Autonomous Tail-Sitter Flights in Unknown Environments

Guozheng Lu*[1], Yunfan Ren*[1], Fangcheng Zhu[1], Haotian Li[1], Ruize Xue[1], Yixi Cai[1], Ximin Lyu[2] and Fu Zhang[1]

***Abstract*—Trajectory generation for fully autonomous flights of tail-sitter unmanned aerial vehicles (UAVs) presents substantial challenges due to their highly nonlinear aerodynamics. In this paper, we introduce, to our best knowledge, the world's first fully autonomous tail-sitter UAV capable of high-speed navigation in unknown cluttered environments. The UAV autonomy is enabled by cutting-edge technologies including LiDAR-based sensing, differential-flatness-based trajectory planning and control with purely onboard computation. In particular, we develop a tail-sitter trajectory planning framework that generates high-speed, collision-free and dynamically-feasible trajectories. To efficiently and reliably solve this nonlinear constrained trajectory optimization, we design an efficient feasibility-first solver, EFOPT, tailored for constrained nonlinear programmings (NLPs). We conducted extensive simulation studies to benchmark EFOPT's superiority in planning tasks against conventional NLP solvers. We also demonstrate exhaustive experiments of aggressive autonomous flights with speed up to $15\,\mathrm{m/s}$ in various real-world environments , including indoor laboratories, underground parking lots and outdoor parks. A video demonstration is available at https://youtu.be/OvqhlB2h3k8, and the EFOPT solver is open-sourced at https://github.com/hku-mars/EFOPT.**

Fig. 1: Autonomous tail-sitter UAV equipped with a LiDAR, driven by real-time onboard perception, planning and control.

I. Introduction

Benefiting from the rapid advances in both electronic devices and control autonomy, small-scale autonomous unmanned aerial vehicles (UAVs) have seen a significant surge over the past decade. Autonomous UAVs are capable of performing tasks independently without human intervention, in both natural and man-made environments. They hold enormous potential for civil and industrial applications, leading to substantial socio-economic impacts. In recent years, autonomous UAVs in conventional fixed-wing and rotary-wing configuration have been pervasively used in a broad range of applications, such as geographical surveys, agriculture monitoring and spraying, and tunnel inspections. However, these two platforms have inherent limitations that restrict their practical use. Fixed-wing airplanes excel in aerodynamic efficiency and fast flight in open spaces, but lack hovering capability and typically rely on runways for takeoff and landing. Conversely, rotary-wing vehicles like multicopters, demonstrate remarkable agility in hovering, vertical takeoff and landing (VTOL), and executing aggressive maneuvers in confined environments. However, they are generally less energy-efficient, making them less ideal for long-range missions.

The hybrid design, combing fixed and rotary-wing elements, offers a complementary solution that boasts both the VTOL capability and aerodynamic efficiency in high-speed flights. These hybrid VTOL UAVs are ideally suited for tasks in diverse and complex environments, such as forest search and rescue, urban and rural infrastructure inspection, and inter-city delivery. The potential profitability of these applications has spurred a variety of VTOL UAV designs, including tilt-rotors [1], [2], tilt-wings [3], rotor-wings [4], dual-systems [5], [6] and tail-sitters [7]–[10]. Among these, tail-sitters have rotors fixed to the wings and utilize their thrust throughout the entire flight regime, eliminating the need of extra tilting mechanisms or redundant propulsion. This feature of mechanical simplicity is especially crucial for small-scale, light-weight, low-cost, portable, and efficient UAVs.

Nevertheless, although the concept of tail-sitters has been around for decades, they are rarely utilized in practical applications compared to other VTOL UAVs, primarily due to control difficulties. Typically, a tail-sitter takes off and lands vertically on its tail, and transitions to a nearly horizontal attitude for forward flight. This transition results in significant variations of aerodynamic forces and moments acting on the vehicle, since they are coupled with the vehicle attitude and airflow. Controlling this type of vehicle has long been a prominent topic in the nonlinear control community, yet it remains an open question. Furthermore, autonomous flights in real-world environments also demand the planning of high-quality trajectories that ensure smoothness, time efficiency, dynamical feasibility, and obstacle avoidance, which presents a considerable challenge for such a nonlinear system.

In the context of autonomous robotic motion planning, especially in real-world environments, safety must always be the first priority. Otherwise, inappropriate trajectories involving actuator saturation or collision with obstacles may lead to

*These two authors contribute equally to this work..

[1]The authors are with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong. {gzlu, renyf, zhufc, haotianl, u3603390, yicicai}@connect.hku.hk, fuzhang@hku.hk. [2] X. Lyu is with School of Intelligent System Engineering, Sun Yat-sen University, Shenzhen, China. lvxm6@mail.sysu.edu.cn.

undesired losses. While trajectory optimization can incorporate with safety constraints to achieve aggressive maneuvers in cluttered environments, it poses a grand challenge for mathematical tools to solve such problems. When dealing with non-convex optimization, off-the-shelf nonlinear programming (NLP) solvers developed on various methods may exhibit varying performance and yield different results. This variability is primarily attributed to how these solvers prioritize optimality and feasibility at different levels. Moreover, the solution efficiency is also essential for online planning in autonomous flights. However, the complex system dynamics of tail-sitters significantly increase the non-convexity and non-smoothness of the optimization, making it more challenging to compute an efficient and feasible solution.

In light of the nonlinear system dynamics and the limitation of off-the-shelf NLP solvers, there has been no tail-sitter UAV capable of performing autonomous flights in unknown environments with obstacle avoidance using only onboard sensors and computation to date.

A. Contributions

In this paper, we present a fully autonomous tail-sitter UAV, as shown in Fig. 1, which is able to fly through unknown, obstacle-dense environments at high speeds, based on a novel framework for real-time trajectory planning. The key contributions of this paper are as follows:

1. We introduce a prototype of autonomous tail-sitter UAV equipped with a LiDAR sensor, and detail its technical pipeline for onboard perception, planning and control.
2. We present an optimization-based trajectory planning framework that generates high-speed, collision-free and dynamically-feasible trajectories specifically for tail-sitters.
3. To address real-time trajectory optimization, we develop an **E**fficient **F**easibility-first **OPT**imization solver, **EFOPT**, which is designed to rapidly find feasible solutions, suitable for autonomous tailsitter flights. We showcase the benchmark performance of our proposed EFOPT against standard off-the-shelf NLP solvers in trajectory optimization problems through extensive simulations.
4. We achieve, to the best of our knowledge, the first tail-sitter with the capability of obstacle avoidance relying on solely onboard sensors and computation. We demonstrate the whole system at high-speed flights in various real-world obstacle-dense environments.

B. Outline

The outline of the rest of the paper is as follows. Section II reviews the related work. Preliminaries of tail-sitter dynamics and differential flatness are first introduced in Section III. The system overview is then presented to depict the UAV platform and entire algorithm framework in Section IV. Section V and VI respectively present the trajectory optimization formulation and solution, which are keys to solve the current bottleneck of tail-sitter autonomous flights. Section VII benchmarks the proposed solver with off-the-shelf NLP solvers in the context of trajectory optimization, followed by real-world experiment results in Section VIII. Limitations and possible extensions of the proposed framework are discussed in Section IX. Finally, conclusion of the paper arrives at Section X.

II. Related Works

A. Tail-sitter planning and control

As a classic nonlinear system, planning and control of tail-sitters have been extensively investigated for decades. These approaches can be generally categorized into two main strategies: the separated control strategy, which involves isolated trajectory generation and controllers designed for specific flight phases, and the global control strategy, which regulates the vehicle maneuvers throughout the entire flight envelope under a unified planning and control framework.

The separated control strategy [11]–[13] usually divides the tail-sitter flights into three phases: vertical flight (including takeoff, landing, and hovering), transition, and level flight, by modeling the system dynamics separately at each phase. In low-speed vertical flights, the dynamics reduce to a rotary-wing model, which can be linearized at the stationary hovering equilibrium and controlled using established multicopter methods, such as loop-shaping [14], robust control [15] and model predictive control (MPC) [16]. Given that the aerodynamics are negligible at low speeds, trajectory generation for multicopters [17] are applicable. Similarly, the tail-sitter dynamics in high-speed level flights are equivalent to a fixed-wing model, allowing the direct application of conventional fixed-wing flight control and planning techniques, such as total energy control system (TECS) [18] and L1 guidance [19].

The transition maneuver between hovering and level flights poses a key challenge for the separate control strategy due to the complicated tail-sitter aerodynamics. During transition, both the angle of attack (AoA) and airspeed undergo dramatic changes, rendering the aerodynamics highly nonlinear. A common linear transition method is to feed a pre-designed profile of linearly decreasing or increasing pitch angle to the attitude controller with a constant altitude command [20], [21], forcing the vehicle to pitch down or up until a mode-switching condition is triggered. However, this linear transition trajectory is not always dynamically feasible and often leads to undesired control errors. Desirable control performance usually necessitates gain-scheduling techniques [22], [23] to enhance the stability margin, and sophisticated controller like iterative learning control [24] to decrease the altitude deviation during transition. Improving control performance can also involve designing dynamically feasible transition trajectories. For example, studies like [22], [25]–[27] formulated the trajectory generation as nonlinear optimization problems with control constraints, to effectively reduce the altitude deviation or transition duration. However, these methods are limited to offline trajectory generation and straight-line transition, thus falling short in accommodating more flexible maneuvers such as bank turnings during transition to avoid obstacles.

The separate control strategy is user-friendly for validating tail-sitter prototypes, particularly when utilizing commercial autopilots [28]. However, its shortcomings including transition without obstacle avoidance and separated controllers'

switching to cause undesired transient response, prevent tail-sitters from agile maneuvers that are required in wide scope of applications. Consequently, a global strategy that exploits a unified dynamic model for both trajectory generation and controller design without any mode switching, appears more promising for practical uses.

Global control strategy can be further categorized into aerodynamic model-free and model-based methods. Considering aerodynamics as external disturbance, model-free methods [29], [30] approximate the vehicle dynamics locally based on quasi-static assumptions, which allows the use of linear theory to stabilize the system. It is a common approach for nonlinear system control, but it is not well-suited for aggressive tail-sitter flights, where increased control inputs and aerodynamic nonlinearity can easily exceed the effective range of local linearization. Another limitation is that the aerodynamics are estimated from the applied inputs and inertial measurement units (IMUs), which are prone to either large measurement noise caused by constant propeller rotation or considerable filter delay; Although this approach circumvents the need for obtaining aerodynamics models, control performance often remains insufficient, as evidenced by large altitude errors during transition in existing works [29], [30].

In comparison, model-based global methods that exploit aerodynamics show promise in control performance, particularly for aggressive flights. There are existing works attempted to develop global controllers using coarse aerodynamic models, such as 2-dimension planner VTOL [31], first-principle classic model [8], and polynomial-parameterized ϕ-theory model [32]. Although the use of simplified aerodynamic models facilitates controller design, tracking performance cannot be always guaranteed as shown in their demonstrations. To compensate for the low-fidelity models, Smeur et al. [33] proposed a global incremental nonlinear dynamic inversion (INDI) controller based on a heuristic aerodynamic model with an assumption of zero flight path angle. Later, Tal et al. [34] introduced a differential flatness transform based on the ϕ-theory model and a zero side force assumption. They developed a global trajectory generation and INDI-based control framework [35] suitable for both coordinated and uncoordinated flight on tail-sitters without vertical surfaces. However, INDI, being an adaptive control technique, estimates and compensates uncertainties, such as model deficiency and wind effect, using IMU data. In consequence, its control performance heavily relies on sensor noise, filter delay, model fidelity and external winds.

To further improve the control performance, researchers have investigated global control methods that fully exploit the UAV actual aerodynamic model, not simplified ones as in [8], [31]–[34]. Zhou et al. [36] proposed a unified control method based on accurate aerodynamic model obtained from wind tunnel test. However, this method requires the attitude command to be computed from the desired acceleration by solving an extremely non-convex optimization problem, which is computationally demanding and hinders real-time implementation. In our recent work [37], we have theoretically proven the tail-sitter differential flatness in coordinated flights using a general classic aerodynamic model without simplification. This discovery significantly streamlines trajectory generation and enables real-time, high-performance control for extremely aggressive aerobatic maneuvers even in outdoor windy environments.

Although the differential flatness property has significantly simplified the tail-sitter trajectory generation problem by reducing the planning in state space to that in flat-output space, the remaining planning problem is still very challenging for real-time computation. As a consequence, the aforementioned Tal's approach in [35] proposed a minimum-snap trajectory generation followed with a separated time allocation optimization to meet dynamic limits, which is computationally intensive and only suitable for offline use. Similarly, our previous work [37] aimed to minimize flight time and control efforts subject to actuator constraints within a single optimization process. However, solving such a non-convex constrained optimization cannot be accomplished in real time. To date, no existing works have proven capable of real-time trajectory planning for aggressive tail-sitter flights.

B. Autonomous flights of fixed-wing and multicopter UAVs

Compared to tail-sitters, planning and control techniques for fixed-wing and multicopter UAVs are well-developed, owing to their relatively simpler aerodynamics. Consequently, a wealth of autonomy techniques have been developed for these UAVs, and are potentially adaptable to tail-sitters.

Fixed-wing airplanes typically operate within a conservative level flight regime, where the wing aerodynamic forces are approximately linear. These airplanes are generally considered in the coordinated flight, a condition where they ideally achieve the maximum aerodynamic efficiency and experience fewer undesirable aerodynamic moment that could cause spins. This coordinated flight condition further simplifies the aerodynamic models with only longitudinal aerodynamic coefficients around zero sideslip angle. Established control methods, such as linear quadratic regulator (LQR) [38] and cascaded proportional-integral-derivative (PID) controller (e.g. TECS [18]) have widespread use in practice. In most case, fixed-wing UAVs operating in open space at high altitude simplify trajectory generation to basic kinematic path planning, connecting waypoints with straight lines and arcs of constant curvature (e.g. L1 guidance [19]). More sophisticated motion planning for aggressive maneuvers that takes dynamical feasibility into consideration, has also been developed by exploiting the differential flatness property [39].

Multicopters have distinctive maneuverability that makes them suitable for various applications operating in confined spaces and close proximity to people. Leveraging the differential flatness property [17], [40], trajectory generation and control for multicopters have been comprehensively studied for specific tasks, such as flying through narrow gaps [17], [41], [42], perching on structures [17], [43], [44], catching a thrown ball [45] and drone racing [46]. Different from flying in laboratory, autonomous flights in unstructured, obstacle-rich environments poses grand challenges to online perception and planning algorithms based on exteroceptive sensors like cameras and LiDAR. Cameras provides informative image

features enabling visual-inertial odometry [47], [48], simultaneous localization and mapping (SLAM) [49]. These techniques are essential to robotic autonomy, but their performance significantly degrades in case of fast movement (e.g., speed over $5\,\mathrm{m/s}$) due to motion blur and changing illumination. In contrast, LiDAR sensors provide dense, accurate and low-latency 3-D point clouds that are more robust for high speed autonomous robots, such as fixed-wing [39] and tail-sitter UAVs. Recent development on real-time LiDAR-based odometry and SLAM [50] has enabled multicopters to navigate much faster in unknown environments [51], [52].

In terms of online trajectory planning for multicopters, existing works can be categorized into four main approaches: search-based, sampling-based, optimization-based and learning-based methods. Search-based methods, such as Dijkstra's [53] and A* [54] algorithms, which search over a graph representing feasible action primitives, have shown promising results in autonomous flights [55]–[57]. However, as the state dimension increases, the computational expense of graph search escalates due to the exponential growth in node numbers. Thus, existing online planning typically considers kinematic primitives rather than true actuator actions. Sampling-based planning algorithms like Probabilistic Roadmap (PRM) [58], Rapidly- exploring Random Tree (RRT) [59] and their variants PRM*, RRT* [60], spread sparse random samples on the graph and connect selected ones to seek collision-free path in configuration space, showing exceptional scalability for large maps. However, in cluttered environments, the computational efficiency of these methods is not guaranteed due to the dependence on map complexity. Optimization-based methods focus on planning trajectory by solving optimal control problems. Instead of directly optimizing the control sequence using shooting [61] or collocation methods [62], real-time planners usually leverage differential flatness properties and efficient trajectory parameterizations, such as polynomials [17], B-splines [63] and Bézier curves [64], to enhance computational efficiency. Obstacle information is usually encoded as constrained terms in the optimization, such as safe flight corridors [64] and Euclidean signed distance field [65]. Recently, machine learning has also demonstrated exceptional results in training end-to-end control policies directly from flight data [66], but it usually requires high-fidelity simulators and techniques for simulation-to-real transfer.

In this paper, we focus on developing optimization-based trajectory planning for tail-sitter autonomous flights. Most online trajectory optimizations for multicopters design minimum-snap trajectories with kinodynamic constraints, giving up accurate vehicle dynamics for computational efficiency. With a predetermined time allocation, minimum-snap trajectory polynomials can be easily formulated into convex optimization, such as quadratic programming (QP) [17], second-order cone programming (SOCP) [64] and mixed-integer quadratic programming (MIQP) [67], leading to efficient solutions. Since heuristic time allocations may lead to odd polynomials, temporal optimization can effectively improve the trajectory quality, at the cost of creating non-convex optimizations. For the sake of efficiency, methods like gradient-descent iteration [68] and spatial-temporal alternative optimization [64] have been applied in practice. Wang et al. [52] proposed a novel polynomial parameterization enabling simultaneous optimization on spatial and temporal parameters, but this results in a nonlinear programming (NLP) problem. They solved the trajectory optimization efficiently using a quasi-Newton method, i.e., LBFGS [69], by treating constraints as soft penalties. However, while tail-sitter is also a differentially flat system like multicopter, its highly nonlienar aerodynamics and aggressive maneuvers necessitate the inclusion of accurate actuator constraints rather than just kinodynamic constraints in trajectory optimization. The resulting optimization is exceedingly complex, and the LBFGS solution can be time-consuming and infeasible (i.e., constraint violations) [37], proving insufficient for online planning.

C. Trajectory optimization methods

As discussed, trajectory optimization at kinematic level, which omits actual system dynamics, can be formulated into convex optimization problems, such as QP, SOCP and MIQP, leading to efficient solutions. Once time allocation and nonlinear system dynamics are involved, the problem becomes non-convex, necessitating advanced NLP methods.

The aforementioned soft penalty method has been a practical solution enabling successful applications on autonomous multicopter flights [52], [56], [65]. However, selecting appropriate penalty weights is usually cumbersome – inadequately small weights fail to yield a feasible solution, while excessively large ones would overwhelm the original objective cost, resulting in conservative results, such as slow speed and long flight time, and can even cause numerical difficulties. Besides, these penalty weights also need frequent adjustment for the change of objective cost, which can vary significantly across different scenarios, such as varying length of trajectory and constraint boundaries.

In addition to the soft penalty method, there are well-established NLP solvers applicable for trajectory optimization. These solvers, designed to find local optima, are based on various optimization methods. For instance, Lagrangian merit function with sequential quadratic programming (SQP) is used in SNOPT [70], interior-point filter line-search algorithm in IPOPT [71] and sequential least-squares QP in NLOPT [72]. These methods vary in their assumptions and prioritization between optimality and feasibility, leading to different results and performance levels. Furthermore, they are intended for general NLP problems, including large-scale ones with a large number of variables and constraints. As a result, their efficiency may not always be sufficient for real-time trajectory optimization. For example, Sun et al. [73] demonstrated that the commercial solver SNOPT often fails to find feasible solution, and even takes over $7\,\mathrm{s}$ to optimize a minimum-jerk trajectory for quadrotor flight below $2\,\mathrm{m/s}$ in several-metres corridor using a $3.3\,\mathrm{GHz}$ Intel CPU.

Customized methods tailored for trajectory optimization can outperform general solvers in certain performance aspects, significantly enhancing the systems they are applied to. For example, Sun et al. [74] presented a bilevel optimization where the lower level utilizes QP to optimize spatial variables with a

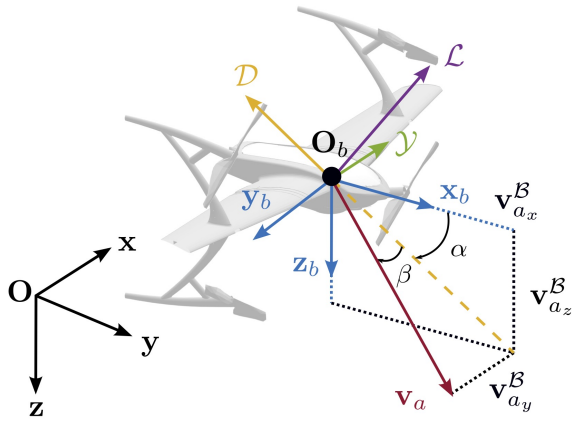


Fig. 2: Coordinate systems and aerodynamic nomenclatures.

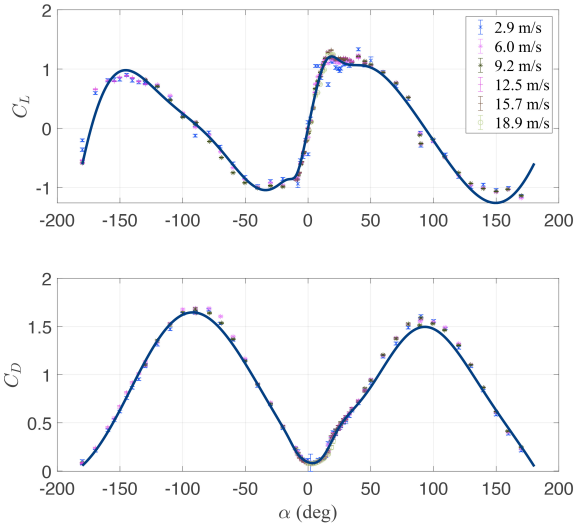


Fig. 3: Longitudinal aerodynamic coefficients C_L and C_D of our previous quadrotor tail-sitter UAV prototype, identified by wind tunnel tests [76].

fixed trajectory duration, while the upper level optimizes the time allocation using a separate NLP. However, this approach was limited to minimum-jerk Bézier-spline trajectory with kinodynamic constraints, and the challenge of solving the upper-level NLP with more complex actuator constraints remains unresolved. Schulman et al. [75] presented TRAJOPT, a ℓ_1 penalty method with sequential convex programming (SCP) for manipulator motion planning. While it showed promising results on finding feasible solutions for manipulator tasks, its experimental performance including success rate, efficiency and optimality, falls short in the context of tail-sitter planning, as shown in benchmark comparisons in Section VII. Our proposed solver, EFOPT, is similar in concept to [75], but differs in key implementations including Hessian calculation, update condition, trust-region strategy and trust-region subproblem solution. These enhancements enable EFOPT to excel in performance of tail-sitter trajectories. Moreover, EFOPT is designed without priori knowledge of the optimization, so it is potentially applicable to other NLPs that require efficient and feasible solutions.

III. Preliminaries

A. Flight dynamics

The coordinate frames and flight dynamics are defined and presented in Fig. 2, following the convention of traditional fixed-wing aircraft. We view the tail-sitter airframe as a rigid body. The translational and rotational dynamics of the aircraft are modeled as follows:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1a}$$

$$\dot{\mathbf{v}} = \mathbf{g} + a_T \mathbf{R}\mathbf{e}_1 + \frac{1}{m}\mathbf{R}\mathbf{f}_a \tag{1b}$$

$$\dot{\mathbf{R}} = \mathbf{R}\lfloor\boldsymbol{\omega}\rfloor \tag{1c}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} + \mathbf{M}_a - \boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega} \tag{1d}$$

where $\mathbf{p}$ and $\mathbf{v}$ are respectively the vehicle position and velocity in the inertial frame, $\boldsymbol{\omega}$ is the angular velocity in the body frame, $\mathbf{R}$ denotes the rotation from the inertial frame to the body frame, m is the total mass of the aircraft, $\mathbf{J}$ is the inertia matrix and $\mathbf{g} = [0\ 0\ 9.8]^T$ is the gravity vector in the inertial frame. a_T and $\boldsymbol{\tau}$ denote the thrust acceleration scalar and control moment vector produced by actuators (e.g., four motors for a quadrotor tail-sitter). $\mathbf{f}_a$ and $\mathbf{M}_a$ are the aerodynamic force and moment in the body frame, respectively. The notation $\lfloor\cdot\rfloor$ denotes the skew-symmetric matrix of a 3-D vector.

Referring to [77], the aerodynamic force $\mathbf{f}_a$ is modeled in the body frame as follows:

$$\mathbf{f}_a = \begin{bmatrix}\mathbf{f}_{a_x}\\ \mathbf{f}_{a_y}\\ \mathbf{f}_{a_z}\end{bmatrix} = \begin{bmatrix}-\cos\alpha & 0 & \sin\alpha\\ 0 & 1 & 0\\ -\sin\alpha & 0 & -\cos\alpha\end{bmatrix}\begin{bmatrix}\mathcal{D}\\ \mathcal{Y}\\ \mathcal{L}\end{bmatrix} \tag{2}$$

where α is the angle of attack. The force components $\mathcal{L}, \mathcal{D}, \mathcal{Y}$ are respectively the lift, drag, and side force, which can be written as products of non-dimensional coefficients, dynamic pressure $\frac{1}{2}\rho V^2$, the reference area S (e.g., the wing area) as follows:

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}\rho V^2 S C_L(\alpha,\beta)\\ \mathcal{D} &= \frac{1}{2}\rho V^2 S C_D(\alpha,\beta)\\ \mathcal{Y} &= \frac{1}{2}\rho V^2 S C_Y(\alpha,\beta)\end{aligned} \tag{3}$$

where ρ is the air density and $V = \|\mathbf{v}_a\|$ is the norm of the airspeed. C_L, C_D, C_Y are the lift, drag, and side force coefficients. The aerodynamic coefficients are functions of the angle of attack α and the sideslip angle β, depending on the design of the airfoil profile and the overall airframe. The accurate aerodynamic coefficients are usually identified by wind tunnel tests [76].

B. Differential flatness in coordinated flights

A tail-sitter vehicle is a differentially flat system in coordinated flights, which is proven in our precious work [37]. We present some important results of the differential flatness

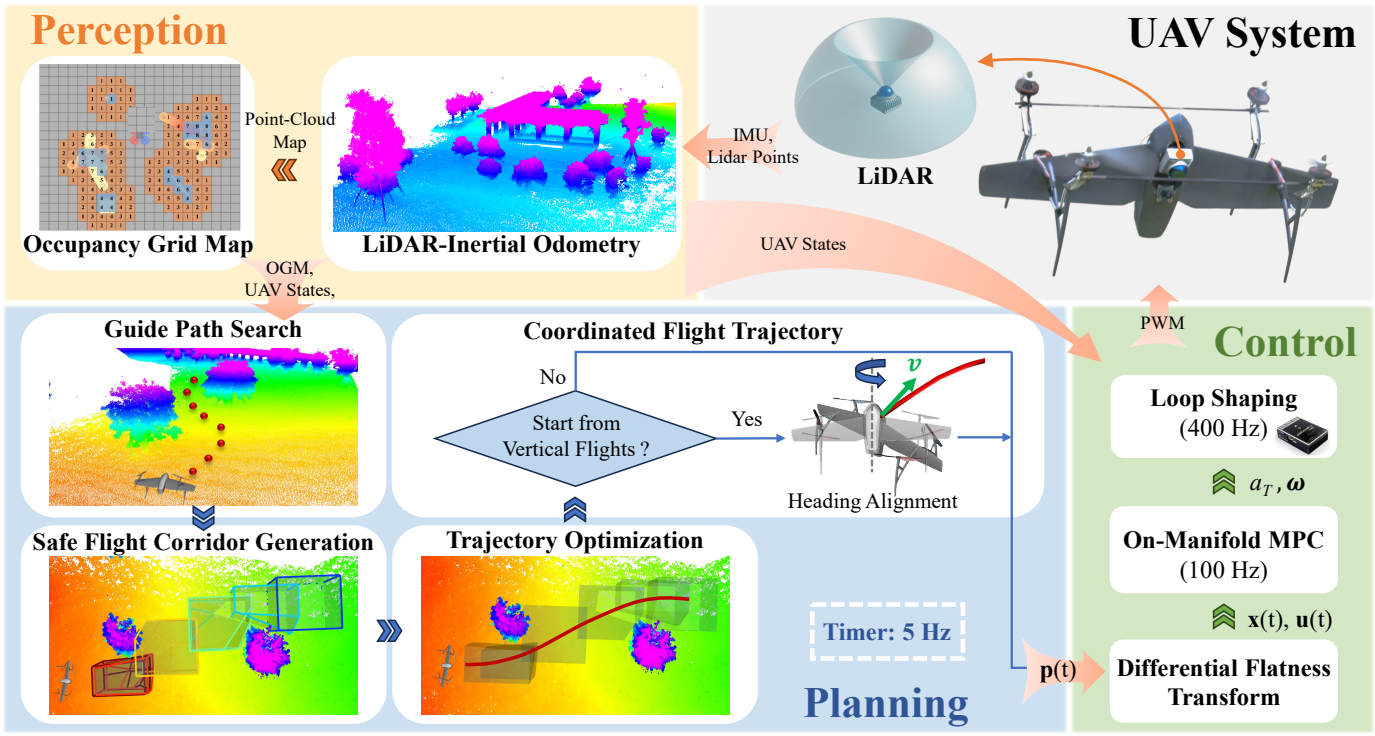


Fig. 4: System overview

that are used in this paper. Readers are referred to [37] for thorough derivation.

Coordinated flight indicates a flight condition where an aircraft has no sideslip (e.g., $\beta = 0, \mathbf{v}_{a_y}^{\mathcal{B}} = 0$) [78]. This flight condition does not restrict the tail-sitter to reach any position in the 3-D space, but significantly improve aerodynamic efficiency and moment stability during level flights that are well-studied in a fixed-wing aircraft. Coordinated flight is also beneficial for perception sensors with small field of view (FoV), as it maintains forward visibility. Moreover, this flight condition simplifies the task of obtaining aerodynamic models, focusing only on longitudinal aerodynamic coefficients around $\beta = 0$ (see Fig. 3).

The flat output is chosen as the vehicle position $\mathbf{p} \in \mathbb{R}^3$. Utilizing $\mathbf{p}$ and its derivatives, all of the system states and inputs can be represented by algebra functions. In this paper, we consider the differential flatness up to the third derivative of position, $\mathbf{p}^{(3)}$. The system states and inputs are defined as follows:

$$\mathbf{x} = (\mathbf{p} \quad \mathbf{v} \quad \mathbf{R}) \in \mathbb{R}^3 \times \mathbb{R}^3 \times SO(3) \tag{4a}$$

$$\mathbf{u} = (a_T \quad \boldsymbol{\omega}) \in \mathbb{R} \times \mathbb{R}^3 \tag{4b}$$

and their corresponding flatness functions are given by [37]:

$$\mathbf{x} = \mathcal{X}(\mathbf{p}^{(0:2)}) \tag{5a}$$

$$\mathbf{u} = \mathcal{U}(\mathbf{p}^{(1:3)}) \tag{5b}$$

where $\mathcal{X}$ and $\mathcal{U}$ denote the state and input flatness functions respectively. The calculation process of this differential flatness transform mapping is detailed in [37]. It is also noted from [37] that the mapping exhibits singularities due to the requirements of coordinated flight condition and the use of classic aerodynamic model, in which the angle of attack α and sideslip angle β are invalid when airspeed is zero. Such singularities have been discussed and resolved in the previous work [37], except the following condition:

$$\mathcal{S}(\mathbf{x}) = \|\dot{\mathbf{v}} - \mathbf{g}\| = 0 \tag{6}$$

which is essentially the free-fall condition and should be actively avoided in the trajectory optimization.

IV. System Overview

In this section, the algorithm pipeline for autonomous flight is presented. The framework consists of three main modules: perception, planning and control, as depicted in Fig. 4. Within the perception module, vehicle states including pose, velocity, and attitude, are estimated by a LiDAR-inertial odometry [50], using 3-D LiDAR points and IMU measurements. Simultaneously, a high-accuracy 3-D point-cloud map is constructed and continuously updated in real time. This map is then downsampled and utilized to maintain a local occupancy grid map (OGM), segmenting the environment into spaces that are obstacle-occupied, unknown and known-free, as proposed in [79].

Next, the planning module generates a desired trajectory based on the OGM and the UAV current states. Initially, a guide path is searched between the current UAV position and the target within a replanning horizon (set to 30 m in this paper) in know-free and unknown spaces, using A* algorithm. Once successfully finding a collision-free path, a safe flight corridor (SFC) comprising of consecutively overlapped polyhedra is generated along the guide path, to segment obstacle-free space from the map. This SFC is then incorporated as

a positional constraint in trajectory optimization. Due to the limited sensing range of the onboard sensor and receding horizon, the planning module consisting of the A* path search, SFC generation and trajectory optimization run at a fixed frequency of 5 Hz, to avoid newly-appeared obstacles. If the initial state involves vertical flights (i.e., hovering, takeoff and landing), the current UAV heading may not be aligned with the velocity direction on the trajectory. In this case, an extra yawing trajectory is inserted at the beginning of the optimized trajectory to align the UAV current heading with the initial velocity direction of the trajectory. With the heading alignment, the tail-sitter can perform smooth coordinated flights on the trajectory. While the A* path search and SFC generation are implemented following existing works [54], [80] respectively, our primary focus in this paper is on the trajectory optimization.

The planned trajectory $\mathbf{p}(t)$ is then forwarded to the control module as a reference. In this module, the trajectory planned in flat-output space is converted into a state-input trajectory $(\mathbf{x}(t), \mathbf{u}(t))$ by the differential flatness transform in (5). Using the current state feedback from LIO, an on-manifold MPC proposed in [37], [81] tracks the state-input trajectory at a frequency of 100 Hz. The consequent optimal thrust acceleration command a_T and body angular velocity commands $\boldsymbol{\omega}$ are sent to an autopilot, where they are tracked by a low-level controller employing loop-shaping techniques (i.e., proportional–integral–derivative (PID) controller and additional Notch filters for flexible modes attenuation [82]) at 400 Hz. The low-level controller then generates throttle and torque commands, which are further converted into pulse width modulation (PWM) signals to drive the actuators (i.e., four motors).

Except the low-level controller implemented on autopilot, all other modules – perception, planning and control modules – run in real time on the onboard computer.

V. Trajectory Optimization Formulation

Although tail-sitter is proven to be a differentially flat system with vehicle position as its flat output similar to multicopters, it is impractical to simply approximate kinodynamic constraints (e.g. velocity, acceleration and jerk) for dynamical feasibility, due to its highly nonlinear aerodynamics. The agility of tail-sitter's maneuvers, actuator saturation and system nonlinearity demand a more accurate consideration of dynamical feasibility. This section presents formulation of trajectory optimization, incorporating with practical constraints for autonomous flights.

We formulate the trajectory planning problem as an optimization to generate a trajectory $\mathbf{p}(t) : \mathbb{R} \in [0, T_f] \mapsto \mathbb{R}^3$ that is dynamically feasible and collision-free, while minimizing the flight time T_f and snap energy $\|\mathbf{p}^{(4)}\|^2$. The dynamic feasibility and obstacle avoidance are guaranteed by respectively incorporating the actuator saturation constraints and positional constraints of safe flight corridors into the optimization.

We parameterize the trajectory as piecewise polynomials using a spatial-temporal deformation proposed in [52]. Given a flight corridor $\mathcal{P}$ comprised of M consecutively overlapped

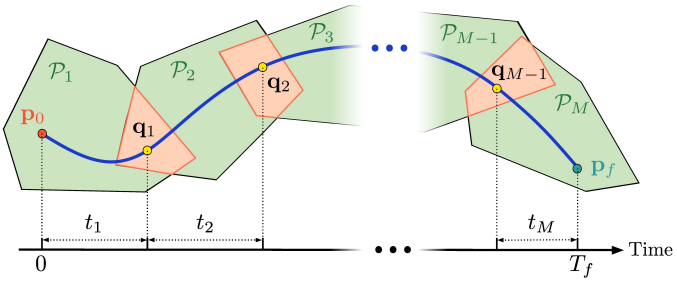


Fig. 5: Trajectory paramterization. The trajectory (blue curve) consecutively connects the inital position $\mathbf{p}_0$, internal waypoint sequence $\mathbf{q}_1, \cdots, \mathbf{q}_{M-1}$, and terminal position $\mathbf{p}_f$. Each trajectory segment has flight duration t_i, and the entire flight duration is T_f.

convex polyhedra $\mathcal{P}_i$, both the trajectory $\mathbf{p}(t)$ and flight duration T_f are divided into M segments, as shown in Fig. 5. Denote $\mathbf{p}(t) = (\mathbf{p}_1(t), \cdots, \mathbf{p}_M(t))$, and $\mathbf{T} = (t_1, \cdots, t_M)$ where $\sum_{i=1}^{M} t_i = T_f$. We introduce sequence of internal waypoints $\mathbf{Q} = (\mathbf{q}_1, ..., \mathbf{q}_{M-1})$. Each waypoint element is inserted at each intersection of the polyhedra, ensuring $\mathbf{q}_i \in (\mathcal{P}_i \cap \mathcal{P}_{i+1})$. Each segment of the trajectory $\mathbf{p}_i(t)$ is confined within the i-th polyhedron $\mathcal{P}_i$, connecting $\mathbf{q}_i$ (or $\mathbf{p}_0$) and $\mathbf{q}_{i+1}$ (or $\mathbf{p}_f$). These trajectory segments create a continuous path from the initial position $\mathbf{p}_0$, through the waypoints sequence $\mathbf{Q}$, to the terminal position $\mathbf{p}_f$. As suggested by [52], this piecewise trajectory is expressed as a set of minimum-snap polynomials with C^3 continuity. Given the initial state $\mathbf{s}_0$, terminal state $\mathbf{s}_f$, the piecewise polynomial trajectory is eventually parameterized by:

$$\mathbf{p}_i(t) = \mathbf{C}_i \left(\mathbf{Q}, \mathbf{T}, \mathbf{s}_0, \mathbf{s}_f\right) \beta(t), \quad t \in [0, t_i] \tag{7}$$

where $\mathbf{C}_i$ is a polynomial coefficient matrix related to internal waypoint $\mathbf{Q}$, time allocation $\mathbf{T}$, and given initial and terminal states $\mathbf{s}_f, \mathbf{s}_f$. $\beta(t) = [t^0, \cdots, t^7]^T$.

The trajectory optimization based on this polynomial parameterization is then formulated as follows:

$$\min_{\mathbf{Q},\mathbf{T}} \int_0^{T_f} \|\mathbf{p}^{(4)}(t)\|^2 dt + \rho T_f \tag{8a}$$

$$\text{s.t. } \mathbf{p}(t) = \mathbf{C}(\mathbf{Q}, \mathbf{T}, \mathbf{s}_0, \mathbf{s}_f)\beta(t), \quad T_f = \sum_{i=1}^{M} t_i \tag{8b}$$

$$\mathbf{x}(t) = \mathcal{X}\left(\mathbf{p}^{(0:2)}(t)\right), \ \mathbf{u}(t) = \mathcal{U}\left(\mathbf{p}^{(1:3)}(t)\right) \tag{8c}$$

$$\mathbf{p}_i(t) \in \mathcal{P}_i, \quad i = 1, \cdots, M \tag{8d}$$

$$\mathbf{x}(t) \in \mathbb{X}, \ \mathbf{u}(t) \in \mathbb{U} \tag{8e}$$

$$\mathcal{S}(\mathbf{x}(t)) \geqslant \epsilon \tag{8f}$$

where $\rho > 0$ is the weight of total flight time T_f. The constraint in (8d) confines the position trajectory within the SFCs for collision safety. The constraints in (8e) describe the kinodynamic and actuator constraints, respectively, where:

$$\|\mathbf{v}(t)\| \leqslant v_{\max}, \quad \|\dot{\mathbf{v}}(t)\| \leqslant a_{\max} \tag{9}$$

with $v_{\max}$ and $a_{\max}$ the maximum velocity and acceleration, and $\mathbb{U} = \{\mathbf{u} \in \mathbb{R}^4 | \ \mathbf{u}_{\min} \leqslant \mathbf{u} \leqslant \mathbf{u}_{\max}\}$ is the boundary of the system inputs (i.e., the thrust acceleration a_T and angular velocity $\boldsymbol{\omega}$). $\mathcal{S}(\mathbf{x})$ in (8f) denotes the singularity condition in

(6) implemented with a small positive value ϵ for numerical stability ($\epsilon = 0.1\,\mathrm{m/s^2}$ in this paper).

The optimization problem in (8) aims to optimize both the internal waypoint $\mathbf{Q}$ and time allocation $\mathbf{T}$, to pursue a time-optimal trajectory with a minimum-snap energy for smoothness, as described in the objective function (8a). In this paper, we assign a large weight to the flight time (i.e., $\rho = 1\text{e}4$) to pursue fast flight. However, while this high weighting enables aggressive trajectories with shorter flight duration and faster speeds, it also significantly increases the tendency towards constraint violations, posing a substantial challenge to optimization solvers.

VI. Efficient Feasibility-First Solution for NPL Optimization

The trajectory optimization in (8) presents a constrained non-convex optimization problem that must be solved both efficiently and robustly to enable real-time autonomous flights. In our previous work [37], we addressed offline trajectory optimization for flying through fixed waypoints without corridor constraint, using the MINCO framework [52], a soft penalty method combined with LBFGS. However, as previously discussed, soft penalty methods typically require significant effort for fine-tuning penalty weights in different scenarios. More importantly, even with meticulous tuning, they often fail in solutions that are either too time-consuming or infeasible, rendering them unsuitable for online planning.

In the context of online planning for autonomous flights, obtaining a feasible solution should always be the first priority for trajectory safety. The safety includes actuator constraints to prevent control failures, and obstacle avoidance (e.g., SFC constraint) to avoid collisions with obstacles. Additionally, computational efficiency is also crucial for timely responses to newly detected obstacles, especially during high-speed flights. While feasibility and efficiency are key requirements, optimality of the solution, although with less priority, is also desirable for achieving high-quality trajectories, characterized by shorter flight times and lower energy consumption. Addressing these needs, we develop an efficient feasibility-first solver, EFOPT, based on ℓ_1 penalty method and the framework of sequential quadratic programming (SQP).

A. Background: ℓ_1 penalty method with sequential quadratic programming

The trajectory optimization such as (8) can be denoted as a general non-convex optimization:

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{10a}$$
$$\text{s.t.} \quad g_i(\mathbf{x}) \leqslant 0, \quad i = 1, 2, \cdots, m \tag{10b}$$
$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \cdots, n \tag{10c}$$

where f, g_i, h_i are non-convex scalar functions.

Sequential convex optimization is an iterative method that solves a non-convex optimization problem by repeatedly constructing a quadratic programming sub-problem, which is an approximation to the original problem around the current iterate $\mathbf{x}$. The original non-convex programming can be solved by iteratively calculating the QP sub-problems as follows:

$$\min_{\mathbf{d}} \tilde{f}(\mathbf{x}+\mathbf{d}) \tag{11a}$$
$$\text{s.t.} \quad \tilde{g}_i(\mathbf{x}+\mathbf{d}) \leqslant 0, \quad i = 1, 2, \cdots, m \tag{11b}$$
$$\tilde{h}_i(\mathbf{x}+\mathbf{d}) = 0, \quad i = 1, 2, \cdots, n \tag{11c}$$

where $\mathbf{d}$ is the update step and the variable vector is iteratively updated $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{d}$ by solving QPs in (11) until the original problem satisfies convergent conditions. $\tilde{f}, \tilde{g}, \tilde{h}$ are convexified quadratic models

$$\tilde{f}(\mathbf{x}+\mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \left(\nabla^2 f(\mathbf{x})\right) \mathbf{d} \tag{12a}$$
$$\tilde{g}(\mathbf{x}+\mathbf{d}) = g(\mathbf{x}) + \nabla g(\mathbf{x})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \left(\nabla^2 g(\mathbf{x})\right) \mathbf{d} \tag{12b}$$
$$\tilde{h}(\mathbf{x}+\mathbf{d}) = h(\mathbf{x}) + \nabla h(\mathbf{x})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \left(\nabla^2 h(\mathbf{x})\right) \mathbf{d} \tag{12c}$$

However, the constructed QP sub-problem in (11) may have no feasible solutions, in the case where the quadratic approximation has considerable deviation from the original problem due to excessive non-convexity. The ℓ_1 penalty method with SQP can be more robust to highly non-convex problems. The original problem in (11) can be solved by iteratively solving an unconstrained minimization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \mu \sum_{i=1}^{m} |g_i(\mathbf{x})|^+ + \mu \sum_{i=1}^{n} |h_i(\mathbf{x})| \tag{13}$$

where $|a|^+ = \max(0, a)$ is ℓ_1 penalty, and μ is a coefficient increased sequentially if (13) is solved but constraints are not satisfied. In each penalty iteration, the problem (13) is further solved by iteratively approximating the unconstrained problem by a QP with trust region constraint:

$$\min_{\mathbf{d}} \tilde{f}(\mathbf{x}+\mathbf{d}) + \mu \sum_{i=1}^{m} |\tilde{g}_i(\mathbf{x}+\mathbf{d})|^+ + \mu \sum_{i=1}^{n} |\tilde{h}_i(\mathbf{x}+\mathbf{d})| \tag{14a}$$
$$\text{s.t.} \quad \|\mathbf{d}\| \leqslant s \tag{14b}$$

where s is a trust region (TR) to restrict each step size such that the approximation is close enough to the original problem. Each QP in (14) is called trust-region sub-problem (TRS).

The penalty coefficient μ is initialized from a small value and increased sequentially. In the beginning, the small coefficient makes the weighted constraint violation less dominant in the merit function (14), allowing to search feasible solutions began from lower objective cost. This property tends to enable the optimization to converge to feasible solution with higher optimality. In the context of trajectory optimization, it means a trajectory with less time duration and (or) control energy, which are critical to high-speed maneuvers. In subsequent iterations, the penalty coefficient is increased to gradually enforce the constraints until its satisfaction.

B. Implementation of EFOPT

Our implementation is a variant of ℓ_1 penalty method for sequential convex programming, presented in [75]. The constraint violation is denoted

$$C = \sum_{i=1}^{m} |g_i(\mathbf{x})|^+ + \sum_{i=1}^{n} |h_i(\mathbf{x})| \tag{15}$$

Denote the merit function as the sum of objective and weighted constraint violation as follows:

$$F(\mathbf{x}) = f(\mathbf{x}) + \mu C \tag{16}$$

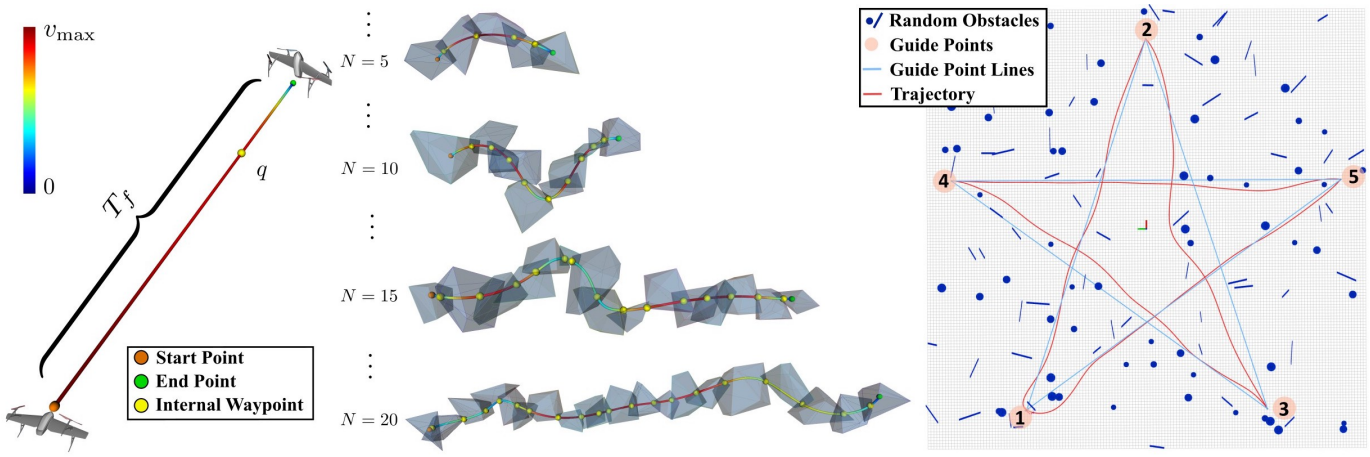
and $\tilde{F}(\mathbf{x})$ denotes a convexification of the merit function in a form of quadratic models in (12).

The implementation of EFOPT is outlined in Algorithm 1, with assumption that gradients of the objective f and constraints g, h are provided. Initially (Line 18-20), the penalty coefficient μ, trust region size s and approximated Hessian $\nabla^2 F$ are initialized, while convergence thresholds xtol and ftol are set to coarse values. Then the solver performs a double loop iteration: the outer loop (Line 22-52) gradually enforces the optimization constraints by iteratively increasing the penalty weight μ in the merit function $F(\mathbf{x})$, and the inner loop (Line 23-42) minimizes the merit function $F(\mathbf{x})$ by sequentially approximating it locally at each iteration $\mathbf{x}$ by a TRS problem. Specifically, in the inner loop, the merit function is first evaluated and convexified into quadratic models (Line 24-25). A TRS in (14) is then constructed to minimize the approximated merit function subject to trust region, which is solved efficiently by Steihaug's conjugate gradient (Steihaug-CG) method [83] (Line 26). Steihaug-CG is known for computational efficiency and has been widely used for trust region sub-problems. The resulting update $\mathbf{d}$ is evaluated on improvements of the merit function F and approximation quality, which is the ratio of actual and approximated decrease of the merit function (Line 27-29). If the merit function decreases (i.e., improves), the solution vector $\mathbf{x}$ is updated and the Hessian matrix is estimated using the BFGS algorithm [84]. After then, the trust region is adjusted for the next iteration: if the approximation quality is poor due to adverse or small improvement in merit function, the trust region size is reduced (Line 34-35), which helps to improve the approximation quality in the next iteration of the inner loop; if the approximation quality is sufficient, the trust region expands for larger updates (Line 36-37); otherwise, the trust region remains unchanged. This adaptive approach, known as Marquardt's strategy, is widely utilized in the Levenberg–Marquardt (LM) algorithm [85]. The inner loop optimization is terminated when the update $\mathbf{d}$ or merit function variation are below the convergence thresholds xtol and ftol (Line 39-41). The thresholds are initially set to coarse values $\text{xtol}^\dagger$, $\text{ftol}^\dagger$ to avoid the successive optimization in inner loop optimization, which could be time consuming. The reason is that, at the beginning of the outer loop optimization, exhaustive optimization of the inner loop is not really necessary since the constraints penalty μ is small and the exact optimal solution of the merit function is likely infeasible anyway. Once the constraints violation is below a coarse threshold $\text{ctol}^\dagger$ (Line 43-45), the convergence thresholds of the inner loop switch to smaller values $\text{xtol}^{\dagger\dagger}$, $\text{ftol}^{\dagger\dagger}$ for greater accuracy. When the inner loop converges or reaches the maximum convexification iteration, the outer loop assesses the constraint satisfaction. If any constraint is not satisfied (i.e., above the fine threshold $\text{ctol}^{\dagger\dagger}$), the penalty weight μ is increased to intensify the constraint penalty in the merit function (Line 46-48). Otherwise the optimization returns the optimal result and terminates (Line 49-51). The outer loop repeats until either constraint satisfaction or the maximum penalty iteration is reached, which means the optimization fails.

Compared to TRAJOPT [75], which is also developed

Algorithm 1: Implementation of EFOPT

1 **Parameters:**
2 m: maximum penalty iteration number
3 n: maximum convexification iteration number
4 μ_0: initial penalty coefficient
5 s_0: initial trust region size
6 λ_1, λ_2: model quality thresholds $(0 < \lambda_1 < \lambda_2)$
7 τ^-, τ^+: trust region expansion and shrinkage factors $(0 < \tau^- < 1 < \tau^+)$
8 k: penalty scaling factor $(k > 1)$
9 $\text{xtol}^\dagger$, $\text{ftol}^\dagger$, $\text{ctol}^\dagger$: coarse thresholds
10 $\text{xtol}^{\dagger\dagger}$, $\text{ftol}^{\dagger\dagger}$, $\text{ctol}^{\dagger\dagger}$: fine thresholds $\left((\cdot)^{\dagger\dagger} < (\cdot)^\dagger\right)$
11 **Variables:**
12 $\mathbf{x}$: current solution vector
13 $\mathbf{d}$: update step
14 μ: penalty coefficient
15 s: trust region size
16 xtol, ftol: convergence thresholds for $\mathbf{x}$ and $F(\mathbf{x})$
17 ctol: constraint tolerance
18 **Initialization:**
19 $\mu \leftarrow \mu_0$, $s \leftarrow s_0$, xtol $\leftarrow \text{xtol}^\dagger$, ftol $\leftarrow \text{ftol}^\dagger$, ctol $\leftarrow \text{ctol}^\dagger$
20 $\nabla^2 F \leftarrow \mathbf{I}$
21 **Optimization:**
22 **for** PenaltyIteration $= 1, 2, \cdots$ m **do**
23 **for** ConvexifyIteration $= 1, 2, \cdots$ n **do**
/* Construct and solve TRS */
24 $F(\mathbf{x})$ = EvaluateMeritFunction(f, g, h, μ) in (16)
25 $\tilde{F}(\mathbf{x})$ = ConvexifyProblem$(F, \nabla F, \nabla^2 F)$ in (12)
26 $\mathbf{d} \leftarrow$ Solve TRS in (14) by Steihaug-CG algorithm
/* Verify update condition */
27 TrueImprove $= F(\mathbf{x} + \mathbf{d}) - F(\mathbf{x})$
28 TRSImprove $= \tilde{F}(\mathbf{x} + \mathbf{d}) - \tilde{F}(\mathbf{x})$
29 ApproxQuality = TrueImprove / TRSImprove
30 **if** TrueImprove < 0 **then**
31 Update $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{d}$
32 Approximate Hessian $\nabla^2 F \leftarrow$ BFGS algorithm
33 **end**
/* Update trust region */
34 **if** ApproxQuality $< \lambda_1$ **then**
35 Shrink TR: $s \leftarrow \tau^- * s$
36 **else if** ApproxQuality $> \lambda_2$ **then**
37 Expand TR: $s \leftarrow \tau^+ * s$
38 **end**
/* Convergence test */
39 **if** $\|\mathbf{d}\| <$ xtol or abs(TrueImprove) $<$ ftol **then**
40 break
41 **end**
42 **end**
/* Switch convergent thresholds */
43 **if** C $<$ ctol **then**
44 xtol $\leftarrow \text{xtol}^{\dagger\dagger}$, ftol $\leftarrow \text{ftol}^{\dagger\dagger}$, ctol $\leftarrow \text{ctol}^{\dagger\dagger}$
45 **end**
/* Increase penalty coefficient */
46 **if** C $>$ ctol **then**
47 Increase penalty: $\mu \leftarrow k * \mu$
48 Reset Hessian $\nabla^2 F \leftarrow \mathbf{I}$
49 **else**
50 return $\mathbf{x}$
51 **end**
52 **end**



(a) Two-variable optimization for a backward transition trajectory.

(b) Hover-to-hover trajectory optimization within 3-D safe flight corridors that have different number of randomly generated polyhedrons.

(c) Collision-free trajectory optimization flying through a sequence of guide points in a random obstacle-dense 3-D map with a size of 150m x150m (bird view).

Fig. 6: Three benchmark simulations to evaluate the performance of different solvers in trajectory optimization.

TABLE I: Primary difference between EFOPT and TRAJOPT.

Difference	EFOPT	TRAJOPT
Convergence threshold	Adaptive	Fixed
Hessian Approximation	BFGS	Finite Difference
Update Condition	True Improvement	Model Improvement
TRS Solver	Steihaug-CG	Gurobi
TR update strategy	Marquardt's strategy	Bang-bang strategy

on ℓ_1 penalty method with SQP, EFOPT exhibits superior computational efficiency and accuracy. There are differences between EFOPT and TRAJOPT discernible from both the article [75] and its source code. Readers are referred to these materials for more detailed information. We summarize four main differences that we believe significantly influence optimization performance in TABLE I and discussed as follows.

1) EFOPT uses adaptive convergence thresholds – coarse thresholds to reduce iteration when the solution is far from the feasible region, and fine thresholds to improve the accuracy of feasible solution. In contrast, TRAJOPT uses a single set of convergence thresholds that may result in either slow convergence or inaccurate solutions at each inner loop optimization.
2) EFOPT adopts efficient BFGS algorithm to approximate the Hessian matrix $\nabla^2 F$, while TRAJOPT relies on time-consuming finite differences to calculate $\nabla^2 f, \nabla^2 g, \nabla^2 h$ numerically.
3) In EFOPT, the decision on the current solution update is the true improvement of the merit function. Conversely, TRAJOPT directly updates $\mathbf{x}$ to the solution $\mathbf{x}$ of TRS. Although $\mathbf{x}$ is optimal for the quadratic approximation, it does not guarantee to improve the original merit function in case of strong nonlinearity or inappropriate trust region sizes.
4) EFOPT uses an efficient C++ implementation of Steihaug's conjugate gradient algorithm to solve trust-region sub-problems, while TRAJOPT employs a commercial solver Gurobi[1] which is designed for general QPs. Repeatedly deploying Gurobi at each inner loop iteration leads to certain overhead in time consumption.
5) EFPOT utilizes Marquardt' strategy to shrink, expand or maintain the trust region size based on the approximation quality, while TRAJOPT' strategy either shrinks or expands the trust region. The bang-bang strategy in TRAJOPT tends to cause fluctuations, resulting in longer convergence or failures in practice.

These differences could lead to significant discrepancy in performance in trajectory optimization, which is evident in our benchmark results presented in Section VII.

VII. Benchmark Results

In this section, we conduct three simulated trajectory optimization to benchmark the performance of our proposed feasibility-first solver, EFOPT, against off-the-shelf NLP solvers, including TRAJOPT [75], SNOPT [70], IPOPT [71], NLOPT [72] and LBFGS-Lite[2]. Fig. 6 depicts the three simulations with detailed explanation provided later.

In all three simulations, the convergence thresholds of all solvers are set $\mathrm{xtol} = 1\mathrm{e}{-6}$ and $\mathrm{ftol} = 1\mathrm{e}{-6}$. Constraint tolerance is set $\mathrm{ctol} = 1\mathrm{e}{-6}$ in the first and second simulation, and relaxed to $\mathrm{ctol} = 1\mathrm{e}{-3}$ in the third simulation for efficiency consideration in online planning. These thresholds are used as the fine thresholds in EFOPT to ensure a fair comparison. In addition, the coarse thresholds in EFOPT are $\mathrm{xtol}^{\dagger\dagger} = 1\mathrm{e}{-3}$, $\mathrm{ftol}^{\dagger\dagger} = 1\mathrm{e}{-3}$ and $\mathrm{ctol}^{\dagger\dagger} = 1\mathrm{e}{-3}$ for fast convergence.

The performance of solvers is sensitive to iteration number. Generally, allowing more iterations tends to improve the success rate but at the cost of increased computational time. We have meticulously tuned the parameters of all solvers in comparison, and found that the default settings of SNOP, IPOPT, NLOPT and LBFGS-Lite maintain a good balance in

[1]https://www.gurobi.com/
[2]https://github.com/ZJU-FAST-Lab/LBFGS-Lite

TABLE II: Solvers' performance in simulation 1.

Solver	Objective($\times 10^4$)	Constraint	Time (ms)
EFOPT	7.3	**0**	**8.8**
TRAJOPT	10.9	3.6×10^{-7}	230.7
SNOPT	**6.5**	0.10	42.6
IPOPT	8.7	**0**	210.3
NLOPT	**6.5**	0.14	353.1
LBFGS-Lite($\mu=1e5$)	9.7	0.05	9.3
LBFGS-Lite($\mu=1e7$)	10.5	1.5×10^{-6}	9.9
LBFGS-Lite($\mu=1e9$)	10.5	2.9×10^{-9}	14.4

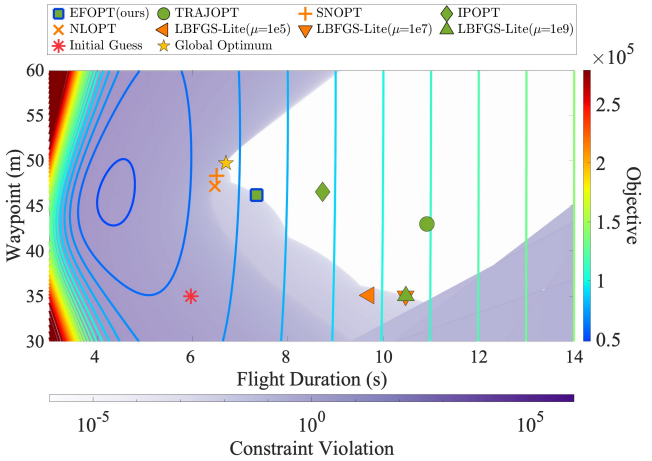
performance. In consequence, these solvers are implemented with their default configurations. However, the default setup of TRAJOPT exhibits low success rate, and we adopt the parameters from EFOPT and increase the maximum convexification iteration to 3000, to improve TRAJOPT's performance. Unlike the other solvers, LBFGS-Lite cannot handle constraints directly, but solves the unconstrained NLP as defined in (13) with a fixed penalty weight μ that should be specified manually. Hence we evaluate its performance under different values of μ in each simulation.

Analytical gradients of the objective function and constraints are provided to improve the convergence speed. Detailed calculations of these gradients are available in our previous work [37]. Besides, we utilized the derivative check function of SNOPT for gradient verification. After confirmation that all tested gradients were accurate, this function was disabled for time consumption statistics. The first two simulations are conducted on a 5.3 GHz Intel i9-10900KF processor, while the last one is carried out on the tail-sitter onboard computer with a 1.8 GHz Intel i7 processor.

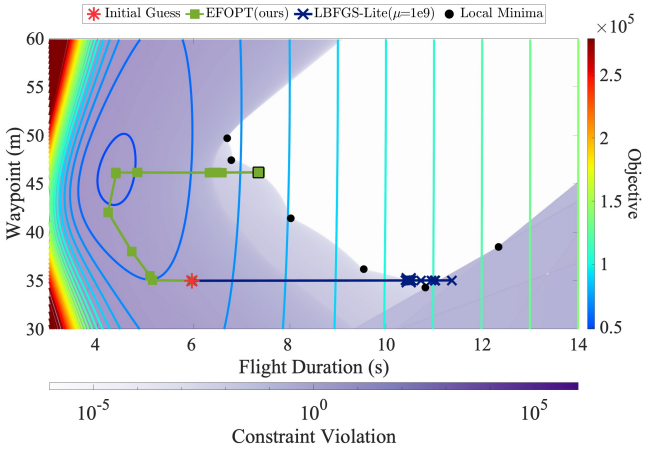
A. *Two-variable optimization*

Visualizing the change of objective function versus the values of optimization variables at different iterations can provide an intuitive presentation of solver convergence behavior. Such visualization is possible when the number of optimization variables are less than or equal to two. Therefore, in this simulation, we consider a typical tail-sitter backward transition flight and formulate a two-variable optimization problem for its trajectory generation. As shown in Fig. 6(a), Given an initial state $\mathbf{s}_0$ in level fight at $15\,\mathrm{m/s}$ and a terminal state $\mathbf{s}_f$ in hovering, the transition flight is a horizontal 2-D straight-line trajectory parameterized by two scalar variables: the total flight duration T_f and an internal waypoint q restricted on the straight line. Each trajectory segment separated by the waypoint q accounts for half of the total flight time T_f, leading to a two-segment polynomial trajectory $\mathbf{C}(q, T_f, \mathbf{s}_0, \mathbf{s}_f)$ as defined in (7). The initial values of of the two optimization variables T_f and q are chosen as heuristic values $(p_f - p_0)/v_{\max}$ and $(p_0 + p_f)/2$, respectively.

In Fig. 7, we visualize the objective values and constraint violations of the two-variable trajectory optimization, as well as results of different solvers in comparison. The objective values are depicted by contours in varying colors, while the



(a) The solutions of different solvers. The white area denotes the feasible region, where the solutions have constraint violation less than tolerant $\mathrm{ctol} = 1e{-}6$. Green markers denote feasible solutions, while orange markers denote infeasible solutions that have constraint violation exceeding the tolerance.



(b) Convergence path of EFOPT and LBFGS-Lite($\mu = 1e9$).

Fig. 7: Convergence progress in the two-variable optimization for the backward transition trajectory in Fig. 6 (a).

degree of constraint violations are indicated by depth of purple color. It is seen from the contours and color map that both the objective and constraint are nonlinear. The white area where constraint violations are less than the tolerant $\mathrm{ctol} = 1e6$, indicates the feasible region. Notably, the objective has a global minimum outside the feasible region. The initial guess marked by red asterisk is infeasible and distant from the global minimum of objective. Fig. 7(a) shows the global optimum (i.e., feasible solution with the lowest objective value, as indicated by the yellow star) and solutions of all compared solvers. None of the solvers can find the global optimum which lies at a corner of the feasible region. More specifically, SNOPT, NLOPT and LBFGS-Lite with lower penalty weights (i.e., $(\mu = 1e5, 1e7)$, whos solutions are marked in orange, come up with infeasible solutions in the purple area. EFOPT, TRAJOPT, IPOPT and LBFGS-Lite($\mu = 1e9$), whose solutions are marked in green, are successful to find feasible solutions. Among feasible solutions, EFOPT achieves the lowest objective value, and both EFOPT and LBFGS-

Lite($\mu = 1e9$) converge near the boundary of the feasible region, while TRAJOPT and IPOPT terminate in the middle of the feasible region, where objective can still decrease without constraint violation in any directions.

Fig. 7(b) presents the convergence paths of EFOPT and LBFGS-Lite($\mu = 1e9$) towards feasible solutions. In EFOPT, the penalty weight μ starts from a small value ($\mu = 1$ in this paper), which makes the objective dominate in the initial merit function in (13). Hence EFOPT first converges towards the objective global minimum along contour gradients, and then the convergence direction shifts towards the feasible region on the right. EFOPT finally terminates at a point on the boundary of the feasible region, without further converging to even nearby local minima that are also feasible (the black dots in Fig. 7(b)). The reason is that the terminal penalty weight has been increased to $\mu = 1e7$. Such a large weight ensures feasibility but meanwhile overwhelms the gradient direction of the merit function. The resultant gradient is barely affected by the objective, hence preventing a gradient-based solver (e.g., EFOPT) to further decrease the objective. On the other hand, LBFGS-Lite ($\mu = 1e9$) uses a large, fixed penalty weight of $\mu = 1e9$, which drives the solution straightly to the feasible region along the constraint gradients. After reaching the feasible region, the solver moves back along the constraint gradient to achieve lower objective values. Eventually, the solver converged at the boundary of the feasible region without further converging to even its nearby local minima (black dots in Fig. 7(b)), a phenomenon similar to our EFOPT due to the same reason. It is noted that the EFOPT reaches termination with a smaller penalty weight of $\mu = 1e7$, while LBFGS-Lite with the same penalty weight fails to converge to a feasible solution. In practice, a larger weight would increase numerical difficulties that may result in failures or slow convergence, which is evident in Fig. 7(b) that LBFGS-Lite spends a plenty of iterations near the convergence result.

TABLE II summarizes the comprehensive performance of all solvers. It is seen that EFOPT has the fastest computation time among all eight solvers in comparison, and has the lowest objective among all methods obtaining feasible solutions. Although SNOPT and NLOPT converge to lower objective, their constraint violation exceeds the threshold ctol$= 1e{-}6$, leading to infeasible solutions.

Remark 1. Both EFOPT ad TRAJOPT are based on ℓ_1 penalty method and SQP. However, in this test, TRAJOPT yields an inferior solution, where the objective value can further decline in any directions without violating any constraints. We believe the reason is that TRAJOPT directly uses the TRS solution as the current update, without evaluating the improvement of the original merit function. Besides, we have attempted to enable SNOPT to find feasible solution by increasing the iteration numbers, but failed. Similar findings have been reported in [74], where SNOPT often terminates without convergence or fails to find a feasible solution.

B. Random safe flight corridor

In the second simulation, we benchmark solvers in the problem of trajectory optimization with different scale of safe flight corridor (SFC) constraints (Fig. 6(b)). A SFC consists of N overlapping polyhedra with N tested from 1 to 20. For each N, we generate 1,000 different SFCs by randomly specifying the face number (in the range of 6 to 24), shape, and position of each polyhedra in the SFC. This results in 20,000 tested SFCs in total, among which the longest SFC spans more than $100\,\mathrm{m}$. The corresponding number of optimization variables when optimizing trajectories in these SFCs ranges from 1 to over 470, thereby testing the solvers' performance in both small-scale and large-scale scenarios. To further increase the variable dimension, we use a barycentric coordinate to represent waytpoint $\mathbf{q}_i$, which should be constrained within the intersected polyhedra $\mathcal{P}_i \cap \mathcal{P}_{i+1}$, by the vertices of the intersected polyhedra as proposed in [52]. That is, $\mathbf{q}_i = f_{\mathcal{B}}(\boldsymbol{\xi}_i)$ with $\boldsymbol{\xi}_i$ the vertices. Hence, The optimization variables become $(\boldsymbol{\xi}, \mathbf{T})$. For each waypoint $\mathbf{q}_i$, there are often many vertices representing the intersected polyhedra, hence the dimension of $\boldsymbol{\xi}_i$ will be significantly higher than that of $\mathbf{q}_i$, which effectively increases the scale of the optimization problem. The initial conditions of $\mathbf{q}_i$ (or its representation $\boldsymbol{\xi}_i$) are set to the mean of vertices of each overlapped polyhedra. Time allocation $\mathbf{T}$ is heuristically initialized as $t_i = \|\mathbf{q}_i - \mathbf{q}_{i-1}\|/v_{\max}$.

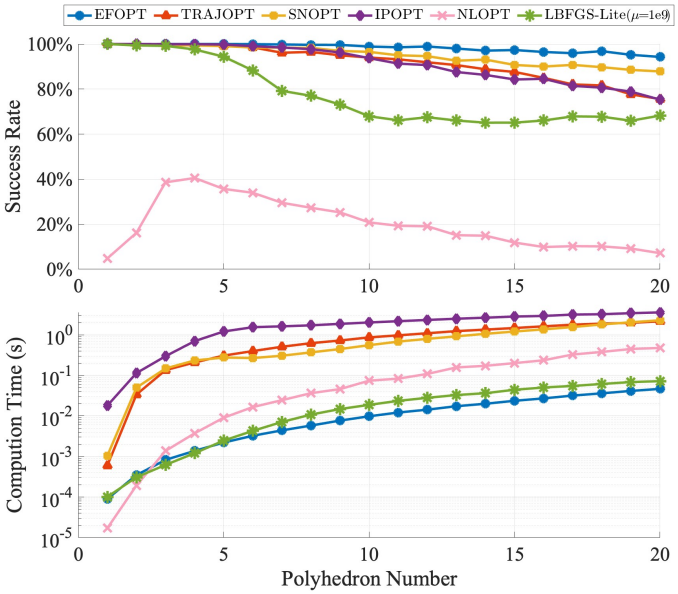


Fig. 8: Success rate and average computation time solvers in hover-to-hover trajectory optimization within randomly generated 3-D SFCs in Fig. 6 (b).

Fig. 8 presents the success rate and average computation time for optimizing trajectories in the 20,000 SFCs. Among the six solvers under evaluation, EFOPT demonstrates the highest success rate across all polyhedron numbers, and maintains the lowest computation time for polyhedron number from 5 to 20. As the polyhedron number increases, EFOPT's success rate slightly decreases from 100% to 94.3%, and its time consumption rises from $0.1\,\mathrm{ms}$ to $466\,\mathrm{ms}$. LBFGS-Lite ($\mu = 1e9$) requires slightly higher computation time, but its success rate drops significantly when the polyhedron number is above 5, reaching the lowest value of 65% at 15 polyhedra. TRAJOPT, SNOPT and IPOPT exhibit similar patterns in

both success rate and computation time. Their computation times are about 100 times longer than EFOPT for nearly all polyhedron number. NLOPT consistently has the lowest success rate for all polyhedron numbers.

C. Obstacle-dense random map

The third simulation assesses the complete planning module by planning obstacle-free trajectories on a randomly generated, obstacle-dense map measuring $150\times150\,\mathrm{m}^2$, as shown in Fig. 6(c). We specify a sequence of five points (i.e., the guide points) to roughly guide the UAV to fly a pentagram-shape trajectory. At each replan cycle, the planning module takes the current guide point in the list as its intermediate local target and generates a smooth trajectory from the current UAV position to the target, using the pipeline of A*, SFC generation, and trajectory optimization detailed in Section IV. Once the current guide point is reached (the UAV is in a its $2\,\mathrm{m}$ neighbor), it will be removed from the list and the next guide point will be used in the next replan cycle. The replan frequency is set to $5\,\mathrm{Hz}$, allowing $200\,\mathrm{ms}$ for the planning module to complete. If the planner is timeout and or the planned trajectory violates any constraint, the UAV continues to follow the previous trajectory. Following the previous trajectory at a planning failure could lead to collision with obstacles. If it occurs, we reset the UAV to the position immediately before the collision and continue the replan process as usual. The success rate is calculated by the ratio of feasible optimization results (regardless of its time consumption) to the total number of trajectory optimization. In this simulation, the optimization variables $(\mathbf{Q},\mathbf{T})$ are initialized in the same way as in simulation (b).

Fig. 9 shows the planning performance of the benchmarked solvers except NLOPT, which was unable to complete the test. As can be seen, EFOPT achieves the second highest success rate with a small gap from the highest one SNOPT. Although SNOPT presents the highest success rate, its time consumption (i.e., $0.3\,\mathrm{s}$ and $8.8\,\mathrm{s}$ in median and maximum) indicates that it frequently fails to update trajectory at $5\,\mathrm{Hz}$ (i.e., timeout), In case of timeout, the UAV slows down or even stops at the end of the previous trajectory. The low flight speed considerably simplifies the optimization problem, which is the main reason of SNOPT's high success rate. In terms of computation time, EFOPT and LBFGS-Lite ($\mu=1\mathrm{e}{-9}$) are the only two methods that can run in real-time. Relatively speaking, although LBFGS-Lite ($\mu=1\mathrm{e}{-9}$) has a lower computation time than EFOPT, its success rate is much lower (19.0% versus 89.6%), causing a bias in its average computation time.

The high success rate and computation efficiency of EFOPT has enabled it to plan feasible trajectories timely, even at high speeds. As a consequence, EFOPT achieves the highest flight speed and the shortest flight duration among all benchmarked solvers (see the lower two plots in Fig. 9). An example trajectory planned by EFOPT is shown in Fig. 10. As can be seen, EFOPT effectively pushes the vehicle speed to the maximum value of $15\,\mathrm{m/s}$ for the majority of the flight time, while maintaining compliance with actuator constraints on thrust acceleration a_T and angular velocity $\boldsymbol{\omega}$.

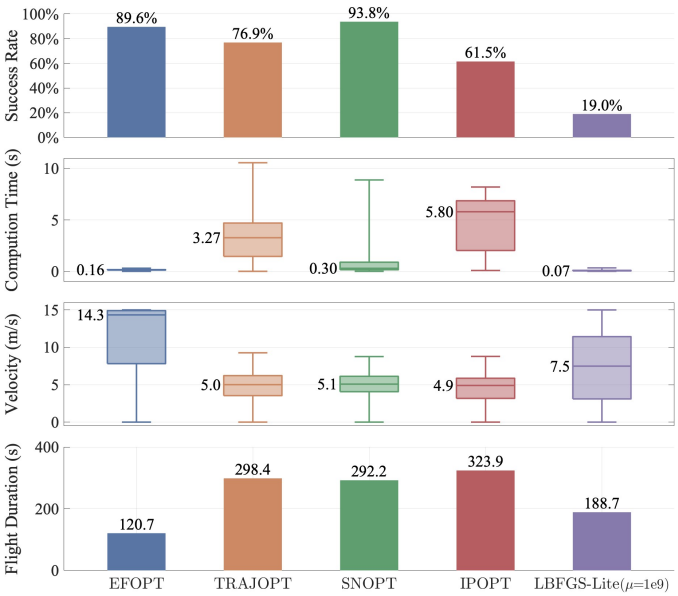


Fig. 9: Benchmark results of local replanning in a 3-D environment with random obstacles as in Fig. 6 (c).

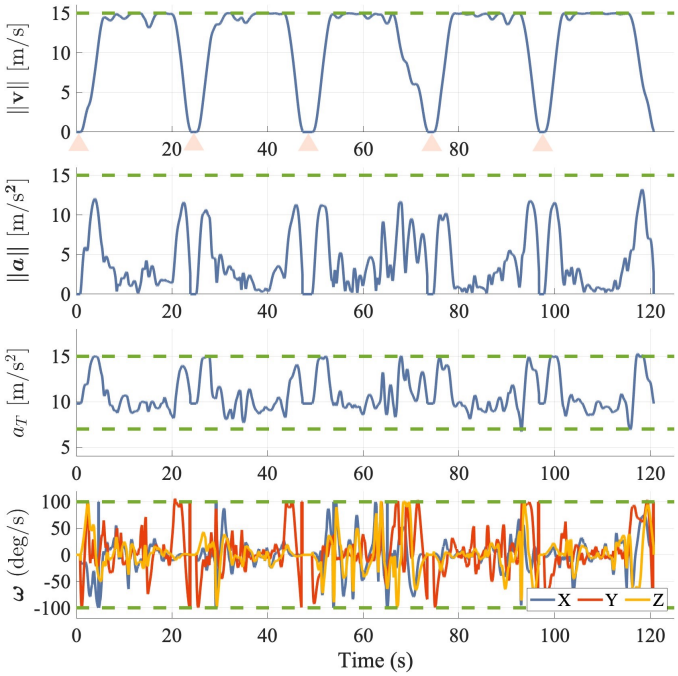


Fig. 10: UAV states and inputs of a trajectory solved by EFOPT in the simulation problem in Fig. 6 (c). The pink triangles in the first figure denote the time when the UAV arrives at the guide points and executes heading alignment toward the next guide point.

VIII. Real World Experiments

In this section, we detail the implementation of an autonomous tail-sitter UAV system, as shown in Fig. 11. To validate our entire algorithm framework, we conduct high-speed autonomous flight tests in various real-world environments, including an indoor drone laboratory, an underground parking lot, and an outdoor park at Sun Yat-sen University, as shown in Fig. 12. Guide points, depicted as yellow stars on the point-cloud maps, are provided to roughly guide the

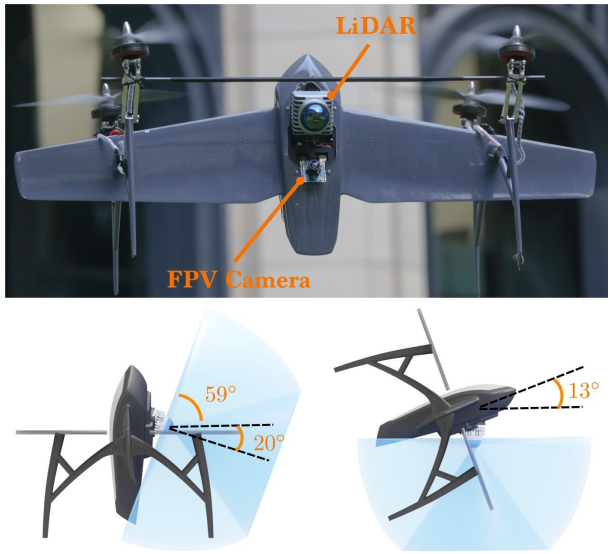


Fig. 11: LiDAR installation on the tail-sitter and its FoV. The LIVOX MID-360 LiDAR is mounted at a tilted angle of 20°, providing a FoV of $59^\circ \times 360^\circ$.

shape of the flight path by acting as local goal positions in the replanning module as detailed in Section VII-C. Obstacles are present between two successive guide points, and sharp turns are needed to fly from one guide point to the next. Avoiding such obstacles and ensuring dynamic feasibility of the trajectory are both achieved in the planning module.

A. System implementation

As shown in Fig. 11, our UAV platform is based on a tail-sitter prototype derived from our previous airframe design [9]. It is equipped with a Livox MID-360 LiDAR[3], featuring a FoV of $59^\circ \times 360^\circ$ and a detection range of $40\,\mathrm{m}$. The onboard computing is implemented on a DJI Manifold 2-C[4] ($1.8\,\mathrm{GHz}$ quad-core Intel i7 CPU), followed by an autopilot PX4 Mini[5]. Additionally, a camera is installed solely for first-person-view (FPV) video capturing. The UAV platform has a total weight of $2.7\,\mathrm{kg}$ and a wingspan of $90\,\mathrm{cm}$. As shown in Fig. 11, the LiDAR is mounted on the airframe's belly at a tilt angle of 20°, ensuring continuous forward visibility for the UAV during a transition with pitch variation of $72^\circ - 13^\circ$, which covers most maneuvers from vertical to level flights of the UAV.

B. Indoor drone laboratory

In the first flight experiment as shown in Fig. 12(a), we place obstacles in an indoor flight space measuring $25 \times 10\mathrm{m}^2$. As marked by yellow stars G1 and G2, two guide points are used, which are also the the start and terminal positions of the UAV. Two stacks of blocks are placed between the initial and terminal positions, preventing the UAV from a straight-line flight. Initially, the UAV hovers at start position G1 and detects the obstacles in front. It plans a trajectory through the free space on the right side. Because the initial Yaw angle does not match to the angle derived from the differential flatness transform on the trajectory, the tail-sitter UAV did not not immediately execute a coordinated flight on the trajectory, but instead adjusts its Yaw angle to align with the trajectory's tangent first. Then the vehicle successfully flies a S-shape trajectory while avoiding obstacles, and arrives at the terminal position G2. Readers are encouraged to watch the video demonstration for the heading alignment behaviour and the complete flights.

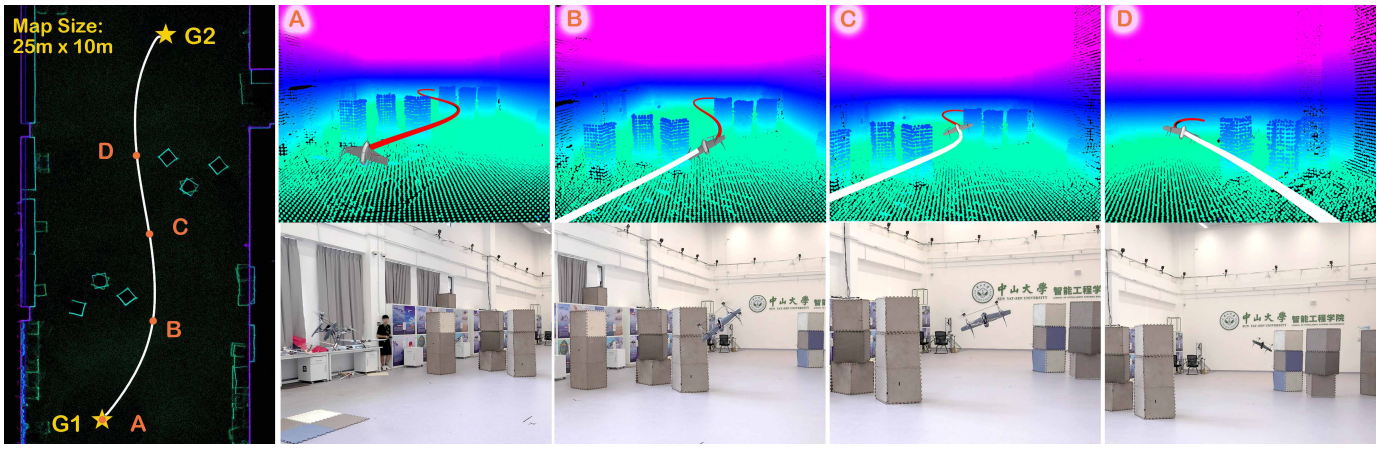
Fig. 13(a) presents the flight data. It is seen that the planned reference velocity quickly reaches the maximum of $8\,\mathrm{m/s}$ in around two seconds, while the angle of attack (AoA) α decreases from 90° to 40°. The optimized trajectory then pulls up the AoA α over 120° to perform aggressive deceleration, and finally reaches hovering status. During the flight, the reference velocity and control inputs of thrust acceleration a_T and angular velocity $\boldsymbol{\omega}$ are confined within the constraint boundaries (i.e., the shaded regions), ensuring the trajectory is feasible to be tracked by controllers and actuators. In consequence, the measurements of vehicle states (e.g., velocity and AoA) and inputs (e.g., thrust acceleration and angular velocity) track their reference accurately under the on-manifold MPC and low-level controllers.

C. Underground parking lot

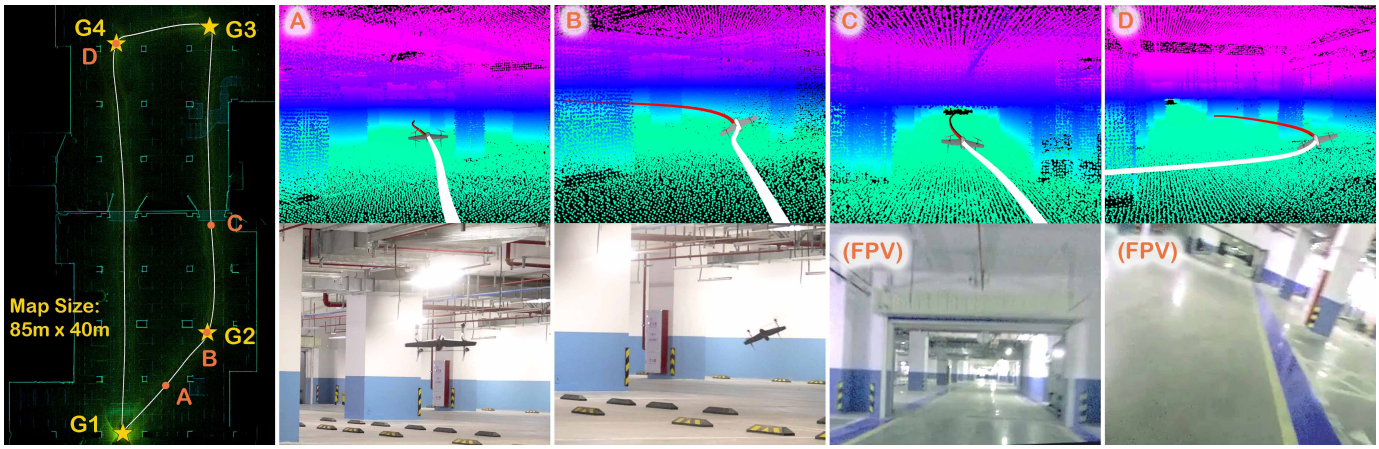
In the second flight experiment, the autonomous tail-sitter UAV flies through a more challenging underground parking lot, as shown in Fig. 12(b). We designated the start and terminal position as the same point G1, and set three other guide points G2-G4 at the turning corners, guiding the UAV to fly a loop trajectory. It is seen that the parking lot presented numerous obstacles, including dense pillars and complex pipes on the ceiling, which was about $5\,\mathrm{m}$ high. Additionally, two gates at points C and its left side limit the passage height to under $3\,\mathrm{m}$. The flight path involves consecutive sharp turns requiring dynamically feasible control inputs and consistently low passage height confining the safe space, a problem posing significant challenges to precise perception, planning, and control. As captured in the snapshots in Fig. 12(b), the tail-sitter transitions swiftly to level flight from the start position to point A, executes a banked turn at B, and successfully passes through the first gate at C by flying a straight-line trajectory. After passing through the gate at C, the UAV makes two consecutive sharp 90° turns at points G3 and G4 (D). It then flies through the second gate and returns to the origin position G1.

As shown in Fig. 13(b), the UAV velocity experiences significant variations during the $25\,\mathrm{s}$ flight, due to the complex environment and the aggressive sharp turns. The velocity peaks at $11.4\,\mathrm{m/s}$, slightly below the constraint bound of $12\,\mathrm{m/s}$. The UAV decelerates to $2\,\mathrm{m/s}$ twice for sharp turns at $11.5\,\mathrm{s}$ and $14.8\,\mathrm{s}$, because of the sharp turns at G3 and G4. After each sharp turns, the vehicle quickly accelerates above $8\,\mathrm{m/s}$. It is also seen that the AoA α also has large fluctuations ranging from 35° to 120° for frequent acceleration and deceleration. Despite the challenging environment and aggressive flight

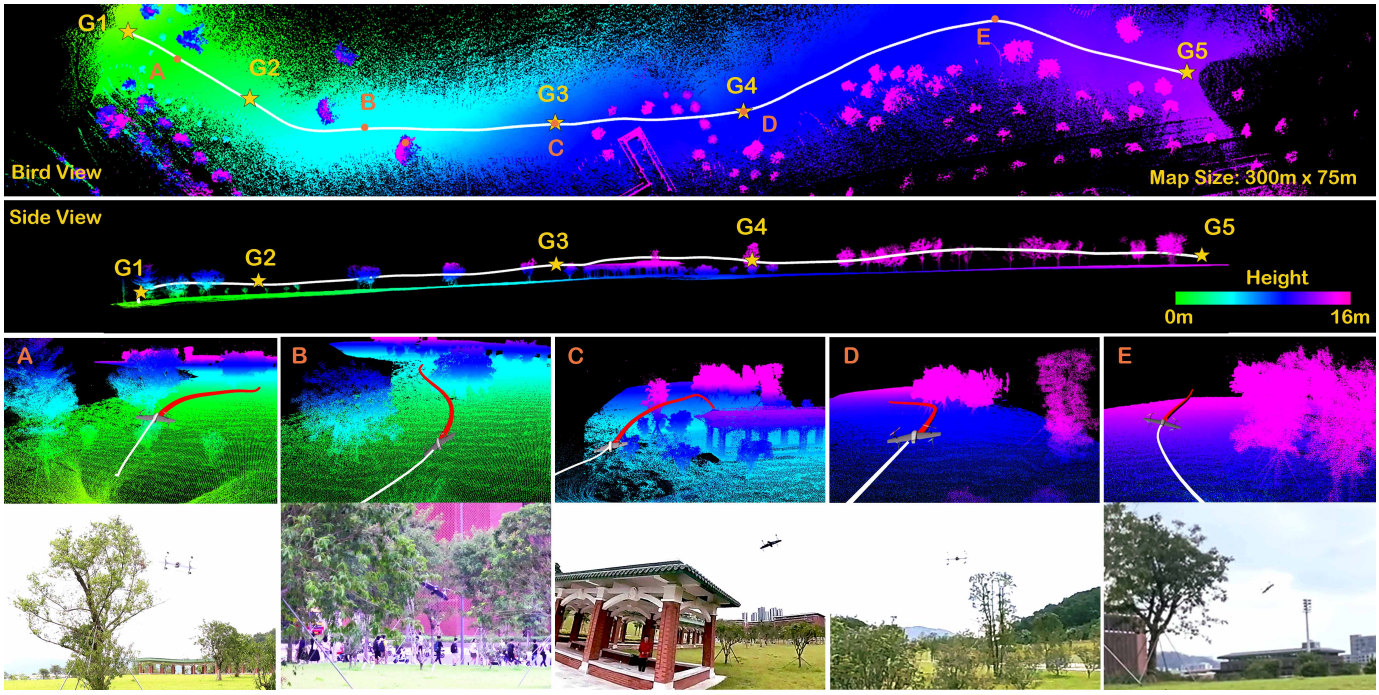
[3]https://www.livoxtech.com/mid-360
[4]https://www.dji.com/manifold-2/specs
[5]https://px4.io/



(a) Indoor drone laboratory.

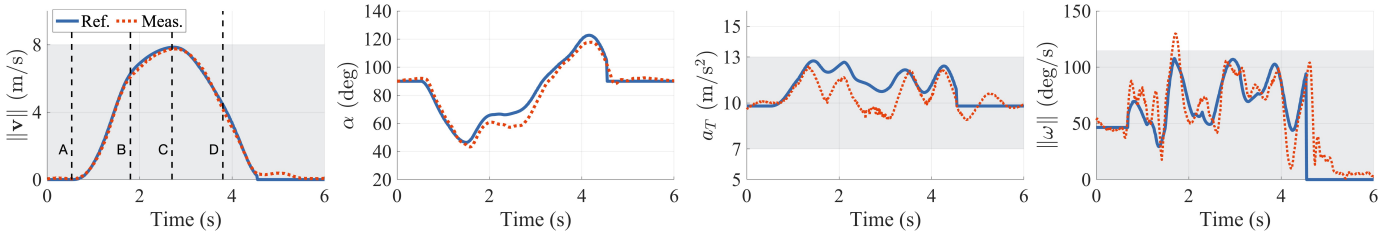


(b) Underground parking lot. Snapshot images A,B are captured by two fixed cameras, while C and D are captured by onboard FPV camera.

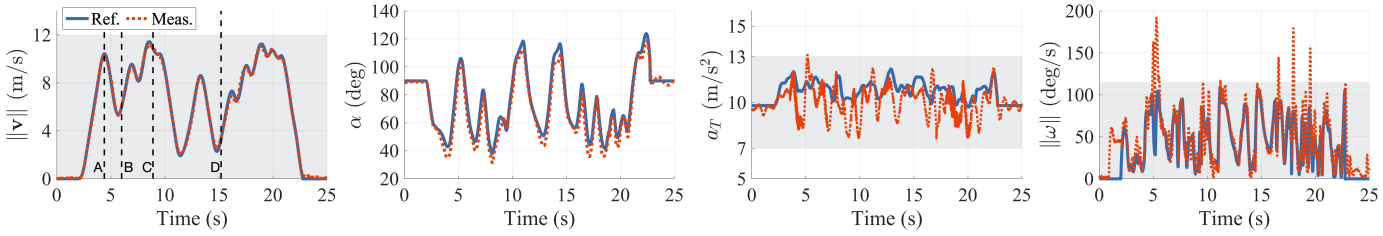


(c) Outdoor park at Sun Yat-sen University. Image C presents the local planner generates a trajectory flying over a pavilion and bushes.

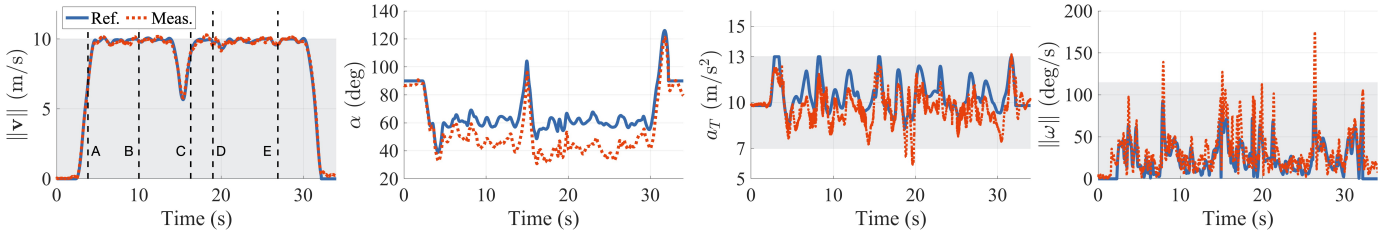
Fig. 12: Autonomous flights in real-world environments. Point-cloud maps are constructed by FAST-LIO2 in real time. Yellow stars (G1-G5) are guide points. Snapshots (A-E) including local trajectories and camera images along the flight trajectories, while their corresponding positions are marked by orange dots on the point-cloud maps.



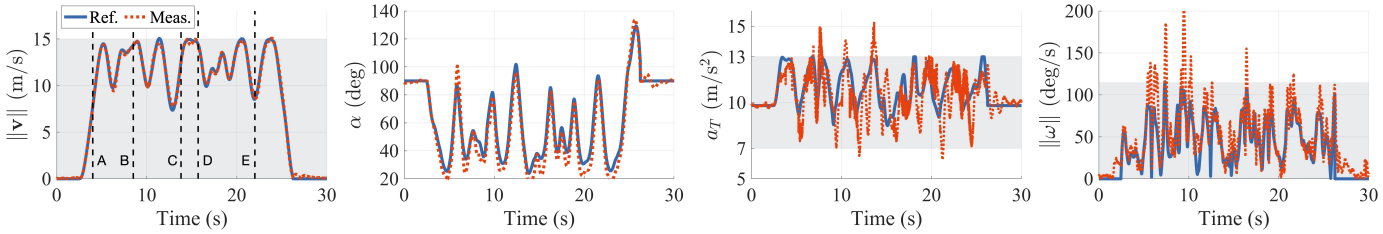
(a) Indoor drone laboratory.



(b) Underground parking lot.



(c) Outdoor park.



(d) Outdoor park at high speeds.

Fig. 13: Flight data of real-world experiments. Shaded regions denote constraint boundaries in the trajectory optimization. Black dashed lines in figures at the first column denotes the time stamps of the snapshots in Fig. 12.

maneuvers, the optimal trajectory successfully restricts the control input a_T and $\boldsymbol{\omega}$ within the constraint boundaries. Fluctuations of measured control inputs are more severe due to the MPC execution for precise control of velocity and AoA.

D. Outdoor park

In the third flight experiment, we verify the system in a large-scale outdoor environment – a park measuring $300 \times 75\mathrm{m}^2$, as shown in Fig. 12(c). We set start and terminal position G1 and G5, along with three guide points G2-G4 as depicted by yellow stars, to roughly guide the flight paths. This flight path cannot guarantee obstacle avoidance, as it is seen that there are trees blocking the path between each two guide points of G2-G5. There is also a pavilion between G3 and G4. Moreover, the park's terrain, a slope rising about $16\,\mathrm{m}$ from left to right, is evident from the varying colors of the point cloud. Therefore, the tail-sitter UAV is required to fly through the obstacles with terrain variation, posing a significant navigation challenge. The flight trajectory (the white curve in Fig. 12(c)) shows that the tail-sitter safely flies the free space beside trees at point A and B, and climbs up to fly over the pavilion and bushes between G3 and G4. The UAV then detects thick and tall trees between G4 and G5, thus it flies along the edge of these trees from point D to E, and finally arrives at termination G5.

We conduct flight tests at two different maximum velocities of $10\,\mathrm{m/s}$ and $15\,\mathrm{m/s}$, as shown in the flight data in Fig. 13(c) and 13(d). In the lower speed test, the vehicle consistently maintains the maximum speed at $10\,\mathrm{m/s}$ for most of the time, except for a slowdown to $6\,\mathrm{m/s}$ at $15\,\mathrm{s}$ right before point C. At that moment, the UAV performs an aggressive climb at height by pulling up the pitch angle (see the equivalent increase in AoA α), preparing to flying over the pavilion and bushes between C and D, as shown in Fig. 12(c). Similar to the previous two flight tests, constraints on control inputs of a_T and $\boldsymbol{\omega}$ are fully satisfied, resulting in accurate tracking in

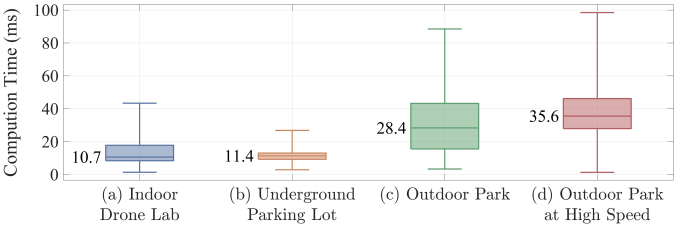


Fig. 14: Computation time of trajectory optimization in real-world experiments.

velocity. However, there is a steady-state error around 15° in the AoA α, as shown in Fig. 13(c). We suspect this error is attributed to the wind disturbance. In spite of AoA errors, the MPC is still robust to stabilize the UAV and tracks the velocity accurately.

In the high-speed test as shown in Fig. 13(d), the UAV frequently reaches the maximum velocity of $15\,\mathrm{m/s}$, but cannot maintain this speed due to the complex environments and actuator limitations. The AoA experiences more significant variations and fluctuations than the lower speed test, to perform aggressive acceleration and deceleration. The minimum and maximum AoA reach 20° and 130°, fully exploiting the FoV limitation of LiDAR and fight envelope. The optimal control inputs are also constrained within the required boundaries. Although the measured control executions and AoA variation are apparently increased, the system still presents accurate tracking performance in velocity and AoA, owing to the dynamically feasible trajectory optimization and real-time MPC controllers.

E. Computation time

The time consumption of trajectory optimization in these four real-world flight tests are presented in Fig. 14. The median computation time rises from $10.7\,\mathrm{ms}$ to $35.6\,\mathrm{ms}$ along test (a) to (d), along the increasing environmental complexity and trajectory aggressiveness. The maximum time to solve a trajectory optimization arrives at $98.5\,\mathrm{ms}$ in the forth test, but is still less than the half of the planning interval of $200\,\mathrm{ms}$, which is sufficient for real-time planning at $5\,\mathrm{Hz}$.

IX. Discussion

In this section, we discuss the limitation and extension of the proposed system.

A. LiDAR selection and installation

The LIVOX MID-360 LiDAR, chosen for its light weight, wide FoV, and capability to capture point clouds from a considerable distance of $40\,\mathrm{m}$, is integral to our prototype's design for autonomous flights. It offers the best thrust-to-weight ratio among the lightest commercial LiDAR sensors. However, as shown in Fig. 4, this sensor has a blind zone at its top. The current installation prioritizes continuous forward FoV, crucial for obstacle detection and avoidance during transition and level flights, at the cost of sacrificing the visibility along the belly. This setup, while effective for our intended demonstrations, is not optimal for applications like geographic surveying and infrastructure inspection, where terrain mapping beneath the airframe is important. In the future, the development of lightwight and wide-FoV LiDAR sensors specifically tailored for UAV applications, could expand the practicality and applicability of our system in real-world scenarios.

B. LiDAR SLAM

LiDAR-based SLAM, leveraging IMU data and point cloud scans, provides accurate and low-latency state estimation as well as dense 3-D maps. However, LiDAR-based SLAM approaches easily fail in environments with few available geometric features, such as open spaces and narrow tunnels. This presents a challenge for tail-sitter UAVs designed to execute multi-scale missions across diverse environments, ranging from cluttered environments to open space at high altitude, where LiDAR's effectiveness diminishes. In this paper, our focus is on trajectory generation with obstacle avoidance for autonomous flights, thus we particularly consider obstacle-dense environments where the deployed LIO, FAST-LIO2, can perform reliably. Moreover, we also found a gap in real-time multiple-sensor fusion and SLAM research that integrates LiDAR, global navigation satellite system (GNSS) and visual cameras. Such integration could be more robust for diverse environments, potentially beneficial for VTOL UAV applications.

C. Uncoordinated flights

We consider a coordinated flight trajectory generation framework in this paper. As mentioned before, coordinated flight condition offers benefits in terms of aerodynamic efficiency, moment stability, and model reduction. However, from practical perspectives, there are specific scenarios, such as sideways flights through narrow gaps, where uncoordinated flight at low speeds may be necessary. The differential flatness transform utilized in our approach supports the specification of an additional yaw angle, in singularity conditions where the airspeed is zero $\|\mathbf{v}_a\| = 0$. This allows low-speed uncoordinated flights by incorporating an extra yaw planning. Hence, the proposed framework can be extended to accommodate such specialized flight maneuvers, enhancing its applicability in diverse operational scenarios.

D. Safe trajectory planning

Our proposed trajectory generation framework plans trajectories in both known-free and unknown spaces. However, there are scenarios where trajectory safety may not be fully guaranteed. For example, unobserved hidden obstacles could exist within the safe flight corridors in complex unknown spaces, and their sudden detection could significantly complicate the trajectory optimization process. In cases where the optimization fails, the reference trajectory may not be updated, potentially leading to collisions. These challenges have been addressed in [67], which suggests planning an additional safe trajectory exclusively in the known-free space. If the primary trajectory optimization fails, the system can switch to this safe trajectory which is absolutely obstacle-free. We aim to

incorporate this safety strategy into future tail-sitter planning frameworks, but adding an extra trajectory planning layer would introduce a considerable computational burden.

E. EFOPT for general NLPs

The proposed EFOPT solver, characterized by its efficient feasibility-first approach, is formulated in a form of general NLPs. Hence, it is potentially suitable to a wide range of NLP applications where finding a feasible solution is the primary concern. However, it's important to note that the current version of EFOPT is tailored for relatively small-scale optimizations. It relies on dense matrix calculations using the Eigen library[6]. As a result, the current EFOPT is more suitable to NLPs involving dozens of variables. Future versions could be adapted to handle larger-scale problems, broadening its applicability.

X. Conclusion

We have proposed a fully autonomous tail-sitter UAV capable of high-speed flights in unknown cluttered environments. The hardware platform and algorithm framework integrated with perception, planning and control technologies, are presented in detail. To address the challenges of constrained, nonlinear trajectory optimization in real-time, we developed EFOPT, an efficient, feasibility-first solver, enabling online planing for high-speed, collision-free and dynamically-feasible trajectory generation. The performance of EFOPT benchmarked against various off-the-shelf NLP solvers through simulations, demonstrates a superior computational efficiency and feasibility satisfaction in the context of tail-sitter trajectory optimizations. Extensive flight experiments in diverse real-world environments provide convincing evidence of the proposed tail-sitter UAV's effectiveness, indicating the system's promising potential for a broad spectrum of practical applications.

Acknowledgment

This work was supported in part by Hong Kong RGC ECS under grant 27202219 and in part by DJI donation.

References

[1] S. Carlson, "A hybrid tricopter/flying-wing vtol uav," in *52nd Aerospace Sciences Meeting*, 2014, p. 0016.

[2] U. Ozdemir, Y. O. Aktas, A. Vuruskan, Y. Dereli, A. F. Tarhan, K. Demirbag, A. Erdem, G. D. Kalaycioglu, I. Ozkol, and G. Inalhan, "Design of a commercial hybrid vtol uav system," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1, pp. 371–393, 2014.

[3] E. Çetinsoy, E. Sirimoğlu, K. T. Öner, C. Hancer, M. Ünel, M. F. Akşit, I. Kandemir, and K. Gülez, "Design and development of a tilt-wing uav," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 19, no. 5, pp. 733–741, 2011.

[4] J. McKenna, "One step beyond, rotor wing," 2007.

[5] R. Park, "Arcturus uav upgrades the jump15 vtol uav," *Airlines & Aviation, Aerospace & Defense*, 2014.

[6] H. Gu, X. Lyu, Z. Li, S. Shen, and F. Zhang, "Development and experimental verification of a hybrid vertical take-off and landing (vtol) unmanned aerial vehicle (uav)," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2017, pp. 160–169.

[7] C. De Wagter, R. Ruijsink, E. J. Smeur, K. G. van Hecke, F. van Tienen, E. van der Horst, and B. D. Remes, "Design, control, and visual navigation of the delftacopter vtol tail-sitter uav," *Journal of Field Robotics*, vol. 35, no. 6, pp. 937–960, 2018.

[8] R. Ritz and R. D'Andrea, "A global controller for flying wing tailsitter vehicles," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2731–2738.

[9] H. Gu, X. Cai, J. Zhou, Z. Li, S. Shen, and F. Zhang, "A coordinate descent method for multidisciplinary design optimization of electric-powered winged uavs," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 1189–1198.

[10] E. A. Tal and S. Karaman, "Global trajectory-tracking control for a tailsitter flying wing in agile uncoordinated flight," in *AIAA AVIATION 2021 FORUM*, 2021, p. 3214.

[11] A. Frank, J. McGrew, M. Valenti, D. Levine, and J. How, "Hover, transition, and level flight control design for a single-propeller indoor airplane," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007, p. 6318.

[12] A. Oosedo, S. Abiko, A. Konno, T. Koizumi, T. Furui, and M. Uchiyama, "Development of a quad rotor tail-sitter vtol uav without control surfaces and experimental verification," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 317–322.

[13] X. Lyu, H. Gu, J. Zhou, Z. Li, S. Shen, and F. Zhang, "A hierarchical control approach for a quadrotor tail-sitter vtol uav and experimental verification," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5135–5141.

[14] J. Zhou, X. Lyu, X. Cai, Z. Li, S. Shen, and F. Zhang, "Frequency domain model identification and loop-shaping controller design for quadrotor tail-sitter vtol uavs," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 1142–1149.

[15] X. Lyu, J. Zhou, H. Gu, Z. Li, S. Shen, and F. Zhang, "Disturbance observer based hovering control of quadrotor tail-sitter vtol uavs using H_∞ synthesis," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2910–2917, 2018.

[16] B. Li, W. Zhou, J. Sun, C.-Y. Wen, and C.-K. Chen, "Development of model predictive controller for a tail-sitter vtol uav in hover flight," *Sensors*, vol. 18, no. 9, p. 2859, 2018.

[17] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

[18] A. Lambregts, "Vertical flight path and speed control autopilot design using total energy principles," in *Guidance and Control Conference*, 1983, p. 2239.

[19] S. Park, J. Deyst, and J. How, "A new nonlinear guidance logic for trajectory tracking," in *AIAA guidance, navigation, and control conference and exhibit*, 2004, p. 4900.

[20] S. Verling, B. Weibel, M. Boosfeld, K. Alexis, M. Burri, and R. Siegwart, "Full attitude control of a vtol tailsitter uav," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 3006–3012.

[21] X. Lyu, H. Gu, Y. Wang, Z. Li, S. Shen, and F. Zhang, "Design and implementation of a quadrotor tail-sitter vtol uav," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3924–3930.

[22] K. Kita, A. Konno, and M. Uchiyama, "Transition between level flight and hovering of a tail-sitter vertical takeoff and landing aerial robot," *Advanced Robotics*, vol. 24, no. 5-6, pp. 763–781, 2010.

[23] Y. Jung and D. H. Shim, "Development and application of controller for transition flight of tail-sitter uav," *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1, pp. 137–152, 2012.

[24] W. Xu, H. Gu, and F. Zhang, "Acceleration based iterative learning control for pugachev's cobra maneuver with quadrotor tailsitter vtol uavs," *work*, vol. 7, p. 12, 2019.

[25] R. Naldi and L. Marconi, "Optimal transition maneuvers for a class of v/stol aircraft," *Automatica*, vol. 47, no. 5, pp. 870–879, 2011.

[26] A. Oosedo, S. Abiko, A. Konno, and M. Uchiyama, "Optimal transition from hovering to level-flight of a quadrotor tail-sitter uav," *Autonomous Robots*, vol. 41, no. 5, pp. 1143–1159, 2017.

[27] B. Li, J. Sun, W. Zhou, C.-Y. Wen, K. H. Low, and C.-K. Chen, "Transition optimization for a vtol tail-sitter uav," *IEEE/ASME transactions on mechatronics*, vol. 25, no. 5, pp. 2534–2545, 2020.

[28] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[29] J. M. Barth, J.-P. Condomines, M. Bronz, J.-M. Moschetta, C. Join, and M. Fliess, "Model-free control algorithms for micro air vehicles

[6] https://eigen.tuxfamily.org

with transitioning flight capabilities," *International Journal of Micro Air Vehicles*, vol. 12, p. 1756829320914264, 2020.
[30] Z.-H. Cheng and H.-L. Pei, "Transition analysis and practical flight control for ducted fan fixed-wing aerial robot: Level path flight mode transition," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3106–3113, 2022.
[31] D. Pucci, T. Hamel, P. Morin, and C. Samson, "Nonlinear control of aerial vehicles subjected to aerodynamic forces," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 4839–4846.
[32] L. R. Lustosa, "The phi-theory approach to flight control design of hybrid vehicles," Ph.D. dissertation, PhD thesis, ISAE-SUPAERO, 2017.
[33] E. J. Smeur, M. Bronz, and G. C. de Croon, "Incremental control and guidance of hybrid aircraft applied to a tailsitter unmanned air vehicle," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 2, pp. 274–287, 2020.
[34] E. Tal and S. Karaman, "Global incremental flight control for agile maneuvering of a tailsitter flying wing," *Journal of Guidance, Control, and Dynamics*, vol. 45, no. 12, pp. 2332–2349, 2022.
[35] E. Tal, G. Ryou, and S. Karaman, "Aerobatic trajectory generation for a vtol fixed-wing aircraft using differential flatness," *IEEE Transactions on Robotics*, 2023.
[36] J. Zhou, X. Lyu, Z. Li, S. Shen, and F. Zhang, "A unified control method for quadrotor tail-sitter uavs in all flight modes: Hover, transition, and level flight," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4835–4841.
[37] G. Lu, Y. Cai, N. Chen, F. Kong, Y. Ren, and F. Zhang, "Trajectory generation and tracking control for aggressive tail-sitter flights," *The International Journal of Robotics Research*, 2023.
[38] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.
[39] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 969–1002, 2015.
[40] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.
[41] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 5774–5781.
[42] Y. Ren, S. Liang, F. Zhu, G. Lu, and F. Zhang, "Online whole-body motion planning for quadrotor using multi-resolution search," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1594–1600.
[43] K. Hang, X. Lyu, H. Song, J. A. Stork, A. M. Dollar, D. Kragic, and F. Zhang, "Perching and resting—a paradigm for uav maneuvering with modularized landing gears," *Science Robotics*, vol. 4, no. 28, p. eaau6637, 2019.
[44] Y.-H. Hsiao, S. Bai, Y. Zhou, H. Jia, R. Ding, Y. Chen, Z. Wang, and P. Chirarattananon, "Energy efficient perching and takeoff of a miniature rotorcraft," *Communications Engineering*, vol. 2, no. 1, p. 38, 2023.
[45] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE transactions on robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.
[46] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.
[47] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2016.
[48] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.
[49] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.
[50] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.
[51] Y. Ren, F. Zhu, W. Liu, Z. Wang, Y. Lin, F. Gao, and F. Zhang, "Bubble planner: Planning high-speed smooth quadrotor trajectories using receding corridors," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 6332–6339.
[52] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, 2022.
[53] E. DIJKSTRA, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
[54] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[55] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.
[56] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.
[57] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1300–1313, 2020.
[58] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
[59] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.
[60] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
[61] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkit—an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.
[62] M. A. Patterson and A. V. Rao, "Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, pp. 1–37, 2014.
[63] L. Tang, H. Wang, Z. Liu, and Y. Wang, "A real-time quadrotor trajectory planning framework based on b-spline and nonuniform kinodynamic search," *Journal of Field Robotics*, vol. 38, no. 3, pp. 452–475, 2021.
[64] 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 Transactions on Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.
[65] V. Usenko, L. Von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 215–222.
[66] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.
[67] J. Tordesillas, B. T. Lopez, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," in *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2019, pp. 1934–1940.
[68] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research: The 16th International Symposium ISRR*. Springer, 2016, pp. 649–666.
[69] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.
[70] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.
[71] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 2006.
[72] S. G. Johnson, "The NLopt nonlinear-optimization package," https://github.com/stevengj/nlopt, 2007.
[73] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, pp. 1–17, 2022.
[74] W. Sun, G. Tang, and K. Hauser, "Fast uav trajectory optimization using bilevel optimization with analytical gradients," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2010–2024, 2021.
[75] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential

convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
[76] X. Lyu, H. Gu, J. Zhou, Z. Li, S. Shen, and F. Zhang, "Simulation and flight experiments of a quadrotor tail-sitter vertical take-off and landing unmanned aerial vehicle with wide flight envelope," *International Journal of Micro Air Vehicles*, vol. 10, no. 4, pp. 303–317, 2018.
[77] B. Etkin and L. D. Reid, *Dynamics of flight*. Wiley New York, 1959, vol. 2.
[78] L. J. Clancy, *Aerodynamics*. John Wiley & Sons, 1975.
[79] Y. Ren, Y. Cai, F. Zhu, S. Liang, and F. Zhang, "Rog-map: An efficient robocentric occupancy grid map for large-scene and high-resolution lidar-based motion planning," *arXiv preprint arXiv:2302.14819*, 2023.
[80] 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, no. 3, pp. 1688–1695, 2017.
[81] G. Lu, W. Xu, and F. Zhang, "On-manifold model predictive control for trajectory tracking on robotic systems," *IEEE Transactions on Industrial Electronics*, 2022.
[82] W. Xu, H. Gu, Y. Qing, J. Lin, and F. Zhang, "Full attitude control of an efficient quadrotor tail-sitter vtol uav with flexible modes," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 542–550.
[83] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.
[84] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2000.
[85] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.