Fast UAV Trajectory Optimization using Bilevel Optimization with Analytical Gradients

Weidong Sun[1] , Gao Tang[2], *Student Member, IEEE* and Kris Hauser[2] *Senior Member, IEEE*

***Abstract*—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 is to obtain the gradient of the objective function which itself needs an optimization to compute. Whereas finite differences must solve many optimization problems to compute the gradient, our method is based on sensitivity analysis of parametric programming: the dual solution (Lagrange multipliers) of the lower-level optimization is used to compute analytical gradients. Since the dual solution is a by-product of the optimization, the exact gradients can be obtained "for free". The framework is demonstrated on generating trajectories in safe corridors for an unmanned aerial vehicle. 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. With a 25 ms cutoff time, our approach achieves over 8 times better suboptimality than the current state-of-the-art.**

I. Introduction

Real-time optimal trajectory generation has long been a challenging but essential component in robotics. Due to nonlinear dynamics, non-convex constraints and high dimensionality, it is difficult to optimize trajectories quickly and reliably. To get around this issue, one would usually 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 time and optimize state trajectories: trajectories are represented using piecewise polynomial splines and optimized by quadratic programming (QP) methods. This has been applied to path planning for ground robots, autonomous cars [1], [2], humanoid robots [3] and UAVs [4]–[6]. However, the time allocated to each piece has to be fixed to maintain convexity of the problem, because time enters the optimization objective and constraints in highly nonlinear fashion. In prior approaches, the allocated time is often chosen heuristically [5], and how to optimize time allocation remains an open question.

*This work was supported by NSF grant #IIS-1816540

[1]W. Sun is with XYZ Robotics, Shanghai, China weidong.sun@xyzrobotics.ai

[2]G. Tang and K. Hauser are with Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, USA {gaotang2,kkhauser}@illinois.edu

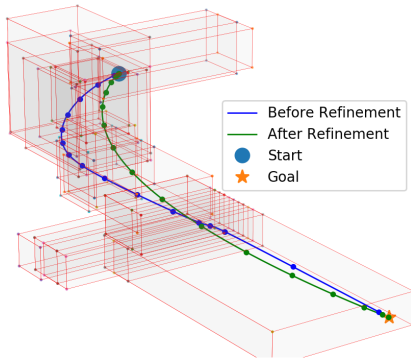


Fig. 1. *Before Refinement* trajectory is computed by the implementation from Gao et al. [5]. *After Refinement* trajectory uses our technique to optimize the path and time allocation among segments. A 19-segment safe corridor is used. Dots indicate equally distributed time steps sampled along the trajectories.

Mellinger et al. [4] applies gradient descent method to refine time allocation. Specifically, the optimization problem is divided into two levels: the lower level optimizes the path while timing is fixed, and the upper level optimizes the time allocation. One unresolved challenge is how to find the gradient of the optimal cost with respect to time allocation. Finite difference is used in [4], but this can be computationally expensive because if n segments are used in the spline, then finite differences requires $n+1$ QPs to be solved for each gradient evaluation. Besides, inappropriate choice of step size may lead to large gradient estimation error.

We propose a bilevel optimization framework that computes optimal time allocations, where the gradient of the objective w.r.t. time allocation is computed from the dual solution (Lagrange multipliers) of the QP problem. We are able to compute the exact gradient "for free", whereas the finite difference method suffers from slow computation and low accuracy. Numerical experiments are conducted on UAV trajectory optimization problems in random environments which shows an 8 times improvement in suboptimality when the optimizer is given a 25 ms cutoff. One example in Fig. 1 shows that our technique is able to refine the trajectory to get a smoother trajectory with lower jerk.

II. Related Work

A. Improving Time Allocation

As described above, trajectory optimization with polynomial splines in piecewise-convex corridors is a relatively solved

problem as long as the spline timing is fixed. However, it has still proven challenging to find an optimal time allocation in real-time. One strategy [5], [6] is to generate a time allocation with heuristics and keep timing fixed during the optimization stage. Heuristics in general can lead to inefficient trajectories such as the one before refinement shown in Fig. 1. Iterative methods such as gradient descent [4], [7] have been used to optimize time allocation. However, gradient is computed by finite difference and $(n+1)$ QPs have to be solved for a problem with n segments for every gradient evaluation, making it slow and inaccurate. We use sensitivity analysis to obtain exact gradient efficiently. Another strategy to determine time allocation is to use sampling [3] in cases where finding a feasible QP is the major issue. Sampling is sufficient when optimality is not emphasized, while in our work we focus on finding an optimal time allocation.

B. Bilevel Optimization

Bilevel optimization [8] refers to a mathematical program where one optimization problem (the upper-level problem) has another optimization problem (the lower-level problem) as one of its constraints, i.e., one optimization task is embedded within another.

Bilevel and multi-level optimization techniques have been employed for switching time optimization for switched systems [9]–[12]. This literature has concentrated on calculating derivatives of an objective function with respect to switching times. In particular, works by Xu et al. [9] and Egerstedt et al. [10] compute the derivatives using Lagrange multiplier methods, which bear some resemblance to sensitivity analysis technique used in this paper. However, these works are based on Pontryagin's Maximum Principle [13] and are difficult to include inequality path constraints, which is often unavoidable in robotic applications.

Applications of bilevel optimization in robotics include trajectory optimization for legged robots [12], and robust control and parameter estimation [14]. Landry et al. [14] present a bilevel optimization solver based on 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. [8] and Colson et al. [15] 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.

III. Methodology

In this section, we describe our framework. We start by an introduction of trajectory optimization, then give a mathematical formulation of the problem we are trying to solve 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 3-D 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 asks 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

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

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]. The safe corridors are generated such that all obstacles are outside. As a result, as long as the trajectory is constrained inside the safe corridors, collision avoidance can be achieved. One such corridor is illustrated in Fig. 1. In general, this method could work for corridors composed of any convex polytope, but axis-aligned boxes are chosen for simplicity and compatibility with the grid data structures commonly used by perception algorithms.

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 (timing of the knot points).

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 i'th 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 j'th control point in the i'th segment and $B_{d,j}$ denotes the j'th Bernstein polynomial of order d defined as

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

The choice of Bézier curve makes it simple to constrain the trajectory inside safe corridors. Since each corridor is convex, and a Bézier curve is inside the convex hull of its control points, it suffices to constrain all control points of each Bézier curve to be inside its corresponding safe corridor.

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 $y = [\Delta t_1, \dots, \Delta t_n]^T \in \mathbb{R}^n_+$.

1) Objective Function: 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:

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

where q is the derivative order and T is a chosen trajectory time.

It has been shown [4], [5] that the first term in Eq. (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. With a slight abuse of notation, we can write Eq.(1) in terms of polynomial coefficients c and time allocation y:

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

where $P_q(y)$ is a symmetric positive semidefinite matrix that is nonlinear in y. Note that we will drop q in $P_q(y)$ since we set $q = 3$ (we penalize jerk) 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. We found that in the UAV case, applying continuity constraints up to acceleration yields good results.

The above constraints can be complied 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-C.
2) The maximum velocity $\|x'(t)\|_\infty$ and maximum acceleration $\|x''(t)\|_\infty$ along the trajectory is bounded, i.e.,

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

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

We encode the trajectory using a piecewise Bézier curve [5], which has the 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 (see Ref [5] for details), safety and dynamically feasible constraints can be imposed as a linear inequality constraint on the flattened coefficients:

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

where matrix G is generally nonlinear in time allocation y.

4) Constraints on Time: Constraints on time allocation y include the sum equals T and the duration of each segment is positive. We encode these constraints as

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

with A, b, C, d properly chosen.

D. Final Formulation

In summary, we collect Eq.(2), (3), (5) and (6), into the problem of Trajectory Optimization using Bézier spline in a Corridor (TOBC):

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

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

E. Formulation of the Bilevel Optimization Problem

To efficiently solve (TOBC), we rewrite it as the bilevel optimization:

$$\begin{aligned} \underset{c,y}{\text{minimize}} \quad & J(c, y) = c^T P(y) c \\ \text{subject to} \quad & c \in \underset{c}{\operatorname{argmin}} \{J(c, y) : G(y)c \le h,\ H(y)c = m\} \\ & Ay \le b \\ & Cy = d. \end{aligned} \tag{TOBC-BO}$$

Note that although the objective functions $J(c, y)$ remain the same in both the lower-level (as 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 quadratic program (QP) because $J(c, y)$ is quadratic in c when y is fixed.

Our solution strategy, is to use a 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.,

$$c^\star(y) \in \underset{c}{\operatorname{argmin}} \{J(c, y) : G(y)c \le h,\ H(y)c = m\}. \tag{7}$$

The descent method indeed has been used to solve bilevel optimization problems [8], and our framework is a variant of

this method. Given an 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 general issue is the availability of gradients, which is addressed in the next section.

F. Gradient Computation

We use a key result from sensitivity analysis of parametric nonlinear programming (NLP) [16] 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 Fiacco [16, Theorem 3.4.1] for details.

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

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

where c is the vector of decision variables and $y \in \mathbb{R}^n$ is a parameter vector.

Given a locally optimal solution $c^\star(y)$ with cost $J^\star(y)$, and Lagrange multipliers $\lambda(y)$ and $\nu(y)$ associated with $g(\cdot)$ and $h(\cdot)$, respectively, under mild assumptions, the gradient of the objective is

$$\nabla_y J^\star(y) = \nabla_y J + \sum_{i=1}^{m} \lambda_i(y) \nabla_y g_i + \sum_{j=1}^{p} \nu_j(y) \nabla_y h_j. \quad (8)$$

The conditions for this theorem to hold are assumed to be true in this paper and we leave as future work how to deal with corner cases such as switching of active inequality constraints. Empirical results show that our method performs well in all the test cases.

G. Solving Bilevel Optimization

Our algorithm, given in Alg 1, uses gradient 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.

Line 5 projects the gradient to satisfy linear constraints of y. For this problem, the linear constraints are $\mathbb{1}^T y = T$ and $y \geq \delta$ for some small δ. The projection is simply $p = g - \mathbb{1}^T g / \|\mathbb{1}\|_1$ to make sure the sum of p is zero so the total traversal time is not changed. If inequality constraint $y \geq \delta$ is violated at some entries, their values are set as δ and the rest are projected. Line 6, which is further illustrated in Algorithm 2, finds a suitable step length α that gives sufficient decrease in the objective function. Since line search is used, this algorithm guarantees that every major iteration improves the cost. The optimality conditions used in our implementation are:

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

Algorithm 1 Refine-Time (y_0)

```
1:  y ← y_0
2:  for i ← 0 to max-iterations do
3:      J, λ, ν ← Solve-QP(P(y), G(y), h, H(y), m)
4:      g ← Get-Gradient(λ, ν)                    ▷ From Eq. (8)
5:      p ← Project-Gradient(g, A, b, C, d)
6:      α, J_α, λ_α, ν_α, y_α ← Line-Search(y, p)
7:      if α not found then
8:          break
9:      if optimality-conditions-satisfied then
10:         break
11:     J, λ, ν, y ← J_α, λ_α, ν_α, y_α
12: return y
```

Algorithm 2 Line-Search (y_s, p)

```
1:  static variable α_0
2:  constant variables τ_g, τ
3:  α ← α_0
4:  for i ← 0 to max-iterations do
5:      y = y_s − αp
6:      J, λ, ν, ← Solve-QP(P(y), G(y), h, H(y), m)
7:      if sufficient-decrease-achieved then
8:          if i = 0 then
9:              α_0 ← τ_g α
10:         else
11:             α_0 ← α
12:         return α, J, λ, ν, y
13:     α ← τα
14: return α not found
```

Algorithm 2 shows our adaptive backtracking line search routine starting from y_s in the direction of p, step length α is returned if found as in Line 12 along with other related results, otherwise "α not found" will be returned as in Line 14 and Alg. 1 is terminated. In our implementation, the Armijo condition [17] is used for checking sufficient decrease in the objective. The initial step length α_0 is updated in an adaptive strategy similar to trust region method. The adaptive strategy needs $\tau_g > 1$ and shrinking parameter $\tau < 1$. In this paper, $\tau_g = 1.5, \tau = 0.2$.

IV. Experiments

Our algorithm, which is released as an open-source package[1], is implemented in a combination of Python and C++. Python is used in constructing the QP problem, performing line search, and gradient calculation, while all QP solvers are implemented in C++. Pybind11[2] is used as the interface between Python and C++.

[1]https://github.com/OxDuke/Bilevel-Planner
[2]https://github.com/pybind/pybind11

A. Problem Description

Experiments use the minimum-jerk as the objective (following Mellinger and Kumar [4]):

$$\begin{aligned} \underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y) c \\ \text{subject to} \quad & \mathbb{1}^T y = T_c \\ & y \geq \delta \\ & G(y)c \leq h \\ & L(y)c = m, \end{aligned} \tag{9}$$

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. We refer [5] for how P, G, L, h, m is defined due to page limit.

B. Numerical Experiments

We evaluate our method on random instances of "forests" using the environment generator of Gao et al. [5]. We generated 100 tests, each with a random environment and randomly sampled feasible start points and goal points. Because we use 6'th order piecewise Bézier curves, the number of variables in each optimization problem is $(6+1) \times 3n$ where n is the number of segments. We note that higher order Bézier curve can also be used. The initial guess of time allocation and fixed total traversal time are computed from the heuristic as in [5]. All the experiments are carried out on a workstation with a 4.0 GHz Intel Core i7-6700K processor, using only one thread.

Table I compares results from different combinations of formulation methods and numerical solvers:

1) LM/FD+Sqopt/Mosek: Solve (9) using bilevel formulation with two gradient computation techniques: LM, which uses Lagrange multiplier gradient computations; FD, which uses finite differences with step length of 0.25×10^{-6}. We tested the following QP solvers: Sqopt [18], an active-set QP solver intended for large and sparse systems; and Mosek [19], an interior-point QP solver.
2) NLP+SNOPT: Directly solving (9) as an NLP using SNOPT [20], a general nonlinear solver for sparse, large-scale problems. 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}.
3) DC+SNOPT: Direct Collocation (DC) [13] is also used to solve problem (9) which directly formulates an NLP solved by SNOPT. We note that DC cannot directly optimize Bézier spline coefficients but discretized states $[p(t), \dot{p}(t), \ddot{p}(t)]$ where $p(t)$ is position and control $u \equiv \dddot{p}(t)$ at each collocation grid. The system dynamics is thus $\frac{d}{dt}[p(t), \dot{p}(t), \ddot{p}(t)] = [\dot{p}(t), \ddot{p}(t), u(t)]$. Due to different representations of trajectory and approximation of the cost integral by summation, its cost function is not the same as the bilevel approach which directly optimizes Bézier spline coefficients and minimizes analytic cost integral. This is why we do not compare the final objectives from DC and other approaches in Table I. Due to page limit, we refer readers to Betts [13] for detailed formulations of DC. The initial guess for DC is the same as NLP+SNOPT. For SNOPT, the maximum number of minor iterations is set to 500,000, and the stopping criteria are set to default.

TABLE I
NUMERICAL EXPERIMENTS (MEAN/MEDIAN), INCLUDING COMPUTATION TIME, SUBOPTIMALITY AND CONSTRAINT VIOLATION.

Method	Comp. Time [ms]	Suboptimality	Cons. Vio.
LM+Sqopt (ours)	71.7 / 54.8	0.025 / 1.8e-7	0.0 / 0.0
LM+Mosek (ours)	127.2 / 117.1	0.016 / 0.0	0.0 / 0.0
FD+Sqopt	220.4 / 168.9	1.179 / 0.144	0.0 / 0.0
FD+Mosek	410.3 / 397.2	7.102 / 0.754	0.0 / 0.0
NLP+SNOPT	22.6 / 18.1	242.75 / 14.66	0.087 / 0.033

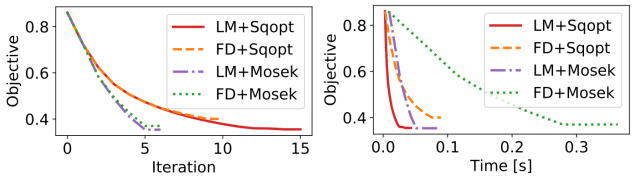
The *Suboptimality* column refers to the relative suboptimality $\frac{J - J^\star}{J^\star}$, where J denotes the objective value achieved by an algorithm, and $J^\star$ denotes the true optimum. In our experiment, we approximate $J^\star$ as the minimum of all the objectives returned by different algorithms.

Overall, LM improves computation time by 2-3 times beyond FD, since the performance bottleneck is moved from gradient computation to the line search step, which ultimately takes about 90% of the total computation time. Sqopt is generally faster than Mosek, and this could be a consequence of active-set methods' ability to perform warm start. Moreover, LM terminates with a much lower suboptimality. We suspect this is because of the improved accuracy of the LM gradient computation.

The NLP+SNOPT formulation does poorly compared to our bilevel optimization framework. 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.

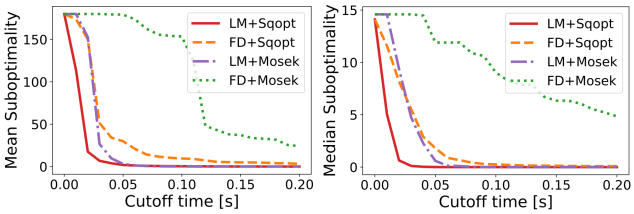
The direct collocation formulation only succeeds on 78% of tests, and is rather slow: the mean and median computation time for the successful cases are 9.6s and 2.5s, respectively. This indicates the DC method is not suitable for real-time applications. The maximum number of iterations is reached without converging on 22% of the tests. Since the formulation of DC is not equivalent to ours, suboptimalities of DC are not reported in Table I.

Fig. 2 examines convergence FD and LM in greater detail on a single test case. The convergence of NLP+SNOPT was



(a) Objective vs iterations (b) Objective vs computation time

Fig. 2. Convergence plot of our Lagrange multiplier method (LM) compared against finite differences (FD) on a representative test case. FD is almost as accurate as LM, since both methods take about the same number of iterations to converge to the same cost while LM achieves a slightly lower objective. However, FD is about 4 times slower in terms of computation time. Although Mosek takes fewer iterations to converge, Sqopt is faster overall because of its warm-start capability.



(a) Mean suboptimality (b) Median suboptimality

Fig. 3. Mean / median cutoff time vs suboptimality over all random test cases, using bilevel optimization.

highly irregular, and is therefore not shown. These plots show that convergence rates are similar in the initial iterations, but LM continues to improve as the descent approaches the optimum. Moreover, each iteration of LM is faster. The results shown in Fig. 3 show convergence as a function of time over all examples. 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.

Observe that due to the properties of the steepest descent method, the first few iterations will give significant decrease in the objective. When LM is run at 40Hz (25 ms cutoff time), which is a reasonable frequency for real-time applications, it achieves a mean and median suboptimality of 12.2 and 0.97 respectively, compared to 100.8 and 7.84 achieved by the FD method of Mellinger and Kumar [4].

V. Conclusion

We present a novel bilevel optimization approach to UAV trajectory optimization, which calculates the gradient of the objective function w.r.t. temporal variables. Our results show that this approach achieves real-time performance and higher quality trajectories than state-of-the-art heuristics. Compared with methods that require solving NLP directly, this approach is any-time and has feasibility guarantee. Future work includes applying this method on physical system and analyzing the convergence rate of our approach.

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 *IEEE International Conference on Robotics and Automation*, 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," *arXiv: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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 1–9.

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE Intl. Conf. on Robotics and Automation*, 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 *IEEE International Conference on Robotics and Automation*, 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 *Robotics Research*. Springer, 2016, pp. 649–666.

[8] 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, no. 2, pp. 276–295, 2018.

[9] X. Xu and P. J. Antsaklis, "Optimal control of switched systems based on parameterization of the switching instants," *IEEE Transactions on Automatic Control*, vol. 49, no. 1, pp. 2–16, 2004.

[10] M. Egerstedt, Y. Wardi, and F. Delmotte, "Optimal control of switching times in switched dynamical systems," in *42nd IEEE International Conference on Decision and Control*, vol. 3, 2003, pp. 2138–2143.

[11] E. R. Johnson and T. D. Murphey, "Second-order switching time optimization for nonlinear time-varying dynamic systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1953–1957, 2011.

[12] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 93–100.

[13] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[14] B. Landry, Z. Manchester, and M. Pavone, "A differentiable augmented lagrangian method for bilevel nonlinear optimization," *arXiv:1902.03319*, 2019.

[15] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[16] A. V. Fiacco, *Introduction to sensitivity and stability analysis in nonlinear programming*. Elsevier, 1983.

[17] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[18] 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.

[19] MOSEK ApS, *MOSEK Optimizer API for C, 8.1.*, 2018. [Online]. Available: https://docs.mosek.com/8.1/capi/index.html

[20] 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.