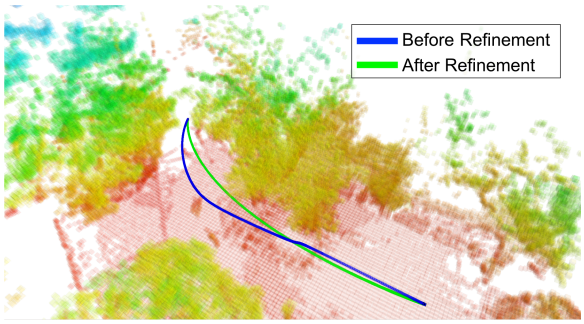
Fast UAV Trajectory Generation Using Bilevel Optimization

Weidong Sun*, Gao Tang*, and Kris Hauser

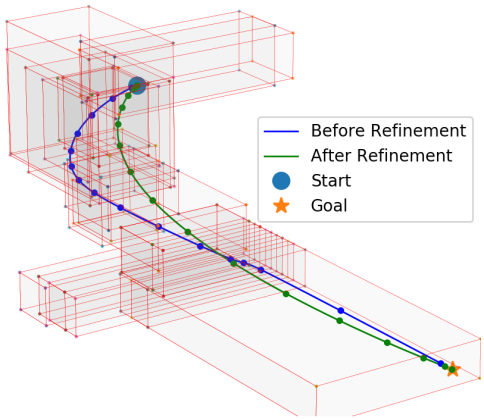
***Abstract*— We present an efficient bilevel optimization framework that solves trajectory optimization problems efficiently by decoupling state variables from timing variables, thereby decomposing a challenging nonlinear programming (NLP) problem into two easier subproblems. With timing fixed, the state variables can be optimized efficiently using convex optimization in the lower level optimization, and the timing variables can be optimized in the upper level. Results from sensitivity analysis of parametric NLPs shows that the dual solution (Lagrange multipliers) of the convex optimization problem can be exploited to calculate the gradient of the time variables. 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 reaches a lower cost over a standard NLP solver, and analytical gradients outperforms finite differences in terms of computation speed. With a 40 ms cutoff time, our approach achieves over 8 times better suboptimality than gradient descent using finite difference gradient approximation.**

I. Introduction

Polynomial trajectories are widely used in motion planning for UAVs. The trajectory generation problem is often formulated as a quadratic programming (QP) with the limitation that each segment has predetermined duration (timing). While the fixed timing can lead to sub-optimal trajectories, huge improvement can be made by refining the timing, as shown in Fig. 1. In Mellinger *et al.* [1], gradient descent with gradient approximated by finite difference is used for optimizing timing for each segment. Consequently, it suffers from the large number of optimization being performed and low accuracy from finite difference. In this work, we use sensitivity analysis to get gradient efficiently and accurately, enabling real-time optimization of segment duration. We present a bilevel 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: a lower-level problem and a upper-level problem. With timing fixed, the state variables can be optimized efficiently using convex optimization in the lower-level problem, and the timing variables can be optimized in the upper-level. Key results from sensitivity analysis of parametric NLP shows that the dual solution (Lagrange multipliers) to the lower-level problem can be used to compute analytical gradients to the upper-level problem under mild regularity conditions [2]. Since the Lagrange multipliers are a by-product that can be obtained "for free" by solving an optimization problem, it is a fast yet accurate way of computing gradients compared to finite difference. Experiments are conducted on generating trajectories for UAV in a cluttered environment, which show that our gradient estimation approach is more efficient than the current state-of-the-art [1] while also being more accurate. Overall, this leads to an 8 times improvement in suboptimality when the optimizer is given a 40 ms cutoff.



(a) Trajectories that fly through a gazebo in the forest



(b) Flying corridor formed by axis-aligned boxes

Fig. 1. Trajectories for flying through an outdoor environment mapped in [3]. The one with unrefined timing, computed by the implementation from [4], shown in blue, is unnecessarily curvy. In Fig.1b, dots along the curves are equally distributed times sampled along the trajectory, which show that the time allocation of the refined trajectory is more evenly distributed. Statistics of these trajectories are shown in Table. I.

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.

* Denotes equal contribution

TABLE I

STATISTICS FOR FIG.1. LENGTH AND COST ARE SHOWN IN (REFINED/UNREFINED)

Length	#Segments	Cost	Computation Time
19.9m / 20.7m	19	0.15 / 45.06	352ms

II. METHOD

A. Spline-Based Trajectory Generation

We represent the trajectory as a d'th order Bézier spline which has n segments and segment durations $\Delta t_1, \ldots, \Delta t_n$. The timing of the i'th connection 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

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

where c_{ij} is the j'th control point of the i'th segment.

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 timing as $y = \begin{bmatrix} \Delta t_1 & \ldots & \Delta t_n \end{bmatrix} \in \mathbb{R}^n$ (temporal part).

The trajectory generation is transcribed as an optimization problem where the objective function is chosen to be the integral of the squared norm of jerk, i.e., we use a minimum jerk objective. As shown in [1], [4], the minimum jerk objective can be written as a quadratic function of the polynomial coefficients, if the time allocation is fixed. Our objective function f_0 is

$$f_0(x, y) = \tfrac{1}{2} x^T P(y) x + q(y)^T x + c(y),$$

where the first and second term encode the minimum jerk objective, the third term, which only depends on y, penalizes total traversal time with $c(y) = c \sum_i \Delta t_i = \mathbb{1}^T y$. In general, the objective $f_0(x, y)$ is quadratic in x (polynomial coefficients) and nonlinear in y (time allocation).

The trajectory is constrained so that:

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.
3) The whole trajectory lies in the safe region to ensure collision avoidance.
4) Velocity and acceleration stay in the bound.

The above constraints can be converted to linear constraints on the Bézier coefficients similar to [4] as:

$$G(y)x \leq h(y),$$
$$L(y)x = m(y),$$

where $G(y), L(y)$ and $h(y), m(y)$ are matrices and vectors which are nonlinear in y.

We also introduce linear timing constraints $Ay \leq b$ and $Cy = d$ to enforce positive durations $y \geq 0$, and possibly fixed total time $\mathbb{1}^T y = T$.

In summary, the trajectory generation problem can be formulated as a bilevel optimization problem [5] as follows:

$$\min_{x,y} \quad f_0(x, y) \tag{1a}$$
$$\text{s.t.} \quad Ay \leq b, \tag{1b}$$
$$Cy = d, \tag{1c}$$
$$x = \operatorname*{argmin}_x \quad f_0(x, y) \tag{1d}$$
$$\text{s.t.} \quad G(y)x \leq h(y), \tag{1e}$$
$$L(y)x = m(y). \tag{1f}$$

where the *lower-level optimization problem* is defined by Eq.(1d)-(1f), the *upper-level optimization problem* is defined by Eq.(1a) - Eq.(1f). Note that although the objective functions remain the same in both the lower-level and upper-level optimization problem, timing y is fixed in the lower-level problem but becomes the variable we want to optimize in the upper-level problem. The lower-level problem is specified to be an QP, which can be solved efficiently and globally optimally in x using "off-the-shelf" solvers.

Our strategy is to use a constrained gradient descent on the function $f_0^\star(y) = f_0(x^\star(y), y)$ with $x^\star$ minimizing the QP for a fixed timing y. The descent method indeed has been used to solve bilevel optimization problems [5], and our framework is a variant of this method. Given an feasible y, we find a direction $-\nabla f_0^\star(y) \in \mathbb{R}^n$ and a step length α that can make a sufficient decrease in $f_0^\star(y)$ while maintaining the feasibility of the new point $y_{new} = y - \alpha \nabla f_0^\star(y)$. The gradient $\nabla_y f_0^\star(y)$ is derived from sensitivity analysis of parametric nonlinear programs under mild regularity conditions:

$$\begin{aligned} \nabla_y f_0^\star(y) &= \nabla_y f_0(x^\star(y), y) \\ &= \nabla_y f_0(x^\star(y), y) + \lambda^T G(y) + \nu^T L(y), \end{aligned}$$

where λ and ν are Lagrange multipliers associated with Eq.(1e) and (1f), respectively.

B. Algorithm

Our bilevel optimization algorithm is given in Algorithm 1, which takes an initial guess y_0 of the time allocation as input. It then iteratively descends $f^\star$ until some optimality conditions are satisfied or the maximum number of iterations is reached.

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 [6]. Line 6, which is a standard backtracking line search, 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

Algorithm 1 Bilevel-Solve (y_0)

1: $y \leftarrow y_0$
2: $J, \lambda \leftarrow$ Solve-QP$(P(y), G(y), h(y), L(y), m(y))$
3: **for** $i \leftarrow 0$ to *max-iterations* **do**
4: $g \leftarrow$ Get-Gradient(λ) ▷ From Eq. (**??**)
5: $p \leftarrow$ Project-Gradient(g, A, b, C, d)
6: $\alpha, J_\alpha, \lambda_\alpha, y_\alpha \leftarrow$ Line-Search(t, p) ▷ Backtracking linesearch
7: **if** α not found **then**
8: **break**
9: **if** optimality-conditions-satisfied **then**
10: **break**
11: $J, \lambda, y \leftarrow J_\alpha, \lambda_\alpha, y_\alpha$
12: **return** t

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 Line 9 check whether the norm of the projected gradient is less than a threshold, or the change of the objective function is less than a threshold.

III. Experiments

A. An example

To illustrate the effect of optimizing time allocation, we generate trajectories for flying through an outdoor environment. The environment is a gazebo in a forest, mapped in [3]. We compare the trajectories generated by our algorithm and the one described in [4]. Both algorithms take exactly the same input (safety corridor, initial/end states, velocity/acceleration bounds, total traversal time, initial guess of the time allocation), the difference is that ours will optimize time allocation and the one from [4] will not. The initial timing is from the heuristics described in [4].

The results, illustrated in Fig.1 and Fig.3, show that a sub-optimal timing can lead to excessive curviness of the trajectory, spikes in acceleration and its higher order derivatives.

B. Numerical Experiments

We demonstrate our method on trajectory planning for UAVs in the presence of obstacles. Our implementation is in Python and C++: Python is used in constructing the QP problem, performing line search and gradient calculation, while all QP solvers run in C++. Pybind11 [7] 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.* [4] to generate convex collision-free 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. 2. 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.

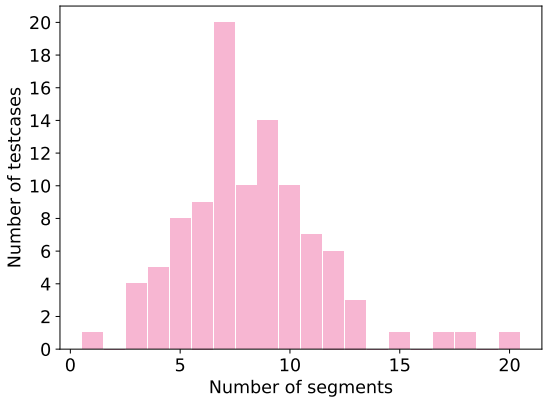


Fig. 2. Test case descriptive statistics.

TABLE II
Experimental results (mean/median). Our method shown in last two rows

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

We establish a minimum-jerk objective function, and fix the total amount of time on the trajectory. The initial timing is set from the heuristic described in [4]. The fixed-duration constraint is used for fair comparison with the unrefined trajectory, but we note that our framework can handle general objective functions and constraints on timing.

Table II summarizes our experiments comparing different combinations of gradient estimation methods and QP solvers. As a baseline, we solve the coupled NLP using the sparse NLP solver SNOPT [8]. Here we simply formulate the joint spatial and temporal optimization problem as an NLP over x and y. We provide analytic gradients to SNOPT for solver robustness, and initialize it with the unrefined time allocation scheme and the spline coefficients resulting with the first QP solution. We compare our bilevel Lagrange multiplier method (LM) against bilevel finite differences (FD), using two QP solvers, Sqopt [9] and Mosek [10], as representatives of active-set and interior-point methods. The Suboptimality column refers to relative suboptimality, which is calculated as $\frac{J - J^\star}{J^\star}$, where J is the objective value achieved by an algorithm, and $J^\star$ is the true optimum, which is set to the minimum of all the objectives returned by different algorithms.

These results show that SNOPT terminates quickly, but is very suboptimal. It is actually terminating prematurely without converging, and it often terminates at an infeasible point, even though it starts from a feasible trajectory. We believe that this is because the NLP is somewhat badly conditioned due to the high-order dependence on timing and spline

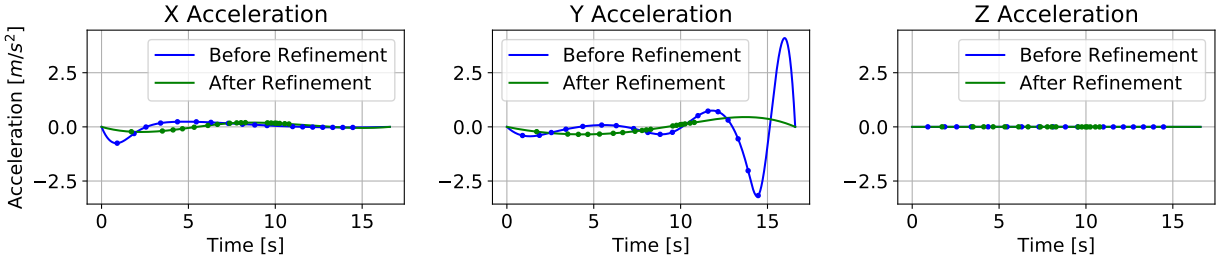


Fig. 3. Acceleration w.r.t. time for the refined and unrefined trajectories shown in Fig.1. Spikes on the acceleration are smoothed out by refining time allocation. Dots along the curves are the knot points, which show the change of time allocation.

coefficients. Bilevel techniques are slower but substantially more reliable. Our LM method improves overall running time by 2-3 times beyond FD, and moreover terminates with a much lower suboptimality. Sqopt is generally faster than Mosek, and this could be a consequence of active-set methods' ability to perform warm start.

The results shown in Fig. 4 explore suboptimality as a function of time. We record the suboptimality achieved at each cutoff time. If no iteration has been finished at the cutoff time, we use the unrefined objective instead. Note that due 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. 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 [1].

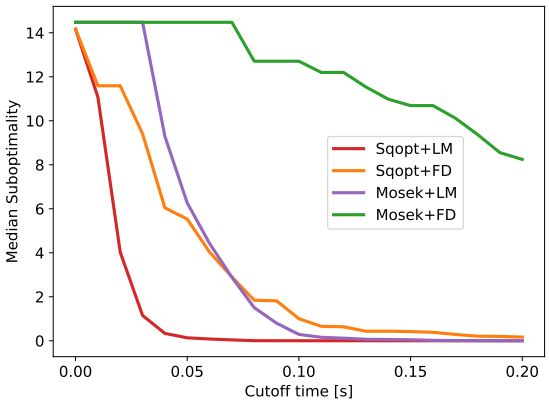


Fig. 4. Computation time vs suboptimality for UAV path planning using bilevel optimization. Our Lagrange multiplier method (LM) is compared against finite differences (FD).

IV. Conclusion

We presented a an efficient bilevel optimization framework to solve trajectory optimization problems. 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.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors." in *ICRA*. IEEE, 2011, pp. 2520–2525.

[2] A. V. Fiacco and Y. Ishizuka, "Sensitivity and stability analysis for nonlinear programming," *Annals of Operations Research*, vol. 27, no. 1, pp. 215–235, Dec 1990. [Online]. Available: https://doi.org/10.1007/BF02055196

[3] F. Pomerleau, M. Liu, F. Colas, and R. Siegwart, "Challenging data sets for point cloud registration algorithms," *The International Journal of Robotics Research*, vol. 31, no. 14, pp. 1705–1711, Dec. 2012.

[4] 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.

[5] 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.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[7] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – seamless operability between c++11 and python," 2017, https://github.com/pybind/pybind11.

[8] 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.

[9] ——, "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.

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