Fast UAV Trajectory Optimization using Bilevel Optimization with Analytical Gradients

Weidong Sun*, Student Member, IEEE, Gao Tang*, and Kris Hauser, Member, IEEE

Abstract—This paper presents an efficient optimization framework that solves trajectory optimization problems efficiently 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 so the time variables can be optimized using a separate outer optimization. This is a bilevel optimization in which the outer objective function itself requires an optimization to compute. The challenge is that gradient optimization methods require the gradient of the objective function with respect to the time variables, which is not available. Whereas the finite difference method must solve many optimization problems to compute a gradient, this paper proposes a more efficient method: the dual solution (Lagrange multipliers) of the convex optimization problem is exploited to calculate the analytical gradient. Since the dual solution is a by-product of the convex optimization problem, the gradient can be obtained "for free" with high accuracy. The framework is demonstrated on solving minimum-jerk trajectory optimization problems in safety corridors for unmanned aerial vehicles (UAVs). Experiments demonstrate that bilevel optimization improves performance over a standard NLP solver, and analytical gradients outperforms finite differences. With a 40 ms cutoff time, our approach achieves over 8 times better suboptimality than the current state-of-theart.

I. INTRODUCTION

REAL-TIME optimal trajectory generation has long been a challenging but essential component in robotics. However, due to nonlinear dynamics, non-convex constraints and high dimensionality, it is difficult to solve these problems in real-time. A popular strategy is to represent trajectories with polynomial splines and to decompose state constraints into convex sets. This allows quadratic programming (QP) methods to be applied, which can solve to global optima efficiently and accurately. This has been applied to path planning for ground robots, autonomous cars [1], [2], humanoid robots [3] and UAVs [4]-[6]. The disadvantage of this approach is that the time allocated for each segment of the spline has to be fixed in order for the optimization to remain convex, because time enters the optimization objective and constraints nonlinearly. In prior approaches, the allocated time is often chosen heuristically, because optimizing the time allocation scheme along with the path makes the problem non-convex and too slow for real-time use.

W.Sun and G.Tang are with the Department of Mechanical Engineering and Materials Science, Duke University. Durham, NC, 27708 USA. e-mail: {weidong.sun, gao.tang}@duke.edu.

K.Hauser is with the Departments of Electrical and Computer Engineering and of Mechanical Engineering and Materials Science, Duke University. Durham, NC, 27708 USA. e-mail: kris.hauser@duke.edu.

One effective strategy proposed in Mellinger $et\ al.$ [4] is a space-time decomposition approach that optimizes the path with the time allocation scheme fixed, and then iteratively refines the time allocation scheme through gradient descent. Specifically, in each iteration, two levels of optimization problems are solved: the lower level optimizes the path with the time fixed, and the upper level finds a better time allocation scheme with gradient descent. One unresolved challenge in this framework is finding a good gradient direction for the upper-level optimization, due to the fact that the objective is the solution of a lower-level constrained optimization problem. Finite differences was used in this prior work, but this can be computationally expensive because if n segments are used in the spline, then finite differences requires n+1 lower-level optimizations to be solved for each gradient evaluation.

The primary contribution in this paper is a bilevel optimization framework for optimal time allocation that estimates the gradient from the dual solution (Lagrange multipliers) of the QP problem. A graphical illustration of this technique is shown in Fig. 1. The dual solution to the lower-level optimization problem can be used to calculate a gradient to the upperlevel one under the assumption of a fixed active set. Since the dual solution is a by-product of the QP problem, we are able to estimate the gradient "for free". Compared with finite differences which suffer from the choice of step size and low accuracy, our approach obtains the exact gradient. Experiments are conducted on UAV trajectory optimization problems in the presence of obstacles, and show that our gradient estimation approach is orders of magnitude faster than the current stateof-the-art [4] while also being more accurate. This leads to an 8 times improvement in suboptimality when the optimizer is given a 40 ms cutoff.

II. RELATED WORK

Despite intensive research, it is still challenging to find an optimal time allocation scheme of a spline trajectory in real-time. One strategy is to generate a time allocation scheme with heuristics [5], [6] and stick with the scheme during the optimization stage. For example, Gao *et al.* [5] use the heuristic of selecting velocity according to the distance to the nearest obstacle, closer the trajectory is to the obstacle, lower the velocity. Though the heuristic is reasonably chosen, the velocity of the trajectory after the optimization stage does not necessarily follow the heuristic. Heuristics, though being cheap to compute, are often not optimal and will sometimes lead to spikes in jerk or snap near connection points, see Fig. 5c for an example.

^{*} Denotes equal contribution

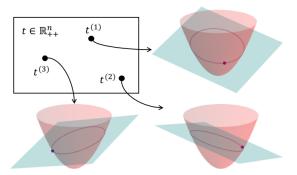


Fig. 1: An illustration of bilevel optimization: $t^{(1)}$, $t^{(2)}$ and $t^{(3)}$ are three feasible time allocation schemes, they are optimized in the *upper-level optimization problem*. Each of these t corresponds to a quadratic programming problem, which is being solved in the *lower-level optimization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

One way to improve a non-optimal time allocation scheme is to refine it iteratively using gradient descent [4], [7]. The refinement process computes the gradient of the objective w.r.t. the time allocation scheme, takes a step after choosing a suitable descent direction and step length. To the best of our knowledge, finite difference is the only way being used for gradient calculation in this scenario. However, estimating gradient with finite difference is time-consuming since the objective function is actually the objective of an optimization problem. For each gradient evaluation, this approach has to solve the same number of optimization problems as the number of segments in the spline, thus making it intractable for real-time performance. Moreover, choosing step sizes for finite difference is hard and the result may not necessarily be a good approximation to the actual gradient.

Another strategy to determine a time allocation scheme is to use sampling [3]. This approach randomly samples the duration of each segment until the corresponding QP can be solved, and is applicable to problems with a rather small feasible set, for example the problem of humanoid locomotion where a small change in time can make the optimization problem infeasible. However it may not scale well to generating trajectories with a large number of segments. Also, the work described in [3] only seeks for a feasible time allocation and does not improve it.

Richter *et al.* [7] proposes a framework that is able to generate collision-free spline trajectories and optimize time allocation. They managed to formulate a QP with only equality constraints. However, such a formulation has the disadvantage: (a) Being collision-free is achieved by iteratively adding new points and solve another QP if there is a collision. This strategy has no bound on the number of iterations needed though their experiments show a few iterations are usually enough. (b) Dynamic feasibility is achieved by checking the max acceleration in every iteration and stop the iteration if the

acceleration reaches the limit. Since computing the extrema of a high order polynomial is hard according to the Abel-Ruffini theorem, this strategy can be slow when we are using high order polynomials. Moreover, their framework solves the optimization problem by re-formulating the equality constrained QP into an unconstrained one, they will not be able to handle general inequality constraints which often exist in trajectory optimization. In our work, we use bilevel optimization as a way to solve a GBD, and exploit theoretical developments from these fields to derive analytical gradients for trajectory optimization.

More broadly, the idea of decomposition and divide-andconquer has been explored in the Benders Decomposition (BD) [8] and Generalized Benders Decomposition (GBD) [9]. which are well-known techniques in operations research. In the GBD formulation, complicating variables are variables which would make the optimization problem considerably more tractable when temporarily fixed. The idea of GBD is to split the original optimization problem into two: a master problem and a subproblem. The master problem is generated by temporarily fixing the *complicating variables* of the original problem, and in BD these are usually integer variables. In the case of trajectory optimization, the complicating variables are times allocated to each segment. This approach is also closely related to bilevel optimization [10], which refers to a mathematical program where one optimization problem has another optimization problem as one of its constraints, that is, one optimization task is embedded within another. The outer optimization task is often referred to as the upper-level optimization problem and the inner optimization task is known to be the lower-level optimization problem. Bilevel optimization is well-known in production and marketing decision making, and we refer readers to [10], [11] for more comprehensive treatments of the topic.

There are analogues to our approach in the field of machine learning. OptNet [12] incorporates a QP solver as a layer into the neural network and is able to provide analytical gradients of the solution to the QP w.r.t. input parameters for back propagation. Gould et al. [13] presents results on differentiating argmin optimization problems w.r.t. optimization variables in the context of bilevel optimization. Their results give exact gradients for the lower-level optimization problem. However, works described in [12], [13] do not consider finding the gradient of the objective function. Moreover, methods proposed by Gould et al. [13] involves inverting a Hessian matrix which could be computationally expensive and often not even invertible, and they only describe the case of equality constraints and use a log-barrier function to tackle inequality constraints. The gradient of the objective function is even cheaper to compute, making our approach very suitable for speeding up trajectory optimization.

III. METHOD

In this section, we describe how our framework can be applied to trajectory optimization. We start by introducing how trajectory optimization works, then give a mathematical formulation of the problem we are trying to solve and end with a description of our algorithm.

A. Spline-Based Trajectory Optimization

Trajectory optimization asks to find a trajectory that minimizes some measure of performance while satisfying all the necessary constraints, e.g. being collision-free and dynamically feasible, and is formulated as

$$\begin{aligned} \min_{x,T} \quad J(x,T) &= \int_0^T \ell(x(y),t) dt + \Phi(x(y)) \\ \text{s.t.} \qquad \qquad x(0) &= x_0 \\ g(x(t)) &\leq 0 \quad \forall t \in [0,T], \\ h(x(t)) &= 0 \quad \forall t \in [0,T], \\ k(y) &\leq 0 \\ s(y) &= 0 \end{aligned}$$

where x encodes a trajectory, with the initial state x_0 prescribed. Often the final state $x(y) = x_f$ is also given, and sometimes the final time T is fixed. J(x,T) is the measure of performance. In order to generate safe and dynamically feasible trajectories, the following constraints must be satisfied:

- Safety: Position constraints that make sure the position lies in a safe region.
- 2) Dynamic Feasibility: Bounds on minimum and maximum velocity and acceleration.

In abuse of notation we write state constraints as $g(x) \leq 0$ and h(x) = 0. More precisely, we have $\ell \equiv \ell(x, x', x'', \dots, t)$, $g \equiv g(x, x', x'', \dots)$, and $h \equiv h(x, x', x'', \dots)$ depending on higher order derivatives.

If we represent the trajectory as a polynomial spline, we can reframe the problem in terms of spatial (spline coefficients) and temporal variables (spline knot points). Specifically, define a spline with n segments and segment durations $\Delta t_1, \ldots, \Delta t_n$. The timing of each knot point is given by $t_i = t_{i-1} + \Delta t_i$ with $t_0 = 0$. The i'th segment is defined over the domain $[t_{i-1}, t_i]$ as

$$x_i(t) = \sum_{j=0}^{d} c_{ij} \left(\frac{t - t_{i-1}}{\Delta t_i}\right)^j \tag{1}$$

for each $i=1,\ldots,n$. We formulate our splines using 6th order polynomials (d=6).

In order for the trajectory to be smooth, equality constraints should be enforced so:

- 1) States at the start and end of the trajectory should match the initial state and (optional) final state.
- 2) Continuities at connection points that ensure a smooth transition between each segment of the trajectory. If the trajectory needs to be C^d continuous, equality constraints up to the d'th order should be applied at all knot points.

We found that in the UAV case, applying continuity constraints up to acceleration yields good results.

Often, the objective function is chosen to be the integral of the squared norm of some high-order derivative of the trajectory, such as minimum jerk. As shown in [4], [5], these types of objectives can be written as a quadratic function of the coefficients of the polynomials, as long as timing is fixed. Each constraint $g(x(y)) \leq 0$ on an individual state at a fixed time $t \in [t_{i-1}, t_i]$ in segment i can also be converted to a

constraint on the coefficients c_{i0}, \ldots, c_{id} . Overall, safety and dynamic feasibility constraints can be enforced at a set of collocation points, leading to a nonlinear program. However, there is no guarantee of the whole trajectory satisfying all the constraints.

Another approach is to encode the trajectory as a Bézier spline. Bézier curves have the nice properties that:

- 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. Thus if the trajectory is represented with Bézier curves and the bounds described above are convex or convex in each segment, then enforcing constraints on all the control points will guarantee that the whole trajectory will satisfy those bounds. (Note that this is only a sufficient condition and tends to be conservative, as the whole trajectory may satisfy all the constraints with some control points lying out of bounds.)

B. Formulation of the Bilevel Optimization Problem

In order to efficiently solve the trajectory optimization problem, we explicitly split the spatial part and the temporal part of the trajectory. We gather all the polynomial coefficients in the flattened vector $x \in \mathbb{R}^{n(d+1)}$ (spatial part), and define the time allocation scheme $y = (\Delta t_1, \dots, \Delta t_n)$ (temporal part).

We require the objective function to be quadratically separable in the form $f(x,y) = \frac{1}{2}x^T P(y)x + q(y)^T x + c(y)$, in which P(y) is a quadratic objective matrix which is positive semidefinite. P(y), q(y) and c(y) are allowed to be nonlinear in y. Separability holds when the objective is expressed as the integral of the squared norm of some high-order derivative of the trajectory, such as minimum jerk. As shown in [4], [5], these types of objectives can be written as quadratic functions of x as long as timing is fixed. c(y) is the part of the objective that is only dependent on y, such as penalizing total time with $c(y) = c \sum_i \Delta t_i = \mathbb{1}^T y$.

If the objective is quadratically separable and the spatial constraints are convex, then a bilevel trajectory optimization problem can be formulated as follows:

$$\begin{split} \min_{x,y} & \quad \frac{1}{2}x^TP(y)x + q(y)^Tx + c(y) \\ \text{s.t.} & \quad Ay \leq b, \\ & \quad Cy = d, \\ & \quad x = \underset{x}{\operatorname{argmin}} & \quad \frac{1}{2}x^TP(y)x + q(y)^Tx + c(y) \\ & \quad \text{s.t.} & \quad G(y)x \leq h(y), \\ & \quad L(y)x = m(y). \end{split}$$

The timing constraints $Ay \leq b$ and Cy = d enforce constraints such as positive durations $y \geq 0$, and possibly fixed total time $\mathbb{1}^T y = T$. We refer interested readers to [2], [3], [5], [6] for different realizations of this general formulation.

Note that although the objective functions remain the same in both the *lower-level* and *upper-level optimization problem*, y is fixed in the *lower-level optimization problem* but becomes the variable we want to optimize in the *upper-level optimization problem*. The lower-level problem is also specified to be

a quadratic program (QP), which can be solved efficiently and globally optimally in x.

Our strategy is to use a constrained gradient descent on the function $f^*(y) = f(x^*(y), y)$ with x^* minimizing the QP for a fixed timing y. The descent method indeed has been used to solve bilevel optimization problems, and our framework is a variant of this method. Given an feasible y, we find a direction $-\nabla f^*(y) \in \mathbb{R}^n$ and a step length α that can make a sufficient decrease in f(y) while maintaining the feasibility of the new point $y_{new} = y + \alpha \nabla f^*(y)$. The general issue is the availability of gradients.

C. Analytical gradient estimation

We now start to examine how the dual solution or Lagrange multipliers from the solution of the *lower-level optimization* problem can be utilized to estimate the gradient of f^* . We assume that the *lower-level* and *upper-level objective functions* are the same. Let us consider a general nonlinear optimization problem formulated as:

$$\min_{x,y} \quad f(x,y)$$
 s.t. $g(x,y) \le 0$,
$$h(x,y) = 0$$
. (2)

The Lagrangian is defined as:

$$\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda^{T} g(x, y) + \mu^{T} h(x, y)$$

Where λ and μ are Lagrange multipliers associated with inequality and equality constraints, respectively.

Suppose that x^* minimizes f(x,y) for a fixed value of y. Let us consider the first-order optimality conditions [14], also known as the Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions must be satisfied at the optimal point. Namely, the following set of equations have to be satisfied by the optimal solution tuple $(x^*, y, \tilde{\lambda}, \tilde{\mu})$:

$$\nabla_{x}\mathcal{L}(x^{\star}, y, \tilde{\lambda}) = 0,$$

$$g(x^{\star}, y) \leq 0,$$

$$h(x^{\star}, y) = 0,$$

$$\tilde{\lambda} \geq 0,$$

$$\tilde{\lambda} \odot g(x^{\star}, y) = 0.$$
(3)

Where ∇_x denotes the gradient with respect to x, \odot denotes an element-wise multiplication, inequalities and equalities should be interpreted element-wise.

The complementary slackness condition [15] tells us that if the i'th element of $\tilde{\lambda}$ is nonzero, then the i'th inequality constraint is active. That is, $g_i(x,y)=0$, where $g_i(x,y)$ denotes the i'th row in the inequality $g(x,y)\leq 0$. If $g_j(x,y)<0$, we say that the j'th constraint is inactive.

The active set is the set of all the indices corresponding to active constraints, which we denote as \mathcal{A} . The inactive set, which contains all the indices corresponding to inactive constraints, is denoted as $\bar{\mathcal{A}}$. It is clear that \mathcal{A} and $\bar{\mathcal{A}}$ are complements of each other. Note that we do not require the active set to be unique.

Now let us split the inequality constraints into two parts: one that contains all the active constraints and the one that contains all the inactive constraints. For all the active inequality constraints, we can think of them as equality constraints since $g_i(x,y)=0, \forall i\in\mathcal{A}$. We now construct a new equality constraint by stacking the active inequality constraints and equality constraints:

$$r(x,y) = 0$$

where $r(x,y) = \begin{bmatrix} g_{\mathcal{A}}(x,y) \\ h(x,y) \end{bmatrix}$, $g_{\mathcal{A}}(y) = \{g_i(y) | \forall i \in \mathcal{A}\}$ denotes the extraction of rows whose indices are in the active set \mathcal{A} . The vector of Lagrangian multipliers corresponding to this new set of equality constraints denoted as z is:

$$z = \begin{bmatrix} \lambda_{\mathcal{A}} \\ \mu \end{bmatrix} \tag{4}$$

At the optimal point, the tuple (x^*, y, \tilde{z}) must satisfy the following due to KKT conditions:

$$\nabla_x f(x^*, y) + \tilde{z}^T \nabla_x r(x^*, y) = 0$$
 (5a)

$$r(x^{\star}, y) = 0 \tag{5b}$$

We would like to examine how the optimal objective changes if a small perturbation is applied to the time allocation scheme while keeping the active set constant. Let us consider an infinitesimal shift $\mathrm{d}y$ applied to the time allocation scheme t. We denote the shifted time allocation scheme as $y' = y + \mathrm{d}y$, the shift in optimal solution due to the perturbation as $\mathrm{d}x = x' - x^*$, the shift in optimal objective due to the perturbation as $\mathrm{d}f = f(x', y') - f(x^*, y)$

If $r(x^*, y)$ is differentiable and we apply first-order Taylor expansion on equation Eq. (5b), we get:

$$r(x^*, y) + \nabla_y r(x^*, y) \, \mathrm{d}y + \nabla_x r(x^*, y) \, \mathrm{d}x = 0 \qquad (6)$$

Since $r(x^*, y) = 0$ due to Eq. (5b), Eq. (6) can be rewritten as:

$$\nabla_y r(x^*, y) \, \mathrm{d}y = -\nabla_x r(x^*, y) \, \mathrm{d}x \tag{7}$$

Let us assume $f(x^*, y)$ is differentiable, approximating df with a first-order Taylor expansion around $f(x^*, y)$ results in:

$$df = f(x', y') - f(x^*, y)$$

$$= f((x^* + dx), (t + dy)) - f(x^*, y)$$

$$= \frac{\partial f}{\partial x}(x^*, y) dx + \frac{\partial f}{\partial y}(x^*, y) dy$$

$$= \nabla_x f(x^*, y) dx + \nabla_y f(x^*, y) dy$$
(8)

Note that we can rewrite Eq. (5a) as:

$$\nabla_x f(x^*, y) = -\tilde{z}^T \nabla_x r(x, y) \tag{9}$$

Now we plug Eq. (9) into Eq. (8):

$$df = -\tilde{z}^T \nabla_x r(x^*, y) dx + \nabla_y f(x^*, y) dy$$
 (10)

Finally, we plug Eq. (7) into Eq. (10):

$$df = \tilde{z}^T \nabla_y r(x^*, y) dy + \nabla_y f(x^*, y) dy$$

= $(\tilde{z}^T \nabla_y r(x^*, y) + \nabla_y f(x^*, y)) dy$ (11)

Since $\mathrm{d}f$ is the shift of the optimal objective, we can conclude that

$$\nabla f^{\star}(y) = \tilde{z}^{T} \nabla_{y} r(x^{\star}, y) + \nabla_{y} f(x^{\star}, y)$$
 (12)

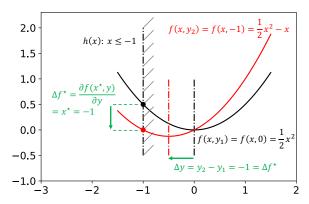


Fig. 2: Black and red parabolas represent $f(x,y_1)$ and $f(x,y_2)$, respectively. Constraint is the vertical line with shade. Black and red line segments are the axes of symmetry of the corresponding parabola. Optimal solutions are black and red dots. The vertial green arrow denotes the decrease in the optimal objective $\Delta f^*(y)$, the horizontal green arrow denotes the shift in y.

as desired. Even more succinctly, since $\tilde{\lambda}_i = 0$ at non-active constraints, we can write this as

$$\nabla f^{\star}(y) = \tilde{\lambda}^{T} \nabla_{y} g(x^{\star}, y) + \tilde{\mu}^{T} \nabla_{y} h(x^{\star}, y) + \nabla_{y} f(x^{\star}, y) \tag{13}$$

D. Two illustrative examples

We illustrate our analytical gradient estimation technique with two toy examples. The first example inspects the case with the objective function being:

$$f(x,y) = \frac{1}{2}x^2 + yx$$

The constraint $h(x, y) \leq 0$ being:

$$x < -1$$

In this example, we assume that the constraint does not depend on y. We pick $y_1 = 0$ and $y_2 = -1$ and inspect the shift of the optimal objective $\Delta f^* = f(x^*, y_2) - f(x^*, y_1)$. Fig. 2 gives a graphical illustration. Since the constraint does not depend on y, we have:

$$\nabla_{u} r(x^{\star}, y) = 0$$

Then according to Eq. (12):

$$\Delta f^{\star} = \nabla_{u} f(x^{\star}, y) = x^{\star} = -1$$

Our second example has an objective that does not depend on y:

$$f(x) = \frac{1}{2}x^2$$

With a constraint $h(x, y) \leq 0$ parameterized by y:

$$x + y < 0$$

Let us consider $y_1 = -1.0$, $y_2 = -0.5$ and inspect the shift of the optimal objective $\Delta f^* = f(x^*, y_2) - f(x^*, y_1)$. Fig. 3 gives a graphical illustration of this example. Since the objective f(x) does not depend on y, we have:

$$\nabla_y f(x^\star, y) = 0$$

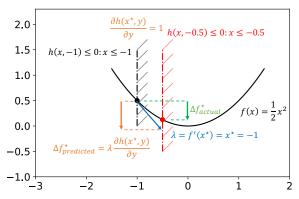


Fig. 3: The black parabola represents the objective function f(x). Black and red line segments with shades are the constraints corresponding to y_1 and y_2 , respectively. Optimal solutions are black and red dots. The blue arrow lies tangent to the parabola represents the Lagrange multiplier λ to the black constraint. The predicted decrease of the optimal objective $\Delta f^{\star}_{predicted}$ is denoted by the orange arrow. The actual decrease of the optimal objective $\Delta f^{\star}_{actual}$ is denoted by the green arrow. Note that the Lagrange multiplier only gives first-order information so the actual decrease is less than predicted.

Then according to Eq. (12):

$$\Delta f_{predicted}^{\star} = \lambda \nabla_y r(x^{\star}, y) = -1$$

E. Outer optimization

Our gradient descent algorithm is given in Algorithm 1, which takes an initial guess y_0 of the time allocation scheme and a maximum number of iterations as input. It then iteratively descends f^* until some optimality conditions are satisfied or the maximum number of iterations is reached.

Algorithm 1 Refine-Time (y_0)

```
2: J, \lambda \leftarrow \text{Solve-QP}(P(y), G(y), h(y), L(y), m(y))
3: for i \leftarrow 0 to max-iterations do
          q \leftarrow \text{Get-Gradient}(\lambda)
                                                                    ⊳ From Eq. (13)
 4:
 5:
          p \leftarrow \text{Project-Gradient}(g, A, b, C, d)
          \alpha, J_{\alpha}, \lambda_{\alpha}, y_{\alpha} \leftarrow \text{Line-Search}(t, p)
 6:
 7:
          if \alpha not found then
                 break
 8:
 9:
          if optimality-conditions-satisfied then
10:
          J, \lambda, y \leftarrow J_{\alpha}, \lambda_{\alpha}, y_{\alpha}
11:
12: return t
```

In Algorithm 1, Line 2 solves a QP problem with a time allocation scheme $y=y_0$ and then returns the objective value J and the dual solution (Lagrange multipliers) λ . J is now the baseline objective and the rest of the algorithm will improve upon it. Line 4 estimates the gradient of the objective w.r.t. time using the Lagrange multipliers λ . Line 5 finds a normalized descent direction from the gradient by projecting the gradient onto the null space of constraints on t [14]. Line 6,

which is further illustrated in Algorithm 2, finds a suitable step length α that gives sufficient decrease in the objective function. If such an α is found, the line search algorithm also returns the objective, Lagrange multipliers and time allocation associated with the optimal α denoted as J_{α} , λ_{α} and y_{α} , respectively. Then the objective, Lagrange multipliers and time allocation scheme are updated in Line 11. If the step length α cannot be found during the iteration, the iteration stops and returns the last t as in Line 8. The optimality conditions used in our implementation are:

- Norm of the projected gradient is less than a threshold, or
- The change of the objective function is less than a threshold.

```
Algorithm 2 Line-Search (y_0, p)
  1: static variable \alpha_0
 2: constant variables \tau_a, \tau_s
 3: \alpha \leftarrow \alpha_0
 4: for i \leftarrow 0 to max-iterations do
  5:
           y = y_0 + \alpha p
           J, \lambda \leftarrow \text{Solve-QP}(P(y), G(y), h(y), L(y), m(y))
 6:
           if sufficient-decrease-achieved then
 7:
                if i = 0 then
 8:
 9:
                      \alpha_0 \leftarrow \tau_q \alpha
10:
                else
                      \alpha_0 \leftarrow \alpha
11:
                return \alpha, J, \lambda, y
12:
13:
           \alpha \leftarrow \tau_s \alpha
14: return \alpha not found
```

Algorithm 2 shows our adaptive backtracking line search routine starting from y_0 in the direction of p, step length α is returned if found as in Line 12 along with the associated objective J_{α} , Lagrange multipliers λ_{α} and time allocation scheme y_{α} , otherwise " α not found" will be returned as in Line 14. In our implementation, the Armijo condition [14] is used for checking sufficient decrease in the objective. The initial step length α_0 for each line search is defined as a static variable in Line 1, it will be updated adaptively. The update strategy is similar to the update of trust region radius in a trust region algorithm: If the line search achieved sufficient decrease after the first iteration, the initial step length α_0 for the next line search will grow as in Line 9; If the line search found an α that leads to sufficient decrease after more than one iterations, then that α will set be as the α_0 for the next line search as in Line 11. Parameters that control the growing and shrinking of α are τ_q and τ_s , respectively, they are defined as constant variables in Line 2. Note that $\tau_q > 1$ and $0 < \tau_s < 1$. Line 5 takes a new step and line 6 solves the QP with the updated time.

IV. EXPERIMENTS

We demonstrate our method on trajectory planning for UAVs in the presence of obstacles. Our implementation of the algorithm is written in Python and C++: Python is used in constructing the QP problem, performing line search and

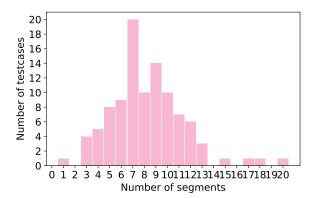


Fig. 4: Test case descriptive statistics.

TABLE I: Experimental results (mean/median)

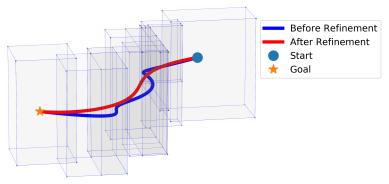
Method	Time [ms]	Suboptimality	Constraint Violation
Sqopt+LM	171.43 / 99.25	0.025 / 1.8e-7	0.0 / 0.0
Sqopt+FD	323.03 / 226.66	1.179 / 0.144	0.0 / 0.0
Mosek+LM	330.46 / 288.35	0.016 / 0.0	0.0 / 0.0
Mosek+FD	1067.16 / 904.25	7.102 / 0.754	0.0 / 0.0
SNOPT	56.44 / 43.34	242.75 / 14.66	0.087 / 0.033

gradient calculation, while all QP solvers run in C++. Pybind11 [16] is used as the interface between Python and C++. All the experiments are carried out on a laptop with a 2.9 GHz Intel Core i5 processor. We make use of the environment generator and planning pipeline from Gao *et al.* to generate convex collision-free corridors for a UAV [5]. This open source software is able to generate random obstacles in a 3D environment, visualize the obstacles and trajectories in Rviz, and generate safe corridors. We generated 100 tests by randomly sample feasible start points and goal points from the environment generator and record the flight corridor. Statistics of the number of segments from these tests are shown in Fig. 4. Because these involve 6th order splines in a 3D state space, the number of variables in each optimization problem is $(7 \times 3 + 1)n$ where n is the number of segments.

We establish a minimum-jerk objective function, and fix the total amount of time on the trajectory. The initial timing is allocated from the heuristic described in [5]. The fixedduration constraint is used for fair comparison with the unrefined trajectory, but we note that our framework is able to handle general objective functions and constraints on timing. Fig. 5 shows an instance of one of these corridors, and how the trajectory can be improved by refining the time allocation scheme.

We compare the results from different combinations of QP solvers and gradient estimation methods listed in Table I. Here we refer to the relative suboptimality, which is calculated as $\frac{J-J^{\star}}{J^{\star}}$, where J denotes the objective value achieved by an algorithm, and J^{\star} denotes the true optimum. In our experiment, we take J^{\star} as the minimum of all the objectives returned by different algorithms.

We test the following solvers: Sqopt [17], an active-set QP solver intended for large and sparse systems; Mosek [18], an interior-point QP solver, OSQP [19], an alternating direction



(a) Blue and red curves show trajectories for unrefined and refined timing, respectively.

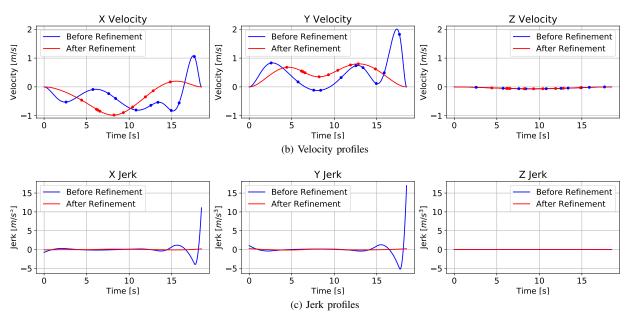


Fig. 5: Refined vs unrefined timing on an example problem. (a) The trajectory must be totally contained in the safe corridor made up by the boxes, and jerk is minimized. (c) The spikes of jerk near the goal are smoothed out after refinement.

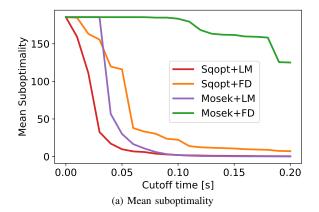
method of multipliers (ADMM) approach, and qpOASES [20], an active-set QP solver designed for small and dense systems. Experiments show that Sqopt and Mosek yield adequate results, but neither of OSQP or qpOASES works well in our experiment, because qpOASES does not scale well to midscale or large problems with sparse structures, and OSQP cannot solve QP to a high accuracy and gives inaccurate Lagrange multipliers.

We compare our Lagrange multiplier method (LM) against finite differences (FD) using Sqopt and Mosek. LM improves overall running time by 2-3 times beyond FD, and moreover terminates with a much lower suboptimality. We suspect this is because of the improved accuracy of the LM gradient estimation. The performance bottleneck of the LM method is moved to the line search step, which takes about 90% of the total computation time while the time spent on gradient estimation is trivial. Sqopt is generally faster than Mosek, and this could be a consequence of active-set methods' ability to perform warm start. We have tried Quasi-Newton methods but they performed poorly against steepest descent. We also observed that the computation time of our method scales roughly linearly with the number of segments.

We also compare against SNOPT [21], which is a general nonlinear solver for sparse, large-scale problems. Here we do not decompose the problem, and simply formulate joint spatial and temporal optimization problem as an NLP. We provide analytic gradients to SNOPT for solver robustness. We initialize SNOPT with the unrefined time allocation scheme and the spline coefficients resulting with the first QP solve. However, even starting from a feasible solution, SNOPT does poorly compared to our bilevel optimization framework. SNOPT tends to terminate prematurely without converging, and often moving to an infeasible point. We believe this is because the joint spatial and temporal NLP is ill-conditioned:

- The QP objective function exhibits high-order dependence on timing.
- 2) Some spatial constraints are very sensitive to the high-order spline coefficients.

The results shown in Fig. 6 explore suboptimality as a function of time. For each cutoff time, we abort the optimization process at that time and record the suboptimality achieved at this time. If no iteration has been finished at the cutoff time, we use the unrefined objective instead. Note that due



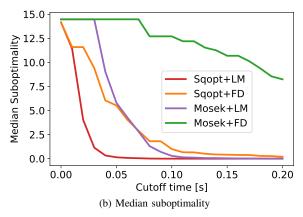


Fig. 6: Timing vs suboptimality for UAV path planning using bilevel optimization. Our Lagrange multiplier method (LM) is compared against finite differences (FD).

to the properties of the steepest descent method, the first few iterations will give significant decrease in the objective but the improvement decreases as more time is spent. In applications, we suggest that our optimizer can run in a separate thread and return the best result at a given timeout. If this algorithm were to run in 25Hz (40 ms cutoff time), which is a reasonable frequency for real-time applications, our method achieves a mean and median suboptimality of 14.73 and 0.87 respectively, compared to 117.33 and 6.91 achieved by [4].

V. CONCLUSION

We presented a novel way to estimate the gradient of the objective function w.r.t. time allocation scheme in a trajectory optimization problem and proposed an efficient bilevel optimization framework based on Generalized Benders Decomposition. Our results indicate that we can achieve a significant decrease in the objective of the trajectory while having real-time performance. This framework may have applications in path planning for ground vehicles, humanoid robots and UAVs. Underpowered robots will particularly benefit from this framework since a refined trajectory is smoother, less aggressive and easier to track than its unrefined counterpart.

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. Also, we are considering the possibility of deploying our framework on a real UAV to further investigate its performance in real-time trajectory optimization.

ACKNOWLEDGMENT

This work is partially supported by NSF Grant #IIS-1253553.

REFERENCES

- M. Wang, Z. Wang, S. Paudel, and M. Schwager, "Safe distributed lane change maneuvers for multiple autonomous vehicles using buffered input cells," in *ICRA*. IEEE, 2018, pp. 1–7.
- [2] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," CoRR, vol. abs/1807.08048, 2018.
- [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 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.
- [4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors." in *ICRA*. IEEE, 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 *ICRA*. IEEE, 2018, pp. 344–351.
- [6] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1688–1695, 2017.
- [7] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *ISRR*, ser. Springer Tracts in Advanced Robotics, vol. 114. Springer, 2013, pp. 649–666.
- [8] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numer. Math.*, vol. 4, no. 1, pp. 238–252, Dec. 1962. [Online]. Available: http://dx.doi.org/10.1007/BF01386316
- [9] A. M. Geoffrion, "Generalized benders decomposition," *Journal of Optimization Theory and Applications*, vol. 10, no. 4, pp. 237–260, Oct 1972. [Online]. Available: https://doi.org/10.1007/BF00934810
- [10] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 276–295, 2018.
- [11] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of Operations Research*, vol. 153, no. 1, pp. 235–256, Sep 2007. [Online]. Available: https://doi.org/10.1007/ s10479-007-0176-2
- [12] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *ICML*, 2017.
- [13] 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," *CoRR*, vol. abs/1607.05447, 2016.
- [14] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.
- [15] S. Boyd and L. Vandenberghe, Convex Optimization. New York, NY, USA: Cambridge University Press, 2004.
- [16] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 seamless operability between c++11 and python," 2017, https://github.com/pybind/pybind11.
- [17] 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," Department of Mathematics, University of California, San Diego, La Jolla, CA, Center for Computational Mathematics Report CCoM 18-2, 2018.
- [18] MOSEK ApS, MOSEK Optimizer API for C, 8.1., 2018. [Online]. Available: https://docs.mosek.com/8.1/capi/index.html
- [19] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," ArXiv e-prints, Nov. 2017.
- [20] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.
- [21] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong, "User's guide for SNOPT 7.7: Software for large-scale nonlinear programming," Department of Mathematics, University of California, San Diego, La Jolla, CA, Center for Computational Mathematics Report CCoM 18-1, 2018.