Alternating Minimization Based Trajectory Generation for Quadrotor Aggressive Flight

Zhepei Wang, Xin Zhou, Chao Xu, Jian Chu, and Fei Gao

***Abstract*—With much research has been conducted into trajectory planning for quadrotors, planning with spatial and temporal optimal trajectories in real-time is still challenging. In this letter, we propose a framework for large-scale waypoint-based polynomial trajectory generation, with highlights on its superior computational efficiency and simultaneous spatial-temporal optimality. Exploiting the implicitly decoupled structure of the problem, we conduct alternating minimization between boundary conditions and time durations of trajectory pieces. Algebraic convenience of both sub-problems is leveraged to escape poor local minima and achieve the lowest time consumption. Theoretical analysis for the global/local convergence rate of our method is provided. Moreover, based on polynomial theory, an extremely fast feasibility checker is designed for various kinds of constraints. By incorporating it into our alternating structure, a constrained minimization algorithm is constructed to optimize trajectories on the premise of feasibility. Benchmark evaluation shows that our algorithm outperforms state-of-the-art waypoint-based methods regarding efficiency, optimality, and scalability. The algorithm can be incorporated in a high-level waypoint planner, which can rapidly search over a three-dimensional space for aggressive autonomous flights. The capability of our algorithm is experimentally demonstrated by quadrotor fast flights in a limited space with dense obstacles. We release our implementation as an open-source ros-package.[1]**

***Index Terms*—Motion and path planning, autonomous vehicle navigation, aerial systems: applications.**

I. Introduction

RECENTLY, our community has witnessed the development of planning methods for quadrotors. Spline-based methods, which decompose the spatial and temporal parameters of a planning problem and focus on its spatial part, are widely applied for real-time applications [1]–[3].

Although spline-based methods can efficiently and accurately generate energy-optimal solutions for online usage, they usually omit temporal planning for simplicity. A typical spatial-temporal joint planning problem has high nonlinearity and nonconvexity coming from its objective and constraints. Since temporal planning has underlying coupling with spatial parameters and implicit gradients, the spatial-temporal joint optimization cannot be solved by general nonlinear programming (NLP) in real-time. Even though existing methods can provide online motion planning without temporal planning, they are often too conservative to be used for autonomous flights with high aggressiveness. To bridge this gap, we propose a framework to split the spatial and temporal aspects of a trajectory optimization problem, then solve them alternately. With our method, we can obtain the energy-time joint optimal trajectory in milliseconds.

Fig. 1. Composite image of the quadrotor aggressive flight in a limited space. Our quadrotor is equipped with a stereo camera, an IMU and an onboard computer. No external positioning system is used. A video of the experiment can be viewed at https://youtu.be/H89ALyWA2NI.

The proposed method is based on the naturally alternating structure of the spatial-temporal trajectory optimization and designed to have guaranteed optimality and efficiency. Our method is motivated by the fact that a polynomial trajectory with an odd order can be uniquely determined by its endpoint derivatives, i.e., the boundary condition, and the time duration. For a piecewise polynomial, once all boundary conditions are fixed, each piece of the trajectory depends only on its time duration, which can be optimized separately. By utilizing the widely-adopted linear-quadratic objective [4] of the optimization, the optimal time durations can be updated efficiently. Moreover, inspired by [2], the closed-form solution is adopted to update derivatives on waypoints. Based on the above observations, the joint optimization can be efficiently processed by an alternating minimization (AM) procedure [5]. With our method, a large-scale joint optimization can be done in a few milliseconds to generate optimal trajectory with the best time allocation.

To the best of our knowledge, our method is the first one that generates trajectories for a quadrotor with the spatial-temporal optimality, in such a limited time. Summarizing our contributions in this work:

Manuscript received February 24, 2020; accepted June 11, 2020. Date of publication June 19, 2020; date of current version June 30, 2020. This letter was recommended for publication by Associate Editor Fabio Ruggiero and Editor Jonathan Roberts upon evaluation of the reviewers' comments. This work was supported by the Fundamental Research Funds for the Central Universities under Grant 2020QNA5013. *(Corresponding author: Fei Gao.)*

The authors are with the State Key Laboratory of Industrial Control Technology, and the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: wangzhepei@zju.edu.cn; iszhouxin@zju.edu.cn; cxu@zju.edu.cn; chuj@zju.edu.cn; fgaoaa@zju.edu.cn).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.3003871

[1]The source code is available at https://github.com/ZJU-FAST-Lab/am_traj

- An unconstrained alternating minimization algorithm is proposed to generate spatial-temporal optimal trajectories efficiently, with proven global/local rates of convergence.
- A computationally efficient feasibility check method is designed for a wide range of constraints.
- A constrained alternating minimization algorithm is constructed to optimize feasible trajectories in a recursive fashion, with global convergence verified.
- The proposed method is integrated into an autonomous quadrotor system then evaluated by real-world experiments as well as extensive benchmarks. The source code is released for the reference of the community.

In what follows, we discuss related literature in Sec. II. Preliminaries of this letter are given in Sec. III. The proposed spatial-temporal trajectory generation method for unconstrained and constrained planning cases are detailed in Sec. IV and V, respectively. Experiments and benchmarks are given in Sec. VI. The letter is concluded in Sec. VII.

II. Related Work

For quadrotor planning, polynomial splines have long been used for trajectory parameterization since [1], because of their flexibility and analytical convenience. In [1], the minimization of squared derivatives is used as the objective of quadratic programming (QP), which can be solved efficiently and accurately. In [2], a closed-form solution of above QP program is proposed. Based on this formulation, intensive works have been recently proposed. In [6]–[8], dynamically feasible trajectories are online generated within a safe flight corridor, which excludes all obstacles in complex environments. However, in these methods, the time allocation of the trajectory is pre-defined or heuristically adjusted. Although heuristics are cheap to compute, the trajectories generated are often far less optimal and over-conservative, making them incapable of high-speed flights.

To address the time allocation problem, Mellinger *et al.* [1] compute the projected gradient with respect to durations on a hyperplane where the sum is fixed. They optimize time allocation through backtracking gradient descent. Temporal scaling is applied on the solution until dynamical feasibility is achieved. Both the finite difference and scaling used in this method are expensive operations when the number of trajectory pieces is large. Liu *et al.* [6] propose a proper scaling factor for rest-to-rest trajectories such that a single scaling operation suffices. Sun *et al.* [9] formulate the problem as a bi-level optimization. They estimate the projected gradient analytically through the dual solution of the low-level QP, which is more accurate than the numerical gradient. Nonetheless, the dynamical feasibility is conservatively treated to formulate the QP. To avoid the constraint on total duration, Richter *et al.* [10] use total duration as a regularization term, thus making each duration an independent variable. The time allocation is optimized through gradient descent, while actuator constraints are also fulfilled by scaling. However, the optimal ratio of time allocation under the constrained case may differ a lot from the unconstrained case. Consequently, scaling can ruin a trajectory where constraints are violated on a very short piece. Burri *et al.* [3] optimize the squared total duration instead. They soften all constraints by penalizing them in the objective and optimize durations through an NLP solver, while the pieces number is limited.

Trajectories with one-layer parameterization and gradient-optimized time allocation in [10] can be unsatisfactory when solution quality or time consumption counts the most. To improve trajectory quality, Gao *et al.* [11] use two-layer parameterization for higher degrees of freedom (DoF). They decouple geometrical and temporal information of a rest-to-rest trajectory. A spatial trajectory is optimized through QP within box-shaped corridors. A temporal trajectory is then optimized by second-order conic programming (SOCP) based on direct collocation [12], which maps time to the spatial trajectory. The spatial trajectory can be re-generated by a better time allocation. By alternating minimization between these two-layer coefficients, the high DoF trajectory can be much improved. In [13], they improve this method by using polyhedron-shaped corridors for higher success rates. Their method increases solution quality, but the computation time is unsuitable for online usage. To reduce time consumption, Almeida *et al.* [14] train a supervised neural network to learn time allocation offline. Thus online refinement of its good initial guesses can be done in real-time. However, the neural network has to be trained case by case.

In this letter, we adopt the time regularized objective [10] as well as the alternating minimization framework [13]. For efficiency, only one-layer parameterization is used, of which the spatial and temporal parts are optimized alternately. Each optimization phase exploits the objective structure and is solved algebraically, making it free from gradient estimation and step-size choosing. To handle various constraints, we also design a simple yet solid feasibility checker. The proposed framework is able to generate aggressive trajectories at extremely high frequency and not limited to the rest-to-rest case.

III. Preliminaries

Differential flatness of quadrotor dynamics is validated by Mellinger *et al.* [1]. It means the trajectory planning for a quadrotor can be done solely in the translational space. The kinodynamic feasibility is implicitly transformed into smoothness of the trajectory. Then, actuator constraints can be enforced by restricting norms on high-order derivatives.

In this letter, we employ the piece-wise polynomial trajectory, with each piece denoted as an N-order polynomial:

$$p(t) = \mathbf{c}^{\mathrm{T}}\beta(t), \quad t \in [0, T], \tag{1}$$

where $\mathbf{c} \in \mathbb{R}^{(N+1)\times 3}$ is the coefficient matrix, T is the duration and $\beta(t) = (1, t, t^2, \ldots, t^N)^{\mathrm{T}}$ is a basis function.

It is worth noting that we only consider odd-order polynomial trajectories. Since N is odd, the mapping is bijective between the coefficient matrix and the boundary condition. To further explain this, consider derivatives of $p(t)$ up to $\lceil (N-1)/2 \rceil$ order:

$$\mathbf{d}(t) = (p(t), \dot{p}(t), \ldots, p^{(\lceil (N-1)/2 \rceil)}(t))^{\mathrm{T}}, \tag{2}$$

we have $\mathbf{d}(t) = \mathbf{B}(t)\mathbf{c}$ where

$$\mathbf{B}(t) = (\beta(t), \dot{\beta}(t), \ldots, \beta^{(\lceil (N-1)/2 \rceil)}(t))^{\mathrm{T}}. \tag{3}$$

We denote $\mathbf{d}_{start}$ and $\mathbf{d}_{end}$ by $\mathbf{d}(0)$ and $\mathbf{d}(T)$, respectively. The boundary condition of this polynomial is described by the tuple $(\mathbf{d}_{start}^{\mathrm{T}}, \mathbf{d}_{end}^{\mathrm{T}})^{\mathrm{T}}$. The following mapping holds:

$$(\mathbf{d}_{start}^{\mathrm{T}}, \mathbf{d}_{end}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{A}(T)\mathbf{c}, \tag{4}$$

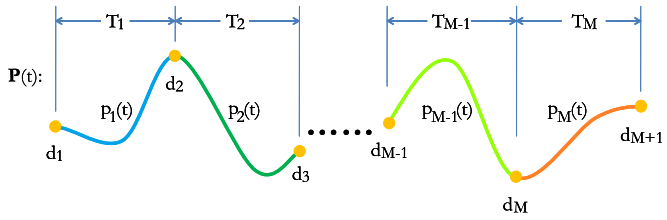


Fig. 2. A trajectory $\mathbf{P}(t)$ contains M pieces. Each piece is fully determined by its duration T_m and boundary condition $\mathbf{d}_m = (d_m^{\mathrm{T}}, d_{m+1}^{\mathrm{T}})^{\mathrm{T}}$.

where $\mathbf{A}(T) = (\mathbf{B}^{\mathrm{T}}(0), \mathbf{B}^{\mathrm{T}}(T))^{\mathrm{T}}$ is the mapping matrix with $2\lceil (N+1)/2 \rceil$ rows and $(N+1)$ columns. $\mathbf{A}(T)$ is a square matrix only if N is an odd number. Otherwise, $\mathbf{A}(T)$ becomes over-determined, which means a polynomial with $\mathbf{c}$ satisfying (4) may not exist for any given $(\mathbf{d}_{start}^{\mathrm{T}}, \mathbf{d}_{end}^{\mathrm{T}})^{\mathrm{T}}$.

Moreover, the inverse $\mathbf{A}(T)^{-1}$ exists and can be obtained with zero overhead when N is an odd number. Burri *et al.* [3] explore the structure of $\mathbf{A}(T)$ and find that $\mathbf{A}(T)^{-1}$ can be computed more efficiently through its Schur-Complement, which only involves submatrix inverse. We take things one step further. Actually, all entries of $\mathbf{A}(T)^{-1}$ are power functions of T, thus Gaussian-Elimination is applied to get its analytic form. Consequently, time-consuming operation is no longer needed when $\mathbf{A}(T)^{-1}$ is computed online. To achieve this, we pre-compute matrices $\mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{S\times S}$ offline, where $S = (N+1)/2$:

$$\mathbf{E}_{ij} = \begin{cases} \prod_{k=1}^{i-1} k & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases},$$

$$\mathbf{F}_{ij} = \begin{cases} \prod_{k=j-i+1}^{j-1} k & \text{if } i \leq j, \\ 0 & \text{if } i > j. \end{cases}$$

$$\mathbf{G}_{ij} = \prod_{k=S-i+j+1}^{S+j-1} k,$$

$$\mathbf{U}_{ij} = \begin{cases} 1/\prod_{k=1}^{i-1} k & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

$$\mathbf{W} = \mathbf{G}^{-1}, \mathbf{V} = -\mathbf{WFU}.$$

Following mapping matrices are computed online:

$$\mathbf{A}(T) = \begin{pmatrix} \mathbf{E} & \mathbf{0} \\ \left(\mathbf{F}_{ij} T^{j-i}\right)_{S\times S} & \left(\mathbf{G}_{ij} T^{S-i+j}\right)_{S\times S} \end{pmatrix},$$

$$\mathbf{A}(T)^{-1} = \begin{pmatrix} \mathbf{U} & \mathbf{0} \\ \left(\mathbf{V}_{ij} T^{j-i-S}\right)_{S\times S} & \left(\mathbf{W}_{ij} T^{j-i-S}\right)_{S\times S} \end{pmatrix}.$$

Therefore, provided with an odd order, we show the practical equivalence between the tuple $(\mathbf{d}_{start}, \mathbf{d}_{end}, T)$ and $(\mathbf{c}, T)$ in the sense of polynomial representation.

Consequently, we consider an M-piece trajectory $\mathbf{P}$ parameterized by time allocation $\mathbf{T} = (T_1, T_2, \ldots, T_M)^{\mathrm{T}}$ as well as boundary conditions $\mathbf{D} = (d_1^{\mathrm{T}}, d_2^{\mathrm{T}}, \ldots, d_{M+1}^{\mathrm{T}})^{\mathrm{T}}$ of all pieces, as shown in Fig. 2. The trajectory is defined by

$$\mathbf{P}(t) := \mathbf{d}_m^{\mathrm{T}} \mathbf{A}(T_m)^{-\mathrm{T}} \beta(t - \sum_{i=1}^{m-1} T_i), \tag{5}$$

where t lies in the m-th piece and $\mathbf{d}_m = (d_m^{\mathrm{T}}, d_{m+1}^{\mathrm{T}})^{\mathrm{T}}$ is a boundary condition of the m-th piece. This definition implicitly involves $(N-1)/2$ order continuity at boundaries of each piece. Normally, some entries in $\mathbf{D}$ are fixed, such as waypoint positions from a high-level planner [2]. We split $\mathbf{D}$ into two parts, the fixed part $\mathbf{D}_F$ which is viewed as constant, and the free part $\mathbf{D}_P$ which is to be optimized. The whole trajectory can be fully determined by $\mathbf{P} = \mathbf{\Phi}(\mathbf{D}_P, \mathbf{T})$.

IV. Spatial-Temporal Trajectory Optimization: *Unconstrained Case*

In this section, we describe our method for jointly optimizing spatial and temporal parameters of a trajectory, for the *Unconstrained Case*, where no constraint is considered.

A. Optimization Objective

We use the time regularized quadratic cost over the whole trajectory, as the objective of the optimization:

$$J(\mathbf{P}) = \int_0^{\sum_{m=1}^{M} T_m} \left(\rho + \sum_{i=D_{min}}^{D_{max}} w_i \left\| \mathbf{P}^{(i)}(t) \right\|^2 \right) \mathrm{d}t, \tag{6}$$

where D_{min} and D_{max} are the lowest and the highest order of derivative to be penalized respectively, w_i is the weight of the i-order derivative and ρ is the weight of time regularization. The weight ρ adjusts the aggressiveness of the trajectory [3], which allows total duration varies adaptively. For now, we consider the unconstrained optimization:

$$\min_{\mathbf{D}_P, \mathbf{T}} J(\mathbf{D}_P, \mathbf{T}) \tag{7}$$

where free boundary conditions and durations are decision variables. $J(\mathbf{D}_P, \mathbf{T}) := J(\mathbf{\Phi}(\mathbf{D}_P, \mathbf{T}))$ is used for brevity.

The cost J_m for the m-th piece can be calculated as

$$J_m = \rho T_m + \mathrm{Tr}\left\{ \mathbf{d}_m^{\mathrm{T}} \mathbf{A}(T_m)^{-\mathrm{T}} \mathbf{Q}(T_m) \mathbf{A}(T_m)^{-1} \mathbf{d}_m \right\}, \tag{8}$$

in which $\mathbf{Q}(T_m)$ is a symmetric matrix [10] consisting of high powers of T_m, and $\mathrm{Tr}\{\cdot\}$ is trace operation which only sums up diagonal costs produced in three dimensions. The overall objective can be formulated as

$$J = \rho \left\| \mathbf{T} \right\|_1 + \mathrm{Tr}\left\{ \begin{pmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{pmatrix}^{\mathrm{T}} \mathbf{C}^{\mathrm{T}} \mathbf{H}(\mathbf{T}) \mathbf{C} \begin{pmatrix} \mathbf{D}_F \\ \mathbf{D}_P \end{pmatrix} \right\}, \tag{9}$$

$$\mathbf{H}(\mathbf{T}) = \bigoplus_{m=1}^{M} \mathbf{A}(T_m)^{-\mathrm{T}} \mathbf{Q}(T_m) \mathbf{A}(T_m)^{-1}, \tag{10}$$

where $\mathbf{H}(\mathbf{T})$ is the direct sum of its M diagonal blocks, and $\mathbf{C}$ is a permutation matrix. We make sure that the setting for J is legal by assuming that the α-sublevel set of $J(\mathbf{D}_P, \mathbf{T})$ for any finite α is bounded and only consists of positive time allocation. For example, consecutive repeating waypoints with identical boundary conditions fixed in $\mathbf{D}_F$ are not allowed.

Algorithm 1: Unconstrained Spatial-Temporal AM.

Input: $\mathbf{D}_P^0, K \in \mathbb{Z}_+, \delta > 0$
Output: $\mathbf{D}_P^*, \mathbf{T}^*$
begin
  $\mathbf{T}^0 \leftarrow \arg\min_{\mathbf{T}} J(\mathbf{D}_P^0, \mathbf{T})$;
  $J_l \leftarrow J(\mathbf{D}_P^0, \mathbf{T}^0), k \leftarrow 0$;
  while $k < K$ **do**
    $\mathbf{D}_P^{k+1} \leftarrow \arg\min_{\mathbf{D}_P} J(\mathbf{D}_P, \mathbf{T}^k)$;
    $\mathbf{T}^{k+1} \leftarrow \arg\min_{\mathbf{T}} J(\mathbf{D}_P^{k+1}, \mathbf{T})$;
    $J_c \leftarrow J(\mathbf{D}_P^{k+1}, \mathbf{T}^{k+1})$;
    if $|J_l - J_c| < \delta$ **then**
      break
    $J_l \leftarrow J_c, k \leftarrow k+1$;
  $\mathbf{D}_P^* \leftarrow \mathbf{D}_P^k, \mathbf{T}^* \leftarrow \mathbf{T}^k$;
  return $\mathbf{D}_P^*, \mathbf{T}^*$;

B. Unconstrained Trajectory Optimization

To optimize (7), we propose an alternating minimization procedure. The basic idea of AM is to divide decision variables into groups then successively update each group with the others fixed. AM is quite efficient when the objective structure can be exploited so that updating by groups is a much cheaper operation than updating jointly.

The procedure is shown in Algorithm 1. Initially, $\mathbf{T}^0$ is solved for the provided $\mathbf{D}_P^0$. After that, the minimization of the objective function is done through a two-phase process, in which only one of $\mathbf{D}_P$ and $\mathbf{T}$ is optimized while the other is fixed.

In the first phase, the sub-problem

$$\mathbf{D}_P^*(\mathbf{T}) = \arg\min_{\mathbf{D}_P} J(\mathbf{D}_P, \mathbf{T}) \tag{11}$$

is solved for each $\mathbf{T}^k$. We employ the unconstrained QP formulation by Richter *et al.* [10], and briefly introduce it here. The matrix $\mathbf{R}(\mathbf{T}) = \mathbf{C}^{\mathrm{T}}\mathbf{H}(\mathbf{T})\mathbf{C}$ is partitioned as

$$\mathbf{R}(\mathbf{T}) = \begin{pmatrix} \mathbf{R}_{FF}(\mathbf{T}) & \mathbf{R}_{FP}(\mathbf{T}) \\ \mathbf{R}_{PF}(\mathbf{T}) & \mathbf{R}_{PP}(\mathbf{T}) \end{pmatrix}, \tag{12}$$

then the solution is be obtained analytically through

$$\mathbf{D}_P^*(\mathbf{T}) = -\mathbf{R}_{PP}(\mathbf{T})^{-1}\mathbf{R}_{FP}(\mathbf{T})\mathbf{D}_F. \tag{13}$$

For efficiency, we solve the sparse linear system

$$\mathbf{R}_{PP}(\mathbf{T})X = -\mathbf{R}_{FP}(\mathbf{T})\mathbf{D}_F \tag{14}$$

through Sparse LU Factorization to get $\mathbf{D}_P^*(\mathbf{T})$ since $\mathbf{H}(\mathbf{T})$ and $\mathbf{C}$ are both sparse.

In the second phase, the sub-problem

$$\mathbf{T}^*(\mathbf{D}_P) = \arg\min_{\mathbf{T}} J(\mathbf{D}_P, \mathbf{T}) \tag{15}$$

is solved for each $\mathbf{D}_P^k$. In this phase, the scale of sub-problem can be reduced into each piece. Due to our representation of trajectory, once $\mathbf{D}_P$ is fixed, the boundary conditions $\mathbf{D}$ isolate each entry in $\mathbf{T}$ from the others. Therefore, T_m can be optimized individually to get all entries of $\mathbf{T}^*(\mathbf{D}_P)$. As for the m-th piece, its cost J_m in (8) is indeed a rational function of T_m. We show the structure of J_m and omit the deduction for brevity:

$$J_m(T) = \rho T + \frac{1}{T^{p_n}} \sum_{i=0}^{p_d} \alpha_i T^i, \tag{16}$$

where $p_n = 2D_{max} - 1$ and $p_d = 2(D_{max} - D_{min}) + N - 1$ are orders of numerator and denominator, respectively. The coefficient α_i is determined by $\mathbf{d}_m$. Due to positiveness of $J_m(T)$, we have $J_m(T) \to +\infty$ as $T \to +\infty$ or $T \to 0^+$. Therefore, the minimizer exists for

$$T_m^*(\mathbf{D}_P) = \arg\min_{T \in (0,+\infty)} J_m(T). \tag{17}$$

To find all candidates, we compute the derivative of (16):

$$\frac{\mathrm{d}J_m(T)}{\mathrm{d}T} = \rho + \frac{1}{T^{1+p_n}} \sum_{i=0}^{p_d} (i - p_n)\alpha_i T^i. \tag{18}$$

The minimum exists in solutions of $\mathrm{d}J_m(T)/\mathrm{d}T = 0$, which can be calculated through any modern univariate polynomial real-roots solver [15]. In this letter, we utilize the Continued Fraction method [16] to isolate all positive roots of any high order (≥ 5) polynomial. The second phase is completed by updating every entry $T_m^*(\mathbf{D}_P)$ in $\mathbf{T}^*(\mathbf{D}_P)$.

C. Convergence Analysis

Algorithm 1 is globally convergent. Moreover, it is faster than conventional gradient descent used in time allocation refinement, under no assumption on convexity.

Theorem 1: Consider the process in Algorithm 1. For any $\mathbf{D}_P^0$, we have $\lim_{K\to\infty} \nabla J(\mathbf{D}_P^K, \mathbf{T}^K) = \mathbf{0}$. For all $K > 0$,

$$\min_{0 \leq k \leq K} \|\nabla J(\mathbf{D}_P^k, \mathbf{T}^k)\|_F^2 \leq M_c \frac{J(\mathbf{D}_P^0, \mathbf{T}^0) - J_c}{K},$$

where M_c and J_c are constants, $\|\cdot\|_F$ is Frobenius norm.

Proof: See [17] for details. ■

Thm. 1 shows that our algorithm shares the same global convergence rate as that of gradient descent with best step-size [18]. The best step-size is practically unavailable. In contrast, our algorithm does not involve any step-size choosing in its iterations. Sub-problems in Eq. (11) and Eq. (15) both are solved exactly and efficiently due to their algebraic convenience. The monotone decrease of objective function shows guaranteed progress in each iteration, while gradient-based methods may try bad step-size, thus making no/negative progress. Therefore, Algorithm 1 is faster than gradient-based methods in practice.

Another key advantage of our algorithm is its capability of escaping from a local minimum in the time optimization. Watching Eq. (9), despite $J(\mathbf{D}_P, \mathbf{T})$ is convex in $\mathbf{D}_P$ as proved in [17], it is a rational function which can have multiple local minima in T_m. Therefore, a case may occur where the initial time allocation falls into one of these local minima instead of the global minimum in $(0, +\infty)$. Under this situation, naturally, the global minimum in time allocation cannot be attained by gradient-based methods. However, with our method, all local minima are compared directly. Thus, the situation can be avoided.

It is worth noting that, here the global optimality is not guaranteed because our algorithm still exploits local structures

of the problem. Although convergence to stationary point is ensured, we argue that strict saddle points are theoretically and numerically unstable for our first-order AM method [19]. Moreover, when the stationary point is a strict local minimum, we show that the convergence rate is faster.

Theorem 2: Let $(\widehat{\mathbf{D}}_P, \widehat{\mathbf{T}})$ denote any strict local minimum of $J(\mathbf{D}_P, \mathbf{T})$ to which Algorithm 1 converges. There exist $K_c \in \mathbb{Z}_+$ and $\gamma \in \mathbb{R}_+$, such that

$$J(\mathbf{D}_P^K, \mathbf{T}^K) - J^* \leq \frac{1}{\gamma(K - K_c) + (J(\mathbf{D}_P^{K_c}, \mathbf{T}^{K_c}) - J^*)^{-1}},$$

for all $K \geq K_c$, where $J^* = J(\widehat{\mathbf{D}}_P, \widehat{\mathbf{T}})$.

Proof: See [17] for details. ■

Thm. 2 shows that a faster convergence rate $O(1/K)$ can be attained for a strict local minimum than the general case in Thm. 1. Note that it is possible to accelerate our method to attain the optimal rate $O(1/K^2)$ of first-order methods [18] or use second-order methods to achieve better performance. However, we still employ the first-order AM process because of its simplicity in implementation and its good performance when the trajectory is far from optimum.

V. Spatial-Temporal Trajectory Optimization: *Constrained Case*

In this section, we present our method to incorporate safety and dynamical feasibility constraints into our optimization process. To begin with, we introduce a computationally efficient feasibility check method that applies to a wide range of constraints. Then this method is used in a constrained trajectory optimization process.

A. Computationally Efficient Feasibility Check

A trajectory piece

$$p(t) = (p_1(t), p_2(t), p_3(t))^{\mathrm{T}} \tag{19}$$

is parameterized as (1). For any given $\mathbf{c}$ and T, it may be of interest whether a constraint $G(p_1^{(i)}(t), p_2^{(i)}(t), p_3^{(i)}(t)) < 0$ is satisfied by the corresponding $p(t)$ for all $t \in [0, T]$. We only consider the case that G is a multivariate polynomial:

$$G(a, b, c) := \sum_{e_1+e_2+e_3 \leq d_g}^{d_c \in \mathbb{R}, e_j \in \mathbb{N}} d_c \cdot a^{e_1} b^{e_2} c^{e_3}, \tag{20}$$

where d_g is the highest degree. Many kinds of constraints can be expressed by G, such as the safe distance constraint to keep away from an obstacle located at $(0, 0, 0)^{\mathrm{T}}$:

$$G_p(p_1(t), p_2(t), p_3(t)) < 0, \quad \forall t \in [0, T],$$
$$G_p(a, b, c) := r_{safe}^2 - (a^2 + b^2 + c^2),$$

or maximum speed constraint:

$$G_v(\dot{p}_1(t), \dot{p}_2(t), \dot{p}_3(t)) < 0, \ \forall t \in [0, T],$$
$$G_v(a, b, c) := a^2 + b^2 + c^2 - v_{max}^2.$$

Provided with any piece $p(t)$, we check whether constraint G is fulfilled for all $t \in [0, T]$. We define $\mathcal{G}(t) := G(p_1^{(i)}(t), p_2^{(i)}(t), p_3^{(i)}(t))$ which is indeed a polynomial of t. The procedure is as follows: Firstly, check the sign of $\mathcal{G}(0)$ and $\mathcal{G}(T)$. Then, If both two endpoints satisfy the constraint, we have to make sure the constraint is not violated inside the interval $(0, T)$. Instead of locating all extrema of $\mathcal{G}(t)$ and checking their values, we only need to check the existence of root of $\mathcal{G}(t) = 0$ in the interval. If the equation has any root in $(0, T)$, then $p(t)$ is infeasible. Fortunately, it is convenient for a polynomial to achieve this leveraging *Sturm's Theory* [20]. Now that neither 0 nor T is a root of $\mathcal{G}(t) = 0$, we compute the Sturm sequence $g_0(t), g_1(t), g_2(t), \ldots$ by

$$\begin{aligned} g_0(t) &= \mathcal{G}(t), \\ g_1(t) &= \dot{\mathcal{G}}(t), \\ -g_{k+1}(t) &= Rem(g_{k-1}(t), g_k(t)), \end{aligned} \tag{21}$$

where $Rem(g_{k-1}(t), g_k(t))$ is remainder in the Euclidean division of $g_{k-1}(t)$ by $g_k(t)$ [20]. When $g_k(t)$ becomes constant, we stop expanding this sequence. Let $V_{sign}(\tau)$ denote the number of sign variations of Sturm sequence at $t = \tau$, in which zero values should be ignored. Then the number of distinct roots inside $(0, T)$ equals $V_{sign}(0) - V_{sign}(T)$. Here the feasibility check is done for $G(\cdot, \cdot, \cdot) < 0$. Sometimes, a constraint has the form $G(\cdot, \cdot, \cdot) \leq 0$. In practice, it can be equally handled by checking $G(\cdot, \cdot, \cdot) < \epsilon$, where ϵ is a small positive real number. What's more, non-polynomial constraint can also be efficiently checked through its Taylor series within acceptable approximation error.

In conclusion, our method converts the feasibility check into the root existence check for polynomials, without computing the root itself. Compared with methods used in [3] and [21], ours is straightforward and involves no redundant operations such as numerical iteration or recursion.

B. Constrained Trajectory Optimization

For the *Constrained Case*, we enforce constraints on norms of derivatives of the trajectory:

$$\min_{\mathbf{D}_P, \mathbf{T}} J(\mathbf{D}_P, \mathbf{T}) \tag{22}$$

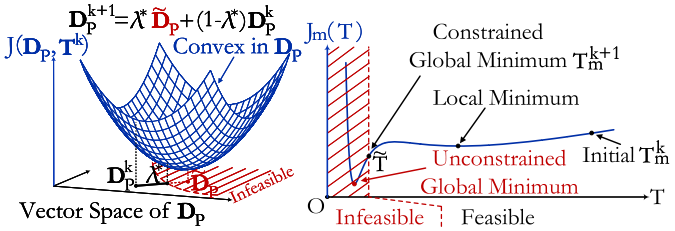
$$\text{s.t. } \|\mathbf{P}^{(n)}(t)\| \leq \sigma_n, \ 0 \leq t \leq \|\mathbf{T}\|_1 \tag{23}$$

$$\mathbf{P} = \mathbf{\Phi}(\mathbf{D}_P, \mathbf{T}), \ 1 \leq n \leq N \tag{24}$$

Generally, the constraint does not have to be like (23). If only a constraint is representable in (20) and its feasible solution can be constructed, then it can be handled in our optimization. With a slight abuse of notation, we use $\mathbf{G}(\mathbf{D}_P, \mathbf{T}) \leq 0$ to denote that $\mathbf{\Phi}(\mathbf{D}_P, \mathbf{T})$ is feasible. $\mathbf{G}(\mathbf{d}_m, T_m) \leq 0$ is used to denote that the m-th piece is feasible. We say $\mathbf{G}(\mathbf{d}_m, T_m) \leq 0$ is tight by meaning that, at least one constraint is tight at a t on the m-th piece.

The constrained version of our method is shown in Algorithm 2. An initial feasible trajectory $\mathbf{P}^0$ can be constructed from conservative time allocation. The spatial-temporal parameters $(\mathbf{D}_P^0, \mathbf{T}^0)$ are then recovered from the trajectory, which is used in the consequent two-phase constrained minimization.

In the first phase, $\mathbf{T}^k$ is fixed. An illustration is provided in Fig. 3(a), where the unconstrained minimum $\widetilde{\mathbf{D}}_P$ is obtained as is done in Algorithm 1. The trajectory $\mathbf{\Phi}(\widetilde{\mathbf{D}}_P, \mathbf{T}^k)$ may not be feasible. Since the feasibility of $(\mathbf{D}_P^k, \mathbf{T}^k)$ is ensured in the



(a) Update of Derivatives. (b) Update of time allocation.

Fig. 3. Illustration for the two phases in constrained optimization.

previous iteration, a line search is done as

$$\min_{\lambda\in[0,1]} J(\mathbf{D}(\lambda),\mathbf{T}^k),\ \text{s.t. } \mathbf{G}(\mathbf{D}(\lambda),\mathbf{T}^k)\le 0, \tag{25}$$

where $\mathbf{D}(\lambda)$ is the convex combination of $\widetilde{\mathbf{D}}_P$ and $\mathbf{D}_P^k$. Convexity of $J(\cdot,\mathbf{T}^k)$ implies the convexity of $J(\mathbf{D}(\cdot),\mathbf{T}^k)$. Moreover, $\lambda=0$ is a feasible solution, while $\lambda=1$ is the unconstrained global minimum. We simply take $\lambda^*=1$ if it is feasible. If not, a bisection procedure is done on the interval [0,1]. In this procedure, the feasibility check method described in Sec. V-A is employed to shrink the interval. The procedure stops at an acceptable interval length, with λ^* taking the feasible lower bound. After that, we update $\mathbf{D}_P^{k+1}$ by $\mathbf{D}(\lambda^*)$. Meanwhile, a set Δ is maintained to store indices of tightened pieces.

In the second phase, $\mathbf{D}_P^{k+1}$ is fixed. An illustration is given in Fig. 3(b). Each entry in $\mathbf{T}^k$ is updated by solving

$$\min_{T\in(0,+\infty)} J_m(T),\ \text{s.t. } \mathbf{G}(\mathbf{d}_m^{k+1},T)\le 0. \tag{26}$$

As stated in Algorithm 1, all extrema of $J_m(T)$ can be computed exactly. However, the constrained minimum may not exist in them. It can be any $\widetilde{T}$ at which some constraints are exactly tightened. When infeasible extremum exists, $\widetilde{T}$ must be located between any infeasible extremum and the neighboring feasible one or T_m^k. A bisection procedure with feasibility check suffices to compute $\widetilde{T}$. After that, we compare $J_m(T)$ on those feasible extrema together with $\widetilde{T}$.

When all iterations are done, the set Δ indicates pieces stuck by active constraints. If Δ is not empty, we recursively apply Algorithm 2 on split sub-trajectories, while boundary conditions of pieces indexed by Δ should be totally fixed. Finally, $\mathbf{P}^*$ is updated and returned. The recursive process is essential, since it ensures that pieces with no room for optimization do not prevent other pieces to decrease the objective. Algorithm 2 is globally convergent to a solution set where constraints are tight or local minimum is attained, which can be checked by Zangwill's theorem [22].

Algorithm 2: Constrained Spatial-Temporal AM.

Input: Feasible $\mathbf{P}^0$, $K\in\mathbb{Z}_+$, $\delta>0$
Output: $\mathbf{P}^*$
begin
  $(\mathbf{D}_P^0,\mathbf{T}^0)\leftarrow\mathbf{\Phi}^{-1}(\mathbf{P}^0)$;
  $J_l\leftarrow J(\mathbf{D}_P^0,\mathbf{T}^0)$;
  $k\leftarrow 0,\Delta\leftarrow\{\}$;
  while $k<K$ **do**
    $\widetilde{\mathbf{D}}_P\leftarrow\arg\min_{\mathbf{D}_P} J(\mathbf{D}_P,\mathbf{T}^k)$;
    $\mathbf{D}(\lambda):=\lambda\widetilde{\mathbf{D}}_P+(1-\lambda)\mathbf{D}_P^k$;
    $\lambda^*\leftarrow\arg\min_{\lambda\in[0,1]} J(\mathbf{D}(\lambda),\mathbf{T}^k)$
      $s.t.\ \mathbf{G}(\mathbf{D}(\lambda),\mathbf{T}^k)\le 0$;
    $\mathbf{D}_P^{k+1}=\mathbf{D}(\lambda^*)$;
    Recover T_m^k from $\mathbf{T}^k$ and $\mathbf{d}_m^{k+1}$ from $\mathbf{D}_P^{k+1}$;
    for $m\leftarrow 1$ **to** M **do**
      if $\mathbf{G}(\mathbf{d}_m^{k+1},T_m^k)\le 0$ *is tight* **then**
        $\Delta\leftarrow\Delta\cup\{m\}$;
      else
        $\Delta\leftarrow\Delta\setminus\{m\}$;
    for $m\leftarrow 1$ **to** M **do**
      Construct $J_m(T)$ from $\mathbf{d}_m^{k+1}$;
      $T_m^{k+1}\leftarrow\arg\min_{T\in(0,+\infty)} J_m(T)$
        $s.t.\ \mathbf{G}(\mathbf{d}_m^{k+1},T)\le 0$;
    Construct $\mathbf{T}^{k+1}$ from T_m^{k+1};
    $J_c\leftarrow J(\mathbf{D}_P^{k+1},\mathbf{T}^{k+1})$;
    if $|J_l-J_c|<\delta$ **then**
      break
    $J_l\leftarrow J_c,k\leftarrow k+1$;
  $\mathbf{P}^k\leftarrow\mathbf{\Phi}(\mathbf{D}_P^k,\mathbf{T}^k)$;
  if Δ *is not empty* **then**
    Split $\mathbf{P}^k$ by removing pieces in Δ;
    Call this **Algorithm** on sub-trajectories;
    Update sub-parts of $\mathbf{P}^k$;
  $\mathbf{P}^*\leftarrow\mathbf{P}^k$;
  return $\mathbf{P}^*$;

VI. Results

A. Comparison of Feasibility Check Methods

Firstly, we compare our feasibility check method with Mueller's recursive check [21], Burri's analytical check [3], as well as the widely used sampling-based check. Mueller's method uses the extremum in each axis as an upper bound of derivative norm, which has closed-form solutions for low-order (≤ 5) trajectories. The bound becomes tighter as the interval to be checked shrinks in recursion. Burri's method directly finds the point with maximal constraint violation by calculating roots of the corresponding polynomial.

In each case, 1000 trajectory pieces are randomly generated with velocity constraints to estimate time consumption. As is shown in Fig. 4, our method outperforms the others because of its resolution independence and scalability with higher polynomial orders. The recursive check and sampling-based check may have false positives under rough temporal resolution. The efficiency of analytical check and recursive check deteriorates with higher orders. In comparison, our method is able to do a solid check within 1 μs.

B. Benchmark for Trajectory Optimization Methods

Secondly, we compare our optimization method with some state-of-the-art waypoint-based methods, i.e., Mellinger's method [1] and Richter's method [2]. Mellinger's method optimizes time allocation with total duration fixed, using backtracking gradient descent (BGD). The dynamical feasibility is ensured

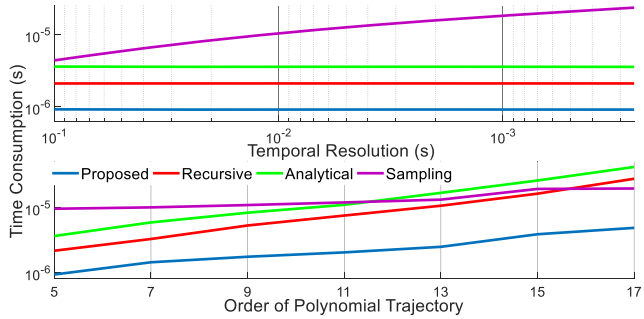
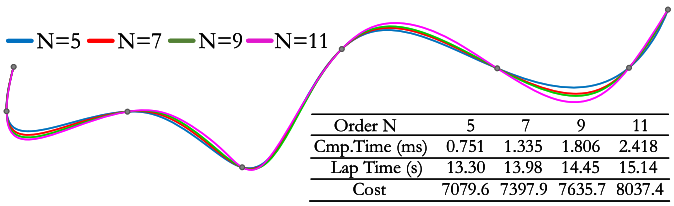
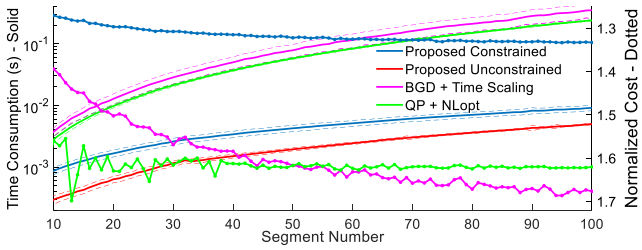


Fig. 4. Computation time for feasibility check of speed constraint, under different temporal resolution (upper) and different polynomial order (lower).

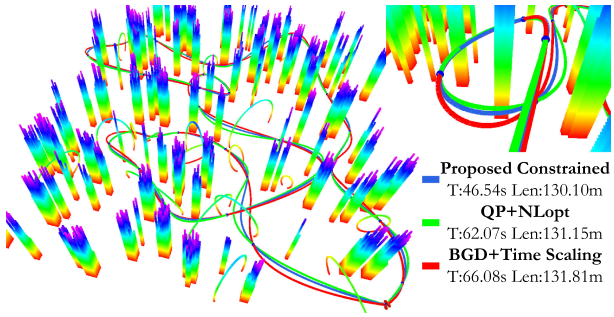


Order N	5	7	9	11
Cmp.Time (ms)	0.751	1.335	1.806	2.418
Lap Time (s)	13.30	13.98	14.45	15.14
Cost	7079.6	7397.9	7635.7	8037.4

Fig. 5. Time regularized minimum jerk trajectory optimization profiles for different polynomial orders in constrained case. Our algorithm works well with different orders. Normally, a large N ensures high-order continuity but increases computation time and trajectory cost.



(a) Benchmark in computation time (solid) and normalized cost (dotted).



(b) Trajectories in a random map.

Fig. 6. Comparisons between our method, BGD with Time Scaling [1] and QP with NLopt [2]. In Fig. 6(a), the performance of different methods are provided. Dashed lines indicate standard deviation. In Fig. 6(b), an intuitive comparison is given. More quantitative comparisons for lap time and trajectory length are included in the attached video.

by Liu's time scaling factor [6]. Richter's method optimizes derivatives on waypoints through an unconstrained QP while time allocation is adjusted by gradient descent and scaling. To use it in constrained case, we soften the constraints by penalizing them in objective function as suggested in [3] and optimize the time allocation through NLopt [23].

The benchmark is done as follows: Sequences of waypoints are generated by random walk with step uniformly distributed over $[-3.0$ m, 8.0 m$]$ for each axis. The maximum speed and acceleration rate are set to 5.0 m/s and 3.5 m/s^2, respectively. Derivatives on the first and last waypoints are set to zero. We set $N = 5$ hereafter because of its highest efficiency shown in Fig. 5. It also captures the most relevant quadrotor dynamics [21]. Moreover, we set $\rho = 512.0$, $D_{max} = D_{min} = 3$, $w_3 = 1.0$. Each method is applied to 1000 sequences of waypoints for a given piece number. The optimization stops until the relative decrease of objective is less than 0.001. The cost is then normalized by that of Algorithm 1. All comparisons are conducted on an Intel Core i7-8700 CPU under Linux environment.

Besides, a more intuitive comparison is also provided. We adopt the high-level waypoint planner proposed by Richter *et al.* [2]. Some compulsory waypoints are pre-chosen in a random map. After that, an optimal path is produced by RRT* [24], which starts from an initial position and passes those compulsory waypoints. All waypoints of the path is used to generate a trajectory. When collision occurs on a particular trajectory piece, the midpoint of its two endpoints is simply added as an additional waypoint. The trajectory is re-optimized, and the process is repeated until the whole trajectory is collision-free.

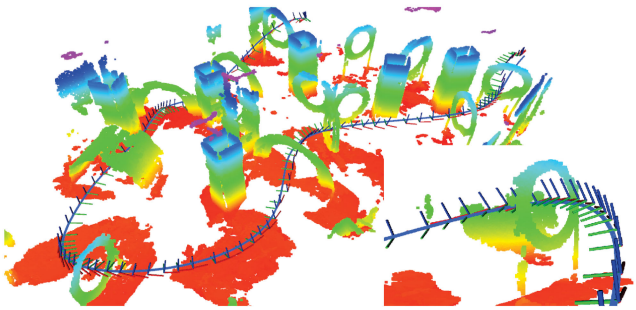
As is shown in Fig. 6, our Algorithm 2 has the fastest speed and the lowest cost when constraints are taken into consideration. Our method is capable of computing trajectories with 60 pieces within 5 ms, i.e., 150 Hz at least. However, both benchmarked methods fail to accomplish real-time computing for trajectories with more pieces. Moreover, our Algorithm 2 always obtains better trajectories in terms of the cost function, while benchmarked methods cannot fully utilize the capability of system dynamics. The main reason is that our method separately minimizes the cost on each piece as much as possible, especially when a relatively large ρ is used. For a small ρ, our method can still outperform the others in computation time while their final costs can be much the same.

C. Aggressive Flight Experiment

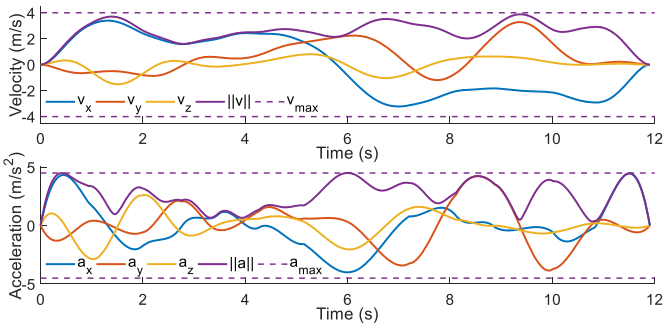
To validate the performance of our method in real-world applications, we deploy it on a self-developed compact quadrotor platform. The proposed method is implemented with C++11, and all tasks are conducted on an onboard computer with Intel Core i7-8550U CPU. The pose of our quadrotor is obtained through a robust visual-inertial state estimator [25]. Besides, no external positioning system nor offboard computing is used. A geometric controller is employed for trajectory tracking control [26].

The experiment is conducted in a complex indoor scene, which is shown in Fig. 1. A globally consistent map of the scene is pre-built, from which some compulsory waypoints are selected offline. An optimal path is generated by the aforementioned technique. Our method generates an optimal trajectory online based on the path within milliseconds. Immediately, the quadrotor starts its aggressive flight. Different from parameters used in benchmark, we set $v_{max} = 4.0$ m/s, $a_{max} = 4.5$ m/s^2 and $\rho = 1024.0$. The aggressive flight along with the generated trajectory is shown in Fig. 7. More details are included in the attached video.

(a) Aggressive flight experiment.



(b) Trajectory of the experiment.



(c) Velocity and acceleration against time.

Fig. 7. Details of our aggressive indoor flight. Fig. 7(a) shows some snapshots of our experiment. In Fig. 7(b), the whole trajectory is visualized in the pre-built map, whose speed is up to $4.0m/s$. In Fig. 7(c), velocity/acceleration profiles are provided. The trajectory fully employs capability of the quadrotor in terms of maximum velocity/acceleration rate.

VII. Conclusion

In this letter, we propose an efficient trajectory generation method for quadrotor aggressive flight, which has guaranteed convergence and feasibility. Benchmarks for components in our method show its superior computation speed, trajectory quality as well as scalability against state-of-the-art methods. Aggressive flight experiments in limited space with dense obstacles validate the practical performance of our method. Currently, in the proposed framework, positions of waypoints are fixed during optimization. However, the method is underlying compatible with waypoints as part of decision variables. Our feasibility checker also supports various safety constraints. In the future, we plan to apply and improve our method in time-critical large-scale motion planning scenarios where complex spatial constraints exist.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.*, Shanghai, China, May 2011, pp. 2520–2525.

[2] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Int. Symp. Robot. Res.*, Singapore, Dec. 2013, pp. 649–666.

[3] M. Burri, H. Oleynikova, M. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Hamburg, Germany, Sep. 2015, pp. 1872–1878.

[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA, USA: Athena-Scientific, 1995.

[5] A. Beck, *First-Order Methods in Optimization*. Philadelphia, PA, USA: SIAM, 2017.

[6] S. Liu *et al.*, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1688–1695, Jul. 2017.

[7] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *Proc. IEEE Int. Conf. Robot. Autom.*, Brisbane, Australia, May 2018, pp. 344–351.

[8] L. Campos-Macías, D. Gómez-Gutiérrez, R. Aldana-López, R. de la Guardia, and J. I. Parra-Vilchis, "A hybrid method for online trajectory planning of mobile robots in cluttered environments," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 935–942, Apr. 2017.

[9] W. Sun, G. Tang, and K. Hauser, "Fast UAV trajectory optimization using bilevel optimization with analytical gradients," 2018, *arXiv: 1811.10753*.

[10] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *Int. J. Robot. Res.*, vol. 34, pp. 969–1002, 2015.

[11] F. Gao *et al.*, "Optimal trajectory generation for quadrotor teach-and-repeat," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1493–1500, Apr. 2019.

[12] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Trans. Autom. Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.

[13] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Trans. Robot.*, to be published, doi: 10.1109/TRO.2020.2993215.

[14] M. M. de Almeida, R. Moghe, and M. R. Akella, "Real-time minimum snap trajectory generation for quadcopters: Algorithm speed-up through machine learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, Montreal, Canada, May 2019, pp. 683–689.

[15] M. Sagraloff and K. Mehlhorn, "Computing real roots of real polynomials," *J. Symbolic Comput.*, vol. 73, pp. 46–86, 2013.

[16] E. P. Tsigaridas and I. Z. Emiris, "Univariate polynomial real root isolation: Continued fractions revisited," in *Proc. Conf. Annu. Eur. Symp.*, Zwitserland, Zurich, Sep. 2006, pp. 817–828.

[17] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Detailed proofs of alternating minimization based trajectory generation for quadrotor aggressive flight," 2020. [Online]. Available: https://arxiv.org/abs/2002.09254

[18] Y. Nesterov, *Lectures on Convex Optimization*. Berlin, Germany: Springer, 2018.

[19] J. D. Lee, I. Panageas, G. Piliouras, M. Simchowitz, M. I. Jordan, and B. Recht, "First-order methods almost always avoid strict saddle points," *Math. Programm.*, vol. 176, pp. 311–337, 2019.

[20] S. Basu, R. Pollack, and M.-F. Roy, *Algorithms in Real Algebraic Geometry*. Berlin, Germany: Springer, 2003.

[21] M. W. Mueller, M. Hehn, and R. DAndrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transact. Robot.*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.

[22] W. I. Zangwill, *Nonlinear programming: A Unified Approach*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1969.

[23] S. G. Johnson, "The NLopt nonlinear-optimization package," [Online]. Available: http://github.com/stevengj/nlopt

[24] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, pp. 846–894, 2011.

[25] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.

[26] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE Control Decis. Conf.*, Atlanta, Georgia, USA, Dec. 2010, pp. 5420–5425.