Fast UAV Trajectory Optimization Using Bilevel Optimization With Analytical Gradients

Weidong Sun [ORCID], Gao Tang [ORCID], *Student Member, IEEE*, and Kris Hauser [ORCID], *Senior Member, IEEE*

Abstract—In the article, we present an efficient optimization framework that solves trajectory optimization problems by decoupling state variables from timing variables, thereby decomposing a challenging nonlinear programming (NLP) problem into two easier subproblems. With timing fixed, the state variables can be optimized efficiently using convex optimization, and the timing variables can be optimized in a separate NLP, which forms a bilevel optimization problem. The challenge of obtaining the gradient of the timing variables is solved by sensitivity analysis of parametric NLPs. The exact analytic gradient is computed from the dual solution as a by-product, whereas existing finite-difference techniques require additional optimization. The bilevel optimization framework efficiently optimizes both timing and state variables which is demonstrated on generating trajectories for an UAV. Numerical experiments demonstrate that bilevel optimization converges significantly more reliably than a standard NLP solver, and analytical gradients outperform finite differences in terms of computation speed and accuracy. Physical experiments demonstrate its real-time applicability for reactive target tracking tasks.

Index Terms—Trajectory optimization, unmanned aerial vehicle.

I. Introduction

REAL-TIME optimal trajectory generation has long been a challenging but essential component in robotics. Due to the prevalence of nonlinear dynamics, nonconvex constraints and high dimensionality of the planning space, it is often difficult to optimize trajectories quickly and reliably. One promising approach is to fix a subset of optimization variables and optimize the rest by convex optimization that can be reliably solved to global optimum. One such example is to fix the time parametrization of a polynomial spline trajectory, which permits optimizing the trajectory with quadratic programming (QP) methods. This technique has been applied to path planning for ground robots, autonomous cars [1], [2], humanoid robots [3], and unmanned aerial vehicles (UAVs) [4]–[6]. However, this method's optimality relies on the choice of a proper time parametrization of the splines, which, given its intricate and highly nonlinear relationship with the optimization objective and the problem's constraints is not trivial. Indeed, most existing methods rely on heuristics for that task [5] and efficient optimization of time-allocation still remains an open question despite efforts in [4] and [7]. The lack of ability to efficiently optimize time allocation leads to two situations. In one situation the optimization takes a long time and is thus unable to be used in highly reactive problems. In the other one, heuristics are used and the trajectory has large jerks, which is more likely to fail due to thrust limit and consumes more energy. It is thus necessary to efficiently optimize the time allocation of UAV trajectories.

One approach to this problem was proposed by Mellinger *et al.* [4], who applied a gradient descent method to refine time allocation. Specifically, they divided the optimization problem in two subproblems (or levels): The lower level optimizes the path while timing is fixed using QP, and the upper level optimizes the time allocation using gradient descent. Similar approach is used in [7].

Nonetheless, finding the gradient of the cost function w.r.t. the time allocation with constraints present in the QP remains an unresolved issue even though analytic gradient is recently given for unconstrained QP [8]. To address this, finite difference method [4] has been employed. But it can be computationally expensive since the number of QPs that needs to be solved at each gradient step grows linearly with the number of spline segments. Moreover, the gradient is inaccurate due to truncation errors and difficulty in the choice of step size. As a result, using finite difference takes longer to converge and tends to converge to a worse cost.

Aiming to address this complexity, we use sensitivity analysis techniques to compute the gradient of the time allocation from the dual solution (Lagrange multipliers) of the QP. By exploiting the dual solution of the QP, our method allows us to compute exact analytical gradients w.r.t. time allocation, bypassing the downsides of finite difference methods: High-computational complexity and low accuracy. The gradient of time allocation is used with gradient descent method to optimize the time allocation.

Our framework combines bilevel optimization and analytic gradient. It decouples spatial and temporal variables and solves them hierarchically. The lower level solves the spatial variables and utilizes off-the-shelf convex solvers to handle the strong sensitivity of polynomial coefficients and constraints for collision

avoidance. The upper level optimizes temporal variables that are numerically better behaved with simple linear constraints using gradient descent with line search and projection.

One potential issue is the nonsmoothness of the upper level optimization. Fiacco [9] shows that the smoothness of the optimal cost of parametric optimization problems, i.e., objective function of the upper level optimization, requires smoothness of both the cost and constraints, strong convexity near optimum, and some constraint qualifications. In fact, these constraint qualifications do not hold universally and as a result the upper level is theoretically a nonsmooth optimization problem. Although this is not a problem in practice, for completeness our method treats objective function discontinuities using a subgradient method when nondifferentiability is detected preventing the progress of gradient descent. We prove the convergence of this method despite potential function nonsmoothness.

The bilevel optimization framework is guaranteed to yield feasible results at any point in its execution, allowing for arbitrary termination. In contrast, alternative solution methods based on nonlinear optimization such as direct collocation (DC) and joint optimization of spatial and temporal variables have no such guarantee due to the problem's strong nonlinearity. Even with feasible initial guesses, our experiments show that these methods do not have a high success rate. Compared with straightforward gradient approximation by finite difference, our analytic gradient is more accurate and computationally efficient. As a result, our method outperforms finite-difference baselines in terms of both computation speed and solution quality. One numerical example showing the effectiveness of our technique is in Fig. 1. Physical experiments using a real quadrotor on point-to-point navigation and dynamic goal tracking further demonstrate the effectiveness and real-time capabilities of our method.

This article is an extension of our conference publication [10], where the contributions include the following.

1) A bilevel optimization framework to optimize time allocation with analytic gradient.
2) Comparison with finite-difference, DC, and joint optimization.

In this article the new contributions include the following.

1) Demonstration of real-time capability in physical experiments.
2) Theoretical justification of the theorem being used.
3) Subgradient descent method to handle nonsmoothness of the cost function and proof of convergence.
4) Test of the algorithm in more complex environments and realistic environments represented by point clouds.
5) Theoretical and empirical scalability analysis to problems with more than 40 segments.

II. RELATED WORK

A. Trajectory Optimization for UAVs

Trajectory optimization solves the problem of computing the optimal trajectory for dynamic systems under some cost function and constraints. It is a widely used method for motion planning of robotic systems. See [11] for an introduction of trajectory optimization for general systems. Specifically, DC [12]
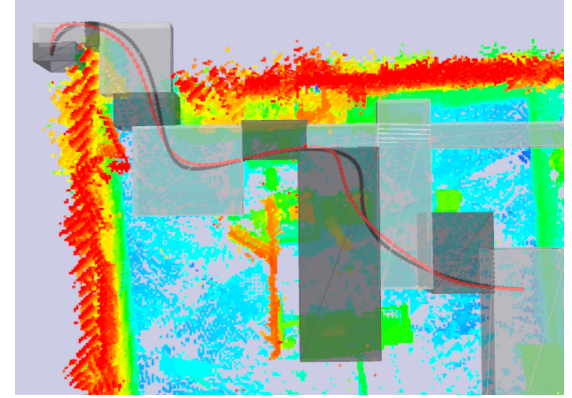
(a)
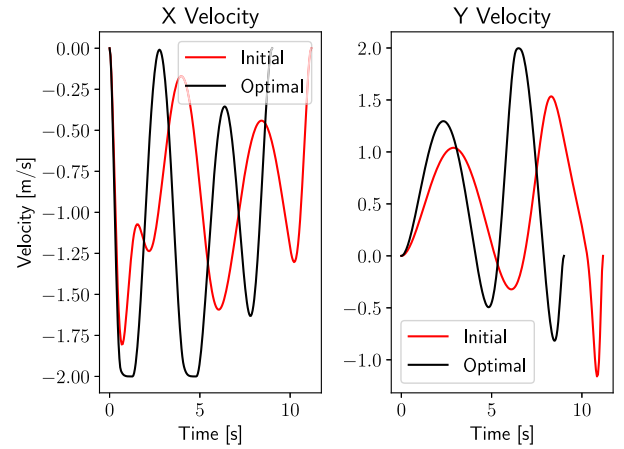
(b)

Fig. 1. Initial trajectory is computed by the heuristic time assignment from Gao *et al.* [5]. Our method optimizes a weighted sum of total jerk and time. a) Trajectories ying through a point cloud environment with the initial trajectory (red) and optimized trajectory using our method (black). The 12 boxes indicate the safe corridor in which the UAV is constrained. (b) Velocity proles of the trajectories in along the x and y direction.

has been widely used. However, pioneered by Mellinger [4], polynomial trajectories which exploit the differential flatness of UAV dynamics are often used for UAVs. With the snap of the trajectory as the cost function, a quadratic optimization suffices to compute the optimal trajectory. Some extensions to this framework include using Bézier curve control points as optimization variables and safe corridor generation to guarantee collision avoidance [5]. However, to use this framework, the temporal variables of the trajectory such as the duration of each segment have to be chosen prior to optimization. They are usually chosen by heuristics and this provides room for optimization of time allocations. Wang *et al.* [13] propose an alternating method but it lacks the ability to handle complex spatial constraints. Recent work [14] uses mixed-integer QP to solve for trajectories and guarantees safety by always having a feasible, safe back-up trajectory. However, the mixed-integer QP does not scale well with the number of convex polyhedra. Our article demonstrates the scalability of our method, which can solve problems in corridors with up to more than 40 polyhedra. Gao *et al.* [15] compute safe and aggressive trajectories in real

time from human-piloted trajectories, and generates them by alternatively optimizing the spatial and temporal trajectories. However, the two parts optimize different objective functions, so convergence is not guaranteed. On the contrary, our bilevel optimization method has been proven to converge to a local minimum of a unified objective.

B. Optimizing Time Allocation

As described above, trajectory optimization with polynomial splines is a well-studied problem as long as the spline timing is fixed. However, finding an optimal time allocation in real time is still challenging. One strategy [5], [6] is to use heuristics such as graph search on a discretized grid to generate a time allocation and keep timing fixed during the optimization stage. Heuristics in general are not optimal and can lead to inefficient trajectories, as shown in Fig. 1, where the initial trajectory has larger jerks than the optimum. Iterative methods such as gradient descent [4], [7] have also been used to optimize time allocation. However, if gradients are computed by finite differences, $(n + 1)$ QPs have to be solved for a problem with n segments for every gradient evaluation, making it slow and inaccurate. Another strategy to determine time allocation is to use sampling [3]. This approach randomly samples the duration of each spline segment until the corresponding QP can be solved, and has been applied successfully to humanoid locomotion problems. Because optimality is not emphasized, this method is only helpful in settings, where obtaining a feasible solution is the major bottleneck.

C. Bilevel Optimization

Bilevel optimization [16] refers to a mathematical program, where one optimization problem (the upper-level optimization problem) has another optimization problem (the lower-level optimization problem) as one of its constraints, i.e., one optimization task is embedded within another.

Bilevel and multilevel optimization techniques have been employed for switching time optimization for switched systems [17]–[20]. These works focus on calculating derivatives of an objective function with respect to switching times. In particular, works by Xu *et al.* [17] and Egerstedt *et al.* [18] compute the derivatives using Lagrange multiplier methods, which bear some resemblance to sensitivity analysis technique used in this article. However, these works are based on Pontryagin's maximum principle [11] and fall short of the capability to include inequality path constraints, which is often unavoidable in robotic applications.

Applications of bilevel optimization in robotics include trajectory optimization for legged robots [20], and robust control and parameter estimation [21]. Landry *et al.* [21] present a bilevel optimization solver based on an augmented Lagrangian method, however, their bilevel method is slower than directly solving the NLP in their experiments.

Many algorithms are available to solve bilevel optimization, and we refer readers to Sinha *et al.* [16] and Colson *et al.* [22] for more comprehensive treatments of the topic. Most closely related to our approach is the *descent method*, which seeks to decrease the upper-level objective while keeping the new point

feasible. Our method might be categorized as a descent method as we solve the upper-level optimization problem by gradient descent using gradients provided by the lower-level optimization problem.

D. Gradients in Bilevel Optimization

Efficiently and accurately computing gradients of the lower-level optimization problem is essential in applying gradient-based methods to solve bilevel optimization problems. The key derivation used in our algorithm is based on sensitivity analysis for parametric NLPs [9]. The method used to compute gradient in this article is very similar to Pirnay *et al.* [23], which provides the optimal sensitivity of solutions to NLP problems.

Efficient computation of gradients of optimization problems is also explored in the field of machine learning. OptNet [24] incorporates a QP solver as a layer into the neural network and is able to provide analytical gradients of the solution to the QP with respect to input parameters for back propagation. Gould *et al.* [25] presents results on differentiating argmin optimization problems with respect to optimization variables in the context of bilevel optimization.

III. METHODOLOGY

In this section, we describe our framework. We start by a mathematical formulation of the trajectory optimization problem for UAVs and end with a description of our algorithm.

A. Trajectory Optimization Preliminaries

In the case of UAV motion planning, differential flatness allows us to plan a trajectory in the UAV's four flat outputs $[x(t)^T \ \psi(t)]^T$ which consist of 3D position $x(t) \in \mathbb{R}^3$ and yaw angle $\psi(t) \in SO(2)$, without explicitly enforcing dynamics [4]. In this work, we plan in $\mathbb{R}^3$ by assuming the yaw stays constant, which is a common practice in UAV motion planning.

Trajectory optimization aims to find a trajectory $x : [0, T] \to \mathbb{R}^d$ that minimizes some measure of performance J while satisfying all the necessary constraints, e.g., being collision-free and dynamically feasible, and is formulated as

$$
\begin{aligned}
\underset{x, T}{\text{minimize}} \quad & J(x, T) = \int_0^T \ell(x, t)dt + \Phi(x(T)) \\
\text{subject to} \quad & x(0) = x_0 \\
& x(T) \in \mathcal{X}_{\text{goal}} \\
& g(x(t)) \leq 0, \quad \forall t \in [0, T] \\
& h(x(t)) = 0, \quad \forall t \in [0, T]
\end{aligned}
$$

where x encodes the trajectory, with the initial state x_0 and goal region $\mathcal{X}_{\text{goal}}$ prescribed. Total traversal time is denoted as T, which is sometimes fixed. The objective $J(x, T)$ is the sum of running cost $\int_0^T \ell(x, t)dt$ and terminal cost $\Phi(x(T))$. Constraints including dynamics, collision avoidance, and other system constraints are encoded in $g(\cdot)$ and $h(\cdot)$.

B. Safe Corridors

To guarantee that the whole trajectory will stay collision-free, we extract a safe corridor from the environment using the implementation from Gao *et al.* [5]. Here we give a brief overview of their method. Given a map represented by an occupancy grid or an Euclidean signed distance field, a start position and a goal position, a safe corridor is generated by taking the following steps.

1) Inflate all the obstacles in the map by a safety radius, so that the UAV can be considered as a point.
2) Find a path that connects the start and the goal using $A^\star$ search or fast marching method.
3) Grow a safe corridor consisting of convex polytopes around the path. In our implementation, we generate axis-aligned boxes by growing an axis-aligned box centered around each node in the path until it hits an obstacle, and then removing redundant boxes.

One such corridor and its corresponding environment[1] is shown in Fig. 1(a). Here, axis-aligned boxes are chosen for simplicity and compatibility with the grid data structures commonly used by perception algorithms. But in general, our method is applicable to corridors composed of any convex polytope.

C. Trajectory Optimization With Piecewise Bézier Curves

In this section we define a trajectory optimization problem in terms of spatial variables c (polynomial coefficients) and temporal variables y (durations of each segment of the curve).

We represent the trajectory as a piecewise Bézier curve of order d with n segments and segment durations $\Delta t_1, \ldots, \Delta t_n$. The timing of each knot point (connection point between two consecutive pieces) is given by $t_i = t_{i-1} + \Delta t_i$ with $t_0 = 0$. The ith segment is defined over the domain $[t_{i-1}, t_i]$ as

$$x(t) = \sum_{j=0}^{d} c_{ij} B_{d,j}\left(\frac{t - t_{i-1}}{\Delta t_i}\right), \quad t \in [t_{i-1}, t_i]$$

for each $i = 1, \ldots, n$, where $c_{ij} \in \mathbb{R}^3$ denotes the jth control point in the ith segment and $B_{d,j}$ denotes the jth Bernstein polynomial of order d defined as

$$B_{d,j}(u) = \frac{d!}{j!(d-j)!} u^j (1-u)^{d-j}.$$

We gather all the polynomial coefficients (control points) in the flattened vector $c \in \mathbb{R}^{3n(d+1)}$ and define the time allocation as $y = [\Delta t_1, \ldots, \Delta t_n]^T \in \mathbb{R}_+^n$.

1) Objective Function: Often, the objective function J is chosen to be the integral of the squared norm of some high-order derivative of the trajectory to penalize control effort. We use a more general objective

$$J(x, T) = \int_0^T \|x^{(q)}(t)\|^2 dt + wT \tag{1}$$

[1]Accessed: Jul. 31, 2020. [Online]. Available: http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/

which is a weighted sum of the integral of the squared L_2-norm of the qth derivative and the traversal time T, with weighting parameter denoted by w.

It has been shown that the first term in (1) can be written as a quadratic function of the coefficients c, with the quadratic matrix $P_q(y)$ determined by time allocation y and the order of derivative q [4], [5]. With a slight abuse of notation, we can write (1) in terms of polynomial coefficients c and time allocation y

$$J(c, y) = c^T P_q(y) c + \mathbb{1}^T y \tag{2}$$

where $P_q(y)$ is a symmetric positive semidefinite matrix that is nonlinear in y and $\mathbb{1}$ is a vector of 1 s. Note that we will drop the subscript q in $P_q(y)$ from now on since it is assumed to be $q = 3$ (we wish to find a minimum jerk trajectory) throughout this work.

2) Constraints on Continuity: Constraints on the trajectory should be enforced so that:

1) states at the start and end of the trajectory should match the initial state and (optional) final state;
2) continuities at knot points which ensure a smooth transition between each segment of the trajectory. If the trajectory needs to be C^k continuous, equality constraints up to the kth order should be applied at all knot points. We found that in the UAV case, applying continuity constraints up to acceleration yields good results, the same as [5].

The above constraints can be compiled into a linear equality constraint on the polynomial coefficients c

$$H(y)c = m \tag{3}$$

where the matrix H is generally nonlinear in time allocation y.

3) Constraints on Safety and Dynamic Feasibility: Safety and dynamical feasibility are ensured by imposing inequality constraints, such that:

1) the whole trajectory stays in the safe corridor discussed in Section. III-B;
2) the maximum velocity $\|x'(t)\|_\infty$ and maximum acceleration $\|x''(t)\|_\infty$ along the trajectory are bounded, i.e.,

$$\|x'(t)\|_\infty \leq v_{\max}, \quad \|x''(t)\|_\infty \leq a_{\max} \quad \forall t \in [0, T] \tag{4}$$

with $v_{\max}$ and $a_{\max}$ prescribed by the capabilities of the vehicle, user preference, or operational norms.

We encode the trajectory using a piecewise Bézier curve [5], which has the following properties.

1) The curve is totally contained in the convex hull of its control points.
2) The derivative of a Bézier curve is again a Bézier curve, with its coefficients being a linear combination of its antiderivative's coefficients.

Using these properties safety and dynamically feasible constraints can be imposed as a linear inequality constraint on the flattened coefficients [5]

$$G(y)c \leq h \tag{5}$$

where matrix G is generally nonlinear in time allocation y.

We note that constraining position, velocity, and acceleration using control points of Bézier curve does introduce some conservativeness since only the ends of the curve reach the

convex boundary even if all control points are at boundary. This effect can be seen in Fig. 1(b), in which the velocity limits of ± 2 m/s are only reached at a few discrete points. An alternative collocation implementation would use a grid along the trajectory, with constraints applied at grid points. Although collocation is less conservative, it would not guarantee feasibility at nongrid points. Another approach is to split the Bézier curve into more pieces so the curve can be represented by more control points. We refer readers to [26] for a more detailed discussion about the conservativeness of Bézier curve in a convex hull and potential alternatives. If the total trajectory time can be changed, one can optimize without velocity and acceleration bounds and simply increase the total trajectory time to satisfy the bounds without losing optimality of the time allocation. In fact, since our objective minimizes jerk, the dynamic feasibility is already considered in the cost function to some extent. No matter what representation is used, only the lower level optimization is affected and the efficacy of our analytic gradient computation and gradient descent method still holds.

4) Constraints on Time: Hard constraints on time allocation y may be imposed, such as a fixed total traversal time or that the duration of each segment must be positive. We encode these constraints as

$$Ay \leq b, \quad Cy = d \tag{6}$$

with A, b, C, d properly chosen.

D. Final Formulation

In summary, we collect (2), (3), (5), and (6), into the problem of trajectory optimization using Bézier spline in a corridor (TOBC)

$$\begin{aligned}
\underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y)c + w\mathbb{1}^T y \\
\text{subject to} \quad & Ay \leq b \\
& Cy = d \\
& G(y)c \leq h \\
& H(y)c = m
\end{aligned} \tag{TOBC}$$

which is nonlinear in time allocation y and convex (quadratic) in spline coefficients c for fixed y. This formulation generalizes the formulations found in [2], [3], [5], and [6].

We mainly study two variants of (TOBC). The hard time variant imposes a fixed traversal time and uses minimum-jerk as the objective (following Mellinger and Kumar [4])

$$\begin{aligned}
\underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y)c \\
& \mathbb{1}^T y = T_c \\
& y \geq \delta \\
& G(y)c \leq h \\
& L(y)c = m
\end{aligned} \tag{7}$$

where T_c is a fixed traversal time, e.g., chosen by a higher level planner, and δ is a small value (we use 1×10^{-6}) which ensures that durations are positive in each corridor. We note that the fixed traversal time T_c is necessary.

The soft time variant uses a weighted sum of jerk and traversal time as the objective (following Richter *et al.* [7]) rather than
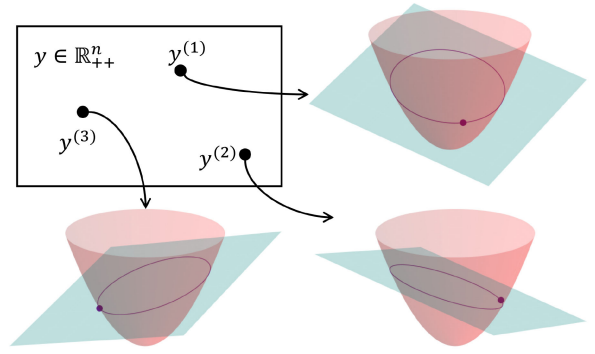

Fig. 2. Illustration of bilevel optimization: $y^{(1)}, y^{(2)}$, and $y^{(3)}$ are three feasible allocations, they are optimized in the upper-level optimization problem. Each of these y corresponds to a QP problem, which is being solved in the lower-level optimization problem. The red paraboloids are quadratic objective functions, cyan planes are equality constraints, no inequality constraints are drawn for illustration purposes. Feasible sets are purple curves, and purple dots are the optimal solutions to each lower-level optimization problem.

imposing hard constraints on traversal time

$$\begin{aligned}
\underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y)c + w\mathbb{1}^T y \\
& y \geq \delta \\
& G(y)c \leq h \\
& L(y)c = m.
\end{aligned} \tag{8}$$

One use case of (8) is tracking a dynamic goal, with the tracking aggressiveness tuned by the weight w.

E. Formulation of the Bilevel Optimization Problem

To efficiently solve (TOBC), we will rewrite it as a bilevel optimization problem, which is defined as follows [16].

Definition 1: A bilevel optimization problem is given by

$$\begin{aligned}
\underset{x_u \in X_U, x_l \in X_L}{\text{minimize}} \quad & F(x_u, x_l) \\
\text{subject to} \quad & x_l \in \underset{x_l \in X_L}{\text{argmin}}\{ f_0(x_u, x_l) : \\
& \qquad f_i(x_u, x_l) \leq 0, i = 1, \dots, m \\
& \qquad h_i(x_u, x_l) = 0, i = 1, \dots, p\} \\
& G_i(x_u, x_l) \leq 0, \quad i = 1, \dots, M \\
& H_i(x_u, x_l) = 0, \quad i = 1, \dots, P
\end{aligned}$$

where the upper-level optimization problem is defined by upper-level objective $F(\cdot)$ and upper-level constraints encoded in $G(\cdot)$ and $H(\cdot)$. The lower-level optimization problem, defined by lower-level objective f_0, with lower-level constraints $\{f_i(\cdot)\}_{i=1}^{m}$ and $\{h_i(\cdot)\}_{i=1}^{p}$, is embedded as a constraint in the upper-level optimization problem. The upper-level and lower-level decision variables are x_u and x_l, respectively. In Fig. 2 we illustrate a simple bilevel optimization problem, where $x_u \equiv y$ encodes the constraints and x_l is the lower-level variable to be solved for every instance of y. In this problem for every y the constraint is different, and so is the optimal solution. In general, both the cost function and constraint may depend on the upper level variable x_u, but here only the constraint depends on x_u. The goal in this problem is to find the optimal y, such that the corresponding lower-level problem has the smallest cost. Our analytic gradient

computes the gradient of the lower level optimal cost with respect to the upper level variable x_u.

Following this definition, we rewrite (TOBC) as:

$$\underset{c,y}{\text{minimize}} \ J(c,y) = c^T P(y) c + w \mathbb{1}^T y$$
$$\text{subject to } c \in \underset{c}{\arg\min}\{J(c,y) : G(y)c \leq h, \ H(y)c = m\}$$
$$Ay \leq b$$
$$Cy = d.$$

(TOBC-BO)

Note that although the objective functions $J(c,y)$ remain the same in both the lower-level (also the first constraint) and upper-level optimization problem, y is fixed in the lower-level optimization problem but becomes the optimization variable in the upper-level optimization problem. The lower-level problem is also a QP because $J(c,y)$ is quadratic in c when y is fixed.

Our solution strategy is to use constrained gradient descent on the function $J^\star(y) = J(c^\star(y), y)$ with $c^\star$ minimizing the QP for every time allocation y, i.e.,

$$c^\star(y) \in \underset{c}{\arg\min}\{J(c,y) : G(y)c \leq h, \ H(y)c = m\}. \quad (9)$$

Gradient descent has, indeed, been used to solve bilevel optimization problems [16], and our framework is a variant of this method. Note that in (TOBC-BO) and (9) we express $c \in \arg\min$ since the solution to the lower problem is in general a set. However, we prove in Section IV that the optimal solution to TOBC is unique. Given a feasible $y \in \mathbb{R}^n$, we find a direction $-\nabla_y J^\star(y) \in \mathbb{R}^n$ and a step length α that can make a sufficient decrease in $J^\star(y)$ while maintaining the feasibility of the new point $y_{\text{new}} = y - \alpha \nabla_y J^\star(y)$. The key issue with this approach is obtaining the gradients, which we address in the next section.

Because gradient descent does require a feasible initial guess of y, it is worth describing how such a feasible point can be determined. For TOBC, feasibility requires meeting velocity and acceleration limits, which requires enough time to be allocated to segments. If the initial and final states are static (with zero velocity and acceleration), one can increase the total trajectory time by scaling the time allocation to make it feasible. In other cases, however, an initial guess may not be feasible. One possible approach to mitigate this problem is to define a relaxed inner problem with slack variables if the initial guess is infeasible. We leave this problem to be addressed in future work.

F. Gradient Computation

We use a key result from sensitivity analysis of parametric nonlinear programming (NLP) [27, Th. 2.3.3] to derive the gradient $\nabla J^\star(y)$. In our problem, the lower-level objective is the same as the upper-level one, which allows us to derive gradients of the upper-level decision variables through sensitivity analysis. For brevity, we give results of first-order sensitivity analysis and refer readers to [27] for details.

Theorem 1: Consider the problem of finding the local solution $c(y)$ of a parametric NLP problem

$$\underset{c}{\text{minimize}} \ J(c,y)$$
$$\text{subject to } g_i(c,y) \leq 0, \quad i = 1, \ldots, m$$
$$\qquad\qquad\quad h_j(c,y) = 0, \quad j = 1, \ldots, p$$

where c is the vector of decision variables and $y \in \mathbb{R}^n$ is a parameter vector. Let $c^\star(y_0)$ be a locally optimal solution, and let λ and ν be Lagrange multipliers associated with $g(\cdot)$ and $h(\cdot)$, respectively.

If the following conditions hold.
1) Functions $J(\cdot)$, $g_i(\cdot)$ (for all i), and $h_j(\cdot)$ (for all j) are twice continuously differentiable in c, and their gradients w.r.t. c and the constraints $g_i(\cdot)$ (for all i) and $h_j(\cdot)$ (for all j) are once continuously differentiable in y in a neighborhood of $(c^\star, y_0)$.
2) Objective $J(c,y)$ is twice continuously differentiable in (c,y) near $(c^\star, y_0)$.
3) The strong second-order sufficient conditions (SSOSC) hold at $c^\star(y_0)$.
4) The gradients $\nabla g_i(c^\star, y_0)$ (for i, such that $g_i(c^\star, y_0) = 0$) and $\nabla h_j(c^\star, y_0)$ (for all j) are linearly independent,

then in a neighborhood of $y = y_0$, the gradient of the objective is

$$\nabla_y J^\star(y) = \nabla_y J + \sum_{i=1}^m \lambda_i(y) \nabla_y g_i + \sum_{j=1}^p \nu_j(y) \nabla_y h_j. \quad (10)$$

Moreover, similar theorem in [9] further requires the strict complementary slackness (SCS), i.e., $\lambda_i > 0$ when $g_i(c^\star, y_0) = 0$ but only requires second-order sufficient condition. With SCS, the active set which is the collection of inequality constraints, where equality holds, i.e., $\{i \in \{1, \ldots, m\} | g_i(c^\star, y_0) = 0\}$ does not change. Theorem in [27] does not require SCS and this means $J^\star(y)$ is differentiable even if some constraints switch between being active and nonactive.

In our problem, conditions 1) and 2) are satisfied by construction; SSOSC can be proved (see Appendix A for details). However, condition 4), also known as the linear independence constraint qualification (LICQ) may fail in some cases. Moreover, this condition can only be verified after the NLP is solved and the pattern of active constraints is known. When LICQ holds, the Lagrangian multipliers are unique and (10) computes the exact gradient of the objective. When LICQ does not hold, however, the Lagrangian multipliers associated with linearly dependent constraints are not unique. The Karush–Kuhn–Tucker (KKT) condition is satisfied for infinite number of multipliers. As a result, the gradient computed by (10) may not be unique. In Section IV-C, we show that under the assumption of Slater's condition [28], equation (10) computes a subdifferential. Slater's condition only requires the existence of a feasible solution in the interior of the convex feasible set and is not a strict assumption.

G. Solving Bilevel Optimization

Our algorithm, given in Algorithm 1, uses a combination of gradient descent and subgradient descent to solve (TOBC-BO). It takes an initial guess of the time allocation y_0 as input. It then iteratively descends $J^\star(y)$ until some optimality conditions are satisfied or the maximum number of iterations is reached. The usual step is a gradient descent step, but if it is detected that the function is nondifferentiable, the algorithm switches to take a subgradient step. Subgradient descent is widely used in training deep neural networks for nonconvex and nonsmooth objective

---

Algorithm 1: Refine-Time $(y_0, \alpha_{\text{sub}})$.

1: $y \leftarrow y_0, n_{\text{sub}} \leftarrow 0, J_{\text{opt}} \leftarrow \infty, y_{\text{opt}} \leftarrow 0$
2: **for** $i \leftarrow 0$ to *max-iterations* **do**
3:      $J, \lambda, \nu \leftarrow$ Solve-QP$(P(y), G(y), h, H(y), m)$
4:      $g \leftarrow$ Get-Gradient(λ, ν) ▷From (10)
5:      $p \leftarrow$ Project-Gradient(g, A, b, C, d)
6:      $\alpha, J, y \leftarrow$ Line-Search(y, p)
7:      **if** α not found **then**
8:          $y \leftarrow y - \alpha_{\text{sub}} p/(n_{\text{sub}} + 1), n_{\text{sub}} \leftarrow n_{\text{sub}} + 1$
9:          $J \leftarrow J(y)$
10:     **else if** optimality-conditions-satisfied **then**
11:         **break**
12:     **end if**
13:     **if** $J < J_{\text{opt}}$ **then**
14:         $(J_{\text{opt}}, y_{\text{opt}}) \leftarrow (J, y)$
15:     **end if**
16: **end for**
18:   **return** y_{opt}

Algorithm 2: Line-Search (y_s, p).

1: **static variable** α_0
2: **constant variables** $\tau_g > 1, 1 > \tau_s > 0$
3: $\alpha \leftarrow \alpha_0$
4: **for** $i \leftarrow 0$ to *max-iterations* **do**
5:      $y = y_s - \alpha p$
6:      $J, \lambda, \nu \leftarrow$ Solve-QP$(P(y), G(y), h, H(y), m)$
7:      **if** sufficient-decrease-achieved **then**
8:          **if** $i = 0$ **then**
9:              $\alpha_0 \leftarrow \tau_g \alpha$
10:         **else**
11:             $\alpha_0 \leftarrow \tau_s \alpha$
12:         **end if**
13:
14:         **return** α, J, y
15:     **end if**
16:     $\alpha \leftarrow \tau_s \alpha$
17: **end for**
19:   **return** "not found", J, y

---

functions. Subgradient descent alone, however, is quite slow due to diminishing step sizes. In our problem, the function is smooth in most regions so gradient descent with line search is usually far more efficient. As we shall see in the experiments, the subgradient step is rarely taken, but can kick the algorithm out of states, where gradient descent gets stuck.

Line 1 solves a QP problem with a time allocation y and then returns the objective value J and the dual solution (Lagrange multipliers) λ and ν. Line 1 computes the estimated gradient of the objective w.r.t. time allocation y with the Lagrange multipliers λ and ν using (10). Line 1 finds a normalized descent direction from the gradient by projecting the gradient onto the null space of equality constraints $Cy = d$ [29]. In the hard-time variant that we consider, the constraint is that total time is a constant, and hence the projected gradient is computed as $p = g - \frac{1}{n} \sum i = 1^n g_i$.

Line 1 calls the line search Algorithm 2 to find a suitable step length α that meets the inequality constraints on y and gives sufficient decrease in the objective function. If α cannot be found, we assume y is near a nonsmooth region, so Algorithm 1 takes a subgradient step without checking for sufficient decrease. If taken, the first subgradient step size α_{sub} is initially set to the initial α at the step when line search fails. If a line search step can be found, the following optimality conditions are checked in Line 7.

1) Norm of the projected gradient is less than 1×10^{-3}.
2) The change of the absolute or relative objective function is less than 1×10^{-3}.

Since the subgradient step may actually increase the objective function's value, the best solution during all iterations is returned (Lines 12–14).

We use an adaptive backtracking method to find a step α to achieve the Armijo sufficient decrease condition [29]. If it fails to find a decrease, "α not found" will be returned as in Line 2. The initial step length α_0 defined in Line 2 will be updated adaptively. The update strategy is similar to the update of trust

region radius in a trust region algorithm [29]: α_0 will grow or shrink based on the decrease achieved in the first iteration, shown in Line 2 and Line 2, respectively. We find adaptive line search useful since it reduces the number of QPs that are solved during line search and time for solving QPs dominates our algorithm's time complexity.

For a candidate step $y \rightarrow y - \alpha p$, where α is the step size and p is the gradient, a nonsmoothness occurs when some point along this line segment fails to meet the LICQ condition and first-order Taylor expansion does not well approximate the function value. However, we do not attempt to detect every point at which the objective is nonsmooth, because the gradient can still make adequate progress (as determined by the sufficient decrease condition) and it is practically computationally expensive to do. Instead, we decide to trigger the subgradient step only when backtracking line search fails to find a sufficient cost decrease. The subgradient steps use a standard diminishing step size to guarantee convergence. We justify this decision further with the convergence analysis of Section IV. Moreover, our experiments in Section V-C suggest that introducing subgradient steps increases the number of iterations, but with the benefit of providing more opportunities to improve the objective.

IV. CONVERGENCE ANALYSIS

Although gradient descent with line search is convergent in smooth regions [29], we must study the behavior of Algorithm 1 in nonsmooth regions when it switches to subgradient methods. The proof is mainly based on results from Davis *et al.* [30] which prove the convergence of the stochastic subgradient descent method for a wide variety of functions including those represented by deep neural networks. We prove convergence by showing the cost function of the upper optimization in our problem satisfies the conditions of the theorem in [30]. We note that the proof relies on the very structure of our problem and is not necessarily true for other bilevel optimization problems.

Specifically, the main property we use is the convexity of the lower level problem.

A. Clarke Subdifferential

For a locally Lipschitz continuous function $f : \mathbb{R}^d \to \mathbb{R}$, the *Clarke subdifferential* [31, Ch2, Th. 8.1] of f at any point x is the set

$$\partial f(x) \equiv \text{conv} \left\{ \lim_{i \to \infty} \nabla f(x_i) : x_i \xrightarrow{\Omega} x \right\} \qquad (11)$$

where Ω is any full-measure subset of $\mathbb{R}^d$, such that f is differentiable at each of its points; conv means the convex hull of the limit of gradient for all sequences in Ω approaching x and a point is *(Clarke) critical* if $0 \in \partial f(x)$. An arc $x(t) : \mathbb{R}_+ \to \mathbb{R}^d$ is called a *trajectory* if it satisfies the differential inclusion

$$\dot{x}(t) \in -\partial f(x(t)) \quad \text{for a.e. } t \geq 0. \qquad (12)$$

An iteration sequence is used to track the trajectory

$$x_{k+1} = x_k + \alpha_k (g_k + \xi_k) \qquad (13)$$

where $\alpha_k > 0$ is a sequence of step sizes that is square summable but not summable, $g_k \in -\partial f(x)$, and ξ_k is noise which is zero in our case. Then by [30, Th. 3.1] the sequence generated by (13) approximates the trajectory of the differential inclusion.

B. Convergence Result

In order for the trajectory of differential inclusion to converge to a critical point, a sufficient condition is that $f(x)$ must locally Lipschitz and semianalytic [30, Th. 5.9]. This condition requires $f(x)$ to be piecewise analytic which fits naturally in our case, where different combinations of active inequality constraints result in piecewise functions. It suffices to show within each piece, that the optimal cost of the lower optimization is an analytic function. Considering that the lower problem is a QP which can be solved by inverting a matrix whose entries are analytic functions of the time allocation y given $y > 0$, the optimal solution is an analytic function, and, consequently, so is the cost function of lower optimization.

It remains to show that each piece is connected, i.e., the optimal lower cost is continuous with respect to the time allocation y. This can be shown by [9, Th. 2.1] which states that the optimal cost is continuous as long as: The set-valued mapping from time allocation to the feasible set (of lower optimization) is continuous; the feasible set is compact; and the objective function is continuous. These conditions are easy to verify in our problem. As a result, the subgradient methods being used in Algorithm 1 converges to a critical point as long as 1) the gradient from (10) is indeed a subdifferential even when LICQ fails to hold and 2) the constraints in the upper level optimization do not affect the convergence of the algorithm. These two conditions are discussed in the next two sections.

C. LICQ Failure

LICQ fails when, at the optimum point, the linearized equality and active inequality constraints are linearly dependent. In that

case, the Lagrangian multipliers associated with those constraints are nonunique. In our problem, the lower optimization is QP and, thus, if LICQ fails to hold, it means that some rows of H and $G_{\mathcal{A}}$ are linearly dependent, where $G_{\mathcal{A}}$ is the collection of inequality constraints that are active. Commercial QP solvers like Sqopt and Mosek have a presolve process that eliminates redundant constraints so the actual QP being solved returns unique multipliers and the redundant constraints have multipliers of zero.

When LICQ fails, we have to assume Slater's condition [28] holds. The Slater's condition imposes restrictions on the feasible set and requires the existence of a point in the feasible set, such that inequality constraints are strictly smaller than 0. The Slater's condition is not strict and only fails to hold for pathological cases in our problem. One pathological example is due to velocity limit and time limit, the drone has to be at the maximum velocity to travel from one end to another. Considering a 1D single corridor case starting from 0 to 1 within 1 s. The velocity limit of 1 m/s requires all velocity inequalities constraints to be active and the feasible set has no interior. This case, however, is close to infeasibility so it rarely occurs. Besides, this case is unstable and small perturbation to y solves the issue. For convex problems, Mangasarian–Fromovitz constraint qualification (MFCQ) [32] holds if Slater's condition holds. Then the optimal cost of the lower problem is directional differentiable [32, Th. 7.3] in any direction and the directional derivative is given by

$$\nabla_y J^*(\bar{y}, z) = \min_{\bar{c} \in S(\bar{y})} \max_{\mu \in M(\bar{c}, \bar{y})} \nabla_y L(\bar{c}, \bar{y}, \mu) z \qquad (14)$$

where z with $\|z\| = 1$ is any direction; $S(\bar{y})$ is the set of minimizers which is singleton in our case due to uniqueness of the optimal solution; μ is the Lagrangian multipliers vector and $M(\bar{c}, \bar{y})$ is the set of valid multipliers that satisfy KKT conditions. In our problem, the optimum is unique due to SSOSC so we can remove the minimum operator. The directional derivative is obtained by essentially solving a linear program to compute the multipliers. According to the proof within [32, Th. 7.3], M is compact if MFCQ holds. Since constraints are linear, M is essentially a polyhedron defined by equality and inequality constraints of KKT equations, so is the set of values of $\nabla_y L(\bar{c}, \bar{y}, \mu)$ due to its linear dependency on μ. For all the vertices, there exists a z, such that the maximum is obtained at the vertex due to convexity. In that direction, $\nabla_y L(\bar{c}, \bar{y}, \mu)$ computes the exact gradient. So according to the definition in (11), $\nabla_y L(\bar{c}, \bar{y}, \mu)$ is within the subdifferential. So the polyhedron of $\{\nabla_y L(\bar{c}, \bar{y}, \mu) | \mu \in M(\bar{c}, \bar{y})\}$ is a subset of the Clarke differential, and so is the gradient computed by any valid μ.

In fact, $\nabla_y L(\bar{c}, \bar{y}, \mu)$ may be unique for all $\mu \in M(\bar{c}, \bar{y})$ such as when the nondependent constraints do not depend on y. In that case, the exact gradient is computed. One example in our problem is when two corridors share a face (denoted as the plane $x = b$) and the constraints include $x_i \leq b, x_j \geq b, x_i = x_j$, where x_i and x_j is the last and first control points of the two corridors, respectively. The first two constraints are the control points that are within their corresponding corridors and the third one is path continuity. At optimum these three constraints are always linearly dependent but do not affect the gradient computation

since they do not depend on the time allocation. Moreover, one redundant inequality constraint is removed during the presolve process of the optimizers.

D. Constraints in the Outer Optimization

This section shows that even with constraints in the outer optimization, the theory in Section IV-A can be applied. The outer optimization in the soft-time variant as in (8) has the form

$$\underset{y}{\text{minimize}} \; J(y) = c^\star + w \mathbb{1}^T y \tag{15}$$
$$y \geq 0.$$

The problem is essentially unconstrained at optimum. The reason is that as $y_i \to 0$, the cost function approaches ∞ and nonpositive durations can be rejected during line search. So the constraint $y \geq 0$ can be ignored in analysis.

The hard-time variant as in (7) has the form

$$\underset{y}{\text{minimize}} \; J(y) = c^\star(y) \tag{16}$$
$$y \geq 0$$
$$\mathbb{1}^T y = T_c$$

with fixed total traversal time T_c.

Here we use a projected (sub)gradient method that computes the (sub)gradient $g \in \mathbb{R}^n$ and projects it onto the hyperplane $\mathbb{1}^T g = 0$ so an update does not change the total traversal time. The projection operator is given by

$$\mathcal{P}g = g - \left(\mathbb{1}^T g / n \right) \mathbb{1} = (I - \frac{1}{n} \mathbb{1} \mathbb{1}^T) g$$

and the update rule is

$$y' \leftarrow y - \alpha \mathcal{P}g = y - \alpha \left(I - \frac{1}{n} \mathbb{1} \mathbb{1}^T \right) g. \tag{17}$$

We show that this is equivalent to (sub)gradient descent on an unconstrained problem

$$\underset{s}{\text{minimize}} \; \tilde{J}(s) = c^\star(y_0 + As) \tag{18}$$
$$y_0 + As \geq 0$$

where $s \in \mathbb{R}^{n-1}$; y_0 is the initial time allocation which satisfies $\mathbb{1}^T y_0 = T_c$; and $A \in \mathbb{R}^{n \times (n-1)}$ is a matrix forming the orthonormal basis of the null space of $\mathbb{1}$. A can be obtained by the singular value decomposition $\mathbb{1} = USV^T$, with $U = [1]$, $S = [\sqrt{n}, 0, \ldots, 0]$, and $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix with first column $\mathbb{1}/\sqrt{n}$ and the remaining rows equal to A, that is $V = [\mathbb{1}/\sqrt{n}|A]$.

The value y corresponding to an iterate s is given by $y_0 + As$. Using the same argument as in the soft-time variant, we can show the inequality constraint does not affect convergence. Each iterate of gradient descent satisfies the constraint $\mathbb{1}^T(y_0 + As) = T_c$. Using chain rule, the (sub)gradient of $\tilde{J}$ w.r.t. s is $A^T g$, where g is a (sub)gradient of J at $y_0 + As$. The update rule is thus

$$s' \leftarrow s - \alpha A^T g$$

so the equivalent time allocation update is

$$y' \leftarrow y_0 + As' = y_0 + As - \alpha AA^T g = y - \alpha AA^T g. \tag{19}$$

Fig. 3. Second floor plan used to generate random test problems. Here the orange and light green lines shows where obstacles along z axis are placed.

Using the orthogonality of V we have

$$I = VV^T = \begin{bmatrix} \mathbb{1}/\sqrt{n} & A \end{bmatrix} \begin{bmatrix} \mathbb{1}^T/\sqrt{n} \\ A^T \end{bmatrix} = (\mathbb{1}\mathbb{1}^T)/n + AA^T$$

which demonstrates that $AA^T = I - \frac{1}{n} \mathbb{1} \mathbb{1}^T$. Hence, the unconstrained update rule (19) is equivalent to (17).

V. EXPERIMENTS

Our algorithm, which is released as an open-source package,[2] is implemented in Python. The QP solvers are called with interfaces to C++ libraries. Since solving QP dominates the running time, our reported computation time is similar to a pure C++ implementation with some overhead from Python.

A. Numerical Experiments

We evaluate our method on random instances of "indoor flight." We take the indoor building environment from [33], extrude it along z axis, and discretize the z direction into 5 cells. We then select some rows and columns in the image as shown in Fig. 3 and fill the bottom 3 or top 3 cells (along z axis) as obstacles according to the color. The start and goal are randomly sampled in the free space so in most cases there is movement along z axis. The environment is shown in Fig. 3. The complexity of the environment allows us to generate problems with more than 40 segments. 200 problems are randomly generated. One example is shown in Fig. 4. We note that despite exclusively using axis-aligned boxes as safe corridors for simplicity, convex polyhedrons may yield less conservative results and our method is able to handle it as well. But there is a tradeoff between the complexity of corridors and optimality of the solution.

We solve the hard time variant as in (7), with the initial guess of time allocation and fixed total traversal time computed from the heuristic introduced by Gao et al. [5]. In this section, the QP solver is Gurobi,[3] a commercial interior-point QP solver. We set a major iteration limit of 50 to limit the total computation time.

[2][Online]. Available: https://github.com/OxDuke/Bilevel-Planner
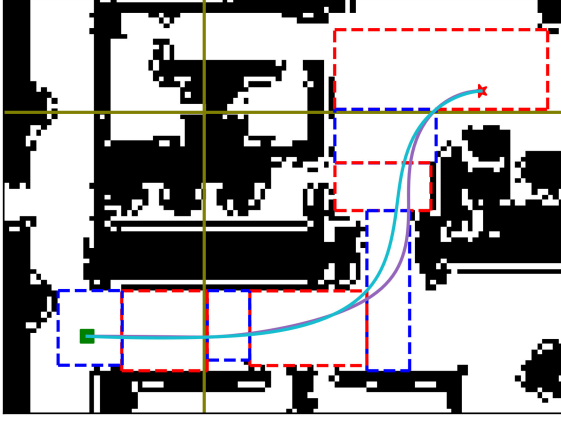[3][Online]. Available: https://www.gurobi.com/

Fig. 4. Random problem instance. The start (square), goal (star), and generated corridor (dashed boxes) are shown. The purple and cyan curve are the optimal trajectories with the initial and final time allocations.

TABLE I
MEAN TOTAL COMPUTATION TIME, AVERAGE MAJOR ITERATION TIME, AND NORMALIZED COST (FINAL COST OVER THE COST FROM HEURISTIC ASSIGNMENT) OF FINITE DIFFERENCE (FD) W.R.T. ANALYTIC GRADIENT (AG)

	Total Time (s)	Avg. Iter. Time (s)	Normalized Cost
FD	15.403	0.497	0.109
AG	0.874	0.024	0.068

The velocity limit is set as 2 m/s. To handle cases with infeasible initial time allocation, we simply multiply the time allocation by a scalar until feasibility is obtained. As a benchmark, we compare our analytic gradient with a finite-difference approximation method under the same bilevel optimization framework. Another method in comparison is to optimize the control points and time allocation simultaneously using nonlinear optimizer SNOPT [34]. Finally, we compare with DC method [11] which discretize the trajectory, formulate an NLP, and solve it using SNOPT. These experiments are carried out on a workstation with a 3.30-GHz Intel Xeon W-2155 processor, using only one thread.

1) Finite Difference Versus Analytic Gradient: In this section, we solve the same problem set as before and compare the analytic gradient from Lagrangian multipliers defined in this article with finite difference approximation, where one gradient computation requires additional n QPs being solved. The average performance is shown in Table 1. Clearly, analytical gradients are not only much faster but also help the optimizer converge to a better solution because of their higher precision.

2) Comparison With Joint Optimization: Joint optimization directly solves (7) as an NLP using SNOPT [34], a general nonlinear solver for sparse, large-scale problems. On the contrary, bilevel optimization decouples temporal and spatial variables and solves them hierarchically. We provide analytic gradients to SNOPT for solver robustness and explore problem sparsity to the best of our ability. We initialize SNOPT with the unrefined time allocation and the spline coefficients computed in the first QP solve. All the stopping criteria are set to default except the optimality tolerance is set to 1×10^{-3}.

TABLE II
SUCCESS RATE OF BILEVEL OPTIMIZATION, JOINT OPTIMIZATION, AND DIRECT COLLOCATION

Bilevel Optimization	Joint Optimization	Direct Collocation
200/200	11/200	44/200

We observed that joint optimization is susceptible to the strong nonlinearity of the problem and only 11 out of 200 problems converged to a feasible solution, as shown in Table 2. Due to the low success rate, it's clearly not suitable for the application. SNOPT tends to terminate prematurely without converging, and often moves to an infeasible point even though it starts from a feasible solution. We believe this is because the joint spatial and temporal NLP is ill-conditioned. The QP objective function exhibits high-order dependence on timing, and some spatial constraints are very sensitive to the high-order spline coefficients. On the other hand, in the bilevel formulation, the ill-conditioned problem is handled by convex solvers, which are known to be more robust. We also note that SNOPT has no guarantee on obtaining a feasible solution while our approach can be terminated at any time and return a feasible solution. Usually nonlinear optimizers require an initial guess close to the optimum values to converge. With the same initial guess of time allocation, our method is able to make progress toward optimum while SNOPT moves from a feasible initial guess to nonfeasible solutions.

3) Comparison With Direct Collocation: We also compared our proposed method with the DC method [11] to solve problem (7) as an alternate NLP formulation. DC optimizes over discretized states $[p(t), \dot{p}(t), \ddot{p}(t)]$, where $p(t)$ is position and control is $u \equiv \dddot{p}(t)$ at each collocation grid point $t_0, \ldots, t_N$. To adjust the timing of each segment, the initial and final times of each segment are introduced as additional decision variables. The state and control trajectories are optimized simultaneously with segment times. Each segment has a fixed grid size. See Appendix B for the details.

Once again, we use SNOPT to solve the NLP. The major iteration limit and total iteration limit are set as 500 and 5000, respectively, to keep the total computation time manageable. Due to the different NLP formulation, the final objective value from DC cannot be directly compared to other approaches so we just compare success rate instead. DC does not perform well on this problem and only 44 out of 200 problems converge to an optimal solution, as shown in Table 2. We observed that with a higher iteration limit, DC can achieve a higher success rate. However, the average computation time with the current settings is already 7.12 s, which is unsuitable for responsive quadrotor flight.

B. Scalability Study

For complex environments, the number of safe corridors may be as large as around 50 and thus imposes a challenge to the computational efficiency. We perform an empirical scalability study to see how our framework performs with increasing number of segments using the same problems in the last section. We also study the effect of QP solvers and compare two types
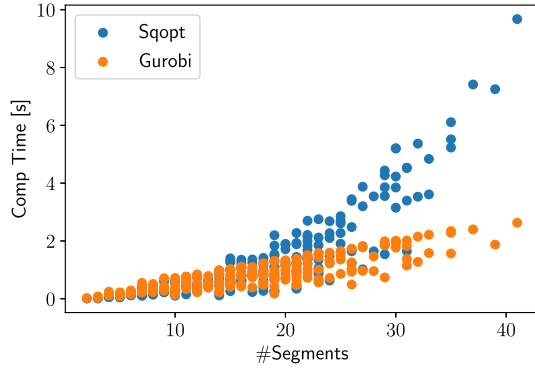
Fig. 5. Total computation time of our algorithm as a function of the number of segments in random problems.
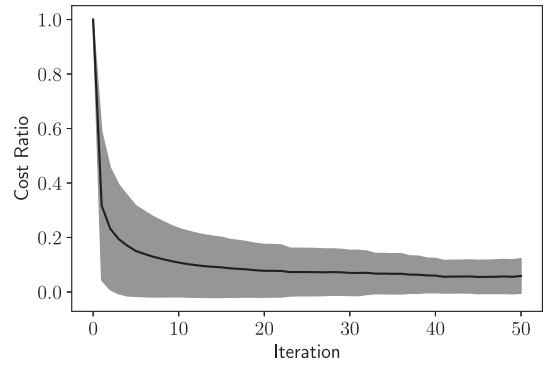

Fig. 6. Profile of the average (solid line) and standard deviation (shaded) of the cost ratio over the initial cost. The cost ratio is computed by dividing the cost at current iteration by the initial cost.

of QP solvers: Active-set solver Sqopt [35] and interior-point solver Gurobi. Both solvers are designed for sparse and large-scale problems. Our previous test suite [10] showed that Sqopt performs better when the average number of segments is below 10. In this article we want to study how it performs when the problem has more segments. We use the same setting as the last section with the two QP solvers.

It turns out the performance of our algorithm in terms of cost function is quite consistent with the two solvers. Numerical errors result in slight difference in the progress of the algorithm in each solver. We note that we set an iteration limit of 50 and around 60 problems out of 200 are terminated after reaching the iteration limit. On average each iteration requires 1.2 QPs being solved. This also indicates the total computation time is roughly proportional to the average time of each QP solving. However, the computation time is quite different for the two solvers, indicating different scalability in QP solving time.

The computation time is different since Sqopt and Gurobi have different scalability with the number of segments which is proportional to the number of optimization variables and constraints. Gurobi implements an interior-point method so its iteration number is roughly constant. In each iteration a sparse block-diagonal matrix is factorized which takes time linearly to the number of segments. Sqopt implements the active-set method which does not scale well since it takes more iterations to identify the correct active set. However, it is faster than Gurobi when the number of segments is below 10, which is consistent with the results in [10]. If finite-difference is used to estimate the gradient, the algorithm has quadratic and cubic scalability when the QP solver is Gurobi and Sqopt, respectively.

The total computation time scales similarly to average QP times since similar numbers of QPs are solved, as shown in Fig. 5. The total computation time scales linearly, when Gurobi is the QP solver. The total computational time shows the potential of applying our algorithm in real time since the major improvement of cost functions occurs in the first few iterations, evidenced in Fig. 6. For large problems with limited computation time, a smaller iteration limit has to be used, although it does not affect the cost function too much.
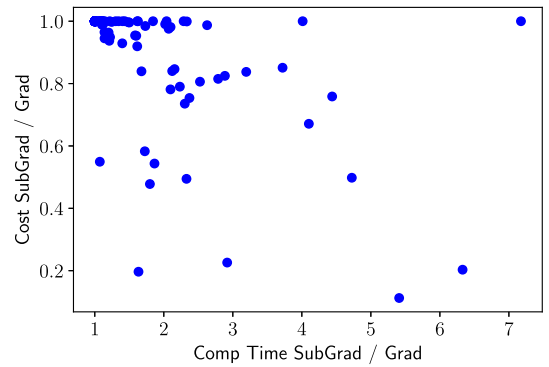

Fig. 7. Enabling subgradient descent can improve cost when pure gradient descent gets stuck. The x and y axes are the ratios of computation time and cost function, respectively, of Algorithm 1 with subgradient descent enabled versus the disabled variant. Lower is better for both axes.

C. Effect of Subgradient Step

Although the numerical experiments above are conducted on challenging 3-D problems, we found that the subgradient step of Algorithm 1 is only triggered once. To better examine the effect of subgradient steps, we designed a problem set that triggers subgradient steps more frequently. These variants project the environments into 2-D to eliminate vertical movement, and velocity limits are disabled. 808 random problem instances of this form were generated, and we compared Algorithm 1 with and without Lines 9–12.

The results are shown in Fig. 7, with both Sqopt and Gurobi as the underlying QP solver. A subgradient step was taken in 15 problems using Squopt, and in 65 problems using Gurobi. The discrepancy between solvers is due to the slightly different gradient estimation, which may lead to different algorithmic behavior overall. The plot combines results from subgradient steps taken with either QP solver. For many problems, the subgradient step decreases the cost substantially. However, it does pay a price in terms of increased computation time. The reason is that the gradient-only variant stalls out more quickly, while the subgradient variant continues to optimize and make progress.
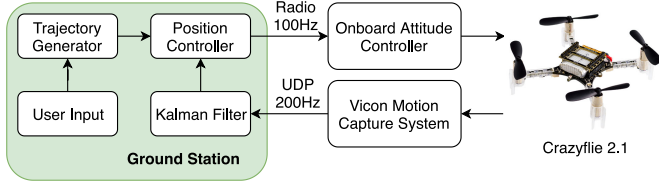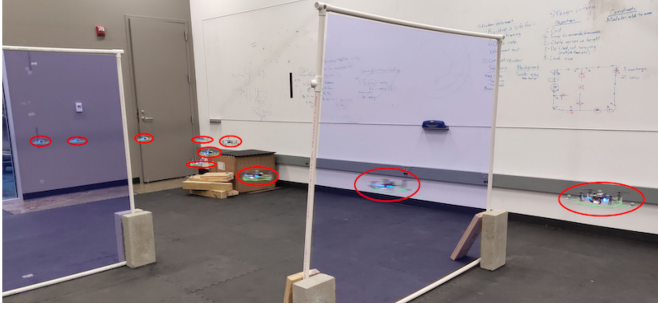
Fig. 8.   Physical quadrotor system setup.

Fig. 9.   Obstacle layout for physical quadrotor experiments, with the space within the frames (tinted blue) are treated as obstacles. Evenly spaced frames from the executed trajectory are overlaid, with the quadrotor circled in red.

D. Physical Experiments

We validate our planner on an indoor obstacle avoidance scenario using a commercially available small-scale quadrotor, Crazyflie 2.1.[4] A video that compiles all the physical experiments is provided as supplementary material.

The system setup is shown in Fig. 8. The position of the quadrotor is captured by the Vicon[5] motion capture system and transmitted to ground control station using Ethernet at 200 Hz. The raw data stream from Vicon goes through a Kalman filter and then serves as feedback for a position controller on the ground control station. The position controller is a feedfoward-feedback controller, with the feedforward term computed from the reference trajectory and its time derivative thanks to the differential flatness property, and the feedback term computed from a proportional-integral-differential controller.

As shown in Fig. 9, we set up two frames as walls, and considered them as obstacles in our algorithm. We run three experiments to demonstrate the effectiveness and real-time capability of our algorithm. Admittedly, in these physical experiments the number of segments is not large. For problems with more segments, a smaller iteration limit can be used without losing too much optimality. Indeed, the property of any-time feasibility of our algorithm is suitable for challenging problems with many segments since the user can set arbitrary number of iterations.

1) Comparison to Time Allocation Heuristics: Our algorithm plans a faster trajectory while achieving the same control effort (jerk) as Gao *el al.* [5], which uses time allocation heuristics. The quadrotor starts at some initial position and is asked to travel to a target position chosen by a human operator using an Rviz[6] interface while avoiding obstacles. Both methods are

[4][Online]. Available: https://www.bitcraze.io/
[5][Online]. Available: https://www.vicon.com/
[6][Online]. Available: http://wiki.ros.org/rviz

TABLE III
COMPARING AGAINST HEURISTIC TIME ASSIGNMENT

Method	Length	Traversal Time	Jerk
Ours	**5.15** m	**4.36** s	39
Gao et al.	5.82 m	5.32 s	39

TABLE IV
COMPARING MOTION AGGRESSIVENESS PARAMETERS

Weight (w)	Traversal Time	Computation time	Jerk
10	5.60 s	10.8 ms	11.2
20	4.96 s	10.7 ms	19.9
40	4.42 s	10.7 ms	36.1
80	4.01 s	9.3 ms	64.7

set up to solve the hard time variant (7), but after running our algorithm, we scale the total traversal time T until the jerks of two trajectories becomes the same. Table 3 indicates that our algorithm yields a 12% shorter and 18% faster trajectory with equivalent jerk.

2) Controlling Aggressiveness Using Time Penalty w: Next we show the soft time variant (8) can handle objectives with various time penalties to control aggressiveness. Results are summarized in Table 4. These indicate that, as expected, when the weight penalty increases, trajectories become faster and more jerky. Also, computation time is not significantly affected by the weight parameter.

3) Tracking a Dynamic Goal: The final experiments show the algorithm running in real time, where the quadrotor tracks a dynamic goal moved by a human while avoiding obstacles. The goal is tracked by Vicon and is used for replanning at 3 Hz. Trajectories are generated using the soft time variant (8) with weight $w = 80$. In these experiments, each optimization takes less than 15 ms.

VI. CONCLUSION

We presented a novel bilevel optimization approach to UAV trajectory optimization, which analytically calculates the gradient of the objective function w.r.t. temporal variables. The optimization method takes into account the nonsmoothness of the upper-level optimization problem. Our results showed that this approach achieved real-time performance and higher quality trajectories than state-of-the-art heuristics. Our method could handle both hard time variant with fixed total time and soft time variant, where the weight was used to adjust trajectory aggressiveness. It could be useful in multiple contexts such as formation flight and tracking dynamic targets.

Future work may include accelerating the gradient descent by exploiting the structure of the problem. For example, acceleration may be achieved through using Newton or quasi-Newton methods. We are also interested in studying extensions of the bilevel optimization approach to other robotic applications like autonomous vehicles and legged locomotion.

APPENDIX A
PROOF OF SSOSC

We prove the SSOSC holds in TOBC, where jerk is minimized amongst Bézier curves of order 6. For brevity we drop dependency on y.

SSOSC [9] states that the Hessian of the Lagrangian evaluated at the optimal point is positive definite on the null space of the gradients of all the active constraints, i.e.,

$$w^T \nabla_{cc}^2 L(c^\star, \lambda^\star, \nu^\star) w > 0, \ \forall w \neq 0 \text{ s.t. } G_\mathcal{A} w = 0; Hw = 0$$

where $G_\mathcal{A}$ collects the rows of active inequality constraints and $L(\cdot)$ is the Lagrangian

$$L(c, \lambda, \nu) = c^T P c + \lambda^T G c + \nu^T H c$$

where λ and ν are the associated Lagrange multipliers.

Our proof shows that there does not exist any $w \neq 0$ that lies in the null space of P and H simultaneously, which proves SSOSC because $\nabla_{cc}^2 L(c^\star, \lambda^\star, \nu^\star) = P$, and since P is a symmetric positive semidefinite matrix, $w^T P w = 0$ implies $Pw = 0$. (Note that we ignore $G_\mathcal{A}$, so the proof holds regardless of which inequality constraints are active.)

Since each of the x, y, z dimensions can be decoupled, we show the proof in one dimension without loss of generality. Matrix P is block diagonal, with blocks denoted $P_i \in \mathbb{R}^{7 \times 7}$, whose entries depend on the duration of the corresponding segment. It can be be shown that

$$\mathcal{N}_{P_i} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 6 & 36 \\ 1 & 7 & 49 \end{bmatrix}$$

is a basis for the null space of P_i, regardless of segment duration. The physical meaning for each column of $\mathcal{N}_{P_i}$ is not moving at all, moving with constant velocity, and moving with constant acceleration, respectively, since these motions induce zero jerk. Then, for a problem with n segments, the matrix

$$\mathcal{N}_P = \begin{bmatrix} [c : c : c]\mathcal{N}_{P_i} & & \\ & \ddots & \\ & & \mathcal{N}_{P_i} \end{bmatrix}$$

is a basis of the null space of P, and has rank $3n$.

The linear equality constraints include initial and final position, velocity, and acceleration which accounts for six constraints. Continuity of position, velocity, and acceleration at conjunction points provide another $3(n-1)$ constraints for a problem with n segments. The linear equality constraint matrix H is defined as, (A.1) shown at bottom of this page, with 3×3 blocks

$$L_i = \begin{bmatrix} 1 & & \\ -d/\Delta t_i & d/\Delta t_i & \\ d(d-1)/\Delta t_i^2 & -2d(d-1)/\Delta t_i^2 & d(d-1)/\Delta t_i^2 \end{bmatrix}$$

$$R_i = \begin{bmatrix} 0 & & 1 \\ 0 & -d/\Delta t_i & d/\Delta t_i \\ d(d-1)/\Delta t_i^2 & -2d(d-1)/\Delta t_i^2 & d(d-1)/\Delta t_i^2 \end{bmatrix}$$

and each 0 block is of size $3 \times (d-5)$. Here, Δt_i is the time allocated to the ith segment and $d = 6$ is the spline degree.

Now we show that $H\mathcal{N}_P$ has full column rank. First, split $\mathcal{N}_{P_i}$ into blocks $[Q_1^T, Q_2^T, Q_3^T]^T$, where $Q_1, Q_3 \in \mathbb{R}^{3 \times 3}$ and $Q_2 \in \mathbb{R}^{1 \times 3}$. Note that Q_1 and Q_3 are square matrices of full rank. With this notation we express the product

$$H\mathcal{N}_P = \begin{bmatrix} [cccc]L_1 Q_1 & & & \\ R_1 Q_3 & -L_2 Q_1 & & \\ & R_2 Q_3 & \ddots & \\ & & \ddots & -L_n Q_1 \\ & & & R_n Q_3 \end{bmatrix}.$$

Observe that each block is nonsingular because it is a product of two square matrices of full rank. The upper $3n$ by $3n$ part is full column rank since each block is nonsingular. Therefore, $H\mathcal{N}_P$ has full column rank.

Consequently, any $w \neq 0$ in the nullspace of P can be expressed as $w = \mathcal{N}_P v$ with $v \neq 0$, which shows that $Hw \neq 0$. Hence there does not exist $w \neq 0$ that lies in the null space of P and H simultaneously, which proves SSOSC as desired.

APPENDIX B
DIRECT COLLOCATION IMPLEMENTATION

This section provides details of the DC method implemented for our numerical experiments. Our implementation follows the standard *Hermite–Simpson* collocation method, see, e.g., Betts [11, Section 5.1]. DC works by discretizing a continuous-time trajectory optimization problem into a NLP with discretized states and controls as decision variables. However, DC is rarely

$$H = \begin{bmatrix} [ccc : ccc : c : ccc]L_1 & 0 & & & & & \\ & 0 & R_1 & -L_2 & 0 & & \\ & & & 0 & R_2 & & \\ & & & & & \ddots & \\ & & & & & -L_n & 0 \\ & & & & & 0 & R_n \end{bmatrix} \tag{A.1}$$

used in UAV trajectory optimization since it does not take advantage of the differential flatness and is usually used directly with the system's actual dynamics equations. In order to use DC to minimize trajectory jerk, we define the system state $x(t) = [p(t), \dot{p}(t), \ddot{p}(t)]^T \in \mathbb{R}^9$ as a stacked vector of position $p(t) \in \mathbb{R}^3$, velocity $\dot{p}(t) \in \mathbb{R}^3$, and acceleration $\ddot{p}(t) \in \mathbb{R}^3$. The control input $u(t)$ is chosen to be the jerk $u(t) = \dddot{p}(t) \in \mathbb{R}^3$.

The system dynamics can be written as

$$\dot{x}(t) = f(x(t), u(t)) = Ax(t) + Bu(t) \quad \text{(B.1)}$$

where

$$A = \begin{bmatrix} 0 & I_{3\times3} & 0 \\ 0 & 0 & I_{3\times3} \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ I_{3\times3} \end{bmatrix}.$$

Consider a safe corridor with N segments, we set up a N-phase trajectory optimization problem. For the ith phase, the trajectory is constrained to stay in the ith convex region. Continuity constraints up to acceleration are applied between consecutive segments. This formulation matches with TOBC. The decision variables include the following.
1) Duration of each phase: $\Delta t_i, i = 1, \ldots, N$.
2) Discretized state trajectory of the UAV for each phase: $x_i(t), t \in [0, \Delta t_i], i = 1, \ldots, N$
3) Discretized control trajectory for each phase: $u_i(t), t \in [0, \Delta t_i], i = 1, \ldots, N$,

and the cost function is

$$J = \sum_{i=1}^{N} \int_0^{\Delta t_i} \|u_i(\tau)\|^2 d\tau + w\Delta t_i. \quad \text{(B.2)}$$

The rest follows the standard DC formulation [11].

REFERENCES

[1] M. Wang, Z. Wang, S. Paudel, and M. Schwager, "Safe distributed lane change maneuvers for multiple autonomous vehicles using buffered input cells," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4678–4684.
[2] H. Fan *et al.*, "Baidu apollo em motion planner," 2018, *arXiv:1807.08048*.
[3] P. Fernbach, S. Tonneau, and M. Taïx, "Croc: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.
[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Intl. Conf. Robot. Automat.*, 2011, pp. 2520–2525.
[5] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 344–351.
[6] S. Liu *et al.*, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3D complex environments," *IEEE Robot. Automat. Lett.*, vol. 2, no. 3, pp. 1688–1695, Jul. 2017.
[7] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Int. J. Robot. Res.*, 2016, pp. 649–666.
[8] Z. Wang, H. Ye, C. Xu, and F. Gao, "Generating large-scale trajectories efficiently using descriptions of polynomials," 2020, *arXiv:2011.02662*.
[9] A. V. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*. New York, NY, USA: Elsevier, 1983.
[10] W. Sun, G. Tang, and K. Hauser, "Fast UAV trajectory optimization using bileveloptimization with analytical gradients," in *Proc. Amer. Control Conf.*, 2020, pp. 82–87.
[11] J. T. Betts, "Survey of numerical methods for trajectory optimization," *J. Guid., Control, Dyn.*, vol. 21, no. 2, pp. 193–207, 1998.
[12] O. V. Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*. Berlin, Germany: Springer, 1993, pp. 129–143.
[13] Z. Wang, X. Zhou, C. Xu, J. Chu, and F. Gao, "Alternating minimization based trajectory generation for quadrotor aggressive flight," *IEEE Robot. Automat. Lett.*, vol. 5, no. 3, pp. 4836–4843, Jul. 2020.
[14] J. Tordesillas, B. T. Lopez, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 1934–1940.
[15] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Trans. Robot.*, vol. 36, no. 5, pp. 1526–1545, Oct. 2020.
[16] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Trans. Evol. Comput.*, vol. 22, no. 2, pp. 276–295, Apr. 2018.
[17] X. Xu and P. J. Antsaklis, "Optimal control of switched systems based on parameterization of the switching instants," *IEEE Trans. Autom. Control*, vol. 49, no. 1, pp. 2–16, Jan. 2004.
[18] M. Egerstedt, Y. Wardi, and F. Delmotte, "Optimal control of switching times in switched dynamical systems," in *Proc. 42nd IEEE Int. Conf. Dec. Control*, vol. 3, 2003, pp. 2138–2143.
[19] E. R. Johnson and T. D. Murphey, "Second-order switching time optimization for nonlinear time-varying dynamic systems," *IEEE Trans. Autom. Control*, vol. 56, no. 8, pp. 1953–1957, Aug. 2011.
[20] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 93–100.
[21] B. Landry, Z. Manchester, and M. Pavone, "A differentiable augmented Lagrangian method for bilevel nonlinear optimization," 2019, *arXiv:1902.03319*.
[22] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Ann. Operations Res.*, vol. 153, no. 1, pp. 235–256, 2007.
[23] H. Pirnay, R. López-Negrete, and L. T. Biegler, "Optimal sensitivity based on IPOPT," *Math. Program. Comput.*, vol. 4, no. 4, pp. 307–331, 2012.
[24] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 136–145.
[25] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo, "On differentiating parameterized argmin and argmax problems with application to bi-level optimization," 2016, *arXiv:1607.05447*.
[26] J. Tordesillas and J. P. How, "MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves," 2020, *arXiv:2010.10726*.
[27] K. Jittorntrum, "Sequential algorithms in nonlinear programming," Ph.D. dissertation, Australian Nat. Univ., Canberra, Australia, 1978.
[28] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. press, 2004.
[29] J. Nocedal and S. Wright, *Numerical Optimization*. New York; Berlin, Germany: Springer, 2006.
[30] D. Davis, D. Drusvyatskiy, S. Kakade, and J. D. Lee, "Stochastic subgradient method converges on tame functions," *Found. Comput. Math.*, vol. 20, pp. 119–154, 2018.
[31] F. H. Clarke, Y. S. Ledyaev, R. J. Stern, and P. R. Wolenski, *Nonsmooth Analysis and Control Theory*. Berlin, Germany: Springer, 2008, vol. 178.
[32] G. Still, *Lectures on Parametric Optimization: An Introduction*. Enschede, The Netherlands: Univ. Twente, 2018.
[33] A. Faust *et al.*, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *Proc. Int. Conf. Robot. Automat.*, 2018, pp. 5113–5120.
[34] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, pp. 99–131, 2005.
[35] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong, "User's guide for SQOPT 7.7: Software for large-scale linear and quadratic programming," Dept. Math., Univ. California, San Diego, La Jolla, CA, Center Comput. Math. Rep. CCoM 18—2, 2018.
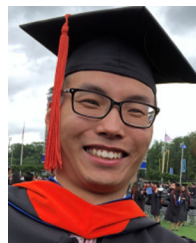
Weidong Sun received the bachelor's degree from the Wuhan University of Technology, Wuhan, Hubei, China, in 2017, and the master's degree from Duke University, Durham, NC, USA, in 2019, under the sponsorship of the China Scholarship Council.

He is currently a Robotics Engineer with XYZ Robotics Inc., Shanghai, China. His research interests include trajectory optimization and motion planning.
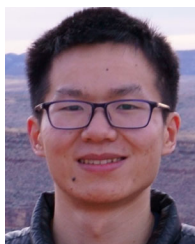
Gao Tang (Student Member, IEEE) received the bachelor's and master's degrees in aerospace engineering from Tsinghua University, Beijing, China, in 2014 and 2016, respectively, and the Ph.D. degree from Duke University, Durham, NC, USA, in 2019. He is currently working toward the Ph.D. degree with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA.

His research interests include motion planning and trajectory optimization.

Kris Hauser (Senior Member, IEEE) received the bachelor's degrees in computer science and mathematics from the University of California, Berkeley, Berkeley, CA, USA, in 2003, and the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 2008.

He was a Postdoc with the University of California, Berkeley. He is currently an Associate Professor with the Department of Computer Science and the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL, USA. He has also held faculty positions with Indiana University, Bloomington, IN, USA, from 2009–2014, and Duke University, Durham, NC, USA, from 2014–2019.

Dr. Hauser was the recipient of the Stanford Graduate Fellowship, Siebel Scholar Fellowship, Best Paper Award at IEEE Humanoids 2015, and an NSF CAREER award.