Fast Path Planning Through Large Collections of Safe Boxes

Tobia Marcucci, Parth Nobel, Russ Tedrake, and Stephen Boyd

***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 it 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*—Motion and Path Planning, Optimization and Optimal Control, Collision Avoidance, Convex Optimization.**

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 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]–[4]. Computation might be subject to strict real-time requirements [5]–[7]. Interactions between multiple robots without central coordination can lead to game-theoretic problems [8]–[11]. In this paper we consider problems in which 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 drone or a legged robot inspecting an industrial plant, or for a mobile robot transporting packages in a large warehouse.

Similarly to previous methods [12], [13], we assume that the environment is described as a collection safe sets, through which our system or robot can move freely without colliding with obstacles. We consider the case where the safe sets are axis-aligned boxes and very large in number (thousands or tens of thousands). Our problem is to find a smooth path that is contained in the union of the safe boxes, and connects given initial and terminal points. Focusing on box-shaped safe sets allows us to substantially accelerate multiple parts of our algorithm (see §VII).

T. Marcucci and R. Tedrake are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. P. Nobel and S. Boyd are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA. Corresponding author is T. Marcucci: `tobiam@mit.edu`.

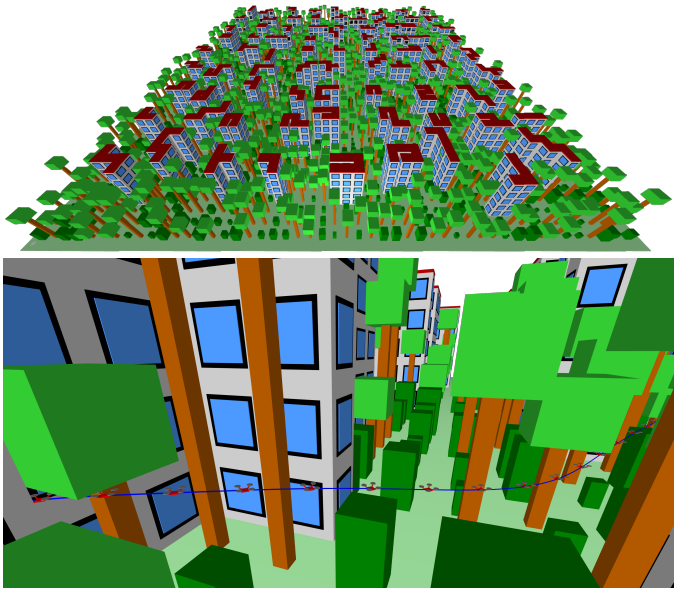

Fig. 1. Path planning problem for a quadrotor flying through a village. *Top.* The village, composed of buildings, trees, and bushes. The free space is decomposed using more than ten thousand safe boxes. *Bottom.* A snapshot of the quadrotor flight. The smooth path connects two opposite corners of the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

Our algorithm 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 we solve a convex optimization problem to label the edges of this graph with approximate distances. These computations are done only once, since the environment is static, and 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 approximately 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 our path [14]. This leads to online planning times of a few hundredths of a second for medium size problems, and of the order of a second for very large problems. Consider that, for a problem like the one in figure 1, previous approaches required hundreds of seconds to design a path through less than ten polytopic regions [12], while our method takes a few seconds to find a path through more than

ten thousand boxes.

The proposed planning algorithm is *complete*, which means it is guaranteed to find a safe smooth path connecting the initial and final positions if such a path exists, and to certify infeasibility of the planning problem otherwise. In addition, by using Bézier curves for the path parameterization, our method designs smooth paths that are guaranteed to be safe at all times, and not only at a finite number of sample points.

Through numerical experiments, we show that the runtimes of our algorithm increase very mildly with the number of safe boxes, and that our method can quickly find high-quality solutions of problems that are well beyond the reach of previous approaches, such as the one in figure 1. The methods of this paper are implemented in a companion open-source package, `fastpathplanning`.

A. Related work

A wide variety of path planning algorithms have been developed over the last fifty years. An excellent overview of the techniques available in the literature is [15, Part 2]. Here we review the methods that are most closely related to ours.

The closest approach to the one presented in this paper is trajectory optimization with GCS (graphs of convex sets) from [13]. Similarly to the method we propose here, GCS designs smooth paths through collections of safe sets that are preprocessed to form a graph. Leveraging the optimization framework from [16], it formulates a tight convex relaxation of the planning problem and it recovers a collision-free trajectory using a rounding strategy. Thanks to this workflow, GCS also provides tight optimality bounds for the trajectories it designs. However, by trying to solve the planning problem through a single convex program, GCS has a few limitations. First, only rough approximate costs on the path acceleration and higher derivatives can currently be enforced with GCS. Secondly, GCS does not scale to the very large numbers of safe sets considered here. The algorithm we present in this paper uses fast graph search to solve the discrete part of the planning problem approximately 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 find high-quality paths for extremely large planning problems very quickly.

The idea of reformulating a planning problem with collision-avoidance constraints as the problem of designing a path through safe sets comes from [12]. That work considers a statement of the path planning problem that, except for the use of boxes as safe sets, is essentially the same as the one in this paper. However, the algorithm proposed in [12], which is based on mixed integer optimization, is computationally expensive and limited to small problems (see the comparison in §VI-D).

More commonly, path planning problems with collision-avoidance constraints are solved using either local nonconvex optimization [17]–[20] or sampling-based algorithms [21]–[23]. The former methods scale to high-dimensional spaces and can handle kinematic and dynamic constraints. However, they suffer from local minima and can often fail in finding a feasible trajectory, especially if the environment is cluttered with obstacles. Sampling-based algorithms, on the other hand, can be more reliable when moving in complex environments, but they do not scale equally well to high dimensions and are less suitable for the design of smooth paths. Similar to [13], 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 a few large safe boxes. This reduces the combinatorial complexity to the minimum and allows us to plan through the open space using efficient convex optimization [24].

A variety of algorithms are available for the approximate convex decomposition of a nonconvex space [25]–[28], and also practical methods tailored to the configuration spaces of kinematic trees have been recently developed [29]–[32]. Simple modifications of these algorithms can be used to efficiently decompose complex robot environments into safe boxes, as required by our algorithm.

In this paper we parametrize continuous paths using Bézier curves. These have a variety of very useful properties for path planning, and have been widely used in this area in the recent years [13], [33]–[37].

B. Outline

This paper is organized as follows. In §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 §III. The first online phase, illustrated in §IV, finds a polygonal curve that is contained in the safe boxes and connects the given endpoints of the path. The second online phase, described in §V, finds a smooth path between the endpoints that (approximately) minimizes the objective function. In §VI we analyze the performance of our algorithm through multiple numerical experiments. In conclusion, in §VII, we describe some extensions of our method to more general planning problems.

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. Path planning problem

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 $\mathcal{S} \subseteq \mathbf{R}^d$ of safe points:

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

We will 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

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

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

We considers paths with D continuous derivatives, and we take our objective to be a weighted sum of the L_2 norm squared 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 traversal 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). Note that in this formulation we specify the initial and terminal positions, but do not constrain the initial and terminal derivatives. A simple variation on our method can handle that case.

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

B. Safety map

The path planning 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 equivalent to finding a function $s : [0, T] \to \{1, \ldots, K\}$ for which

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

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 refer to $t_1, \ldots, t_{N-1}$ as the *safety transition times*. Note that at these times we have $p(t_j) \in \mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$.

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

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

where the variables are the path p and the safety map s. We observe that if the safety map s is fixed, problem (4) reduces to a convex optimal control problem with single variable p, quadratic objective, and linear constraints.

C. Feasibility

We will say that a safety map of the form (3) is *feasible* if it satisfies

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

The path planning problem (4) is feasible if and only if a feasible safety map exists. To see this, we note that for any path and safety map that are feasible for (4), the safety map is feasible according to definition (5). Conversely, suppose a safety map is feasible, with $p_j \in \mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$, $j = 1, \ldots, N-1$. Then the polygonal curve with nodes $p^{\text{init}} = p_0, p_1, \ldots, p_N = p^{\text{term}}$ is entirely contained in the safe set $\mathcal{S}$. We will construct a feasible path that has D continuous derivatives, and moves along the polygonal curve (and so is safe). Choose any times $0 = t_0 < t_1 < \cdots < t_N = T$. Choose a parametrization of the polygonal curve that satisfies the interpolation conditions $p(t_j) = p_j$, $j = 0, \ldots, N$, as well as $p^{(i)}(t_j) = 0$, $i = 1, \ldots, D$ 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 to the path planning problem (4).

D. Method outline

We give here a high-level description of our path planning algorithm, with the details illustrated in future sections. Our method has three main phases, summarized below. The first is done offline, before we have specified the initial and terminal points or the objective coefficients; the second and third phases are done whenever a path is to be found.

Offline preprocessing: Offline preprocessing is done using only the safe set, *i.e.*, the collection of safe boxes. In this phase we construct a weighted graph with each vertex associated to a point in the intersection of two boxes, and with each edge corresponding to a pair of points that lie in the same safe box. Note that this graph, when considered as a subset of $\mathbf{R}^d$, lies entirely in the safe set $\mathcal{S}$. We refer to the points associated with the graph vertices as *representatives*.

Polygonal phase: Here we find a polygonal curve $\mathcal{C}$ that starts at p^{init}, ends at 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 graph constructed offline. Then it is shortened through an iterative process, where we alternate between minimizing the curve length for a fixed sequence of safe boxes, and updating the sequence of safe boxes for a fixed polygonal curve. The former is a small convex optimization problem and the latter only involves negligible computation.

Smooth phase: In this phase the sequence of safe boxes $s_1, \ldots, s_N$ identified in the polygonal phase, and traversed by the polygonal curve $\mathcal{C}$, is fixed. Through a simple heuristic, we use $\mathcal{C}$ also to estimate initial transition times $t_1, \ldots, t_{N-1}$. Then we alternate between optimizing the smooth path p and improving the transition times. As noted above, the first step is a convex optimal control problem, since we are solving problem (4) for a fixed safety map s, defined as in (3). For the update of the transition times we use a heuristic that is also based on convex optimization.

Summary: Overall, the online part of our method can be summarized as follows:

procedure FAST PATH PLANNING (high-level outline)
    `# polygonal phase`
    find initial box sequence $s_1, \ldots, s_N$ and safe polygonal curve $\mathcal{C}$ connecting p^{init} to p^{term}
    while not converged **do**
        fix box sequence and shorten polygonal curve
        fix polygonal curve and improve box sequence
    end while
    `# smooth phase`
    freeze box sequence
    estimate transition times $t_1, \ldots, t_{N-1}$
    while not converged **do**
        fix transition times and optimize path p
        fix path and optimize transition times
    end while
end procedure

The proposed method is approximate, but it finds paths that have typically low cost and that are guaranteed to be safe at all instants of time (as opposed to only at a finite number of sample points). In addition, the algorithm is complete, *i.e.*, it is guaranteed to find a feasible path whenever problem (2) is feasible, and to detect infeasibility otherwise.

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.

The line graph allows us to efficiently construct polygonal curves that are guaranteed to be safe (*i.e.*, contained in the safe set $\mathcal{S}$). 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 [38, §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 $y_v \in \mathbf{R}^d$ 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:

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

Here the variables are the representative points y_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 [24, §4.4.2] [39]. We also note that this problem only needs to be solved to modest, or even low, accuracy.

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

C. Example

We illustrate the preprocessing steps on a small path planning problem that will serve as a running example throughout the paper. This problem has $K = 9$ safe boxes in $d = 2$ dimensions and is depicted in figure 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

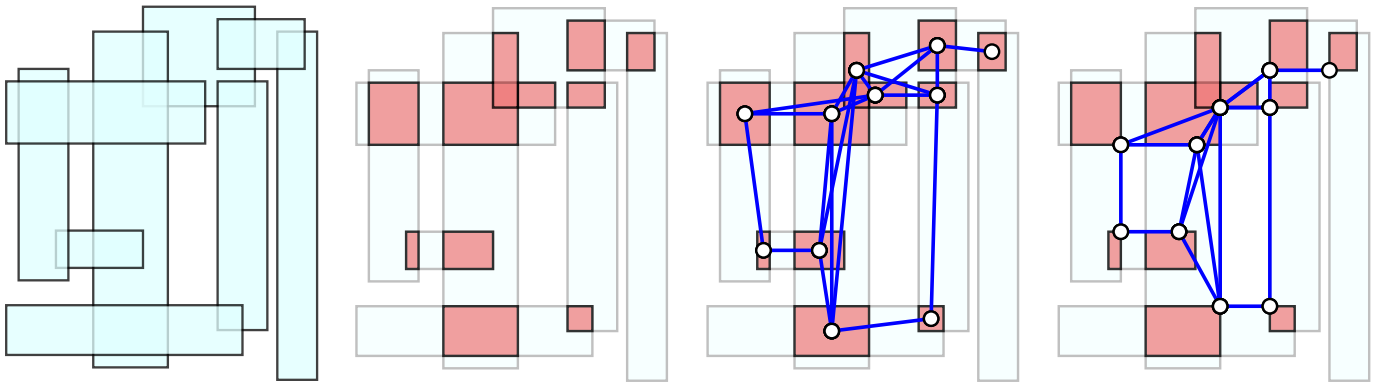

Fig. 2. Offline preprocessing of the safe boxes. *Left.* Safe boxes. *Center left.* Pairwise intersections of the safe boxes. *Center right.* Line graph, with vertices in the box intersections and edges connecting intersections that share a box. *Right.* Line graph with optimized representative points.

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 we 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 line segments is contained in at least 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}, *i.e.*, the distance between them is ∞, then the path planning problem (4) is infeasible. Conversely, if there is a path between these two vertices, the original path planning problem is feasible, since a feasible trajectory can be constructed as in §II-C.

The problem of identifying 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, this takes negligible time [38]. Using an optimized implementation of Dijkstra's algorithm (*e.g.*, the one provided by `scipy` [40]), the search for 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 of short length. However, the online knowledge of the initial and terminal points, which were unknown during the preprocessing stage, allows us to shorten this curve further. This is done iteratively: we alternate between solving a convex optimization problem 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 $\mathcal{C}$ be 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 $z_0, \ldots, z_N \in \mathbf{R}^d$ be the curve nodes, with $z_0 = p^{\text{init}}$ and $z_N = p^{\text{term}}$. For $j = 1, \ldots, N$, let also $s_j \in \{1, \ldots, K\}$ be the index of the safe box that covers the line segment between z_{j-1} and z_j. We fix the boxes s_j that the curve $\mathcal{C}$ traverses, and we optimize the position of the nodes z_j so that the length of the curve is minimized. This leads to the problem

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{N} \|z_j - z_{j-1}\|_2 \\ \text{subject to} & z_0 = p^{\text{init}}, \quad z_N = p^{\text{term}}, \\ & z_{j-1},\ z_j \in \mathcal{B}_{s_j}, \quad j = 1, \ldots, N, \end{array} \tag{7}$$

with variables $z_0, \ldots, z_N$. 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 [14].

Improvement of the box sequence: After solving problem (7) the nodes z_j minimize the curve length for the given box sequence, but the insertion of a new box in the sequence can potentially give us room to further shorten the current curve (see, *e.g.*, the example at the end of this section). In our second step we seek a new box sequence that contains the current curve and that is guaranteed to result in a shorter curve. Thanks to the fact that our safe sets are boxes, this step will only involve negligible computations and will be extremely fast.

For each node z_j in our curve, we consider inserting between the indices s_j and s_{j+1} in our box sequence a third safe box $\mathcal{B}_k$ that contains z_j. (As mentioned above, given our offline computations, the stabbing problem of finding all the boxes that contain a given point can be solved very efficiently.) Without loss of generality, we assume that the points z_{j-1}, z_j, and z_{j+1} are distinct, since if two nodes coincide we can eliminate one. In §A we show that this box insertion leads to a reduction of the curve length if and only if the optimal value of the following problem is larger than one:

$$\begin{array}{ll} \text{minimize} & \|\lambda\|_2 \\ \text{subject to} & L_1\lambda \geq L_1\lambda_1, \quad U_1\lambda \leq U_1\lambda_1, \\ & U_2\lambda \geq U_2\lambda_2, \quad L_2\lambda \leq L_2\lambda_2. \end{array} \tag{8}$$

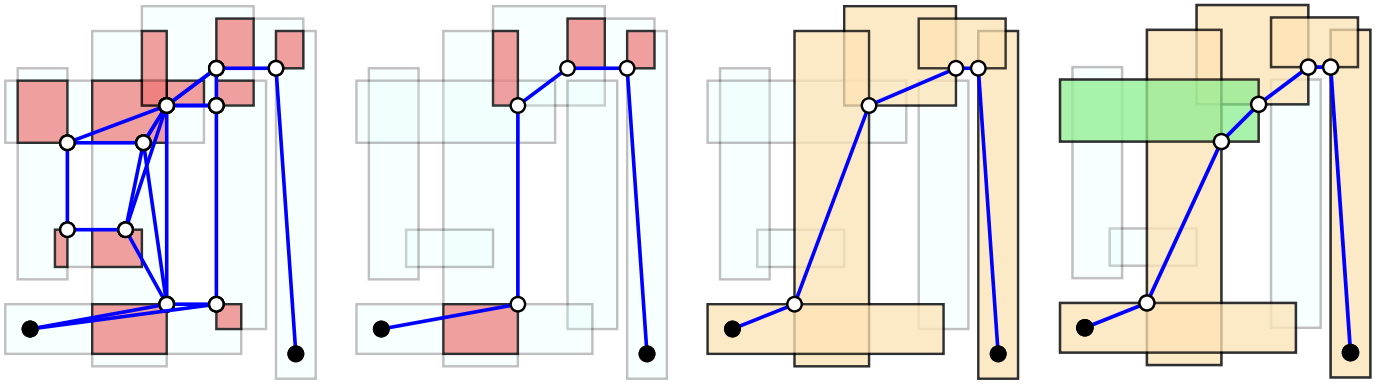

Fig. 3. Polygonal phase of the algorithm. *Left.* Line graph augmented with p^{init} and p^{term}, shown as black disks. *Center left.* Shortest path from p^{init} to p^{term}. *Center right.* The safe box sequence is fixed and the polygonal curve is shortened via convex optimization. *Right.* A new 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.

Here the variable is $\lambda \in \mathbf{R}^d$. The vectors $\lambda_1,\ \lambda_2 \in \mathbf{R}^d$ are defined as

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

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

We observe that problem (8) can be solved in closed form in negligible time. In fact, since L_1, U_1, L_2, and U_2 are selection matrices, each inequality in (8) is simply a bound on a subset of the entries of the variable λ. The constraints in (8) can then be expressed as

$$l \le \lambda \le u,$$

for two suitable vectors $l \in (\mathbf{R} \cup \{-\infty\})^d$ and $u \in (\mathbf{R} \cup \{\infty\})^d$. Problem (8) is then an orthogonal projection onto an axis-aligned box, and its minimizer $\lambda^\star$ can be computed explicitly as

$$\lambda^\star = \min\{u, \max\{l, 0\}\},$$

where the minimum and maximum are elementwise.

For each index $j = 1, \ldots, N-1$ 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 §A.) After updating the box sequence, the distance minimization problem (7) is solved again. This process is repeated until the condition above fails for every curve node j and every box k.

C. Example

Figure 3 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 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 small example the polygonal phase converges in a single iteration.

V. Smooth phase

We now describe the final phase of our algorithm, which starts from the polygonal curve $\mathcal{C}$ computed in the previous phase, and constructs a smooth path p that is feasible for our planning problem, and has small objective value.

In this phase the sequence $s_1, \ldots, s_N$ of safe boxes traversed by the curve $\mathcal{C}$ is frozen. Our first step is to find an initial estimate for the transition times $t_1, \ldots, t_{N-1}$. This gives us an initial safety map s, that is then improved by alternating between two convex optimization problems. The first is an optimal control problem, where we fix the safety map and we optimize the path. The second is a re-timing problem, where we improve the transition times for a fixed path.

The optimal control problems we solve are infinite dimensional. To solve them numerically we parametrize our path as 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 the basic properties of this family of curves.

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:

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

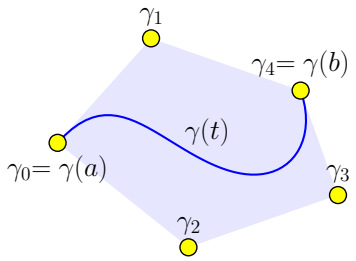

Fig. 4. Bézier curve with control points $\gamma_0, \ldots, \gamma_M$, with $M = 4$. The curve starts at $\gamma(a) = \gamma_0$, ends at $\gamma(b) = \gamma_M$, and is entirely contained in the convex hull of the control points, shown shaded.

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 figure 4, 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.*,

$$\gamma(a) = \gamma_0, \qquad \gamma(b) = \gamma_M. \tag{9}$$

Control polytope: Since each point on a Bézier curve is a convex combination of the control points, the curve is contained in the convex hull of the control points, *i.e.*,

$$\gamma(t) \in \mathbf{conv}\{\gamma_0, \ldots, \gamma_M\}, \tag{10}$$

for $t \in [a, b]$. This convex hull is called the *control polytope* of the Bézier curve γ. It follows that if all control points lie in 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 linear combinations of the ones of γ:

$$\gamma_n^{(1)} = \frac{M}{b-a}(\gamma_{n+1} - \gamma_n), \quad n = 0, \ldots, M-1. \tag{11}$$

Iterating this, we see that the derivatives of any order of a Bézier curve are also Bézier curves, and their control points are related to the ones of γ through linear constraints. Combining this observation with the endpoint property (9), we see that the requirement that a piecewise Bézier curve has D continuous derivatives can be expressed as a set of linear equality constraints on the control points of the Bézier curves that compose it (see (15) and (16)).

L_2 norm squared: The square of the L_2 norm of a Bézier curve can be expressed as a convex quadratic function of the control points [41, §3.3]:

$$\int_a^b \|\gamma(t)\|_2^2 \, dt = \frac{b-a}{2M+1} \sum_{m=0}^{M} \sum_{n=0}^{M} \frac{\binom{M}{m}\binom{M}{n}}{\binom{2M}{m+n}} \gamma_m^T \gamma_n. \tag{12}$$

This allows us to express the L_2 norm squared of any derivative of the curve γ, which is itself a Bézier curve, as a convex quadratic function of its control points.

B. Initialization of the transition times

The first step of the smooth phase is the estimation of the transition times $t_1, \ldots, t_{N-1}$. For $j = 1, \ldots, N-1$, we simply let t_j be the time at which we reach node z_j by traveling along the polygonal curve $\mathcal{C}$ at constant speed, with total traversal time T. Thus the time window $T_j = t_j - t_{j-1}$ allocated for box $\mathcal{B}_{s_j}$ is proportional to the distance between the nodes z_{j-1} and z_j. We combine these initial transition times with the box sequence of the curve $\mathcal{C}$ as in (3). This defines our initial safety map s.

C. Optimization of the path shape

As observed before, given a safety map, problem (4) reduces to a convex optimal control problem. To solve this problem numerically we parametrize a path p as a piecewise Bézier curve.

Given the transition times $t_1, \ldots, t_{N-1}$, we divide the path p into N subpaths,

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

Each subpath p_j is a Bézier curve of degree M, with control points $p_{j,0}, \ldots, p_{j,M} \in \mathbf{R}^d$ and domain $[t_{j-1}, t_j]$. Leveraging the endpoint property (9), the boundary conditions in our path planning problem are enforced simply as

$$p_{1,0} = p^{\text{init}}, \qquad p_{N,M} = p^{\text{term}}. \tag{13}$$

Property (10) tells us that a Bézier curve lies within its control polytope. Therefore, to ensure that each subpath p_j is entirely contained in the corresponding safe box $\mathcal{B}_{s_j}$, it is sufficient to constrain the control polytope of p_j to lie in the box:

$$l_{s_j} \le p_{j,n} \le u_{s_j}, \quad j = 1, \ldots, N, \ n = 0, \ldots, M. \tag{14}$$

This implies that each subpath is safe, $p_j(t) \in \mathcal{B}_{s_j}$ for all $t \in [t_{j-1}, t_j]$, and that the whole path p is safe, $p(t) \in \mathcal{S}$ for all $t \in [0, T]$. The subpath derivatives $p_j^{(i)}$ are themselves Bézier curves. As in (11), the control points of these derivatives are defined recursively by the linear difference equation

$$p_{j,n}^{(i)} = \frac{M-i+1}{T_j}\left(p_{j,n+1}^{(i-1)} - p_{j,n}^{(i-1)}\right), \quad i = 1, \ldots, D, \ j = 1, \ldots, N, \ n = 0, \ldots, M-i, \tag{15}$$

where $T_j = t_j - t_{j-1}$ is the duration of the subpath p_j. Using this, the continuity and differentiability of our path is simply enforced as a set of linear equality constraints:

$$p_{j,M-i}^{(i)} = p_{j+1,0}^{(i)}, \quad i = 0, \ldots, D, \ j = 1, \ldots, N-1. \tag{16}$$

(Note that initial and terminal values for any derivative of our path can be specified using the control points $p_{1,0}^{(i)}$ and $p_{N,M-i}^{(i)}$ as in (13).)

Using the formula in (12), each term in our objective function (1) can be expressed as the following convex quadratic function of the control points:

$$\begin{aligned} J_{i,j} &= \int_{t_{j-1}}^{t_j} \|p_j^{(i)}(t)\|_2^2 \, dt \\ &= \frac{T_j}{2(M-i)+1} \sum_{m=0}^{M-i} \sum_{n=0}^{M-i} \frac{\binom{M-i}{m}\binom{M-i}{n}}{\binom{2(M-i)}{m+n}} \left(p_{j,m}^{(i)}\right)^T p_{j,n}^{(i)}. \end{aligned} \tag{17}$$

Overall, we then have the optimization problem

$$\begin{array}{ll} \text{minimize} & J = \sum_{i=1}^{D} \alpha_i \sum_{j=1}^{N} J_{i,j} \\ \text{subject to} & \text{(13), (14), (15), and (16).} \end{array} \tag{18}$$

The variables are the control points $p_{j,n}^{(i)}$ for $i = 0, \ldots, D$, $j = 1, \ldots, N$, and $n = 0, \ldots, M - i$. Since the objective J is convex quadratic and all the constraints are linear, this is a quadratic program (QP) [24, §4.4]. The difference equations (15) couple only the control points of consecutive subpaths. These QPs have then the structure of optimal control problems with banded constraints. This allows us to solve them in time that scales only linearly in N [14].

Of course, the QP (18) is an approximation of the original infinite dimensional path planning problem. Nonetheless, piecewise Bézier curves give us an excellent finite dimensional basis for the infinite dimensional space of paths that are continuously differentiable D times. In addition, the solution of (18) is guaranteed to be safe at all times.

Choice of degree: If the degree of the Bézier curves is chosen so that $M \geq 2D+1$, then the QP (18) is guaranteed to be feasible. This degree condition 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 form of the polygonal curve $\mathcal{C}$, while satisfying the differentiability constraints.

While this minimum degree guarantees feasibility, we have found that curves of smaller degree almost always yield feasible QPs. In any case, the degree must be at least $D + 1$ (*i.e.*, $D + 2$ control points), since the continuity of the path derivatives in (16) is a set of $D+1$ linear equality constraints.

D. Re-timing

The last piece of our algorithm addresses the question of how to improve the transition times $t_1, \ldots, t_{N-1}$ for a fixed path p, obtained when solving the QP (18).

To obtain a new tentative timing, we solve a convex optimization problem where we imagine scaling the duration T_j of each subpath p_j by a constant factor $\eta_j > 0$, for $j = 1, \ldots, N$. We note immediately that this scaling does not preserve the feasibility of our path. In fact, by scaling the duration of two adjacent subpaths p_j and p_{j+1} by different factors η_j and η_{j+1} we break the path differentiability at the transition time t_j. Here we will use the factors η_j to guess a new timing, and a new (feasible) path will be constructed by re-solving the QP (18).

The effect of the time scaling on the cost of our path is to multiply each objective term $J_{i,j}$, defined in (17), by η_j^{1-2i}. Our first term in the objective of the re-timing problem is then

$$J_1 = \sum_{i=1}^{D} \alpha_i \sum_{j=1}^{N} J_{i,j} \eta_j^{1-2i},$$

where the terms $J_{i,j}$ are constant coefficients that have the value of the corresponding functions before the re-timing. Note that J_1 is a convex function of the factors $\eta_j > 0$, since for $i \geq 1$ we have $1 - 2i < 0$.

The second term in our re-timing objective is a Lagrangian penalty that approximates (relaxes) the differentiability constraints in (16) for small variations of the factors around the nominal value $\eta_1 = \cdots = \eta_N = 1$. For $i = 1, \ldots, D$ and $j = 1, \ldots, N-1$, let $\nu_{i,j} \in \mathbf{R}^d$ be the optimal Lagrange multipliers of the differentiability constraints (16), obtained when solving the QP (18). We substitute the differentiability constraints with the cost penalty

$$\sum_{i=1}^{D} \sum_{j=1}^{N-1} \nu_{i,j}^T \left(p_{j,M-i}^{(i)} - p_{j+1,0}^{(i)} \right).$$

Note that, since the multipliers $\nu_{i,j}$ are optimal, the optimal solution of the QP (18) is unchanged after this substitution. Expressing this penalty as a function of our re-timing variables η_j, we obtain

$$\sum_{i=1}^{D} \sum_{j=1}^{N-1} w_{i,j} (\eta_j^{-i} - \eta_{j+1}^{-i}),$$

where $w_{i,j} = \nu_{i,j}^T p^{(i)}(t_j)$ and p is the fixed path found by solving (18). Since the latter expression is, in general, not convex in the scaling factors, we linearize it around the nominal point $\eta_1 = \cdots = \eta_N = 1$. This yields our second cost term,

$$J_2 = \sum_{i=1}^{D} \sum_{j=1}^{N-1} i w_{i,j} (\eta_{j+1} - \eta_j),$$

where all the constants have been dropped.

The first constraint in our re-timing problem is a linear equality that ensures that the total duration of our path is unchanged after re-timing:

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

Finally, similarly to a gradient type method, we use a trust region constraint to explicitly control the mismatch between the subpath derivatives. Specifically, we limit the maximum change in adjacent scaling factors as $|\eta_{j+1} - \eta_j| \leq \kappa$, where $\kappa > 0$.

Overall, our re-timing problem is then

$$\begin{array}{ll} \text{minimize} & J_1 + J_2 \\ \text{subject to} & \sum_{j=1}^{N} \eta_j T_j = T, \\ & |\eta_{j+1} - \eta_j| \leq \kappa, \quad j = 1, \ldots, N-1. \end{array} \tag{19}$$

The variables are the factors $\eta_1, \ldots, \eta_N$, which are subject to the implicit constraint $\eta_j > 0$. The re-timing problem (19) is convex, since the objective is convex, the first constraint is linear, and the second constraint can be expressed as $2(N - 1)$ linear inequalities. More precisely, it can be verified that problem (19) is representable as an SOCP [39, §2.3]. Given that this problem has only N variables and sparse structure, it can be solved extremely quickly.

After solving problem (19) and updating the transition times $t_1, \ldots, t_{N-1}$, the QP (18) is solved again and a new (fully differentiable) path is obtained. If the optimal objective value decreases, compared to its value before re-timing, we accept the new times and update our path. Otherwise we keep the

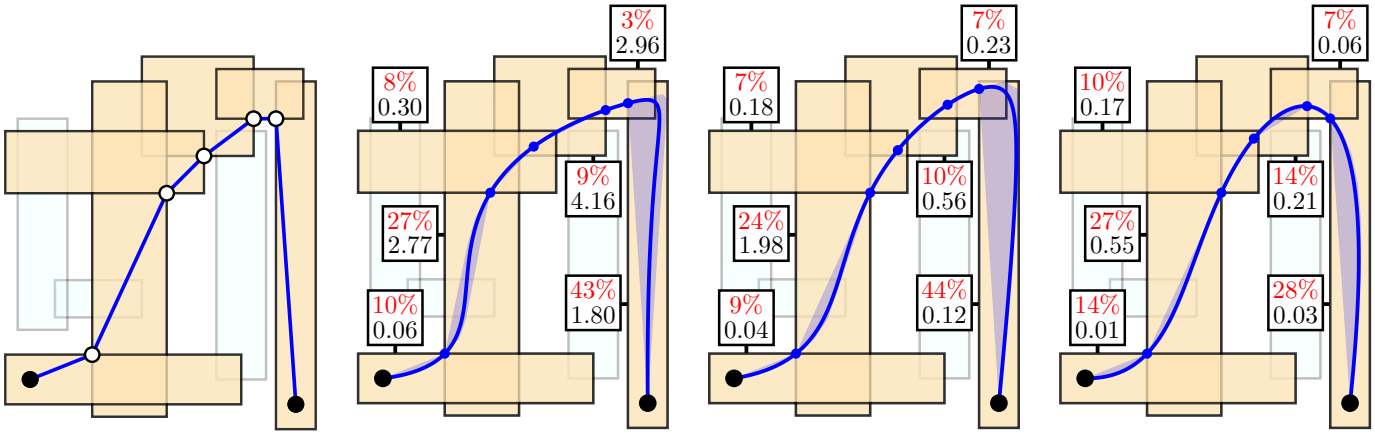

Fig. 5. Smooth phase of our algorithm. The shaded sets are the control polytopes of the Bézier curves. The labels show the relative time duration (top) and the cost (bottom) of each Bézier subpath. *Left.* The curve obtained in the polygonal phase of the algorithm with the corresponding sequence of safe boxes. *Center left.* The transition times are estimate using the polygonal curve and an initial smooth path is optimized. *Center right.* The transition times are improved using convex optimization and the path is optimized a second time. *Right.* The path at the last (sixth) iteration.

previous times and path. Independently of the success of the iteration, we then decrease the value of κ. This process is repeated until κ becomes smaller than a fixed tolerance $\varepsilon > 0$.

We decrease κ using

$$\kappa = \frac{1}{\omega} \max_{j=1,\dots,N-1} |\eta^\star_{j+1} - \eta^\star_j|,$$

where $\eta^\star_1, \dots, \eta^\star_N$ is the solution of (19) and $\omega > 1$ is a parameter. Note that in case of an unsuccessful update of the transition times, $\omega > 1$ guarantees that at the next iteration the factors $\eta^\star_1, \dots, \eta^\star_N$ are infeasible for the new re-timing problem (19) with the updated value of κ.

We have found that for most problems the value of κ can be simply initialized to one. Small values of ω (*e.g.*, $\omega = 2$) tend to work well when the initial transition times from §V-B are inaccurate, while larger values of ω (*e.g.*, $\omega = 5$) are more effective otherwise. In the numerical experiments illustrated below we use $\omega = 3$. For the termination tolerance a reasonable choice is $\varepsilon = 10^{-2}$.

E. 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 take objective weights $\alpha_3 = 1$ and $\alpha_1 = \alpha_2 = 0$, *i.e.*, our objective is the L_2 norm squared of the third derivative (or jerk) of the path. We use Bézier curves of degree $M = 2D+1 = 7$, which ensure feasibility of the QP (18) and success of our algorithm. Given the objective weights of this problem, the choice of the traversal time T does not affect the shape of our path, but only scales its cost. Therefore, to simplify the analysis, we select T so that the global minimum of the problem is equal to one.

Figure 5 illustrates the smooth phase of our algorithm. The blue shaded areas are the Bézier control polytopes within each box. Each box traversed by the path is labeled with two numbers: the percentage on top is the ratio between the duration T_j of the subpath p_j paired with the box and the total traversal time T; the number at the bottom is the cost $J_{3,j}$ of the subpath p_j. The left figure shows the curve computed in the polygonal phase, with the corresponding sequence of safe boxes. In the center left figure, we show the path obtained by solving the QP (18), with the initial transition times computed as in §V-B. We solve problem (19) to update the transition times, and we then solve the QP (18) a second time. The resulting path is depicted in the center right figure. After six iterations the smooth phase converges, with the resulting path depicted in the right figure.

The initial path (center left) has total cost 12.04. The path after the first iteration (center right) has cost 3.13, which is 74% smaller than the initial one. The final path (right) has cost 1.04, which is 91% cheaper than the initial one, and within only 4.0% of the global minimum (which has unit value). We also observe that our simple heuristic to initialize the transition times was quite inaccurate. However, our algorithm fixes this in very few iterations.

VI. Numerical Experiments

In this section we illustrate our method with numerical experiments. Every experiment was carried out 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 (initial) version of `fastpathplanning` uses `CVXPY` [42] to construct the convex optimization problems and pass them to the solver. This introduces an overhead that, in some cases, can be significant. 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 paper.

A. Software package

The algorithm presented in this paper 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` [43]. For the solution of the shortest path problem in the line graph we use `scipy` [40]. The convex optimization problems are specified using `CVXPY` [42], and solved with the `Clarabel` solver [44].

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 # traversal 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, intial and terminal points, traversal 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 the path planning 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

$$K = P^2 = 25,\ 100,\ 400,\ 1{,}600,\ 6{,}400,\ 25{,}600.$$

The traversal 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 $M = 2D + 1 = 7$ (which ensures that our algorithm will succeed). 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 ($K = 25{,}600$) is depicted in figure 6.

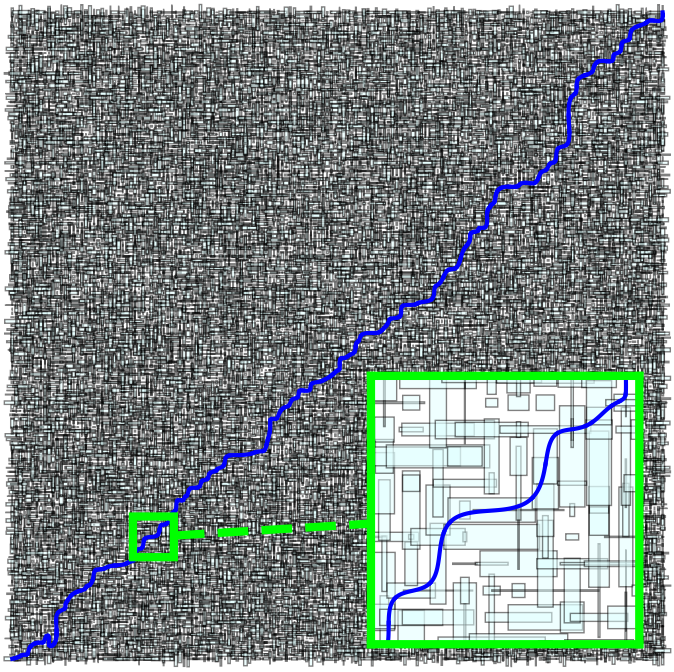

Fig. 6. Largest problem instance in the scaling study, with $K = 25{,}600$ safe boxes and final path shown.

Total boxes K	25	100	400	1,600	6,400	25,600
Vertices $\lvert\mathcal{V}\rvert$	38	179	804	3,298	12,816	52,308
Edges $\lvert\mathcal{E}\rvert$	127	708	3,583	15,613	58,351	241,348
Path boxes N	7	12	35	53	113	241

TABLE I
PROBLEM INSTANCE DIMENSIONS.

The computation times are shown in figure 7, broken down into offline preprocessing, polygonal phase, and smooth phase. For the largest problem instance (figure 6) the offline processing time is 31 seconds, and a smooth path is generated online in 2.3 seconds. Accounting for some fixed overhead, we can see that the preprocessing times grow approximately linearly with the number of boxes K (unit slope in the log-log plot), while the online runtimes grow even more slowly. For each problem instance in this analysis, 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 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 4 and 6.

C. Large example

In our second example we plan a path for a quadrotor in an environment with a very large number of obstacles. The configuration space of a quadrotor is six dimensional: three

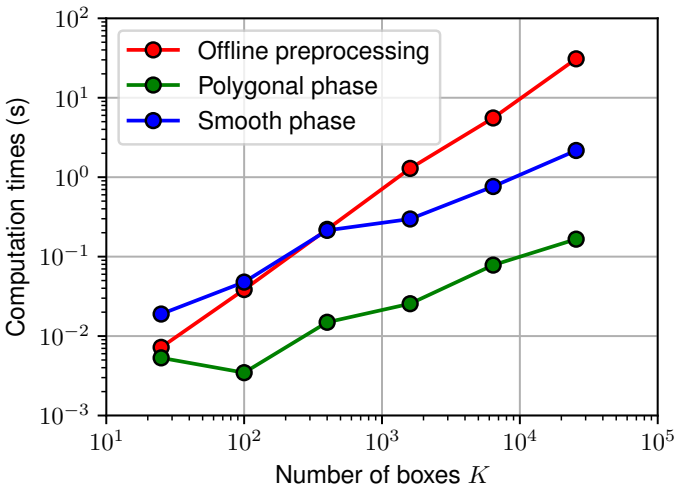

Fig. 7. Computation times for the scaling study, broken down into offline preprocessing, polygonal phase, and smooth phase.

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 [45]. This very 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 figure 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 from $[1.0, 4.5]$. 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 need 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 figure 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 [45], a natural objective function when planning the path of a quadrotor is the L_2 norm squared of the fourth derivative (or snap). Thus we set our cost weights to $\alpha_4 = 1$ and $\alpha_1 = \alpha_2 = \alpha_3 = 0$. We design a path that is continuously differentiable $D = 4$ times, and we use Bézier curves of degree $M = 2D+1 = 9$. The traversal 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 $p^{\text{init}} = (P,P,0)$. In [45] it is also shown that for the quadrotor to start and stop horizontally, with zero translational and angular velocity, the following boundary conditions are necessary:

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

These constraints are easily handled by our algorithm as mentioned in §V-C.

The offline preprocessing of the safe boxes takes 110 seconds, with the representative points in (6) computed using the commercial solver `MOSEK 10.0`. The polygonal phase takes 0.39 seconds, and it converges in 5 iterations. The smooth phase takes 3.50 seconds and 6 iterations. The number of boxes in the final path is 133. The bottom of Figure 1 shows the quadrotor flying along the path generated by our algorithm. A video of the path can be found at `https://youtu.be/p1xlcNiER0o`.

In this example, as well as in any other problem where we only penalize the path snap, the initial transition times from §V-B yield an initial trajectory with very high cost. However, the first iteration of the smooth phase is already sufficient to reduce the cost by 92.1%, and the final trajectory has a cost that is 99.4% smaller than the initial one.

D. Comparison with mixed integer optimization

Among the existing algorithms for path planning, a very natural approach to solving problem (2) is mixed integer (global) optimization [12]. We conclude our experiments with a short comparison of our method with these techniques. As a benchmark we use our simple running example illustrated in figures 2 to 5 (since on larger instances the mixed integer approach quickly becomes completely impractical).

To solve problem (2) with mixed integer optimization, we parametrize a path as a piecewise Bézier curve with N subpaths of equal duration. The degree of each subpath is $M = 2D + 1 = 7$. We write a mixed integer quadratic program (MIQP) that is essentially identical to the QP (18), except for the safety condition (14) that is substituted with a disjunctive constraint. This disjunctive constraint requires that each subpath p_j is contained in at least one safe box $\mathcal{B}_k$, and is encoded using the binary variables $\sigma_{j,k} \in \{0,1\}$,

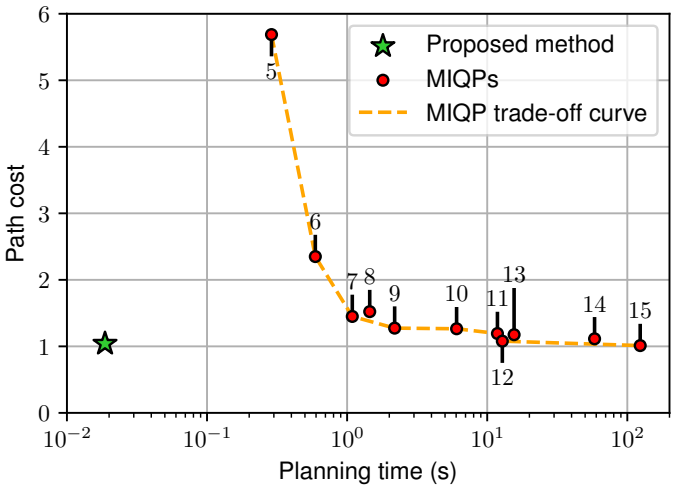

Fig. 8. Comparison of the proposed path planner with mixed integer optimization. The number attached to each MIQP solution is the number N of subpaths used in the trajectory parameterization.

$j = 1, \dots, N$ and $k = 1, \dots, K$. Since our safe sets are axis-aligned boxes, this constraint takes the following simple form:

$$\sum_{k=1}^{K} l_k \sigma_{j,k} \leq p_{j,n} \leq \sum_{k=1}^{K} u_k \sigma_{j,k}, \qquad j = 1, \dots, N, \;\; n = 0, \dots, M,$$

where the binary variables are also subject to the "one-hot" condition

$$\sum_{k=1}^{K} \sigma_{j,k} = 1, \quad j = 1, \dots, N.$$

The MIQPs are solved with the commercial solver `MOSEK 10.0`. We highlight that, since in the worst case the solver has to enumerate all the K^N possible assignments of binary variables, the runtimes of this approach can grow exponentially with N.

The path designed by our method for the running example is illustrated in the right of figure 5 and has cost 1.04. (We recall that the data of this problem are chosen so that the global minimum is equal to one.) The offline preprocessing (figure 2), the polygonal phase (figure 3), and the smooth phase (figure 5) of our algorithm take 2.4, 3.7, and 12.6 milliseconds, respectively. The sum of these three times (18.7 milliseconds) and the cost of our path are reported in figure 8 with a green star. Using mixed integer optimization, we design a sequence of trajectories with increasing number of subpaths, $N = 5, \dots, 15$, and we mark the associated runtimes and costs with red dots in figure 8. Close to each dot we show the corresponding number N of subpaths.

Mixed integer optimization requires $N = 7$ subpaths and 1.09 seconds to find a path that has cost comparable to ours. As shown by the trade-off curve, the best mixed integer path is obtained for $N = 15$. This has cost 1.01 and requires 124 seconds to be found. Compared to ours, this is a decrease in cost of only 2.7% and an increase in runtime by a factor larger than 6,000.

VII. Extensions

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

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 [38]. We also leveraged this assumption in the polygonal phase, specifically in the multiple stabbing problems and in the improvement of the box sequence in §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 traversal time: In some applications specifying a fixed traversal 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 θT with fixed weight $\theta > 0$) that prevents our original objective J from making the traversal time arbitrarily large. Our approach can be extended to these problems very naturally. In the initialization of the transition times in §V-B, we now require an initial guess for the total duration of the path. This guess is then improved by solving the retiming problem (19), where we let the traversal time T be an optimization variable and we add the duration penalty (*e.g.*, θT) to the cost function. Note that even with these changes problem (19) is still representable as an SOCP.

Constraints on the path derivatives: Convex constraints on the path derivatives can also be incorporated in our framework very easily. In fact, the derivatives of the path designed by our method are piecewise Bézier curves, and, similarly to the safety constraints in (14), these derivatives can be forced to lie in a convex set at all times simply by constraining their control points. If the traversal time T is fixed, the addition of these constraints breaks the completeness of our algorithm. Specifically, the feasibility argument in §II-C does not hold anymore, and the optimization of our piecewise Bézier path in (18) 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 path 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 (18) into a single QP, 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 our retiming technique. Finally, periodic trajectories that visit all the waypoints can be generated by asking our path to satisfy $p^{(i)}(0) = p^{(i)}(T)$, $i = 0, \ldots, D$. These conditions translate immediately to linear constraints on the control points of the Bézier subpaths p_1 and p_N.

ACKNOWLEDGMENTS

This research was supported by the Office of Naval Research (ONR), Award No. N00014-22-1-2121. Indeed, this work is a direct consequence of the collaboration fostered by this grant.

Parth Nobel was supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1656518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Stephen Boyd was partially supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR, and by Office of Naval Research grant N00014-22-1-2121.

REFERENCES

[1] S. LaValle and R. Sharma, "On motion planning in changing, partially predictable environments," *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 775–805, 1997.

[2] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The international journal of robotics research*, vol. 17, no. 7, pp. 760–772, 1998.

[3] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 2210–2215.

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

[5] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of guidance, control, and dynamics*, 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 Transactions on control systems technology*, vol. 17, no. 5, pp. 1105–1118, 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," *Transportation Research Part C: Emerging Technologies*, 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 *Robotics: Science and systems*, vol. 2. Ann Arbor, MI, USA, 2016, pp. 1–9.

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

[10] A. Liniger and J. Lygeros, "A noncooperative game approach to autonomous racing," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 884–897, 2019.

[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 Transactions on Robotics*, vol. 36, no. 5, pp. 1389–1403, 2020.

[12] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation*. IEEE, 2015, pp. 42–49.

[13] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *arXiv preprint arXiv:2205.04422*, 2022.

[14] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2009.

[15] S. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[16] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *arXiv preprint arXiv:2101.11565v4*, 2021.

[17] F. Augugliaro, A. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1917–1922.

[18] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[19] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[20] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2020.

[21] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[23] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[25] J.-M. Lien and N. Amato, "Approximate convex decomposition of polygons," in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 17–26.

[26] N. Ayanian and V. Kumar, "Abstractions and controllers for groups of robots in environments with obstacles," in *2010 IEEE International Conference on Robotics and Automation*, May 2010.

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

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

[29] 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*. Springer, 2022, pp. 328–348.

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

[31] H. Dai, A. Amice, P. Werner, A. Zhang, and R. Tedrake, "Certified polyhedral decompositions of collision-free configuration space," *arXiv preprint arXiv:2302.12219*, 2023.

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

[33] M. Flores, *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational b-spline basis functions*. California Institute of Technology, 2008.

[34] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2427–2433.

[35] M. Elbanhawi, M. Simic, and R. Jazar, "Continuous path smoothing for car-like robots using b-spline curves," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 23–56, 2015.

[36] F. Koolen, "Balance control and locomotion planning for humanoid robots using nonlinear centroidal models," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[37] N. Csomay-Shanklin, A. Taylor, U. Rosolia, and A. Ames, "Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control Lyapunov functions," *arXiv preprint arXiv:2204.00152*, 2022.

[38] A. Zomorodian and H. Edelsbrunner, "Fast software for box intersections," in *Proceedings of the sixteenth annual symposium on computational geometry*, 2000, pp. 129–138.

[39] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear algebra and its applications*, vol. 284, no. 1-3, pp. 193–228, 1998.
[40] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
[41] R. Farouki and V. Rajan, "Algorithms for polynomials in Bernstein form," *Computer Aided Geometric Design*, vol. 5, no. 1, pp. 1–26, 1988.
[42] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.
[43] A. Hagberg, D. Schult, and P. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.
[44] P. Goulart and Y. Chen, "Clarabel solver documentation," 2023. [Online]. Available: https://oxfordcontrol.github.io/ClarabelDocs/stable/
[45] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

APPENDIX A
DERIVATION OF BOX UPDATE FORMULA

In this appendix we derive problem (8), which is used in the polygonal phase of our algorithm to improve the sequence of boxes that the curve $\mathcal{C}$ must traverse. We consider three consecutive nodes z_{j-1}, z_j, and z_{j+1}, obtained by solving problem (7). As observed, these can be assumed to be distinct, and node z_j lies in the intersection of $\mathcal{B}_{s_j}$ and $\mathcal{B}_{s_{j+1}}$. Let $\mathcal{B}_k$ be a third box that contains z_j. Here we answer the question of whether inserting this box between the boxes s_j and s_{j+1} gives us room to shorten our polygonal curve.

To simplify the notation, just for this appendix, let $a = z_{j-1}$, $b = z_{j+1}$, and $z = z_j$. In addition let l and u be the lower and upper bound that delimit the axis-aligned box $\mathcal{B}_{s_j} \cap \mathcal{B}_{s_{j+1}}$. Similarly, 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 $\mathcal{B}_k \cap \mathcal{B}_{s_{j+1}}$. To answer the question above, we need to compare the optimal value of two optimization problems. The first problem is

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

where the only variable is z. The second problem is

$$\begin{array}{ll} \text{minimize} & \|z_1-a\|_2 + \|z_2-z_1\|_2 + \|b-z_2\|_2 \\ \text{subject to} & l_1 \le z_1 \le u_1, \quad l_2 \le z_2 \le u_2, \end{array} \tag{20}$$

where the variables are z_1 and z_2. The insertion of the box k is length decreasing if and only if the optimal cost of the second optimization is less than the one of the first.

Let $z^\star$ be the solution of the first problem, which is known to us since we have solved (7). Observe that the solution $z_1 = z_2 = z^\star$ is feasible for the second problem. This implies that the insertion of box k is length decreasing if and only if $z_1 = z_2 = z^\star$ is not optimal for the second problem. To check the optimality of this variable assignment, we look for Lagrange multipliers of problem (20) that satisfy complementary slackness and are dual feasible.

Complementary slackness reads

$$\begin{array}{ll} \lambda_1 = (z^\star - a)/\|a - z^\star\|_2, & (z^\star - l_1)^T \nu_1^+ = 0, \\ \lambda_2 = (b - z^\star)/\|b - z^\star\|_2, & (z^\star - l_2)^T \nu_2^+ = 0, \\ & (u_1 - z^\star)^T \nu_1^- = 0, \\ & (u_2 - z^\star)^T \nu_2^- = 0. \end{array}$$

Here the multipliers $\lambda_1,\ \lambda_2 \in \mathbf{R}^d$ are paired with the first and last objective terms in (20), $\nu_1^+,\ \nu_1^- \in \mathbf{R}^d$ 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 (20) are

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

where the multiplier $\lambda \in \mathbf{R}^d$ is paired with the second cost term in (20).

Let L_1 and U_1 be the matrices that select the entries where $l_1 < z^\star$ and $z^\star < u_1$, respectively. 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

$$\begin{array}{lll} \|\lambda\|_2 \le 1, & L_1\lambda \ge L_1\lambda_1, & U_1\lambda \le U_1\lambda_1, \\ & U_2\lambda \ge U_2\lambda_2, & L_2\lambda \le L_2\lambda_2. \end{array}$$

Here the only variable is λ, as the values of λ_1 and λ_2 are fixed (and known) by the complementary slackness conditions. This set of conditions gives problem (8).

Finally, we also observe that the norm of λ can be interpreted as the force exchanged by the points z_1 and z_2. This motivates our heuristic of inserting the box that maximizes the optimal value (8), when multiple box insertions are length decreasing.

Tobia Marcucci received the B.S.E. and M.S.E. in Mechanical Engineering from the University of Pisa in 2013 and 2015, respectively. From 2015 to 2017 he was Ph.D. student at the Research Center "E. Piaggio" and the Istituto Italiano di Tecnologia (IIT). Since 2017 he is at the Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT, to continue his Ph.D. studies. Since 2022 he is also a graduate visiting researcher in the Department of Electrical Engineering at Stanford University. His research lies at the intersection of convex and combinatorial optimization, with applications to robotics, motion planning, and optimal control.

Parth Nobel is a a Ph.D. student at Stanford University in Electrical Engineering and, since 2022, a Visiting Scholar at UC Berkeley in Electrical Engineering and Computer Science. He earned his B.S. in Electrical Engineering and Computer Science from UC Berkeley in 2021. His research centers on applying convex optimization and randomized numerical linear algebra to statistics, signal processing, and various other application areas.

Russ Tedrake is the Toyota Professor of Electrical Engineering and Computer Science, Aeronautics and Astronautics, and Mechanical Engineering at MIT, the Director of the Center for Robotics at CSAIL, and the leader of Team MIT's entry in the DARPA Robotics Challenge. Russ is also the Vice President of Robotics Research at the Toyota Research Institute. He is a recipient of the NSF CAREER Award, the MIT Jerome Saltzer Award for undergraduate teaching, the DARPA Young Faculty Award in Mathematics, the 2012 Ruth and Joel Spira Teaching Award, and was named a Microsoft Research New Faculty Fellow. Russ received his B.S.E. in Computer Engineering from the University of Michigan, Ann Arbor, in 1999, and his Ph.D. in Electrical Engineering and Computer Science from MIT in z004, working with Sebastian Seung. 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.

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 Professor of Electrical Engineering at Stanford University, Stanford, CA, USA. He 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). His current research focus is on convex optimization applications in control, signal processing, machine learning, and finance.