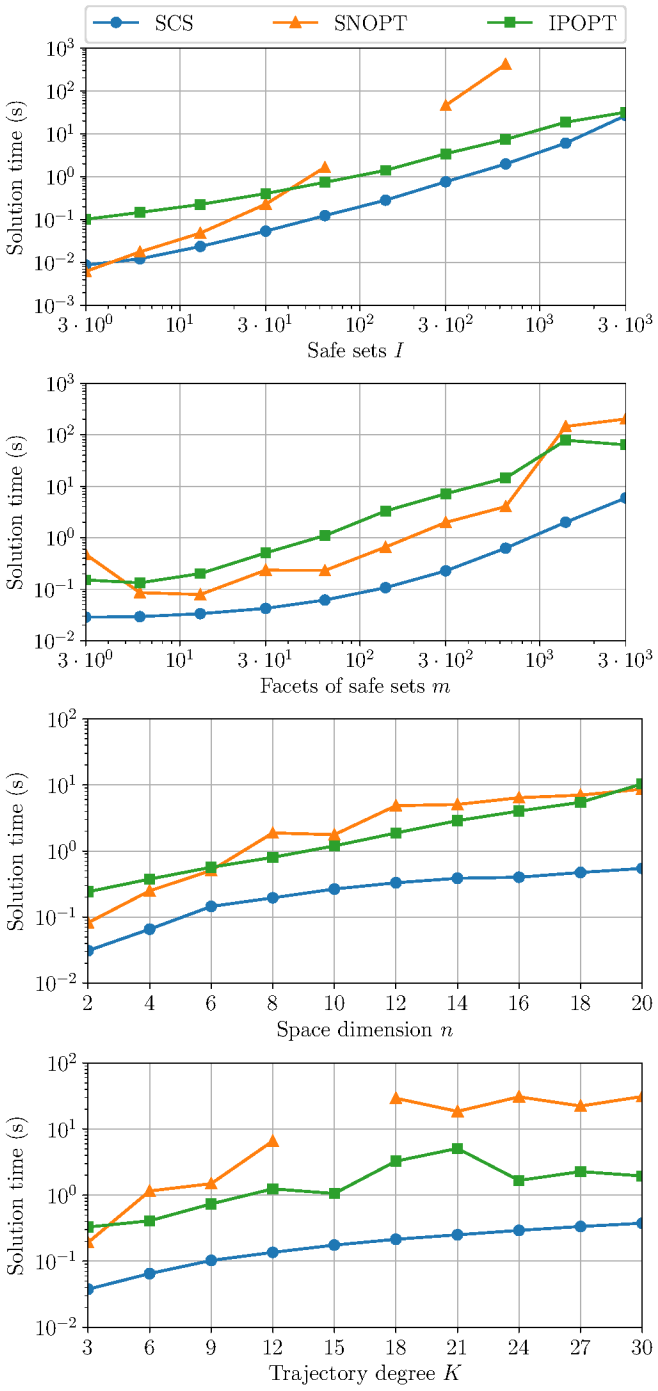

Fig. 6. Comparison of SCS with the solvers `SNOPT` and `IPOPT`. The runtimes of the three methods are analyzed as functions of multiple problem data. Missing markers correspond to solver failures. The runtimes of SCS grow almost linearly in each experiment (note that the horizontal axis has logarithmic scale in the first two panels and linear scale in the last two).

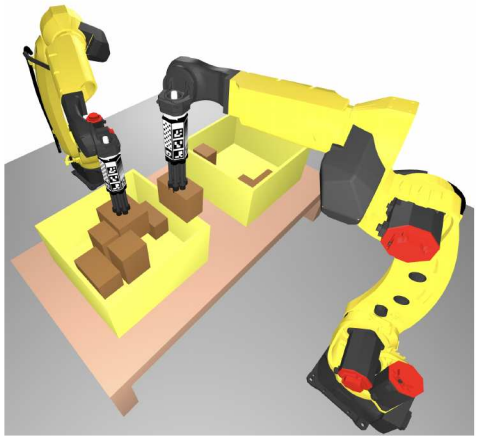

Fig. 7. Sparrow robots that move packages between bins in minimum time.

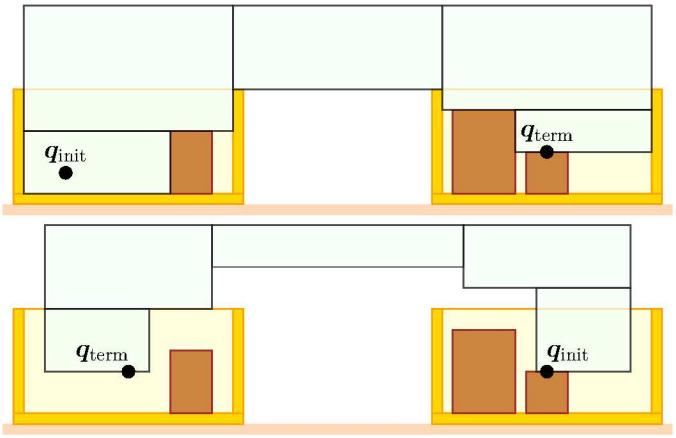

Fig. 8. Two-dimensional illustration of the three-dimensional safe sets $\mathcal{Q}_i$ used for the package-transfer task. The top panel shows the sets for picking the rightmost package. The bottom panel shows the sets for its placement. The latter are shrunk to avoid the collision of the transported package.

runtime grows by 210). The number of subproblems necessary for SCS to converge is equal to 5 for every value of m.

In the third panel of Fig. 6, we let the space dimension n grow from 2 to 20, while we set $I = 20$, $m = 2n$, and $K = 3$. The nonconvex solvers find again identical trajectories, and SCS has a maximum cost gap of 3.2%. SCS is again the fastest, and its runtimes increase a little more than linearly with n (the space dimension grows by 10 and the runtimes by 17.6). The number of SCS subproblems ranges between 5 and 16.

In the fourth panel of Fig. 6, we let $I = 20$, $m = 6$, $n = 3$, and increase the degree K from 3 to 30. All the methods return similar trajectories: the maximum cost difference between SCS and the nonconvex solvers is 0.4%. SCS is the fastest and its runtimes grow linearly with K (the degree increases by 10 and the runtimes by 9.9). `IPOPT` performs better than `SNOPT`, which also fails in one instance. SCS always converges after 5 subproblems.

C. Minimum-time package transfer with two Sparrow robots

We use SCS to plan the motion of two Sparrow robots that transfer packages between bins in simulation. We also benchmark SCS against the trust-region method proposed in [28, §V], as well as a simple waypoint-based motion planner representative of those commonly used in industry.

The package-transfer task is illustrated in Fig. 7. The two robots face each other, and between them is a table with two bins. One bin contains ten packages and the other is empty. The goal is to move all the packages in the first bin to the second as quickly as possible. The final package positions in the second bin must mirror the initial positions in the first bin. Packages are represented as axis-aligned boxes (these can be the packages themselves, or bounding boxes of products with more complex shape). The bins have side 0.6 and height 0.3, and the distance between their centers is 1. The package sides are drawn uniformly at random between 0.1 and 0.25. Also the initial package positions are drawn uniformly at random within the corresponding bin, and sampled packages are rejected when they collide with existing packages.

We solve the task using a state machine. At each iteration, if a robot has completed its previous pick or place motion, we plan its next motion neglecting the presence of the other robot. If this results in a collision, we let the robot idle until the next iteration. If the state machine stalls (neither arm can execute its motion without colliding with the other), we retract one arm and allow the other to move. Each time a robot plans a picking motion, it targets the package closest to its side of the table. Trajectories are planned directly in the three-dimensional task space, and the full robot configuration is retrieved through inverse kinematics. We let the sets $\mathcal{V}$ and $\mathcal{A}$, that constrain the gripper velocity and acceleration, be spheres of radius 10 centered at the origin. (In practice, these sets can be shaped to prevent package delamination, and ensure that the robots can track the designed task-space trajectories.) We use Bézier curves of degree $K = 5$ and set the termination tolerance to $\varepsilon = 0.01$.

For each pick and place motion, the three-dimensional task space is decomposed into five box-shaped safe sets $\mathcal{Q}_i$, illustrated in two dimensions in Fig. 8. The first and fifth sets allow the gripper to reach the trajectory endpoints, without colliding with the packages in the bins. The second and fourth sets cover the space above the packages in the two bins. The third is a transfer region that connects the spaces above the bins. As shown in the bottom panel of Fig. 8, these sets are shrunk during a place motion to avoid collisions of the transported package (packages are always picked above their centers).

We consider 50 randomly generated package-transfer problems. As the low-level motion planner for the state machine just described, we compare SCS against the following alternatives:

- The trust-region method from [28, §V], modified as described in §C to deal with minimum-time problems.
- A simple waypoint-based motion planner, which lifts a package vertically, moves it horizontally above the

TABLE I
PACKAGE-TRANSFER BENCHMARK

	Task-completion time (s)			Motion-planning time (ms)		
Planner	min	mean	max	min	mean	max
SCS	8.86	9.96	11.34	4	15	49
Trust region	10.43	12.73	14.87	15	46	163
Waypoint	13.17	14.97	16.74	2	3	12

desired destination, and places it down. Where each trajectory segment is executed in minimum time.

The three methods use the same constraints and trajectory parameterization. The first two share also the same initialization strategy and termination tolerance. Tab. I shows the statistics for the task-completion time and the runtime of each motion planner. SCS generates the best trajectories: in fact, the average completion time for the overall package-transfer task is about 10 s for SCS, 13 s for the trust-region method, and 15 s for the waypoint-based planner. In other words, SCS allows us to transfer 28% and 50% more packages per unit of time than the trust-region and the waypoint-based planners, respectively. The runtimes of SCS are approximately five times longer than those of the waypoint-based planner, but they remain very low for practical use. The trust region method is roughly three times slower than SCS. The videos of five of these package-transfer tasks are provided as Supplementary Material.

We conclude by emphasizing that the trust-region and the waypoint-based planners are natural baselines for the task considered in this section. The first provides the same completeness guarantees as SCS, designs smooth trajectories, and has relatively low runtimes. The second is widespread in warehouse automation thanks to its good performance and high reliability. In our experience, off-the-shelf nonconvex trajectory optimization faces significant challenges with this package-transfer task: it struggles with the many collision geometries in Fig. 7, relies on handcrafted warm starts, can take seconds to converge (unless we use accelerated hardware [35]), and can also fail to converge. Sampling-based planners can be more reliable, but generate polygonal curves that require additional smoothing. They excel in tasks where finding a collision-free trajectory is the main challenge, and trajectory cost is secondary. However, our package-transfer task presents the opposite challenge.

REFERENCES

[1] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[2] Joshua Bialkowski, Sertac Karaman, and Emilio Frazzoli. Massively parallelizing the RRT and the RRT*. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3513–3518. IEEE, 2011.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[4] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *IEEE International Conference on Robotics and Automation*, pages 1476–1483. IEEE, 2016.

[5] Shao Yuan Chew Chia, Rebecca H Jiang, Bernhard Paus Graesdal, Leslie Pack Kaelbling, and Russ Tedrake. GCS*: Forward heuristic search on implicit graphs of convex sets. *arXiv preprint arXiv:2407.08848*, 2024.

[6] Hongkai Dai, Alexandre Amice, Peter Werner, Annan Zhang, and Russ Tedrake. Certified polyhedral decompositions of collision-free configuration space. *The International Journal of Robotics Research*, 43(9):1322–1341, 2024.

[7] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 109–124. Springer, 2015.

[8] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *IEEE International Conference on Robotics and Automation*, pages 42–49. IEEE, 2015.

[9] Steven Diamond, Reza Takapoui, and Stephen Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

[10] Rida Farouki and V Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, 1988.

[11] Philip E Gill, Walter Murray, and Michael A Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[12] Paul J Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. *arXiv preprint arXiv:2405.12762*, 2024.

[13] Taylor A Howell, Brian E Jackson, and Zachary Manchester. ALTRO: A fast solver for constrained trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[14] Jeffrey Ichnowski, Michael Danielczuk, Jingyi Xu, Vishal Satish, and Ken Goldberg. Gomp: Grasp-optimized motion planning for bin picking. In *IEEE International Conference on Robotics and Automation*, pages 5270–5277. IEEE, 2020.

[15] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 4569–4574. IEEE, 2011.

[16] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE Conference on Decision and Control*, pages 7681–7687. IEEE, 2010.

[17] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

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

[19] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11, Computer Science Department, Iowa State University*, 1998.

[20] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[21] Mirko Leomanni, Gabriele Costante, and Francesco Ferrante. Time-optimal control of a multidimensional integrator chain with applications. *IEEE Control Systems Letters*, 6:2371–2376, 2022.

[22] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[23] Thomas Lipp and Stephen Boyd. Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311, 2014.

[24] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Robotics and Automation Letters*, 2 (3):1688–1695, 2017.

[25] Danylo Malyuta, Taylor P Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behçet Açıkmeşe. Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently. *IEEE Control Systems Magazine*, 42(5):40–113, 2022.

[26] Tobia Marcucci. *Graphs of Convex Sets with Applications to Optimal Control and Motion Planning*. PhD thesis, Massachusetts Institute of Technology, 2024.

[27] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *Science robotics*, 8(84):eadf7843, 2023.

[28] Tobia Marcucci, Parth Nobel, Russ Tedrake, and Stephen Boyd. Fast path planning through large collections of safe boxes. *IEEE Transactions on Robotics*, 40:3795–3811, 2024.

[29] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[30] Savva Morozov, Tobia Marcucci, Alexandre Amice, Bernhard Paus Graesdal, Rohan Bosworth, Pablo A Parrilo, and Russ Tedrake. Multi-query shortest-path problem in graphs of convex sets. *arXiv preprint arXiv:2409.19543*, 2024.

[31] Jia Pan and Dinesh Manocha. GPU-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2):187–200, 2012.

[32] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96:293–320, 2003.

[33] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[34] Xinyue Shen, Steven Diamond, Madeleine Udell, Yuantao Gu, and Stephen Boyd. Disciplined multi-convex programming. In *29th Chinese Control and Decision Conference*, pages 895–900. IEEE, 2017.

[35] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. Curobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation*, pages 8112–8119. IEEE, 2023.

[36] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[37] Russ Tedrake, Ian Manchester, Mark Tobenkin, and John Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[38] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. Motions in microseconds via vectorized sampling-based planning. In *IEEE International Conference on Robotics and Automation*, pages 8749–8756. IEEE, 2024.

[39] Marc Toussaint. Newton methods for k-order markov constrained motion problems. *arXiv preprint arXiv:1407.0414*, 2014.

[40] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.

[41] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

[42] Qianhao Wang, Zhepei Wang, Mingyang Wang, Jialin Ji, Zhichao Han, Tianyue Wu, Rui Jin, Yuman Gao, Chao Xu, and Fei Gao. Fast iterative region inflation for computing large 2-D/3-D convex regions of obstacle-free space. *arXiv preprint arXiv:2403.02977*, 2024.

[43] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2009.

[44] Peter Werner, Alexandre Amice, Tobia Marcucci, Daniela Rus, and Russ Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. In *IEEE International Conference on Robotics and Automation*, pages 10359–10365. IEEE, 2024.

[45] Peter Werner, Thomas Cohn, Rebecca H Jiang, Tim

Seyde, Max Simchowitz, Russ Tedrake, and Daniela Rus. Faster algorithms for growing collision-free convex polytopes in robot configuration space. *arXiv preprint arXiv:2410.12649*, 2024.

[46] Peter Werner, Richard Cheng, Tom Stewart, Russ Tedrake, and Daniela Rus. Superfast configuration-space convex set computation on GPUs for online motion planning. *arXiv preprint arXiv:2504.10783*, 2025.

[47] Yuwei Wu, Igor Spasojevic, Pratik Chaudhari, and Vijay Kumar. Optimal convex cover as collision-free space approximation for trajectory generation. *arXiv preprint arXiv:2406.09631*, 2024.

[48] Lujie Yang, Tobia Marcucci, Pablo A Parrilo, and Russ Tedrake. A new semidefinite relaxation for linear and piecewise-affine optimal control with time scaling. *arXiv preprint arXiv:2504.13170*, 2025.

[49] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

APPENDIX A
FINITE-DIMENSIONAL PROGRAMS

We illustrate the finite-dimensional versions of the programs presented in this paper. Here, candidate trajectories are parameterized using Bézier curves as shown in §VII.

We start from the nonconvex program (7). The constraints of its finite-dimensional counterpart are as follows (the objective is unchanged):

$$\boldsymbol{q}_{1,0} = \boldsymbol{q}_{\text{init}}, \tag{20a}$$
$$\boldsymbol{q}_{I,K} = \boldsymbol{q}_{\text{term}}, \tag{20b}$$
$$\dot{\boldsymbol{q}}_{1,0} = \dot{\boldsymbol{q}}_{I,K-1} = \boldsymbol{0}, \tag{20c}$$
$$\boldsymbol{q}_{i,k} \in \mathcal{Q}_i, \qquad k \le K,\ i \le I, \tag{20d}$$
$$\dot{\boldsymbol{q}}_{i,k} \in T_i \mathcal{V}, \qquad k \le K-1,\ i \le I, \tag{20e}$$
$$\ddot{\boldsymbol{q}}_{i,k} \in T_i^2 \mathcal{A}, \qquad k \le K-2,\ i \le I, \tag{20f}$$
$$T_i > 0, \qquad i \le I, \tag{20g}$$
$$\boldsymbol{q}_{i,K} = \boldsymbol{q}_{i+1,0}, \qquad i \le I-1, \tag{20h}$$
$$\dot{\boldsymbol{q}}_{i,K-1}/T_i = \dot{\boldsymbol{q}}_{i+1,0}/T_{i+1}, \qquad i \le I-1, \tag{20i}$$
$$\dot{\boldsymbol{q}}_{i,k} = K(\boldsymbol{q}_{i,k+1} - \boldsymbol{q}_{i,k}), \qquad k \le K-1,\ i \le I, \tag{20j}$$
$$\ddot{\boldsymbol{q}}_{i,k} = (K-1)(\dot{\boldsymbol{q}}_{i,k+1} - \dot{\boldsymbol{q}}_{i,k}), \qquad k \le K-2,\ i \le I. \tag{20k}$$

For all $i \le I$, the variables here are the traversal times T_i and the control points $\boldsymbol{q}_{i,k}$ for $k \le K$, $\dot{\boldsymbol{q}}_{i,k}$ for $k \le K-1$, and $\ddot{\boldsymbol{q}}_{i,k}$ for $k \le K-2$.

The finite-dimensional version of the subproblem with fixed transition velocities (11) differs from the nonconvex program (20) in just two ways. First, the acceleration constraint (20f) and the traversal-time lower bound (20g) are replaced with the convex conditions

$$\ddot{\boldsymbol{q}}_{i,k} \in \bar{T}_i(2T_i - \bar{T}_i)\mathcal{A}, \qquad k \le K-2,\ i \le I,$$
$$2T_i \ge \bar{T}_i, \qquad i \le I.$$

Second, the continuity constraint (20i) is split into two linear constraints:

$$\dot{\boldsymbol{q}}_{i,K-1} = \boldsymbol{v}_i T_i,\ \dot{\boldsymbol{q}}_{i+1,0} = \boldsymbol{v}_i T_{i+1}, \qquad i \le I-1.$$

The finite-dimensional version of the subproblem with fixed transition points (15) has similar constraints:

$$\begin{aligned}
&\boldsymbol{r}_{1,0} = S_1 \boldsymbol{q}_{\text{init}},\\
&\boldsymbol{r}_{I,K} = S_I \boldsymbol{q}_{\text{term}},\\
&\dot{\boldsymbol{r}}_{1,0} = \dot{\boldsymbol{r}}_{I,K-1} = \boldsymbol{0},\\
&\boldsymbol{r}_{i,k} \in S_i \mathcal{Q}_i, && k \le K,\ i \le I,\\
&\dot{\boldsymbol{r}}_{i,k} \in \mathcal{V}, && k \le K-1,\ i \le I,\\
&\ddot{\boldsymbol{r}}_{i,k} \in \bar{T}_i(2 - \bar{T}_i S_i)\mathcal{A}, && k \le K-2,\ i \le I,\\
&0 < S_i \le 2/\bar{T}_i, && i \le I,\\
&\boldsymbol{r}_{i,K} = \boldsymbol{p}_i S_i,\ \boldsymbol{r}_{i+1,0} = \boldsymbol{p}_i S_{i+1}, && i \le I-1,\\
&\dot{\boldsymbol{r}}_{i,K-1} = \dot{\boldsymbol{r}}_{i+1,0}, && i \le I-1,\\
&\dot{\boldsymbol{r}}_{i,k} = K(\boldsymbol{r}_{i,k+1} - \boldsymbol{r}_{i,k}), && k \le K-1,\ i \le I,\\
&\ddot{\boldsymbol{r}}_{i,k} = (K-1)(\dot{\boldsymbol{r}}_{i,k+1} - \dot{\boldsymbol{r}}_{i,k}), && k \le K-2,\ i \le I.
\end{aligned}$$

For $i \le I$, here the decision variables are the traversal times T_i and the control points $\boldsymbol{r}_{i,k}$ for $k \le K$, $\dot{\boldsymbol{r}}_{i,k}$ for $k \le K-1$, and $\ddot{\boldsymbol{r}}_{i,k}$ for $k \le K-2$.

The initialization phase requires solving problem (17) repeatedly. Using the Bézier parameterization, the constraints of this problem become

$$\begin{aligned}
&\boldsymbol{r}_{i,0} = S_i \boldsymbol{p}_{i-1},\\
&\boldsymbol{r}_{i,K} = S_i \boldsymbol{p}_i,\\
&\dot{\boldsymbol{r}}_{i,0} = \dot{\boldsymbol{r}}_{i,K-1} = \boldsymbol{0},\\
&\dot{\boldsymbol{r}}_{i,k} \in \mathcal{V}, && k \le K-1,\\
&\ddot{\boldsymbol{r}}_{i,k} \in T_i \mathcal{A}, && k \le K-2,\\
&T_i \ge 1/S_i,\ S_i > 0,\\
&\dot{\boldsymbol{r}}_{i,k} = K(\boldsymbol{r}_{i,k+1} - \boldsymbol{r}_{i,k}), && k \le K-1,\\
&\ddot{\boldsymbol{r}}_{i,k} = (K-1)(\dot{\boldsymbol{r}}_{i,k+1} - \dot{\boldsymbol{r}}_{i,k}), && k \le K-2,
\end{aligned}$$

with variables T_i, S_i, $\boldsymbol{r}_{i,k}$ for $k \le K$, $\dot{\boldsymbol{r}}_{i,k}$ for $k \le K-1$, and $\ddot{\boldsymbol{r}}_{i,k}$ for $k \le K-2$.

APPENDIX B
PARAMETRIC PROBLEM FOR RUNTIME ANALYSIS

We detail the construction of the problem in §X-B, which is parametric in the number I of safe sets, the number m of facets of each safe set, and the space dimension n.

We let $\boldsymbol{x}_0 = \boldsymbol{0} \in \mathbb{R}^n$. Then, for all $i \le I$, we define the points

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \boldsymbol{e}_j,$$

where $\boldsymbol{e}_j$ is the jth element of the standard basis, and j is the reminder of i/n. In other words, the point $\boldsymbol{x}_i$ is obtained by shifting $\boldsymbol{x}_{i-1}$ along the jth dimension by one. The sequence of points $\boldsymbol{x}_0, \dots, \boldsymbol{x}_I$ is then an n-dimensional staircase with I links.

For $i \le I$, we let $\mathcal{E}_i \subset \mathbb{R}^n$ be an ellipsoid centered around the line segment that connects $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$. The main axis

of $\mathcal{E}_i$ is aligned with the vector $\boldsymbol{x}_i - \boldsymbol{x}_{i-1}$ and has length $4/3$, while all the other axes have length $1/3$. Each safe set $\mathcal{Q}_i$ is a conservative polytopic approximation with m facets of the corresponding ellipsoid $\mathcal{E}_i$. In $n = 2$ dimensions, this construction over-approximates the unit circle with a regular polytope, which is then mapped to the polytope $\mathcal{Q}_i$ through the same affine transformation that maps the unit circle to the ellipse $\mathcal{E}_i$. We let the initial point be $\boldsymbol{q}_{\text{init}} = \boldsymbol{x}_0$ and the terminal point be $\boldsymbol{q}_{\text{term}} = \boldsymbol{x}_I$. The constraint sets $\mathcal{V}$ and $\mathcal{A}$ are spheres centered at the origin of radius 10 and 1, respectively.

APPENDIX C
TRUST-REGION METHOD FOR MINIMUM-TIME PROBLEMS

We briefly describe how the trust-region method from [28, §V] can be adapted for minimum-time problems.

We start from problem (7), whose only nonconvex constraints are the velocity continuity (4b) and the acceleration constraint (5c). As in the problem with fixed transition points (15), we introduce the variables

$$\dot{\boldsymbol{r}}_i = \dot{\boldsymbol{q}}_i / T_i, \qquad\qquad i \leq I. \tag{21}$$

Recall that the derivatives $\ddot{\boldsymbol{r}}_i$ of these functions correspond to $\ddot{\boldsymbol{q}}_i / T_i$. Using these additional variables, the velocity continuity (4b) becomes a linear constraint

$$\dot{\boldsymbol{r}}_i(1) = \dot{\boldsymbol{r}}_{i+1}(0), \qquad\qquad i \leq I - 1,$$

while the acceleration constraint (5c) becomes a convex constraint of the kind discussed in §I-B:

$$\ddot{\boldsymbol{r}}_i(s) \in T_i \mathcal{A}, \qquad\qquad s \in [0, 1],\ i \leq I.$$

This yields a program whose nonconvexity is due exclusively to the equality constraint (21).

Following [28, § V-C], we solve the nonconvex program by alternating between two convex programs: a "tangent" and "projection" program. In the tangent program, we linearize constraint (21) around the current solution, and try to improve the trajectory shape and timing jointly. The linearization error is controlled by a trust-region constraint whose size and shape are defined as in [28, § V-C]. Because of the linearization error, the solution of the tangent program might not be feasible for the original nonconvex program. Therefore, in the projection step, we fix the new trajectory timing and solve the resulting convex program, hoping to obtain a feasible solution with cost lower than the current one. The numerical solution of these infinite-dimensional convex subproblems follows the steps in §VII.