Fast Path Planning Through Large Collections of Safe Boxes

Tobia Marcucci, Parth Nobel, Russ Tedrake, *Member, IEEE*, and Stephen Boyd, *Fellow, IEEE*

***Abstract*—We present a fast algorithm for the design of smooth paths (or trajectories) that are constrained to lie in a collection of axis-aligned boxes. We consider the case where the number of these safe boxes is large, and basic preprocessing of them (such as finding their intersections) can be done offline. At runtime, we quickly generate a smooth path between given initial and terminal positions. Our algorithm designs trajectories that are guaranteed to be safe at all times, and detects infeasibility whenever such a trajectory does not exist. Our algorithm is based on two subproblems that we can solve very efficiently: finding a shortest path in a weighted graph, and solving (multiple) convex optimal-control problems. We demonstrate the proposed path planner on large-scale numerical examples, and we provide an efficient open-source software implementation, `fastpathplanning`.**

***Index Terms*—Collision avoidance, convex optimization, motion and path planning, optimization and optimal control.**

I. Introduction

PATH planning is a problem at the core of almost any autonomous system. Driverless cars, drones, autonomous aircraft, robot manipulators, and legged robots are just a few examples of systems that rely on a path-planning algorithm to navigate in their environment. Path-planning problems can be challenging on many fronts. The environment can be dynamic, i.e., change over time, or uncertain because of noisy sensor measurements [1], [2], [3], [4]. Computation might be subject to strict real-time requirements [5], [6], [7]. Interactions between multiple robots without central coordination can lead to game-theoretic problems [8], [9], [10], [11]. In this article, we consider problems where a single smooth path needs to be found through an environment that is fully known and static, but potentially very large and complicated to navigate through. For example, this is the case for a mobile robot that transports products in a large warehouse [12], a drone that delivers packages in a mapped environment [13] (see also Fig. 1), a quadruped that inspects an industrial plant [14], [15], or a robot arm that unloads packages from many identical shelves and sorts them into bins [16].

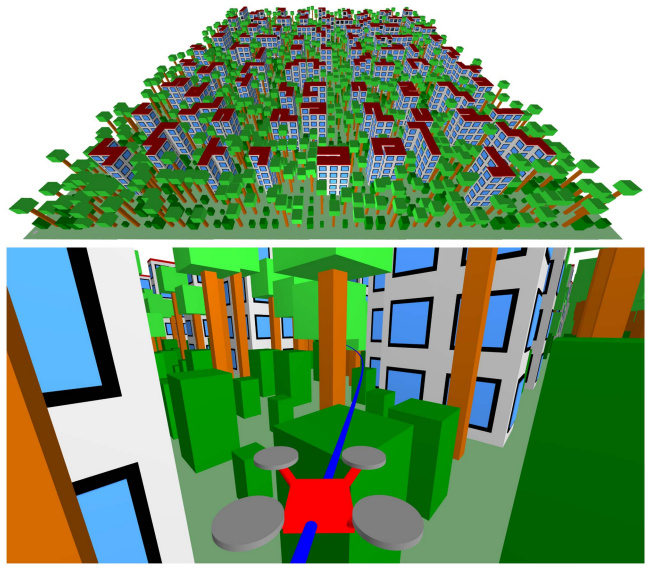

Fig. 1. Path planning for a quadrotor flying through a simulated village. *Top.* The village, composed of buildings, trees, and bushes. The free space is decomposed using more than ten thousand safe boxes. *Bottom.* A snapshot of the quadrotor flight. The smooth path connects two opposite corners of the village and is guaranteed to be collision free at all times. The online planning time is only a few seconds.

Like previous methods [17], [18], we assume that the environment (or configuration space) is described as a collection of safe sets, through which our robot can move freely without colliding with obstacles. Our problem is to find a smooth path that is contained in the union of the safe sets, and connects given initial and terminal points. We consider the case where the safe sets are axis-aligned boxes and large in number (thousands or tens of thousands). Focusing on boxes allows us to substantially accelerate multiple parts of our algorithm. While the large number of boxes is necessary to adequately represent even highly complex robot environments. Note that the decomposition of the complex spaces into boxes can be approximate (conservative), and can be computed using simple variations of existing algorithms [19], [20], [21], [22], [23], as well as methods tailored

Manuscript received 2 January 2024; revised 7 June 2024; accepted 21 July 2024. Date of publication 26 July 2024; date of current version 12 August 2024. This work was supported by the Office of Naval Research (ONR) under Grant N00014-22-1-2121. Indeed, this work is a direct consequence of the collaboration fostered by this grant. The work of Parth Nobel was supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant DGE-1656518. The work of Stephen Boyd was supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR. This article was recommended for publication by Associate Editor D. Panagou and Editor N. Amato upon evaluation of the reviewers' comments. *(Corresponding author: Tobia Marcucci.)*

Tobia Marcucci and Russ Tedrake are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: tobiam@mit.edu).

Parth Nobel and Stephen Boyd are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2024.3434168, provided by the authors.

Digital Object Identifier 10.1109/TRO.2024.3434168

to the configurations spaces of kinematic trees [24], [25], [26], [27].

Our path-planning method is composed of an offline and an online part. In the offline preprocessing, we construct a graph that stores the intersections of the safe boxes and solve a convex program to label the edges of this graph with approximate distances. These computations are done only once, since the environment is static, and they require from a fraction of a second to a few tens of seconds, depending on the problem size. In the online part, we first use the graph constructed offline to design a polygonal curve of short length that connects the given initial and terminal points. Then, we solve a sequence of convex optimal-control problems to transform the polygonal curve into a smooth path that minimizes a given objective function. The online runtimes of our algorithm are dominated by these control problems, which, however, are solvable in a time that increases only linearly with the number of boxes traversed by the path [28]. This results in online planning times on the scale of a hundredth of a second for medium-size problems and of a second for very large problems. Consider that, for a problem like the quadrotor flight in Fig. 1, existing techniques take seconds to optimize a path through a few tens of safe boxes [18]. Within the same time, our planner designs a path through more than 10 000 boxes.

The proposed algorithm is *complete:* it always finds a smooth path connecting the initial and final positions if such a path exists, and certifies infeasibility of the planning problem otherwise. In addition, by using Bézier curves for the path parameterization, our smooth trajectories are guaranteed to be safe at all times, and not only at a finite number of sample points. Our method is *heuristic:* although it designs paths that have typically low cost, it is not guaranteed to solve the planning problem optimally, or within a fixed percentage of the optimum.

The techniques of this article are implemented in a companion open-source package, `fastpathplanning`.

A. Related Work

A wide variety of path-planning algorithms has been developed over the last 50 years. An excellent overview of the techniques available in the literature can be found in [29, Part 2]. Here, we review the methods that are most closely related to ours.

The closest approach to the one presented here is graphs of convex sets (GCS) from [18]. Similarly to our method, GCS designs trajectories through collections of safe sets that are preprocessed to form a graph. Leveraging the optimization framework from [30], it formulates a tight convex relaxation of the planning problem and it recovers a collision-free trajectory using a cheap rounding strategy. Thanks to this workflow, GCS also provides tight optimality bounds for the trajectories it designs. On the other hand, by trying to solve the planning problem through a single convex program, GCS has a few limitations. First, at present, GCS does not efficiently handle costs or constraints on the path acceleration and higher derivatives, which are a central component of the problems analyzed in this article. Second, GCS does not scale to the very large numbers of safe sets considered here. The proposed algorithm is different in spirit from GCS: it leverages fast graph search to heuristically solve the discrete part of the planning problem and, only at a later stage, it uses convex optimization to shape the continuous path. This division sacrifices the optimality guarantees but retains the algorithm completeness, and it allows us to quickly find paths of low cost for planning problems of very large scale.

A natural approach for designing smooth paths that avoid obstacles optimally is mixed-integer programming. Earlier mixed-integer formulations dealt with polyhedral obstacles, and used a binary variable for each facet of each obstacle to enforce the constraint that a trajectory point is not in collision [31], [32], [33]. Conversely, the formulation from [17] leverages the algorithm from [22] to cover (all or part of) the collision-free space with convex regions, and ensures safety by forcing each trajectory segment to lie entirely in at least one convex region. This makes the mixed-integer program more efficient, since each trajectory segment requires only one binary variable per safe set. The path planning problem considered in [17] is essentially the same as ours, but our algorithm can solve dramatically larger problems in a fraction of the time (see the comparison in Section VII-D).

Two popular approaches for collision-free path planning are local nonconvex optimization [34], [35], [36], [37], [38], [39] and sampling-based algorithms [40], [41], [42]. The former methods can handle kinematic and dynamic constraints, but suffer from local minima and can often fail in finding a feasible trajectory if the environment has many obstacles. Although multiple strategies have been proposed to mitigate this issue [43], [44], [45], sampling-based algorithms are typically more reliable when the environment is complex (in fact, they are *probabilistically complete*). However, sampling-based methods can struggle in high dimensions and are less suitable for the design of smooth paths. Similar to [18], the approach we propose here can be thought of as a generalization of sampling-based algorithms, where collision-free samples are substituted with collision-free sets. Instead of sampling the environment densely, we fill it with large safe boxes. This reduces the combinatorial complexity and allows us to plan through the open space using efficient convex optimization [46]. (Specifically, given the offline preprocessing of the safe boxes performed by our method, in this article, we target the same applications as *multiple-query* sampling-based planners, such as probabilistic roadmaps [40].)

Decompositions of the environment into safe sets (or cells) are also common in feedback motion planning. There, a feedback plan is constructed by composing a navigation function with a piecewise-smooth vector field: the former decides the discrete transitions between the cells and the latter causes all states in a cell to flow into the next cell [29, Sec. 8.4]. In a similar fashion, the method in [47] leverages discrete abstractions [48] to generate provably correct control policies for planar robots moving in polygonal environments. In robust motion planning, funnels [37], [49], tubes [50], barrier functions [9], [51], and positively invariant sets [52], [53], [54], [55] are frequently used to abstract away the continuous dynamics and reduce the planning problem to a discrete search. While similar in spirit to our algorithm, the methods presented in those papers consider problems of different nature from our. We do not aim to synthesize a feedback policy, nor do we reason about dynamics

and disturbances explicitly. Our goal is to design safe smooth paths of low cost, and the main challenge in our problem is the environment complexity (i.e., the number of safe boxes).

Last, in this article, we use Bézier curves to parameterize smooth paths. These curves enjoy several properties that make them particularly well suited for convex optimization, and have been widely used in path planning and optimal control over the last 15 years (see, e.g., [56], [57], [58]).

B. Outline

The rest of this article is organized as follows. In Section II, we state the path-planning problem and give a high-level overview of our algorithm. The algorithm can be broken down into three parts, one offline and two online. The offline preprocessing, which does not use either the endpoints of the path or the specific objective function, is described in Section III. The first online phase, illustrated in Section IV, finds a polygonal curve of short length that is contained in the safe boxes and connects the given path endpoints. The second online phase, described in Section V, transforms the polygonal curve into a safe smooth path of low cost. In Section VI, we summarize the main properties of our path planner. In Section VII, we evaluate the performance of our algorithm through multiple numerical experiments. Finally, Section VIII concludes this article.

II. Path Planning

In this section, we state the path-planning problem and we describe at a high-level the components of our algorithm.

A. Problem Statement

We consider the design of a smooth path in $\mathbf{R}^d$ from a given initial point $p^{\text{init}} \in \mathbf{R}^d$ to a given terminal point $p^{\text{term}} \in \mathbf{R}^d$. We represent the path as the function $p : [0, T] \to \mathbf{R}^d$, where T is the time taken to traverse the path. In addition to the initial and terminal-point constraints

$$p(0) = p^{\text{init}}, \qquad p(T) = p^{\text{term}}$$

we require that the path stay in a given set $\mathcal{S} \subset \mathbf{R}^d$ of safe points

$$p(t) \in \mathcal{S}, \quad t \in [0, T].$$

We assume that the safe set $\mathcal{S}$ is a union of K axis-aligned boxes

$$\mathcal{S} = \bigcup_{k=1}^{K} \mathcal{B}_k$$

with

$$\mathcal{B}_k = \{x \in \mathbf{R}^d \mid l_k \le x \le u_k\}, \quad k = 1, \ldots, K.$$

Here, the inequalities are elementwise, and the box bounds satisfy $l_k \le u_k$ for $k = 1, \ldots, K$.

We considers paths with D continuous derivatives, and we take our objective to be a weighted sum of the squared L_2 norm of these derivatives

$$J = \sum_{i=1}^{D} \alpha_i \int_0^T \|p^{(i)}(t)\|_2^2 \, dt \tag{1}$$

where $p^{(i)}$ denotes the ith derivative of p, and α_i are nonnegative weights.

The path-planning problem is

$$\begin{array}{ll} \text{minimize} & J \\ \text{subject to} & p(0) = p^{\text{init}}, \quad p(T) = p^{\text{term}} \\ & p(t) \in \mathcal{S}, \quad t \in [0, T]. \end{array} \tag{2}$$

The optimization variable is the path p. The problem data are the objective weights α_i, the final time T, the initial and terminal points p^{init} and p^{term}, and the safe set $\mathcal{S}$ (specified by the box bounds l_k and u_k). This statement includes only the essential components of a path-planning problem. For example, here we specify the initial and terminal positions, but do not constrain the initial and terminal derivatives. In Section VIII, we will discuss multiple of these simple extensions, and highlight the necessary modifications to our method.

The path-planning problem (2) is infinite dimensional, but we will restrict candidate paths to piecewise Bézier curves, which are parameterized by a finite set of control points.

B. Safety Map

Problem (2) has convex quadratic objective, two linear equality constraints, and the safety constraint, which, in general, is not convex. The safety constraint is an infinite collection of disjunctive constraints, that force the point $p(t)$, for each $t \in [0, T]$, to lie in at least one of the boxes $\mathcal{B}_k$. Ensuring safety of a path p is then equivalent to finding a function $s : [0, T] \to \{1, \ldots, K\}$ such that

$$p(t) \in \mathcal{B}_{s(t)}, \quad t \in [0, T].$$

The value $s(t) \in \{1, \ldots, K\}$ represents the choice of a safe box for the path at time t, and the overall function s can be thought of as a *safety map* for our path.

Our safety maps will have a finite number of transitions, i.e., will be of the form

$$s(t) = \begin{cases} s_1 & t \in [t_0, t_1] \\ s_2 & t \in (t_1, t_2] \\ \vdots & \\ s_N & t \in (t_{N-1}, t_N] \end{cases} \tag{3}$$

where $0 = t_0 < t_1 < \cdots < t_N = T$. We will refer to $s_1, \ldots, s_N$ as the *box sequence* of the safety map s, and to $T_1 = t_1 - t_0, \ldots, T_N = t_N - t_{N-1}$ as the *traversal times*.

In terms of the safety map, the path-planning problem is

$$\begin{array}{ll} \text{minimize} & J \\ \text{subject to} & p(0) = p^{\text{init}}, \quad p(T) = p^{\text{term}} \\ & l_{s(t)} \le p(t) \le u_{s(t)}, \quad t \in [0, T] \end{array} \tag{4}$$

where the variables are the path p and the safety map s. We observe that if the box sequence $s_1, \ldots, s_N$ is fixed, problem (4) reduces to a nonconvex but continuous optimal-control problem, with the path p and the traversal times $T_1, \ldots, T_N$ as decision variables. If we also fix the traversal times, then the safety map is entirely specified, and problem (4) becomes a convex optimal-control problem with quadratic objective and linear constraints.

C. Feasibility

We will say that a safety map is *feasible* if it satisfies

$$\begin{aligned} &p^{\text{init}} \in \mathcal{B}_{s_1}, \quad p^{\text{term}} \in \mathcal{B}_{s_N} \\ &\mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}} \neq \emptyset, \quad j = 1, \ldots, N-1. \end{aligned} \tag{5}$$

The first condition says that the first and last boxes in the box sequence cover the initial and terminal points, respectively. The second condition requires that every two consecutive boxes intersect; thus the box sequence can be traversed by a continuous path.

Importantly, the path-planning problem (4) is feasible if and only if a feasible safety map exists. To see this, note that if a path p and a safety map s are feasible for (4), then the safety map must satisfy both conditions in (5). (In particular, the second condition follows from the continuity of p, which ensures that $p(t_j) \in \mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$ for all $j = 1, \ldots, N-1$.) For the other direction, suppose a safety map is feasible, and let $p_j \in \mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$ for $j = 1, \ldots, N-1$. Then, the polygonal curve with nodes $p^{\text{init}} = p_0$, $p_1, \ldots, p_{N-1}, p_N = p^{\text{term}}$ is entirely contained in the safe set $\mathcal{S}$. Through the following steps, we construct a path p that has D continuous derivatives, and moves along the polygonal curve (and so is safe). We select any times $0 = t_0 < t_1 < \cdots < t_N = T$. We choose any smooth time parameterization of the polygonal curve that satisfies the interpolation conditions $p(t_j) = p_j$ for $j = 0, \ldots, N$, as well as the derivative constraints $p^{(i)}(t_j) = 0$ for $i = 1, \ldots, D$ and $j = 1, \ldots, N-1$. While the polygonal curve has kinks, the path p is differentiable D times since it comes to a full stop at each kink. By pairing this path with the feasible safety map, we have a feasible solution of problem (4).

D. Method Outline

We give here a high-level description of the three phases in our path-planning algorithm, with the details illustrated in future sections.

Offline Preprocessing: The offline preprocessing uses only the safe set $\mathcal{S}$, i.e., the safe boxes $\mathcal{B}_1, \ldots, \mathcal{B}_K$. In this phase, we construct a *line graph* G whose vertices correspond to points in the intersection of two boxes, and whose edges connect pairs of points that lie in the same box. When considered as a subset of $\mathbf{R}^d$, this graph lies entirely in the safe set, and it can be used to quickly design safe polygonal curves that connect given initial and terminal points. The points associated with the vertices are called *representative points*, and are optimized to minimize the total Euclidean length of the edges in the line graph. This serves as a heuristic to reduce the length of the polygonal curves.

procedure OFFLINE PREPROCESSING
  compute intersections of safe boxes $\mathcal{B}_1, \ldots, \mathcal{B}_K$
  construct line graph G
  optimize representative points
end procedure

Polygonal Phase: Here, we find a polygonal curve $\mathcal{C}$ that connects p^{init} to p^{term}, is entirely contained in the safe set $\mathcal{S}$, and has small length. The curve is initialized by solving a shortest-path problem in the line graph constructed offline. Then, it is shortened through an iterative process, where we alternate between minimizing the curve length for a fixed box sequence, and updating the box sequence for a fixed polygonal curve.

procedure POLYGONAL PHASE
  connect p^{init} to p^{term} with safe polygonal curve $\mathcal{C}$
  while not converged **do**
    fix box sequence $s_1, \ldots, s_N$ and shorten curve
    fix curve and improve box sequence
  end while
end procedure

Smooth Phase: In this phase, we freeze the box sequence $s_1, \ldots, s_N$ identified in the polygonal phase, and traversed by the curve $\mathcal{C}$. As observed above, this reduces problem (4) to a continuous but nonconvex optimal-control problem. To solve this control problem, we first use a simple heuristic to estimate initial traversal times $T_1, \ldots, T_N$. Then, we alternate between two convex optimal-control problems. In the first, we fix the traversal times (thus we specify the whole safety map s) and optimize the shape of the path. In the second, we attempt to improve the traversal times by solving a local convex approximation of the nonconvex optimal-control problem.

procedure SMOOTH PHASE
  fix box sequence $s_1, \ldots, s_N$
  estimate traversal times $T_1, \ldots, T_N$
  while not converged **do**
    fix traversal times and optimize path p
    attempt improvement of traversal times
  end while
end procedure

III. OFFLINE PREPROCESSING

In this section, we describe the offline part of our algorithm. The steps below are also illustrated through a simple example at the end of the section.

A. Line Graph

We start by computing the line graph associated with the safe boxes. The vertices of this graph are pairs of safe boxes that intersect, and the edges connect pairs of intersections that share a box. Formally, the line graph is an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with vertices

$$\mathcal{V} = \{\{k, l\} \subseteq \{1, \ldots, K\} \mid \mathcal{B}_k \cap \mathcal{B}_l \neq \emptyset,\ k \neq l\}$$

and edges

$$\mathcal{E} = \{\{v, w\} \subseteq \mathcal{V} \mid v \cap w \neq \emptyset,\ v \neq w\}.$$

The name line graph is motivated by the fact that G can be equivalently defined as the line graph of the *intersection graph* of our collection of boxes.

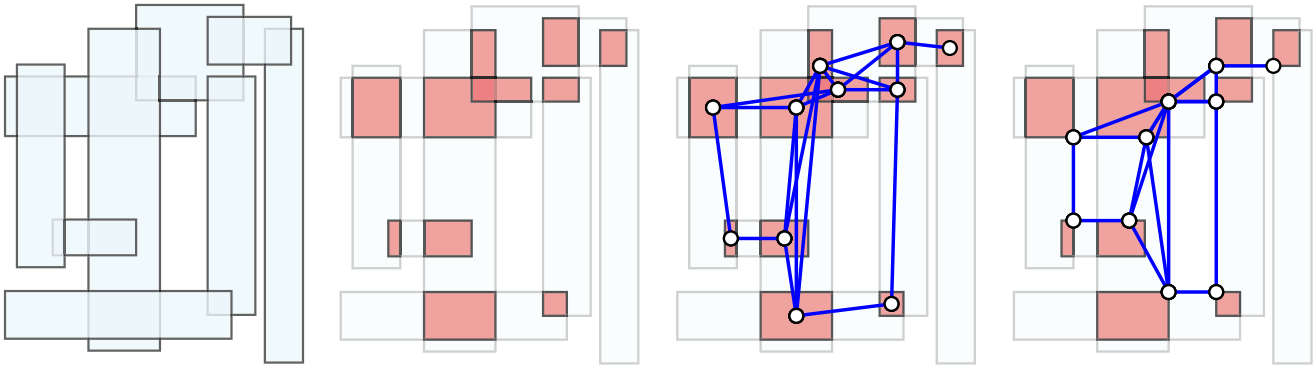

Fig. 2. Offline preprocessing of the safe boxes. *Left.* Safe boxes. *Center left.* Pairwise intersections of the safe boxes. *Center right.* Line graph, with vertices in the box intersections and edges connecting intersections that share a box. *Right.* Line graph with optimized representative points.

The line graph allows us to efficiently construct polygonal curves that are guaranteed to be safe. Consider a path in the line graph. For each vertex in this path, choose a point in $\mathbf{R}^d$ in the corresponding box intersection. Then form the polygonal curve passing through these points. Each line segment in this curve is associated with an edge in the line graph, and therefore with a safe box. By construction, this safe box contains the line segment entirely. It follows that the whole polygonal curve is safe.

Since computing the intersection of two boxes is a trivial operation, we can construct the line graph G very efficiently, even when the number K of boxes is very large. Our implementation is based on the technique from [59, Sec. 2].

B. Representative Points

Our next step is to choose a representative point for each vertex of the line graph, i.e., for each pair of intersecting boxes. As a heuristic method to shorten the polygonal curves constructed as described above, we position these points close to their neighbors in the line graph. More formally, denoting with $x_v \in \mathbf{R}^d$ the representative point of vertex $v \in \mathcal{V}$, we minimize the sum of the Euclidean distances between all pairs of representative points that are connected by an edge

$$\begin{array}{ll} \text{minimize} & \sum_{\{v,w\}\in\mathcal{E}} \|x_v - x_w\|_2 \\ \text{subject to} & x_v \in \mathcal{B}_k \cap \mathcal{B}_l, \quad v = \{k,l\} \in \mathcal{V}. \end{array} \tag{6}$$

Here, the variables are the representative points x_v, $v \in \mathcal{V}$. Each of these points is constrained in the corresponding box intersection, which is itself an axis-aligned box. This is a convex optimization problem that can be represented as a second-order cone program (SOCP) and efficiently solved [46, Sec. 4.4.2], [60].

After optimizing the position of the representative points x_v as in (6), each edge $\{v, w\}$ of the line graph is assigned the weight $\|x_v - x_w\|_2$.

C. Example

We illustrate the offline preprocessing on a small problem that will serve as a running example throughout the article. This problem has $K = 9$ safe boxes in $d = 2$ dimensions and is depicted in Fig. 2. The left figure shows the safe boxes, and the center left figure shows their intersections (with some overlapping when more than two boxes intersect). These intersections correspond to the $|\mathcal{V}| = 11$ vertices of the line graph. In the center right figure, we show the $|\mathcal{E}| = 20$ edges of the line graph as line segments connecting the centers of the box intersections. The right figure shows the optimized representative points, which minimize the total Euclidean distance over the edges of the line graph, i.e., a solution of (6). Note that some of the 20 edges overlap in this figure. Observe also that the line graph, considered as a subset of $\mathbf{R}^2$, is entirely contained in the safe set, since each edge is in at least one safe box.

IV. Polygonal Phase

We now describe the first online phase of our algorithm, where we design a safe polygonal curve $\mathcal{C}$ of short length that connects p^{init} to p^{term}. An illustration of the steps below can be found at the end of the section, where we continue our running example.

A. Shortest-Path Problem

We use the line graph G to initialize the polygonal curve $\mathcal{C}$. We augment the line graph with two new vertices with representative points p^{init} and p^{term}. An edge is added between p^{init} and all the intersections of safe boxes that contain p^{init}, i.e., all the vertices $\{k, l\} \in \mathcal{V}$ such that $p^{\text{init}} \in \mathcal{B}_k$ or $p^{\text{init}} \in \mathcal{B}_l$. An analogous operation is done for p^{term}. As for the other edges in the line graph, these new edges are assigned a weight equal to the Euclidean distance between the representative points that they connect. We then find a shortest path from the initial point to the terminal point, and recover an initial polygonal curve $\mathcal{C}$ by connecting the representative points along this path. As noted above, this curve is safe because each of its segments is contained in a safe box.

This shortest-path step determines whether or not our path-planning problem is feasible. If there is no path in the augmented line graph between the vertices associated with p^{init} and p^{term}, then the path-planning problem (4) is infeasible. Conversely, if there is a path between these two vertices, then the path-planning problem is feasible since a feasible trajectory can be constructed as in Section II-C.

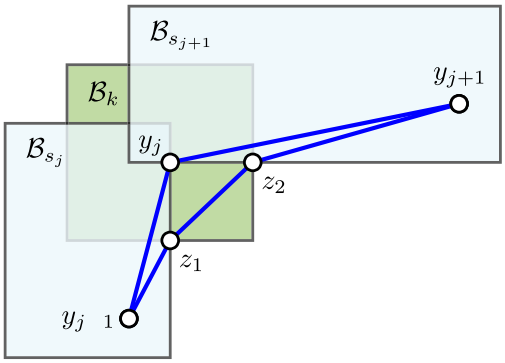

Fig. 3. Shortening of the polygonal curve $\mathcal{C}$ through the insertion of a new box ($\mathcal{B}_k$ in green) in the current box sequence.

The problem of identifying all the safe boxes that contain the initial and terminal points is known as *stabbing problem* and, given the precomputations done to construct the line graph, it takes negligible time [59]. Using an optimized implementation of Dijkstra's algorithm (e.g., the one provided by `scipy` [61]), finding a shortest path is also very fast.

B. Shortening of the Polygonal Curve

Thanks to the optimization of the representative points in (6), our initial polygonal curve $\mathcal{C}$ is typically short. However, the online knowledge of the initial and terminal points, which were unknown during the preprocessing, can allow us to shorten this curve further. This is done iteratively: we alternate between solving a convex program that minimizes the curve length for a fixed box sequence, and improving the box sequence for a fixed polygonal curve.

Optimization of the Polygonal Curve: Denote with $\mathcal{C}$ the curve at the current iteration (initialized with the solution of the shortest-path problem). Let N be the number of segments in $\mathcal{C}$ and $y_0, \dots, y_N \in \mathbf{R}^d$ be the curve nodes, with $y_0 = p^{\text{init}}$ and $y_N = p^{\text{term}}$. For $j = 1, \dots, N$, let also s_j be the index of the safe box that covers the line segment between y_{j-1} and y_j. We fix the box sequence $s_1, \dots, s_N$ traversed by the current curve, and we optimize the position of the nodes y_j so that the length of the curve is minimized. This leads to the problem

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{N} \|y_j - y_{j-1}\|_2 \\ \text{subject to} & y_0 = p^{\text{init}}, \quad y_N = p^{\text{term}} \\ & y_j \in \mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}, \quad j = 1, \dots, N-1 \end{array} \tag{7}$$

with variables $y_0, \dots, y_N$. This is a small SOCP with banded constraints that can be solved very efficiently, in time that is only linear in the number N of segments [28].

The optimal nodes from (7) define our new curve $\mathcal{C}$. We will assume that these nodes are distinct, since if two nodes coincide we can always eliminate one.

Improvement of the Box Sequence: After solving problem (7), the nodes $y_0, \dots, y_N$ minimize the curve length for the given box sequence $s_1, \dots, s_N$. However, as Fig. 3 illustrates, the insertion of a new box can potentially give us room to further shorten our polygonal curve. In our second step, we seek a new sequence of boxes that contains the current curve and is guaranteed to yield a length decrease. Since our safe sets are boxes, this step will be extremely quick.

For all $j = 1, \dots, N-1$, we solve a stabbing problem to find all the boxes $\mathcal{B}_k$ that contain the curve node y_j. Then, we consider inserting the index k between s_j and s_{j+1} in our box sequence. As shown in Fig. 3, this insertion leads to a new instance of problem (7), where the variable y_j is replaced by two variables

$$z_1 \in \mathcal{B}_{s_j} \cap \mathcal{B}_k, \qquad z_2 \in \mathcal{B}_k \cap \mathcal{B}_{s_{j+1}}.$$

Choosing $z_1 = z_2 = y_j$ gives us a feasible solution of this new instance of (7), and does not change the length of our curve $\mathcal{C}$. Therefore, the insertion of $\mathcal{B}_k$ leads to a shorter curve if and only if this feasible solution is not optimal.

We fix $z_1 = z_2 = y_j$ and check if the optimality conditions of the new instance of (7) can be satisfied. As explained in Appendix, this check reduces to finding a vector $\lambda \in \mathbf{R}^d$ that satisfies the following inequalities:

$$\begin{array}{lll} \|\lambda\|_2 \le 1, & L_1(\lambda - \lambda_1) \ge 0, & L_2(\lambda - \lambda_2) \le 0 \\ & U_1(\lambda - \lambda_1) \le 0, & U_2(\lambda - \lambda_2) \ge 0. \end{array} \tag{8}$$

Here, the vectors $\lambda_1,\ \lambda_2 \in \mathbf{R}^d$ are fixed and given by

$$\lambda_1 = \frac{y_j - y_{j-1}}{\|y_j - y_{j-1}\|_2}, \qquad \lambda_2 = \frac{y_{j+1} - y_j}{\|y_{j+1} - y_j\|_2}.$$

The matrices L_1 and U_1 select the indices of the inactive inequalities in the box constraint $y_j \in \mathcal{B}_{s_j} \cap \mathcal{B}_k$ (L_1 for the lower bounds and U_1 for the upper bounds). Similarly, L_2 and U_2 select the inactive inequalities in $y_j \in \mathcal{B}_k \cap \mathcal{B}_{s_{j+1}}$.

Checking if the inequalities in (8) are satisfiable is very quick. In fact, since L_1, L_2, U_1, and U_2 are selection matrices, the corresponding inequalities in (8) simply impose bounds on a subset of the entries of λ. We express these bounds as $c_1 \le \lambda \le c_2$, for two suitable vectors $c_1 \in (\mathbf{R} \cup \{-\infty\})^d$ and $c_2 \in (\mathbf{R} \cup \{\infty\})^d$. Then, the vector λ of minimum Euclidean norm that lies within these bounds can be computed as

$$\lambda^\star = \min\{c_2, \max\{c_1, 0\}\} \tag{9}$$

where the minimum and maximum are elementwise. We conclude that $\lambda^\star$ has norm greater than one if and only if the system of inequalities (8) has no solution, which is equivalent to the insertion of the box $\mathcal{B}_k$ shortening our curve $\mathcal{C}$.

For each index $j = 1, \dots, N-1$ such that the norm of $\lambda^\star$ is greater than one, we insert a new box in our sequence. If multiple boxes satisfy this condition for the same index j, we select one for which the norm of $\lambda^\star$ is largest (this heuristic is motivated in Sec. A). After updating the box sequence, we optimize the curve $\mathcal{C}$ by solving the new instance of problem (7). This process is iterated until the condition above fails for every curve node j and every box k.

C. Example

Fig. 4 continues our running example, and illustrates the construction of the polygonal curve. The initial position p^{init} and terminal position p^{term} are shown as black disks in the bottom

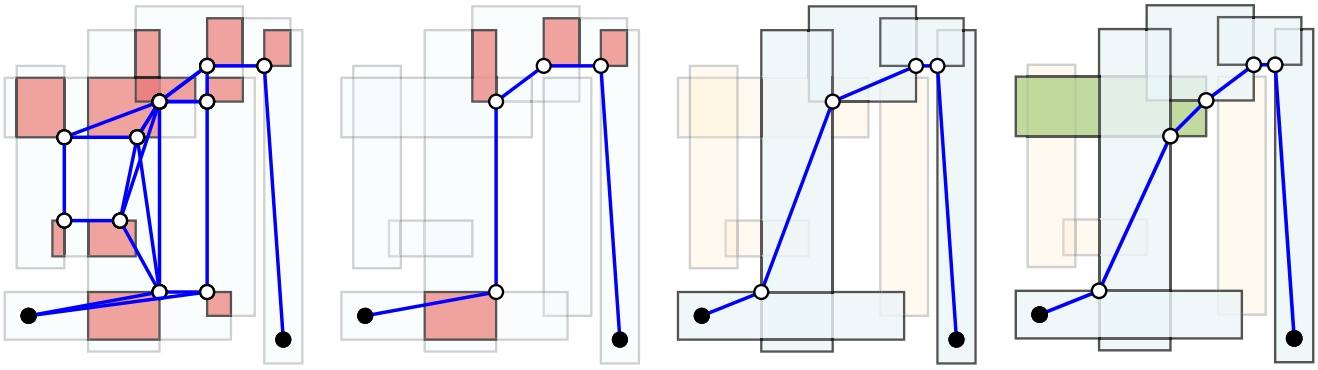

Fig. 4. Polygonal phase of the algorithm. *Left.* Line graph augmented with p^{init} and p^{term}, shown as black disks. *Center left.* Shortest path from p^{init} to p^{term}. *Center right.* The safe box sequence is fixed and the polygonal curve is shortened via convex optimization. *Right.* A new box (shown in green) is inserted in the sequence and the curve is shortened a second time. Since no further shortening is possible, the polygonal phase converges in one iteration.

left and bottom right, respectively. The left figure shows the augmented line graph, where these two points are connected to their adjacent vertices. The initial point p^{init} has two adjacent vertices, while the terminal point p^{term} has only one. The center left figure shows the shortest path from the initial point to the terminal point. In the center right figure, we fix the boxes that the curve must traverse, and we minimize the curve length by solving the SOCP (7). In the right figure, a new box is inserted in the box sequence and the curve nodes are optimized again. In this example, the polygonal phase converges in a single iteration.

V. Smooth Phase

The smooth phase is the final phase of our algorithm. It starts from the polygonal curve $\mathcal{C}$ and constructs a smooth path p that is feasible for our planning problem, and has small objective value. For the path parameterization, we use a piecewise Bézier curve, i.e., a sequence of Bézier curves that connect smoothly. (Sometimes, this is also called a composite Bézier curve.) We start this section by reviewing some basic properties of this family of curves. Next, we describe the optimal-control problems that we solve to design our smooth path. Finally, we conclude our running example.

A. Bézier Curves

A Bézier curve is constructed using Bernstein polynomials. The Bernstein polynomials of degree M are defined over the interval $[a, b] \subset \mathbf{R}$, with $b > a$, as

$$\beta_n(t) = \binom{M}{n} \left(\frac{t-a}{b-a}\right)^n \left(\frac{b-t}{b-a}\right)^{M-n}, \quad n = 0, \ldots, M.$$

For $t \in [a, b]$ the Bernstein polynomials are nonnegative and, by the binomial theorem, they sum up to one. Therefore, the scalars $\beta_0(t), \ldots, \beta_M(t)$ can be thought of as the coefficients of a convex combination. Using these coefficients to combine a given set of *control points* $\gamma_0, \ldots, \gamma_M \in \mathbf{R}^d$, we obtain a Bézier curve

$$\gamma(t) = \sum_{n=0}^{M} \beta_n(t)\gamma_n.$$

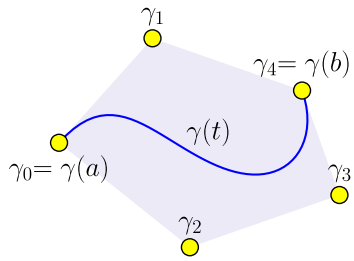

Fig. 5. Bézier curve with control points $\gamma_0, \ldots, \gamma_M$, $M = 4$. The curve starts at $\gamma(a) = \gamma_0$, ends at $\gamma(b) = \gamma_M$, and is entirely contained in the convex hull of the control points, shown shaded.

The Bézier curve $\gamma : [a, b] \to \mathbf{R}^d$ is a polynomial function of degree M. An example of a Bézier curve is shown in Fig. 5, for $d = 2$ and $M = 4$. Below, we list some important properties that we will use later in this section.

Endpoints: A Bézier curve starts at its first control point and ends at its last control point, i.e.,

$$\gamma(a) = \gamma_0, \qquad \gamma(b) = \gamma_M. \tag{10}$$

With this property, a piecewise Bézier curve (our path) can be made continuous simply by equating the last control point of each curve piece with the first control point of the next piece.

Control Polytope: Since each point on a Bézier curve is a convex combination of the control points, the entire curve is contained in the convex hull of the control points

$$\gamma(t) \in \mathbf{conv}\{\gamma_0, \ldots, \gamma_M\} \tag{11}$$

for all $t \in [a, b]$. This convex hull is called the *control polytope* of the Bézier curve γ. From this property, it follows that if all control points lie in a convex set (in our case a box), then so does the Bézier curve.

Derivatives: The derivative $\gamma^{(1)}$ of a Bézier curve γ is also a Bézier curve. It has degree $M - 1$ and its control points are given by the difference equation

$$\gamma_n^{(1)} = \frac{M}{b-a}(\gamma_{n+1} - \gamma_n), \quad n = 0, \ldots, M-1. \tag{12}$$

Iterating this, we see that the derivative $\gamma^{(i)}$ of any order $i \geq 1$ is a Bézier curve of degree $M - i$. Moreover, the derivatives of a piecewise Bézier curve are also piecewise Bézier curves, and their continuity can be enforced using the endpoint property (10).

Squared L_2 Norm: The square of the L_2 norm of a Bézier curve γ can be expressed as a function of the control points using the following expression [62, Sec. 3.3]

$$\int_a^b \|\gamma(t)\|_2^2 \, dt = (b-a)Q(\gamma_0, \ldots, \gamma_M) \tag{13}$$

where Q is a convex quadratic function defined as

$$Q(\gamma_0, \ldots, \gamma_M) = \frac{1}{2M+1} \sum_{m=0}^{M} \sum_{n=0}^{M} \frac{\binom{M}{m}\binom{M}{n}}{\binom{2M}{m+n}} \gamma_m^T \gamma_n.$$

This formula allows us also to compute the squared L_2 norm of a piecewise Bézier curve and its derivatives.

B. Nonconvex Optimal-Control Problem

In the smooth phase, we limit our attention to paths that are piecewise Bézier curves and traverse the same box sequence $s_1, \ldots, s_N$ as the curve $\mathcal{C}$. This reduces the path-planning problem (4) to an optimal-control problem that is finite dimensional and has only continuous variables, but is nonconvex. This section illustrates this control problem, and the next section describes our approach to its solution.

Variables: The variables in problem (4) are the safety map s and the path p. Here, the box sequence is fixed, therefore, a safety map is specified only through the traversal times $T_1, \ldots, T_N$, which are the first variables in our control problem. For the path parameterization, we use a piecewise Bézier curve with N pieces (one per safe box that our path must traverse). Each piece, or subpath, is a Bézier curve

$$p_j : [t_{j-1}, t_j] \to \mathbf{R}^d, \qquad j = 1, \ldots, N.$$

These Bézier curves have degree equal to M, and their control points

$$p_{j,0}, \ldots, p_{j,M} \in \mathbf{R}^d, \quad j = 1, \ldots, N \tag{14}$$

are the second group of variables in our control problem.

For $i = 1, \ldots, D$, the derivatives $p_j^{(i)}$ of the subpaths are Bézier curves of degree $M - i$. Our last set of variables are the control points these derivatives

$$p_{j,0}^{(i)}, \ldots, p_{j,M-i}^{(i)} \in \mathbf{R}^d, \quad i = 1, \ldots, D, \; j = 1, \ldots, N. \tag{15}$$

For simplicity of notation, we will sometimes denote the control points in (14) as $p_{j,0}^{(0)}, \ldots, p_{j,M}^{(0)}$, where the superscript represents the zeroth derivative.

Constraints: We assemble the constraints of our control problem by leveraging the properties of the Bézier curves.

Using the endpoint property (10), the boundary conditions in problem (4) are enforced simply as

$$p_{1,0} = p^{\text{init}}, \qquad p_{N,M} = p^{\text{term}}. \tag{16}$$

Similarly, the continuity and differentiability of our path are enforced as

$$p_{j,M-i}^{(i)} = p_{j+1,0}^{(i)}, \quad i = 0, \ldots, D, \; j = 1, \ldots, N-1. \tag{17}$$

Property (11) tells us that a Bézier curve lies within its control polytope. Therefore, to ensure that a subpath p_j is entirely contained in the corresponding safe box $\mathcal{B}_{s_j}$, it is sufficient to constrain its control points

$$l_{s_j} \leq p_{j,n} \leq u_{s_j}, \quad j = 1, \ldots, N, \; n = 0, \ldots, M. \tag{18}$$

Since each subpath p_j is constrained in a safe box, the whole piecewise Bézier curve p will be safe.

The control points of the subpath derivatives need to satisfy a difference equation analogous to (12)

$$p_{j,n}^{(i)} = \frac{M-i+1}{T_j} \left(p_{j,n+1}^{(i-1)} - p_{j,n}^{(i-1)} \right),$$
$$i = 1, \ldots, D, \; j = 1, \ldots, N, \; n = 0, \ldots, M-i. \tag{19}$$

Note that these equality constraints are nonlinear, since both the control points and the traversal times are variables in our optimization problem.

Last, the traversal times need to be positive

$$T_j > 0, \qquad j = 1, \ldots, N \tag{20}$$

and sum up to the final time

$$\sum_{j=1}^{N} T_j = T. \tag{21}$$

Objective Function: We split the integrals in our objective function (1) into the sum of N terms (one per subpath)

$$J = \sum_{i=1}^{D} \alpha_i \sum_{j=1}^{N} J_{i,j}.$$

We use (13) to express each term as a function of the control points and the traversal times

$$J_{i,j} = \int_{t_{j-1}}^{t_j} \|p_j^{(i)}(t)\|_2^2 \, dt = T_j Q\left(p_{j,0}^{(i)}, \ldots, p_{j,M-i}^{(i)}\right) \tag{22}$$

for $i = 1, \ldots, D$ and $j = 1, \ldots, N$. Note that the quadratic function Q is convex, but its product with the traversal time T_j makes our objective nonconvex.

Optimization Problem: Collecting all the components, we obtain the optimization problem

$$\begin{aligned} &\text{minimize} && J \\ &\text{subject to} && \text{constraints (16) to (21).} \end{aligned} \tag{23}$$

This program has the structure of an optimal-control problem where the difference equation (19) acts as a dynamical system that links the variables over time. Together with the nonconvex objective terms (22), this nonlinear difference equation makes the problem nonconvex. However, similarly to its infinite-dimensional counterpart (4), this problem simplifies to a convex quadratic program (QP) [46, Sec. 4.4] if we fix the

traversal times. In fact, this makes the difference equation linear and the objective function convex quadratic.

Curve Degree and Feasibility: If the degree of the subpaths satisfies $M \geq 2D+1$, then problem (23) is guaranteed to be feasible. In fact, similarly to the discussion in Section II-C, this minimum degree ensures that each subpath can be a line segment, with the first D derivatives equal to zero at the endpoints. The overall path p can then take the shape of the safe polygonal curve $\mathcal{C}$, while satisfying the differentiability constraints. Note also that the degree M must be at least $D+1$, since the continuity and differentiability constraints (17) fix the value of $D+1$ control points per subpath. In the rest of this article, we will use curves of degree $M = 2D+1$, so that problem (23) will always be feasible. Although, in practice, we have found that also values of M closer to $D+1$ almost always yield feasible problems.

C. Solution Via Convex Alternations

We solve the nonconvex program (23) by alternating between a *projection problem* and a *tangent problem*, both of which are convex optimal-control problems. As the other parts of our planning method, this step is heuristic: it is guaranteed to find a feasible solution of (23), but this solution needs not to be optimal.

Initialization: We start by computing an initial estimate of the traversal times $T_1, \ldots, T_N$ that satisfies the constraints (20) and (21). To do so, we imagine travelling along the polygonal curve $\mathcal{C}$ at constant speed. The window of time T_j that we allocate for the jth box is then equal to the distance between the nodes y_j and y_{j-1}, divided by the total length of the curve $\mathcal{C}$ and multiplied by the final time T.

Although this heuristic is very simple, we have found that it works well for most problems. More precise initial estimates of the traversal times are certainly possible but, in our experience, they are rarely worth their increased complexity.

Projection Problem: In this step, we fix the current value of the traversal times (initialized as just described) and we solve the control problem (23). As observed above this is a convex QP, which has the effect of projecting the current iterate onto the nonconvex feasible set of (23). Thanks to their optimal-control structure and their banded constraints, these QPs are solvable in a time that grows only linearly with the number N of boxes traversed by our path [28].

Tangent Problem: In this step, we attempt to improve the estimate of the traversal times by solving a convex approximation of (23).

Let us introduce auxiliary variables that represent the products of the traversal times and the control points of the path derivatives

$$q_{j,n}^{(i)} = T_j p_{j,n}^{(i)} \tag{24}$$

for $i = 1, \ldots, D$, $j = 1, \ldots, N$, and $n = 0, \ldots, M-i$. Using these variables, the nonlinear difference equation (19) becomes linear

$$q_{j,n}^{(i)} = (M-i+1)\left(p_{j,n+1}^{(i-1)} - p_{j,n}^{(i-1)}\right)$$

and the nonconvex objective terms (22) become quadratic over linear [46, Sec. 3.2.6]

$$J_{i,j} = \frac{Q\left(q_{j,0}^{(i)}, \ldots, q_{j,M-i}^{(i)}\right)}{T_j}.$$

(Recall that quadratic-over-linear functions, with the numerator convex and the denominator positive, are convex and representable through a second-order cone [60, Sec. 2.3].)

The only nonconvexity left in our problem is the nonlinear equality constraint (24), which we simply linearize around the current traversal times $\bar{T}_j$ and control points $\bar{p}_{j,n}^{(i)}$ (obtained by solving the projection problem)

$$q_{j,n}^{(i)} = -\bar{T}_j \bar{p}_{j,n}^{(i)} + T_j \bar{p}_{j,n}^{(i)} + \bar{T}_j p_{j,n}^{(i)}.$$

Since this linearization might be inaccurate away from the nominal point, we also add a trust-region constraint

$$\frac{1}{1+\kappa} \leq \frac{T_j}{\bar{T}_j} \leq 1+\kappa, \qquad j = 1, \ldots, N. \tag{25}$$

This sets a limit of $\kappa > 0$ to the maximum relative variation of the traversal times.

The resulting problem is an SOCP that approximates the nonconvex program (23) locally, and tries to improve the current solution by taking a step in the tangent space of the nonlinear equation (24). Like the projection problem, it can be solved in a time that increases only linearly with N. From its solution, we only retain the optimal traversal times $T_1^\star, \ldots, T_N^\star$, and then we solve a new projection problem to obtain a new feasible path. If the optimal objective value decreases, compared to the previous projection problem, we accept the new times and update our path. Otherwise we keep the previous times and path.

Trust Region Update: After each iteration, independently of its success, we decrease the value of the trust-region parameter κ. A simple way to do so would be to divide κ by a parameter $\omega > 1$. However, using this rule, we might have that two consecutive iterations produce the same unsuccessful update of the traversal times. Specifically, if one iteration is unsuccessful and the transition times computed in the tangent problem are not at the boundary of the trust region (25), then these times might still be feasible (and thus optimal) after we shrink the trust region. To prevent this phenomenon, we use a slightly more sophisticated update

$$\kappa^+ = \frac{1}{\omega}\left(\max\left\{\frac{\bar{T}_1}{T_1^\star}, \frac{T_1^\star}{\bar{T}_1}, \ldots, \frac{\bar{T}_N}{T_N^\star}, \frac{T_N^\star}{\bar{T}_N}\right\} - 1\right) \leq \frac{\kappa}{\omega}.$$

Here, κ and κ^+ are the current and the updated trust-region parameters, respectively. The term in the parenthesis is the minimum value of κ that, in hindsight, would have activated at least one of the trust-region constraints (25). If one of these constraints was already active, then this rule reduces to $\kappa^+ = \kappa/\omega$. If none of the trust-region constraints was active, and the iteration was unsuccessful, then the trust region is shrunk enough to make the solution of the last tangent problem infeasible for the next.

Termination: The tangent problem has optimal value smaller than or equal to its preceding projection problem. We terminate our algorithm when this gap, normalized by the cost of

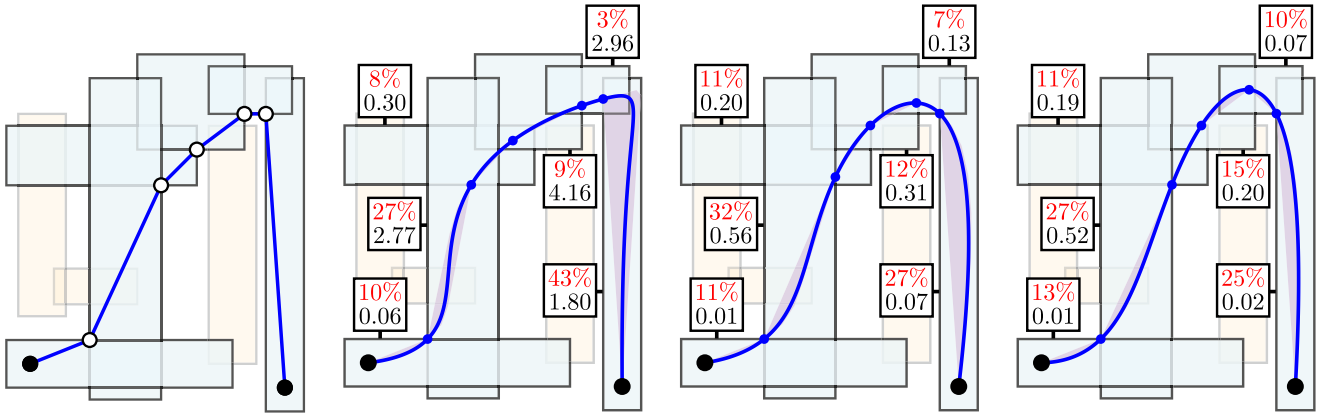

Fig. 6. Smooth phase of our algorithm. *Left.* The curve from the polygonal phase with the corresponding sequence of safe boxes. *Center left.* The polygonal curve is used to estimate initial traversal times and a first smooth path is optimized. For each box traversed by the path, the labels show the traversal time (normalized by the final time) at the top, and the cost of the trajectory piece at the bottom. The red shaded sets are the control polytopes of the Bézier curves. *Center right.* The traversal times are improved and the path is optimized a second time. *Right.* The path at the last (fourth) iteration, whose cost is within 0.01% of the global minimum.

the projection problem, is smaller than a fixed tolerance $\varepsilon > 0$. In which case, we solve one last projection problem and we return the best path that we have found.

Choice of the Parameters: We have found that for most problems the value of κ can be simply initialized to one. Large values of ω (e.g., $\omega = 5$) tend to work well when our initialization of the traversal times is accurate, while smaller values (e.g., $\omega = 2$) are more effective otherwise. In the numerical experiments discussed in this article, we use $\omega = 3$. For the termination tolerance, a reasonable choice is $\varepsilon = 10^{-2}$.

D. Example

We conclude our running example by illustrating the smooth phase of the path-planning algorithm. We seek a path that is $D = 3$ times continuously differentiable, and has total duration T equal to one. We use Bézier curves of degree $M = 2D + 1 = 7$. We take objective weights $\alpha_1 = \alpha_2 = 0$ and $\alpha_3 > 0$, i.e., our objective penalizes the squared L_2 norm of the third derivative (or jerk) of the path. To simplify the analysis, the weight α_3 is chosen so that the global minimum of the problem is equal to one.

In the left of Fig. 6, we have the curve computed in the polygonal phase, with the corresponding sequence of safe boxes highlighted in cyan. In the center left of Fig. 6, we show the path obtained by solving the first projection problem, with the traversal times initialized using the constant-velocity heuristic. Each box traversed by the path is labeled with two numbers: the percentage on top is the ratio between the traversal time T_j of that box and the final time T; the number at the bottom is the cost $J_{3,j}$ of the subpath p_j. The red shaded areas are the Bézier control polytopes within each box. We solve the tangent problem, we update the traversal times, and we solve the projection problem a second time. The resulting path is depicted in the center right of Fig. 6. After four iterations the smooth phase terminates, with the resulting path depicted in the right of Fig. 6.

The initial path (center left) has cost 12.04. The path after the first iteration (center right) has cost 1.27, which is 89% smaller. The final path (right) has cost 1.0001, and is essentially the global minimum of the problem. Although our simple heuristic to initialize the traversal times was not accurate, our algorithm converges in very few iterations.

VI. Algorithm Efficiency and Guarantees

We briefly summarize the main properties of our path-planning method.

Completeness: Our algorithm is complete: it finds a safe smooth path connecting the initial and final positions if such a path exists, and it certifies infeasibility otherwise. Feasibility is decided almost immediately with the solutions of the shortest-path problem at the beginning of the polygonal phase. If this problem is infeasible, then the initial and terminal points cannot be connected by a continuous curve. Conversely, if the shortest-path problem is feasible, then our algorithm can always recover a smooth feasible path as described at the end of Section V-B. Of course, if the safe boxes approximate a more complex space, then the completeness of our method is up to the conservatism of this approximation.

Suboptimality: Our algorithm is heuristic, and not guaranteed to solve problem (2) optimally (or within a fixed optimality tolerance). In practice, the main source of suboptimality is the choice of the box sequence, which is frozen after the polygonal phase, and does not take into account the actual objective of our path-planning problem. Solving the nonconvex problem (23) is another source of suboptimality, and our alternating method in Section V-C is designed to prioritize a low number of iterations over the cost of the final path. Two other (milder) approximations are the path parameterization using Bézier curves, and the sufficiency of the safety constraint (18).

Some heuristic steps in our path planner do not contribute to the algorithm completeness, but can play an important role in limiting the optimality losses just described. For example,

we could take the centers of the box intersections as representative points, instead of optimizing them as in (6). However, the downstream shortest-path problem would typically select less efficient box sequences with this choice. Two other main heuristic components of our method are the iterative shortening of the polygonal curve in Section IV-B and the initialization of the traversal times in Section V-C.

Runtimes: The offline and online runtimes of our method are dominated by the convex optimization problems. The SOCP (6) is the preprocessing step that takes most time, but is efficiently solvable even for path-planning problems of very large scale. In addition, we note that this subproblem only needs to be solved to modest, or even low, accuracy. The SOCP (7) in the polygonal phase takes negligible time, since it is very small and has banded constraints. Furthermore, this phase usually converges within four or five iterations. (More formally, the iterations of this phase can be bounded by the number K of safe boxes; since each iteration adds at least one box to our sequence, and box repetitions are not optimal). The projection QPs and the tangent SOCPs in the smooth phase can be large problems, but they take a time that is only linear in the number of traversed boxes and can be solved to modest precision. Note also that the trust region (25) shrinks geometrically during the iterations of the smooth phase. Therefore, after a handful of iterations (typically four to eight with the parameters given in Section V-C) the projection and the tangent problems are essentially identical, and the smooth phase terminates.

VII. Numerical Experiments

In this section, we analyze the performance of our method through multiple numerical experiments. Every experiment was run using the default values in our software implementation `fastpathplanning`, which we briefly describe below. The computations were carried out on a computer with 2.4 GHz 8-Core Intel Core i9 processor and 64 GB of RAM.

For code readability and fast prototyping, the current version of `fastpathplanning` uses `CVXPY` [63] to construct the convex optimization problems and pass them to the solver. This introduces an overhead that for some problems can be even a few times larger than the actual solver times. Since by communicating directly with the solver this overhead can be made negligible, the time spent within `CVXPY` has been eliminated from the runtimes reported in this article.

A. Software Package

The algorithm presented in this article is implemented in the open-source Python software package `fastpathplanning`, which is available at `https://github.com/cvxgrp/fastpathplanning`. For the graph computations (e.g., the construction of the line graph) we use `NetworkX 3.2` [64]. For the solution of the shortest-path problem in the line graph, we use `scipy 1.11.3` [61]. The convex optimization problems are specified using `CVXPY 1.4.1` [63], and solved with the `Clarabel 0.6.0` solver [65].

The following is a basic example of the usage of `fastpathplanning`.

```
1 import fastpathplanning as fpp
2
3 # offline preprocessing
4 L = ... # lower bounds of the safe boxes
5 U = ... # upper bounds of the safe boxes
6 S = fpp.SafeSet(L, U)
7
8 # online path planning
9 p_init = ... # initial point
10 p_term = ... # terminal point
11 T = 1 # final time
12 alpha = [1, 1, 5] # cost weights
13 p = fpp.plan(S, p_init, p_term, T, alpha)
14
15 # evaluate solution
16 t = 0.5 # sample time
17 p_t = p(t)
```

The matrices `L` and `U` contain the lower bound l_k and the upper bound u_k of each safe box $\mathcal{B}_k$, $k = 1, \ldots, K$. These have dimension $K \times d$, and are not explicitly defined in the code above. In line 6, they are used to instantiate the safe set $\mathcal{S}$ (as the object `S`). This line is where the offline preprocessing is done, i.e., we construct the line graph and optimize the representative points. In line 13, the function `plan` finds a smooth path `p`, given the safe set, initial and terminal points, final time, and objective coefficients. The number D of continuous derivatives that our path will have is equal to the length of the list `alpha`. By default, the degree of the Bézier curves is set to $M = 2D + 1$. The path object `p` can be called like a function by passing a time $t \in [0, T]$ as in line 17. (It also contains other attributes such as the list of Bézier control points and the safe boxes $s_1, \ldots, s_N$ that the path traverses.)

B. Scaling Study

In our first example, we consider path-planning problems in $d = 2$ dimensions, and analyze the performance of our algorithm as a function of the number K of safe boxes.

We generate an instance of problem (2) as follows. We construct a square grid with P^2 points with integer coordinates $\{1, \ldots, P\}^2$. We let each point in this grid be the center of a safe box $\mathcal{B}_k$. Each box elongates either horizontally or vertically, with equal probability. The short and long sides of a box are drawn uniformly at random from the intervals $[0, 0.5]$ and $[0, 2]$, respectively.

We use this procedure to generate six feasible path-planning problems with grids of side $P = 5, 10, 20, 40, 80, 160$. The number of boxes in these problems is then

$$K = P^2 = 25,\ 100,\ 400,\ 1{,}600,\ 6400,\ 25\,600.$$

The final time is taken to be $T = P$ and the cost weights are $\alpha_1 = 0$ and $\alpha_2 = \alpha_3 = 1$. The path is continuously differentiable $D = 3$ times, and the Bézier curves have degree $M = 2D + 1 = 7$. The initial position is the center of the bottom-left box, $p^{\text{init}} = (1, 1)$, and the terminal position is the center of the top-right box, $p^{\text{init}} = (P, P)$. The largest of these instances (with $K = 25{,}600$ safe boxes) is depicted in Fig. 7.

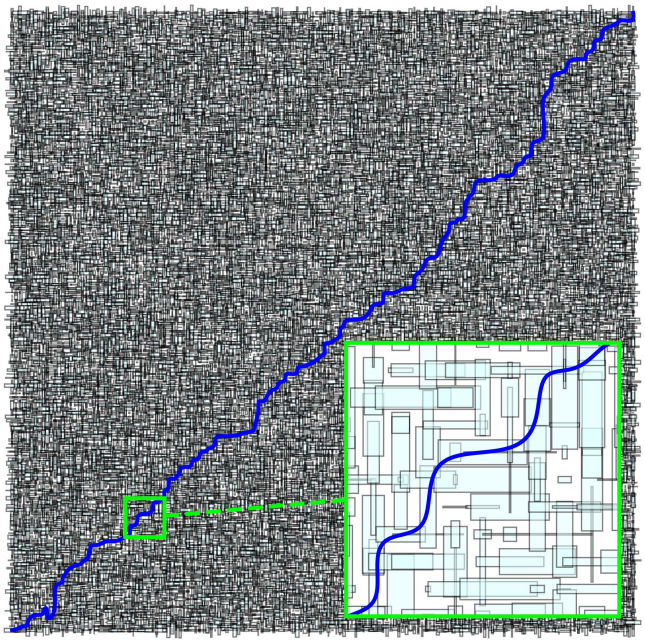

Fig. 7. Largest problem instance in the scaling study, with $K = 25{,}600$ safe boxes and final path shown.

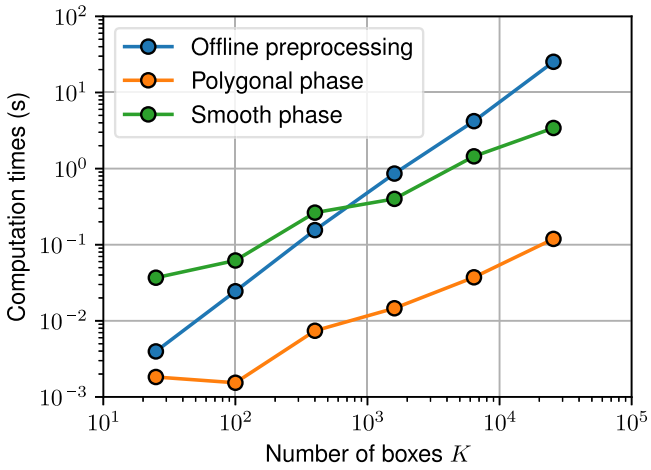

Fig. 8. Computation times for the scaling study, broken down into offline preprocessing, polygonal phase, and smooth phase.

The computation times are shown in Fig. 8, broken down into offline preprocessing, polygonal phase, and smooth phase. The smaller instances are solved in a few hundredths or tenths of seconds. For the largest instance in Fig. 7 the offline processing time is 25 s, while the online polygonal and smooth phases take 0.12 and 3.4 s, respectively. Accounting for some fixed overhead, we see that the preprocessing times grow almost linearly with the number of boxes K (unit slope in the log-log plot), while the online runtimes grow even more slowly. Table I shows the number of vertices $|\mathcal{V}|$ and edges $|\mathcal{E}|$ in the line graph G and the number of boxes N traversed by the final path, for all the problems in this analysis.

We report that for both the polygonal and the smooth phase the number of iterations is essentially unaffected by the size of the problem. In the polygonal phase, the number of iterations ranges between 1 and 4, in the smooth phase between 5 and 6.

TABLE I
SIZE OF THE INSTANCES IN THE SCALING STUDY

Total boxes K	25	100	400	1600	6400	25 600
Vertices $\lvert\mathcal{V}\rvert$	38	179	804	3298	12 816	52 308
Edges $\lvert\mathcal{E}\rvert$	127	708	3583	15 613	58 351	241 348
Path boxes N	7	12	35	53	113	241

C. Large Example

In our second example, we plan a path for a quadrotor in an environment with many obstacles. The configuration space of a quadrotor is six dimensional: three coordinates specify the position of the center of mass, and three coordinates specify the orientation. However, given any path for the center of mass that is differentiable four times, a dynamically feasible trajectory for the quadrotor's orientation, together with the necessary control thrusts, can always be reconstructed [66]. This convenient property is called *differential flatness*, and it allows us to plan the flight of a quadrotor by solving a path-planning problem in only $d = 3$ dimensions.

The quadrotor environment is shown at the top of Fig. 1, and it resembles a village with multiple buildings and dense vegetation. This village is constructed over a square grid with $P^2 = 50^2 = 2{,}500$ points, which divide the ground into $(P-1)^2$ square cells of unit side. The cell indexed by $(i, j) \in \{1, \dots, P-1\}^2$ has bottom-left coordinate $(i, j) \in \mathbf{R}^2$ and top-right coordinate $(i+1, j+1) \in \mathbf{R}^2$. Each cell contains one of the following obstacles: a building, a bush, or a tree. There are a total of $9^2 = 81$ buildings. The cells that each building occupies are identified through a random walk of length 5 that starts in the cell with index $(i, j) \in \{5, 10, \dots, 40, 45\}^2$. Therefore, each building can cover up to six cells, and neighboring buildings can potentially be connected. The buildings are constructed so that the quadrotor, whose collision geometry is overestimated with a sphere of radius 0.1, cannot collide with them while flying in another cell. The height of each building is equal to 5.0. In the cells that are not occupied by a building, we have either a bush or a tree, with equal probability. Bushes and trees are positioned in the center of their cells. A bush has square base of side chosen uniformly at random between 0.2 and 0.7, and its height is twice the side of its base. The foliage of a tree is represented as a cube of side 0.8. The center of the foliage has height that is drawn uniformly at random between 1.0 and 4.5. The trunk of a tree has square section with side 0.2.

To construct the safe set $\mathcal{S}$, we decompose the free space in each cell independently using axis-aligned boxes. The buildings occupy their cells entirely, so for these cells we do not use any safe box. The free space around a bush is decomposed using five safe boxes: four around the bush and one on top. Similarly, for a tree we have four safe boxes around the trunk and one safe box on top of the foliage. These boxes are appropriately shrunk to take into account the collision geometry of the quadrotor. The total number of safe boxes needed to decompose the environment in

Fig. 1 using this method is $K = 10{,}150$. The resulting line graph has $|\mathcal{V}| = 70{,}907$ vertices and $|\mathcal{E}| = 1{,}022{,}782$ edges.

As shown in [66], a natural objective function when planning the path of a quadrotor is the squared L_2 norm of the fourth derivative (or snap). Thus, we set our cost weights to $\alpha_1 = \alpha_2 = \alpha_3 = 0$ and $\alpha_4 = 1$. We design a path that is continuously differentiable $D = 4$ times, and we use Bézier curves of degree $M = 2D + 1 = 9$. The final time is taken to be $T = P = 50$. The quadrotor takes off at the bottom left of the environment $p^{\text{init}} = (1, 1, 0)$, and lands in the top right $p^{\text{init}} = (P, P, 0)$. Using the results from [66], it can be seen that for the quadrotor to start and stop horizontally, with zero translational and angular velocity, the following boundary conditions are necessary:

$$p^{(i)}(0) = p^{(i)}(T) = 0, \quad i = 1, \ldots, 3.$$

The small modifications necessary for our algorithm to handle these constraints are described in Section VIII.

The offline preprocessing of the safe boxes takes 101 s, with the representative points in (6) computed using the commercial solver `MOSEK 10.0`. The polygonal phase takes 0.22 s, and it converges in 5 iterations. The smooth phase takes 7.5 s and 8 iterations. The number of boxes in the final path is 135. The bottom of Fig. 1 shows the quadrotor flying along the path generated by our algorithm. A video of the quadrotor flight can be found at `https://youtu.be/t9UWIi9NyxM`.

In this example, as well as in any other problem, where we only penalize the path snap, our initial guess of the traversal times is quite inaccurate, and the initial trajectory has very high cost. However, the first iteration of the smooth phase is already sufficient to reduce the cost by 84.3%, and the final trajectory has a cost that is 99.3% smaller than the initial one.

D. Comparison With Mixed-Integer Optimization

A very natural approach to solving problem (2) is mixed-integer (global) optimization [17]. We conclude our experiments with a comparison of our method with these techniques. As a benchmark, we use our simple running example illustrated in Sections III–V, since the mixed-integer approach is impractical for larger problems.

To solve problem (2) using mixed-integer optimization, we parameterize a path as a piecewise Bézier curve with N subpaths of equal duration $T_j = T/N$, $j = 1, \ldots, N$. We write a mixed-integer program that is identical to problem (23), except for the traversal times T_j that here have fixed value, and the safety condition (18) that is substituted with a disjunctive constraint. This disjunctive constraint requires that each subpath p_j be contained in at least one safe box $\mathcal{B}_k$, and is encoded using the binary variables $\sigma_{j,k} \in \{0, 1\}$, $j = 1, \ldots, N$ and $k = 1, \ldots, K$. Since our safe sets are axis-aligned boxes, this constraint takes the following simple form:

$$\sum_{k=1}^{K} l_k \sigma_{j,k} \leq p_{j,n} \leq \sum_{k=1}^{K} u_k \sigma_{j,k}$$

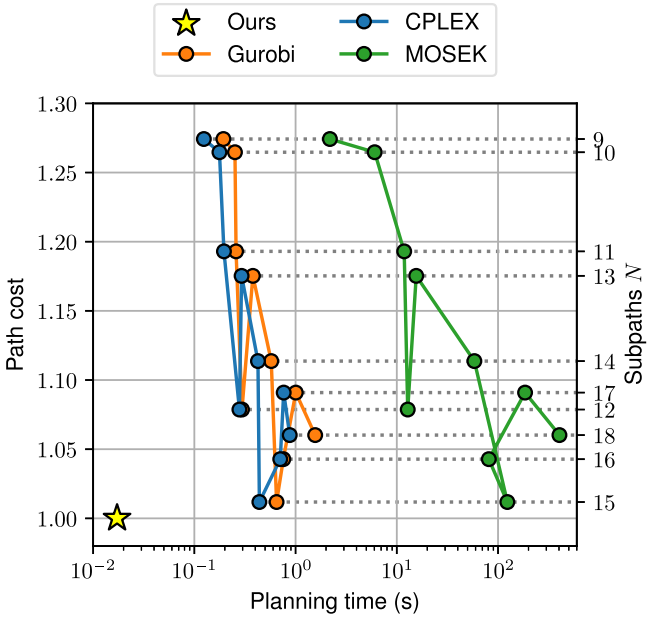

Fig. 9. Comparison of our algorithm with mixed-integer optimization, in terms of planning time and cost of the designed path. The yellow star represents our method. The three curves show the performance of different commercial mixed-integer solvers (`CPLEX`, `Gurobi`, and `MOSEK`) as the number N of subpaths used in the path parameterization increases. The value of N corresponding to each mixed-integer program can be read on the vertical axis on the right.

for $j = 1, \ldots, N$ and $n = 0, \ldots, M$. The binary variables are also subject to the "one-hot" constraint

$$\sum_{k=1}^{K} \sigma_{j,k} = 1$$

for $j = 1, \ldots, N$. The resulting problem is a mixed-integer quadratic program (MIQP).

We solve a sequence of MIQPs for an increasing number of subpaths in our piecewise Bézier curve: $N = 9, \ldots, 18$. The minimum value $N = K = 9$ is chosen since the optimal path might have to visit each safe box. Setting the degree of the Bézier curves to $M = 2D + 1 = 7$, we then have that the mixed-integer approach features the same completeness guarantee as our method, i.e., the MIQP is feasible if and only if the original planning problem (2) is feasible. Larger values of N yield a more flexible path parameterization and can decrease the MIQP optimal value. However, they also increase the MIQP solution times, which in the worst case are proportional to the number K^N of possible assignments of the binary variables. Note that, since $N \geq K$, this worst-case runtime is super-exponential in the number K of boxes.

The path designed by our method for the running example is illustrated in the right of Fig. 6, has cost 1.0001, and is essentially the global minimum of the problem (which has unit cost). The offline preprocessing (see Fig. 2), the polygonal phase (see Fig. 4), and the smooth phase (see Fig. 6) of our algorithm take 1.0, 1.7, and 14.5 ms, respectively. The sum of these three times (17.2 ms) and the cost of our path are reported in Fig. 9 with a yellow star.

For the solution of the MIQPs, we consider three state-of-the-art commercial solvers: `CPLEX 22.1.1`, `Gurobi 10.0`, and `MOSEK 10.0`. Fig. 9 reports the solution times and the path costs for these solvers, as functions of the number N of subpaths. For $N = 9$, the MIQP has an optimal value of 1.27, which is higher than our method since the duration of each subpath is fixed in the MIQP, and the solver cannot finely optimize the traversal times. The fastest solver is `CPLEX` which takes 124 ms, and is six times slower than our approach. The number of subpaths that leads to the MIQP of lowest optimal value is $N = 15$. This optimal value is 1.01, which is closer to but still larger than the cost of the path found with our method. `CPLEX` is the fastest solver also in this case, and takes 439 ms (26 times slower than our algorithm).

As expected, the mixed-integer approach becomes quickly impractical as the problem size grows. For instance, by solving via MIQP the smallest problem in the scaling study in Section VII-B (with $N = K$ and $M = 2D + 1$), we get a path that is approximately three times cheaper than ours. However, our planner takes only 43 ms (including the offline preprocessing), while `CPLEX` takes 584 s to solve the MIQP, and 25 s of branch and bound before finding a feasible path with cost lower than ours. The other solvers are even slower.

VIII. Extensions

We conclude by briefly mentioning how the techniques presented in this article can be extended to more general path-planning problems.

Initial and Final Derivatives: Our method handles boundary values on the path derivatives very easily. We only need to specify them in problem (23) using the control points $p_{1,0}^{(i)}$ and $p_{N,M-i}^{(i)}$, with $i \in \{1, \dots, D\}$, as done for the endpoint constraints in (16).

An additional modification to our algorithm that is useful in presence of boundary conditions on the derivatives concerns the estimate of the traversal times in Section V-C. Instead of initializing the traversal times by traveling the whole polygonal curve $\mathcal{C}$ at constant speed, we travel at constant speed only the central part of $\mathcal{C}$, and in the first and last segments we set to a constant the smallest derivative that gives us enough variables to satisfy the boundary conditions and the differentiability constraints. For example, in the quadrotor problem in Section VII-C, we need constant seventh derivative in the initial and final segments of $\mathcal{C}$ to find a time parameterization whose first three derivatives vanish at the endpoints, and that is continuously differentiable four times.

Finally, we note that with boundary conditions on the derivatives the degree of the first and last Bézier curves might need to be increased to preserve the completeness of our algorithm. For example, if the initial position is close to a boundary of the safe set, and the initial velocity points outwards, we may need many control points to design a sharp turn that does not leave the safe set.

Convex Safe Sets: The assumption that the safe sets are axis-aligned boxes is very convenient in the offline part of our algorithm, since the pairwise intersections between a collection of boxes can be found very efficiently [59]. We also leveraged this assumption in the polygonal phase, specifically in the multiple stabbing problems and in the improvement of the box sequence in Section IV-B. In case of more generic convex safe sets these computations are more demanding and can significantly slow down our algorithm. For example, checking if two convex sets intersect requires solving a convex optimization problem, e.g., a linear program when the sets are polyhedra. However, if each convex safe set is equipped with an axis-aligned bounding box, part of the efficiency of our approach can be recovered.

Unspecified Final Time: In some applications specifying a fixed final time T is not straightforward, and it is preferable to let the planning algorithm select this value automatically. In these cases, we also add a penalty on T (e.g., a linear cost $\alpha_0 T$ with fixed weight $\alpha_0 > 0$) that prevents our original objective J from making the final time arbitrarily large. Our approach can be extended to these problems very naturally. In the initialization of the traversal times in Section V-C, we now require an initial guess also for the total duration of the path. This guess is then improved by solving the tangent problem in Section V-C, where the final time T is now a variable in the linear constraint (21), and the cost function includes the time penalty (e.g., $\alpha_0 T$).

Derivative Constraints: Convex constraints on the path derivatives are also easily incorporated in our framework. In fact, the path derivatives are piecewise Bézier curves, and, similarly to the safety constraints in (18), they can be forced to lie in a convex set at all times by constraining their control points. If the final time T is fixed, the addition of these constraints breaks the completeness of our algorithm. Specifically, the feasibility argument in Section II-C does not hold anymore, and the optimization of our piecewise Bézier path in (23) might be infeasible even if the original path-planning problem is feasible. However, if we let T be an optimization variable as described above, then the algorithm completeness is recovered. This because any derivative constraint (that contains the origin in its interior) can be satisfied by travelling along a curve sufficiently slowly.

Multiple Waypoints: In some path-planning problems, we need to design a single smooth path that interpolates or passes through a given sequence of intermediate waypoints in order. To extend our approach to these problems, the steps in the polygonal phase are repeated to connect each pair of consecutive waypoints, yielding a single polygonal curve that satisfies all the interpolation constraints. Similarly, in the smooth phase, we concatenate multiple problems of the form (23) into a single program, where each piecewise Bézier curve has fixed endpoints and is constrained to connect smoothly with its neighbors. The time at which the overall path visits each waypoint is then automatically selected by the smooth phase. Finally, periodic trajectories that visit all the waypoints can be generated by asking our path to satisfy $p^{(i)}(0) = p^{(i)}(T)$, $i = 0, \dots, D$. These conditions translate immediately to linear constraints on the control points of the initial and final Bézier subpaths.

Dynamic and Unknown Environments: This article addresses the problem of computing smooth trajectories quickly through a known environment that is mostly static, and where only the initial and terminal points change on each query. Such problems are of great industrial interest, and are relevant to a

variety of robotic systems, e.g., mobile robots, unmanned aerial vehicles, and robot arms. If the environment is dynamic but the motion of the obstacles is known or predictable (as it is often the case, e.g., in aerospace applications), then a variant of our method can be used to design paths through a decomposition of the space-time into boxes or convex sets. If the environment is partially unknown, we can consider using techniques similar to those developed in sampling-based motion planning [67], [68] to rapidly adjust our collection of safe sets in response to online observations. Specifically, if a new obstacle appears, then we can shrink or prune the safe sets that intersect with it. Conversely, if an obstacle disappears, then we can let a new safe region take its place. However, computing the safe regions from perception data at real-time rates remains an active area of research.

APPENDIX

In this appendix, we derive the inequalities (8), which are used in the polygonal phase to improve the box sequence traversed by the curve $\mathcal{C}$.

To simplify the notation, in this appendix, we let $a = y_{j-1}$, $b = y_{j+1}$, and $y = y_j$. In addition, we denote with l and u the lower and upper bounds that delimit the axis-aligned box $\mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$. Similarly, we let l_1 and u_1 delimit the box $\mathcal{B}_{s_j} \cap \mathcal{B}_k$, and l_2 and u_2 delimit the box $\mathcal{B}_k \cap \mathcal{B}_{s_{j+1}}$. We compare the optimal values of the two problems illustrated in Fig. 3. The first is

$$\begin{array}{ll} \text{minimize} & \|y-a\|_2 + \|b-y\|_2 \\ \text{subject to} & l \le y \le u \end{array}$$

where the only variable is y. The second is

$$\begin{array}{ll} \text{minimize} & \|z_1-a\|_2 + \|z_2-z_1\|_2 + \|b-z_2\|_2 \\ \text{subject to} & l_1 \le z_1 \le u_1, \quad l_2 \le z_2 \le u_2 \end{array} \tag{26}$$

where the variables are z_1 and z_2. Let $y^\star$ be the solution of the first problem, which is known to us since we have solved (7). We want to check if choosing $z_1 = z_2 = y^\star$ is optimal for the second problem. To do so, we look for Lagrange multipliers of problem (26) that satisfy complementary slackness and are dual feasible [46, Sec. 5.5].

Complementary slackness reads

$$\begin{aligned} \lambda_1 &= \tfrac{y^\star - a}{\|y^\star - a\|_2}, \quad (y^\star - l_1)^T \nu_1^+ = (y^\star - l_2)^T \nu_2^+ = 0 \\ \lambda_2 &= \tfrac{b - y^\star}{\|b - y^\star\|_2}, \quad (u_1 - y^\star)^T \nu_1^- = (u_2 - y^\star)^T \nu_2^- = 0 \end{aligned}$$

where the multipliers $\lambda_1,\ \lambda_2 \in \mathbf{R}^d$ are paired with the first and last objective terms in (26), $\nu_1^+,\ \nu_1^- \in \mathbf{R}^d$ with the lower and upper limits in the first box constraint, and $\nu_2^+,\ \nu_2^- \in \mathbf{R}^d$ with the second box constraint. The constraints of the dual of problem (26) are

$$\begin{gathered} \nu_1^+,\ \nu_1^-,\ \nu_2^+,\ \nu_2^- \ge 0 \\ \|\lambda\|_2,\ \|\lambda_1\|_2,\ \|\lambda_2\|_2 \le 1 \\ \lambda - \lambda_1 + \nu_1^+ + \nu_1^- = \lambda_2 - \lambda + \nu_2^+ + \nu_2^- = 0 \end{gathered}$$

where the multiplier $\lambda \in \mathbf{R}^d$ is paired with the second cost term in (26).

We let L_1 and U_1 be the matrices that select the entries, where $l_1 < y^\star$ and $y^\star < u_1$, respectively. We let L_2 and U_2 be defined similarly but for the limits l_2 and u_2. After a few manipulations, the two sets of conditions above reduce to the inequalities in (8). The only variable is λ, since the values of λ_1 and λ_2 are fixed (and known) by the complementary slackness conditions.

Finally, we observe that the norm of the Lagrange multiplier $\lambda^\star$ in (9) can be interpreted as the elastic force exchanged between the points z_1 and z_2 in Fig. 3, and is indicative of the cost decrease that we incur by letting these points separate. This motivates our heuristic of inserting the box for which the vector $\lambda^\star$ has largest norm.

REFERENCES

[1] S. LaValle and R. Sharma, "On motion planning in changing, partially predictable environments," *Int. J. Robot. Res.*, vol. 16, no. 6, pp. 775–805, 1997.

[2] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[3] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proc. IEEE/RJS Int. Conf. Intell. Robots Syst.*, 2005, pp. 2210–2215.

[4] N. D. Toit and J. Burdick, "Robot motion planning in dynamic, uncertain environments," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 101–115, Feb. 2011.

[5] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. Guidance, Control, Dyn.*, vol. 25, no. 1, pp. 116–129, 2002.

[6] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, Sep. 2009.

[7] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transp. Res. Part C: Emerg. Technol.*, vol. 60, pp. 416–442, 2015.

[8] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions," in *Proc. Robot.: Sci. Syst.*, Ann Arbor, MI, USA, 2016, vol. 2, pp. 1–9.

[9] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Trans. Robot.*, vol. 33, no. 3, pp. 661–674, Jun. 2017.

[10] A. Liniger and J. Lygeros, "A noncooperative game approach to autonomous racing," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 884–897, May 2020.

[11] R. Spica, E. Cristofalo, Z. Wang, E. Montijano, and M. Schwager, "A real-time game theoretic planner for autonomous two-player drone racing," *IEEE Trans. Robot.*, vol. 36, no. 5, pp. 1389–1403, Oct. 2020.

[12] N. V. Kumar and C. S. Kumar, "Development of collision free path planning algorithm for warehouse mobile robot," *Procedia Comput. Sci.*, vol. 133, pp. 456–463, 2018.

[13] T. Benarbia and K. Kyamakya, "A literature review of drone-based package delivery logistics systems and their implementation feasibility," *Sustainability*, vol. 14, no. 1, 2021, Art. no. 360.

[14] M. Hutter et al., "Towards a generic solution for inspection of industrial sites," in *Proc. Field Service Robot.: Results 11th Int. Conf.*, 2018, pp. 575–589.

[15] C. Gehring et al., "ANYmal in the field: Solving industrial inspection of an offshore HVDC platform with a quadrupedal robot," in *Proc. Field Service Robot.: Results 12th Int. Conf.*, 2021, pp. 247–260.

[16] N. Correll et al., "Analysis and observations from the first Amazon picking challenge," *IEEE Trans. Automat. Sci. Eng.*, vol. 15, no. 1, pp. 172–188, Jan. 2018.

[17] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *Int. Conf. Robot. Automat.*, 2015, pp. 42–49.

[18] T. Marcucci, M. Petersen, D. v. Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Sci. Robot.*, vol. 8, no. 84, 2023, Art. no. eadf 7843.

[19] J.-M. Lien and N. Amato, "Approximate convex decomposition of polygons," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 17–26.

[20] N. Ayanian and V. Kumar, "Abstractions and controllers for groups of robots in environments with obstacles," in *Proc. Int. Conf. Robot. Automat.*, 2010, pp. 3537–3542.

[21] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien, "Fast approximate convex decomposition using relative concavity," *Comput.-Aided Des.*, vol. 45, no. 2, pp. 494–504, 2013.

[22] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Berlin, Germany: Springer, 2015, pp. 109–124.

[23] P. Werner, A. Amice, T. Marcucci, D. Rus, and R. Tedrake, "Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs," 2023, *arXiv:2310.02875*.

[24] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake, "Finding and optimizing certified, collision-free regions in configuration space for robot manipulators," in *Algorithmic Foundations of Robotics XV*. Berlin, Germany: Springer, 2022, pp. 328–348.

[25] M. Verghese, N. Das, Y. Zhi, and M. Yip, "Configuration space decomposition for scalable proxy collision checking in robot planning and control," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3811–3818, Apr. 2022.

[26] H. Dai, A. Amice, P. Werner, A. Zhang, and R. Tedrake, "Certified polyhedral decompositions of collision-free configuration space," *Int. J. Robot. Res.*, 2023, doi: 10.1177/02783649231201437.

[27] M. Petersen and R. Tedrake, "Growing convex collision-free regions in configuration space using nonlinear programming," 2023, *arXiv:2303.14737*.

[28] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[29] S. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[30] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM J. Optim.*, vol. 34, no. 1, pp. 507–532, 2024.

[31] T. Schouwenaars, B. D. Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *Proc. Eur. Control Conf.*, 2001, pp. 2603–2608.

[32] A. Richards, J. Bellingham, M. Tillerson, and J. How, "Coordination and control of multiple UAVs," in *Proc. AIAA Guid., Navigation, Control Conf. Exhibit*, 2002, pp. 4588–4598.

[33] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 477–483.

[34] F. Augugliaro, A. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *IEEE/RJS Int. Conf. Intell. Robots Syst.*, 2012, pp. 1917–1922.

[35] J. Schulman et al., "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, 2014.

[36] X. Liu and P. Lu, "Solving nonconvex optimal control problems by convex optimization," *J. Guidance, Control, Dyn.*, vol. 37, no. 3, pp. 750–765, 2014.

[37] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robot. Res.*, vol. 36, no. 8, pp. 947–982, 2017.

[38] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE 2019 Int. Conf. Robot. Automat.*, 2019, pp. 6741–6747.

[39] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 3, pp. 972–983, May 2021.

[40] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[41] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Ames, IA, USA, Tech. Rep. TR 98-11, 1998.

[42] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[43] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 489–494.

[44] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE Int. Conf. Robot. Automat.*, 2011, pp. 4569–4574.

[45] K. Hauser, "Learning the problem-optimum map: Analysis and application to global optimization in robotics," *IEEE Trans. Robot.*, vol. 33, no. 1, pp. 141–152, Feb. 2017.

[46] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[47] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 864–874, Oct. 2005.

[48] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, Jul. 2000.

[49] R. Tedrake, I. Manchester, M. Tobenkin, and J. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robot. Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[50] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *2017 IEEE Int. Conf. Robot. Automat.*, 2017, pp. 5883–5890.

[51] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3861–3876, Aug. 2017.

[52] A. Weiss, F. Leve, M. Baldwin, J. R. Forbes, and I. Kolmanovsky, "Spacecraft constrained attitude control using positively invariant constraint admissible sets on $SO(3) \times \mathbb{R}^3$," in *2014 Amer. Control Conf.*, 2014, pp. 4955–4960.

[53] A. Weiss, C. Petersen, M. Baldwin, R. S. Erwin, and I. Kolmanovsky, "Safe positively invariant sets for spacecraft obstacle avoidance," *J. Guidance, Control, Dyn.*, vol. 38, no. 4, pp. 720–732, 2015.

[54] K. Berntorp, R. Bai, K. F. Erliksson, C. Danielson, A. Weiss, and S. D. Cairano, "Positive invariant sets for safe integrated vehicle motion planning and control," *IEEE Trans. Intell. Veh.*, vol. 5, no. 1, pp. 112–126, Mar. 2020.

[55] C. Danielson, K. Berntorp, A. Weiss, and S. D. Cairano, "Robust motion planning for uncertain systems with disturbances using the invariant-set motion planner," *IEEE Trans. Autom. Control*, vol. 65, no. 10, pp. 4456–4463, Oct. 2020.

[56] J. -w. Choi, R. Curry, and G. Elkaim, "Path planning based on bézier curve for autonomous ground vehicles," in *Proc. Adv. Elect. Electron. Eng.-IAENG Special Ed. World Congr. Eng. Comput. Sci.* 2008, pp. 158–166.

[57] M. Flores, "Real-time Trajectory Generation for Constrained Nonlinear Dynamical Systems Using Non-Uniform Rational B-Spline Basis Functions". California Institute of Technology, Pasadena, CA, USA, 2008.

[58] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 2427–2433.

[59] A. Zomorodian and H. Edelsbrunner, "Fast software for box intersections," in *Proc. 16th Annu. Symp. Comput. Geometry*, 2000, pp. 129–138.

[60] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra Appl.*, vol. 284, no. 1-3, pp. 193–228, 1998.

[61] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[62] R. Farouki and V. Rajan, "Algorithms for polynomials in Bernstein form," *Comput. Aided Geometric Des.*, vol. 5, no. 1, pp. 1–26, 1988.

[63] S. Diamond and S. Boyd, "CVXPY: A python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.

[64] A. Hagberg, D. Schult, and P. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python Sci. Conf.*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[65] P. J. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," 2024, *arXiv:2405.12762*.

[66] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.

[67] L. Jaillet and T. Siméon, "A PRM-based motion planner for dynamically changing environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, vol. 2, pp. 1606–1611.

[68] J. V. D. Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *proc. IEEE Int. Conf. Robot. Automat.*, 2006, pp. 2366–2371.

Tobia Marcucci received the B.S.E. and M.S.E. degrees in mechanical engineering from the University of Pisa, Pisa, Italy, in 2013 and 2015, respectively.

From 2015 to 2017, he was the Ph.D. student with the Research Center "E. Piaggio" and the Istituto Italiano di Tecnologia (IIT). Since 2017, he has been with the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, Cambridge, MA, USA, to continue his Ph.D. studies. Between 2022 and 2023, he has also spent one year with the Department of Electrical Engineering, Stanford University, Stanford, CA, USA, as a Graduate Visiting Researcher. His research lies at the intersection of convex and combinatorial optimization, with applications to robotics, motion planning, and optimal control.

Parth Nobel received the B.S. degree in electrical engineering and computer science from UC Berkeley, Berkeley, CA, USA, in 2021. He is currently working toward the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA.

Since 2022, he has been a Visiting Scholar in electrical engineering and computer science with UC Berkeley. His research centers on applying convex optimization and randomized numerical linear algebra to statistics, signal processing, and various other application areas.

Russ Tedrake (Member, IEEE) received the B.S.E. degree in computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 1999, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2004.

After graduation, he joined the MIT Brain and Cognitive Sciences Department as a Postdoctoral Associate. During his education, he has also spent time at Microsoft, Microsoft Research, and the Santa Fe Institute. He is currently the Toyota Professor of electrical engineering and computer science, aeronautics and astronautics, and mechanical engineering with MIT, working with Sebastian Seung, and the Director of the Center for Robotics with CSAIL, and the leader of Team MIT's entry in the DARPA Robotics Challenge.

Dr. Russ is currently the Toyota Professor of Electrical Engineering and Computer Science with the Massachusetts Institute of Technology, the Director of Center of Robotics with the Computer Science and Artificial Intelligence Laboratory, and the Vice President of Robotics Research with the Toyota Research Institute, Los Altos, CA, USA.

Stephen Boyd (Fellow, IEEE) received the A.B. degree in mathematics from Harvard University, Cambridge, MA, USA, in 1980, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, Berkeley, CA, USA, in 1985.

He is currently the Samsung Professor of engineering, and a Professor of electrical engineering with Stanford University, Stanford, CA, USA. His current research focus is on convex optimization applications in control, signal processing, machine learning, and finance.

Dr. Boyd is a member of US National Academy of Engineering (NAE), a foreign member of the Chinese Academy of Engineering (CAE), and a foreign member of the National Academy of Engineering of Korea (NAEK).