Fast-Racing: An Open-source Strong Baseline for $\mathrm{SE}(3)$ Planning in Autonomous Drone Racing

Zhichao Han, Zhepei Wang, Chao Xu, and Fei Gao

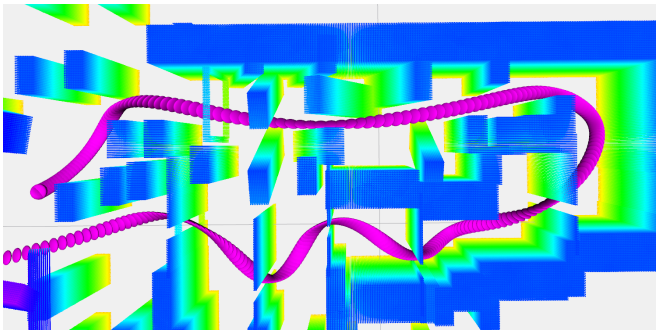
***Abstract*— With the autonomy of aerial robots advances in recent years, autonomous drone racing has drawn increasing attention. In a professional pilot competition, a skilled operator always controls the drone to agilely avoid obstacles in aggressive attitudes, for reaching the destination as fast as possible. Autonomous flight like elite pilots requires planning in $\mathrm{SE}(3)$, whose non-triviality and complexity hindering a convincing solution in our community by now. To bridge this gap, this paper proposes an open-source baseline, which includes a high-performance $\mathrm{SE}(3)$ planner and a challenging simulation platform tailored for drone racing. We specify the $\mathrm{SE}(3)$ trajectory generation as a soft-penalty optimization problem, and speed up the solving process utilizing its underlying parallel structure. Moreover, to provide a testbed for challenging the planner, we develop delicate drone racing tracks which mimic real-world set-up and necessities planning in $\mathrm{SE}(3)$. Besides, we provide necessary system components such as common map interfaces and a baseline controller, to make our work plug-in-and-use. With our baseline, we hope to future foster the research of $\mathrm{SE}(3)$ planning and the competition of autonomous drone racing.**

I. Introduction

As the capability of sensing and computing advances, the community is continuously pushing the boundary of autonomy of unmanned aerial vehicles (UAVs), making them fly into many industrial and commercial scenarios. In recent years, researchers have put attentions to a challenging problem, autonomous drone racing, such as IROS (Intelligent Robots and Systems) Autonomous Drone Race [1] and AlphaPilot [2]. Drone racing [3] is a competition where human operators are required to pilot the drone to avoid obstacles as fast as possible in a given tough track.

Autonomous drone racing interleaves many technologies including dynamic modeling, flight control, state estimation, and planning; and provides a good testbed to trial these modules in extreme situations. As the maturity of UAV design and control, state estimation, and planning have been the bottleneck of drone racing for a long time. Recently, several visual-inertial odometry (VIO) benchmarks and datasets [4] [5] for high-speed drone racing have been presented publicly, which significantly boost state estimation for aggressive motion.

However, there is always a lack of complete benchmarks for the planning module in drone racing. Investigating the motion planning problem for drone racing, it is necessary to plan in $\mathrm{SE}(3)$ for fully exploiting the maneuverability of drones to pass challenging tracks with high aggressiveness. Imagining the drone racing situation, professional pilots always control the drone to aggressively fly against obstacles in a large attitude to reach the destination speedily[1]. Besides, birds flying at high speeds in a dense forest, such as skylarks, always alternate their attitudes to avoid obstacles. All evidence shows that $\mathrm{SE}(3)$ planning is imperative, no matter for human piloting or biological flight in nature. Nevertheless, there exists no open-sourced baseline for $\mathrm{SE}(3)$ planning at the moment, which is not conducive to $\mathrm{SE}(3)$ planning research. The reason is that $\mathrm{SE}(3)$ planning is much more complicated than traditional planning in $\mathbb{R}^3$ because the former requires group operations while conventional optimizers require decision variables in Euclidean space.Moreover, an appropriate simulation environment is necessary to verify the planning algorithm. However, existing autonomous drone



(a) The drone flies through narrow gaps and passages.

(b) Visualization of the SE(3) motion in our simulator.

Fig. 1: The two figures show the capability of our $\mathrm{SE}(3)$ planner and one of our environments.

All authors are with the State Key Laboratory of Industrial Control Technology, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China and with Huzhou Institute of Zhejiang University, Huzhou 313000, China. (*Corresponding author: Fei Gao; Chao Xu)*

E-mail: {zhichaohan, fgaoaa}@zju.edu.cn

[1]Drone Racing League `https://www.youtube.com/watch?v=GTifvVZBNWs`

racing simulation environments are not challenging for the planning module, such as tracks in [6]. To address the above issues, we propose a strong baseline for $\mathrm{SE}(3)$ planning. Our $\mathrm{SE}(3)$ planner generates safe trajectories with arbitrary $\mathrm{SE}(3)$ constraints, as shown in Fig.1a. Then, we build highly-challenging racing tracks and finish them with our planner.

In our paper, we firstly model a quadrotor by its shape, then derive the collision-free condition for the whole trajectory and specify the $\mathrm{SE}(3)$ trajectory optimization problem formulation. Moreover, following [7], we simplify the trajectory optimization without sacrificing optimality so that it is solved cheaply and efficiently. Furthermore, to improve time efficiency, we implement a parallel architecture that calculates gradients at all points along a trajectory simultaneously. In addition, we build a highly challenging drone racing simulation platform that contains tracks, map interfaces, and a controller. Last but not least, we apply our planner in the drone racing tracks as the baseline and show the results for comparisons. To sum up, contributions of this paper are:

1) We propose a general $\mathrm{SE}(3)$ collision-free constraint form for multirotor in all shapes.
2) We implement a high-performance trajectory optimization based on parallel computing.
3) We present a plug-in-and-use simulation platform specially designed for $\mathrm{SE}(3)$ planning.
4) We release the proposed $\mathrm{SE}(3)$ planner and simulation publicly to the community, for fostering future research in $\mathrm{SE}(3)$ planning and autonomous drone racing[2].

II. Related Works

A. Autonomous Drone Racing

Drone racing attracts wide public attention in recent years and also develops significantly with the improvement of algorithms and computing. The AlphaPilot Challenge [2] is the largest one recently, which builds the qualification round based on the MIT FlightGoggles simulation framework [8]. Then, the winners are required to finish the hardware implementation in a later competition. In addition, IROS conference also hosts an annual hardware drone racing challenge [1] since 2016. The competition consists of navigating autonomously through a number of gates. In IROS 2018 autonomous drone race, the UZH team showed a winning performance[3] whose drone took 30 seconds to complete the track. The Airsim Drone Racing Lab [6] introduces a drone racing simulation framework, for bridging the gap between autonomous systems and researchers who specialize in software. Moreover, it hosts a competition with three tiers, which focus on perception, trajectory planning, and control, respectively.

Drone racing involves lots of aspects such as flight control, state estimation, and motion planning. High-quality and accessible benchmarks are key to fostering the research community forward. In the domain of state estimation, open-sourced datasets (EuRoC [9], MVSEC [10], Zurich Urban MAV Dataset [11]) boost the development of visual-based algorithms in extreme situations. In addition, the recently released Blackbird dataset [4] and UZH-FPV dataset [5], which both target high-speed drone racing, build a foundation for aggressive motion estimation.

B. $\mathrm{SE}(3)$ Planning

Motion planning is a crucial part of drone racing and has made excellent progress in recent years. Trajectories of UAVs or other differentially flat systems [12] are usually parameterized as piecewise polynomials since their derivatives can be used to obtain explicit expressions for the system states and control inputs [13]. Mellinger et al. [12] present a trajectory smoothness metric, and use anchor waypoints along the trajectory to ensure collision avoidance. Gao et al. [14] construct a flight corridor that consists of a sequence of connected free-space 3-D grids as the safety constraint, and use Bernstein basis polynomial to represent the trajectory. Although these works [12] [14] [15] [16] generate trajectories efficiently, they require planning in a configuration space where the drone's attitude is neglected. As described in Sect. I, $\mathbb{R}^3$ planners are much inferior when applied to extremely challenging environments or trying to beat elite pilots. These environments, for instance, contain multiple narrow and tilted gaps that are only traversable if the drone drifts with large attitudes. To this end, several works [17] [18] [19] have been proposed. However, these works all over-simplify the $\mathrm{SE}(3)$ Planning problem, and only generate trajectories with given attitude constraints. Instead of exploring the whole $\mathrm{SE}(3)$ solution space, they are dominated by the boundary conditions of attitude, thus only applicable to rather simple environments such as only one titled gap exist. Based on kinodynamic searching, Liu et al. [20] present a $\mathrm{SE}(3)$ planner which is resolution complete. However, this method is not applicable to challenging situations in practice. Since it suffers from a combinatorial explosion, a fine resolution quickly induces unacceptable computing overhead, in both time and memory. Recently, Wang et al. [7] propose a planning framework that generates high-quality SE(3) trajectories even in complicated scenes. This method shows an efficient way to handle general spatial and temporal constraints in trajectory generation for multirotor. Still, Wang et al. [7] do not fully squeeze the potential and finalize the implementation of their method, making it consume unnecessary computation in large-scale problems. In this paper, we exploit the parallel nature of the optimization program [7], and further extend it to incorporate a drone with general shapes. In this way, we significantly improve the performance and scalability of the $\mathrm{SE}(3)$ trajectory planning, and finally present an easy-to-use baseline.

III. A High-performance $\mathrm{SE}(3)$ Planner

In this section, we introduce our $\mathrm{SE}(3)$ trajectory planner with explicit spatial constraints. We present a general way

[2] https://github.com/ZJU-FAST-Lab/Fast-Racing

[3] IROS 2018 Autonomous Drone Race https://www.youtube.com/watch?v=9AvJ3-n-82w

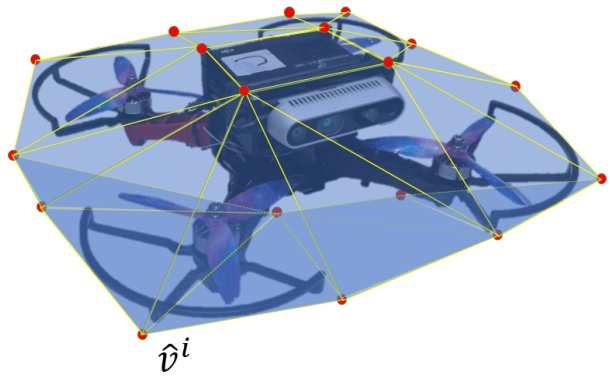


Fig. 2: The drone is completely wrapped in a convex polyhedron. The red points are the vertices of the bounding volume.

to model a quadrotor and derive $\mathrm{SE}(3)$ optimization formulation with attitude constraints. Then, we propose a parallel architecture to solve the optimization problem efficiently.

A. Safety Constraint

We use the flight corridor as the explicit spatial constraint to ensure obstacle avoidance. The generation steps are as follows:

1) A path search is adopted to obtain a feasible path from the start to the goal.
2) We start at the first point w_p of the path and follow the path to find the farthest unblocked point w_f but not exceeding the limit distance. (The unblocked point indicates the line between w_p and w_f is collision-free.)
3) RILS [21] is used to generate a convex polyhedron based on the line in step(2).
4) The interval point of the line is regarded as w_p again. Then repeating back to step(2) until the endpoint is included in the convex hull.

The whole algorithm is shown in Alg.1.

B. Problem Formulation

Thanks to the differential flatness property, the attitude of a quadrotor R_b can be written as an algebraic function of finite derivatives of its four differentially flat outputs $O_f = [p_x, p_y, p_z, \psi]^{\mathrm{T}} := [\mathbf{x}^{\mathrm{T}}, \psi]^{\mathrm{T}}$ explicitly. Here $\mathbf{x} = [p_x, p_y, p_z]^{\mathrm{T}}$ is the coordinate of the center of mass in the world frame, and ψ is the yaw angle. Utilizing the result from [12], R_b is as follows:

$$f_b = \ddot{\mathbf{x}} + g_w e_3, \tag{1}$$
$$r_{bz} = f_b / \left\| f_b \right\|_2, \tag{2}$$
$$r_{ix} = [\cos\psi, \sin\psi, 0]^{\mathrm{T}}, \tag{3}$$
$$r_{by} = \frac{r_{bz} \times r_{ix}}{\left\| r_{bz} \times r_{ix} \right\|_2}, \tag{4}$$
$$r_{bx} = r_{by} \times r_{bz}, \tag{5}$$
$$R_b = [r_{bx}, r_{by}, r_{bz}], \tag{6}$$

where g_w is the gravitational acceleration and $e_3 = [0, 0, 1]^{\mathrm{T}}$. Then, we model the quadrotor as a convex polyhedron which can be generated by using k-DoP (the Boolean intersection of extents along k directions) or specified manually, as shown in Fig.2. We define the i-th vertex $\hat{v}^i$ of the bounding volume at time t as:

$$\hat{v}^i(t) = R_b(t) q_v^i + \mathbf{x}(t) \quad \forall i \in \{1, 2, \cdots, K\}, \tag{7}$$

where K is the number of vertexs and $q_v^i = [q_{vx}^i, q_{vy}^i, q_{vz}^i]^{\mathrm{T}}$ is the coordinate of the i-th vertex in the body frame. Note q_v^i is constant once the quadrotor model is identified.

Algorithm 1 Flight Corridor Generation

```
Initialize()
path ← PathSearch(start,end)
w_p ← start
while True do
    w_f ← FindFarPoint(w_p,path,MaxDis)
    FlightCorridor.pushback(ConvexGen(w_p, w_f))
    if end in FlightCorridor then
        return FlightCorridor
    end if
    w_p ← IntervalPt(w_p,w_f)
end while
```

In Sect. III-A, we build a flight corridor as a series of closed convex polyhedra which can be defined as:

$$\begin{aligned} &\mathcal{P}_j = \left\{ \mathbf{x} \in \mathbb{R}^3 | (\mathbf{x} - \hat{p}_j^k)^{\mathrm{T}} \vec{n}_j^k \le 0, k = 1, 2, \cdots, N_j \right\}, \\ &\forall j \in \{1, 2, \cdots, M\}, \end{aligned} \tag{8}$$

where M is the number of polyhedra and each polyhedron $\mathcal{P}_j$ consists of N_j hyperplanes which can be represented by a normal vector $\vec{n}_j^k$ outwards and a point $\hat{p}_j^k$ on it. Constraining the whole body of a quadrotor inscribed by a convex polyhedron written as:

$$\begin{aligned} &(\vec{n}_j^k)^{\mathrm{T}} (\hat{v}^i(t) - \hat{p}_j^k) \le 0, \\ &\forall t \in [T_{j-1}, T_j], \forall j \in \{1, \cdots, M\}, \\ &\forall i \in \{1, \cdots, K\}, \forall k \in \{1, \cdots, N_j\}. \end{aligned} \tag{9}$$

We adopt piecewise-polynomials to represent our trajectory in each dimension out of x, y, z:

$$\mathbf{x}(t) = \begin{cases} p_1(t) = \sum_{n=0}^{2s-1} c_{1n} (t - T_0)^n & T_0 \le t < T_1, \\ \vdots & \vdots \\ p_M(t) = \sum_{n=0}^{2s-1} c_{jn} (t - T_{M-1})^n & T_{M-1} \le t < T_M. \end{cases} \tag{10}$$

The desired trajectory is optimized on both the control effort and time regularization to ensure smoothness and aggressiveness. Moreover, for enforcing safety, spatial constraints are added along the whole trajectory. Also, feasibility constraints are necessary to guarantee that the trajectory is physically

Fig. 3: Some features of our drone racing tracks: Left - A gap which is narrower than drone's diameter in track(a), Middle - A few snapshots from our racing environments, Right - The goal gate.

feasible. The optimization problem formulation is:

$$\begin{aligned}\min_{u_j(t),\forall j}\ H &= \sum_{j=1}^{M}\int_{T_{j-1}}^{T_j} u_j(t)^{\mathrm{T}}u_j(t)dt + \rho(T_M - T_0)\\ &+ \underbrace{W_v\sum_{j=1}^{M}\int_{T_{j-1}}^{T_j}\mathcal{K}\left(||p_j^{(1)}(t)||_2^2 - v_{max}^2\right)dt}_{\textit{velocity feasible penalty}}\\ &+ \underbrace{W_a\sum_{j=1}^{M}\int_{T_{j-1}}^{T_j}\mathcal{K}\left(||p_j^{(2)}(t)||_2^2 - a_{max}^2\right)dt}_{\textit{acceleration feasible penalty}}\\ &+ \underbrace{W_c\sum_{j=1}^{M}\int_{T_{j-1}}^{T_j}\sum_{i=1}^{K}\sum_{k=1}^{N_j}\mathcal{K}\left((\hat{v}^i(t)-\widetilde{p}_j^k)^{\mathrm{T}}\vec{n}_j^k\right)dt}_{\textit{collision-free penalty}},\end{aligned} \tag{11}$$

$$s.t.\quad u_j(t) = p_j^{(s)}(t),\ \forall t\in[T_{j-1},T_j], \tag{12}$$

$$p_j^{[d]}(T_j) = p_{j+1}^{[d]}(0),\ p_j^{[\widetilde{d}]}(T_j) = \overline{p}_j, \tag{13}$$

$$p_j(T_j)\in\mathcal{C}_j\cap\mathcal{C}_{j+1},\ T_j - T_{j-1} > 0, \tag{14}$$

$$\forall j\in\{1,\cdots,M\}, \tag{15}$$

$$p_1^{[s-1]}(0) = \overline{p}_0,\ p_M^{[s-1]}(T_M) = \overline{p}_f, \tag{16}$$

where $\mathcal{K}(x) = \max(x,0)^3$ is the cubic penalty, ρ is the weight of time regularization, W_v, W_a, W_c are the corresponding penalty weights, $\overline{p}_j \in \mathbb{R}^{3(\widetilde{d}+1)}$ is the interval condition, $\overline{p}_0, \overline{p}_f \in \mathbb{R}^{3s}$ are the initial and final states, respectively.

C. Gradient Calculation

We denote the j-th piece of the polynomial trajectory as:

$$p_j(t) = \mathbf{C}_j^{\mathrm{T}}\beta(t - T_{j-1}),\ t\in[T_{j-1},T_j] \tag{17}$$

where $\mathbf{C}_j\in\mathbb{R}^{2s\times 3}$ is the coefficient matrix of the piece and $\beta(t) = [1,t,\cdots,t^{2s-1}]^{\mathrm{T}}$ is the time vector. We define the coefficient matrix $\mathbf{C}$ and time allocation vector $\mathbf{T}$:

$$\mathbf{C} = (\mathbf{C}_1^{\mathrm{T}},\mathbf{C}_2^{\mathrm{T}},\cdots,\mathbf{C}_M^{\mathrm{T}})^{\mathrm{T}}\in\mathbb{R}^{2Ms\times 3}, \tag{18}$$

$$\mathbf{T} = (T_1 - T_0, T_2 - T_1,\cdots,T_M - T_{M-1})^{\mathrm{T}} \tag{19}$$

$$= (\delta_1,\delta_2,\cdots,\delta_M)^{\mathrm{T}}\in\mathbb{R}_+^M. \tag{20}$$

Therefore, the objective function (11) is the function of $\mathbf{C}$ and $\mathbf{T}$: $H = H(\mathbf{C},\mathbf{T})$. Also, we denote the velocity, acceleration, jerk and attitude at timestamp t of the j-th piece of the trajectory as $v_j(t) = [v_{j_x}(t), v_{j_y}(t), v_{j_z}(t)]^{\mathrm{T}}$, $a_j(t) = [a_{j_x}(t), a_{j_y}(t), a_{j_z}(t)]^{\mathrm{T}}$, $j_j(t) = [j_{j_x}(t), j_{j_y}(t), j_{j_z}(t)]^{\mathrm{T}}$ and $R_{b_j}(t) = [r_{bx_j}(t), r_{by_j}(t), r_{bz_j}(t)]^{\mathrm{T}}$. Then we derive the gradient with respect to $\mathbf{C}$ and $\mathbf{T}$ as:

$$\frac{\partial H(\mathbf{C},\mathbf{T})}{\partial\mathbf{C}} = \frac{\partial\sum_{j=1}^{M}\int_0^{\delta_j}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))^{\mathrm{T}}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))dt}{\partial\mathbf{C}} + \frac{\partial(E_v+E_a+E_c)}{\partial\mathbf{C}}, \tag{21}$$

$$\frac{\partial H(\mathbf{C},\mathbf{T})}{\partial\mathbf{T}} = \frac{\partial\sum_{j=1}^{M}\int_0^{\delta_j}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))^{\mathrm{T}}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))dt}{\partial\mathbf{T}} + \rho\mathbf{1} + \frac{\partial(E_v+E_a+E_c)}{\partial\mathbf{T}}, \tag{22}$$

$$\begin{aligned}E_v &= W_v\sum_{j=1}^{M}\int_0^{\delta_j}\mathcal{K}\left(\left\|\mathbf{C}_j^{\mathrm{T}}\beta^{(1)}(t)\right\|_2^2 - v_{max}^2\right)dt,\\ E_a &= W_a\sum_{j=1}^{M}\int_0^{\delta_j}\mathcal{K}\left(\left\||\mathbf{C}_j^{\mathrm{T}}\beta^{(2)}(t)\right\|_2^2 - a_{max}^2\right)dt,\\ E_c &= W_c\sum_{j=1}^{M}\int_0^{\delta_j}\sum_{i=1}^{K}\sum_{k=1}^{N_j}\mathcal{K}\left((\hat{v}_j^i(t)-\widetilde{p}_j^k)^{\mathrm{T}}\vec{n}_j^k\right)dt,\\ \hat{v}_j^i(t) &= R_{bj}(t)q_v^i + \mathbf{C}_j^{\mathrm{T}}\beta(t),\end{aligned} \tag{23}$$

where $\mathbf{1}\in\mathbb{R}^M$ is an all-ones vector with an appropriate size. Since $\sum_{j=1}^{M}\int_0^{\delta_j}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))^{\mathrm{T}}(\mathbf{C}_j^{\mathrm{T}}\beta^{(s)}(t))dt$ is a polynomial composed of $\mathbf{C}$ and $\mathbf{T}$, its derivative can be explicitly and easily calculated. Therefore, we mainly deduct the derivative for the penalty terms E_v, E_a, E_c. To simplify the calculation, we convert the integration in E_v, E_a, E_c to the form of discrete summation and uniformly discretize each piece of the polynomial trajectory as L constraint points. The E_v, E_a, E_c are transformed as follows:

$$\begin{aligned}E_v &\approx W_v\sum_{j=1}^{M}\sum_{l=0}^{L-1}\mathcal{K}\left(\left\|\mathbf{C}_j^{\mathrm{T}}\beta^{(1)}(\frac{l}{L}\delta_j)\right\|_2^2 - v_{max}^2\right)\frac{\delta_j}{L},\\ E_a &\approx W_a\sum_{j=1}^{M}\sum_{l=0}^{L-1}\mathcal{K}\left(\left\|\mathbf{C}_j^{\mathrm{T}}\beta^{(2)}(\frac{l}{L}\delta_j)\right\|_2^2 - a_{max}^2\right)\frac{\delta_j}{L},\\ E_c &\approx W_c\sum_{j=1}^{M}\sum_{l=0}^{L-1}\sum_{i=1}^{K}\sum_{k=1}^{N_j}\mathcal{K}\left(\left(\hat{v}_j^i(\frac{l}{L}\delta_j) - \widetilde{p}_j^k\right)^{\mathrm{T}}\vec{n}_j^k\right)\frac{\delta_j}{L}.\end{aligned} \tag{24}$$

The penalty term and its derivative are always zero if constraints are not violated. Therefore, we only need to derive

the gradient of the point that violates constraints, denoted as $\partial E_{v_{jl}}/\partial \mathbf{C}$, $\partial E_{v_{jl}}/\partial \mathbf{T}$, $\partial E_{a_{jl}}/\partial \mathbf{C}$, $\partial E_{a_{jl}}/\partial \mathbf{T}$, $\partial E_{c_{jl}}/\partial \mathbf{C}$, $\partial E_{c_{jl}}/\partial \mathbf{T}$. Moreover, we denote $v_j(\frac{l}{L}\delta_j)$, $a_j(\frac{l}{L}\delta_j)$, $j_j(\frac{l}{L}\delta_j)$ and $\hat{v}_j^i(\frac{l}{L}\delta_j)$ as $\mathbf{v} = [v_x, v_y, v_z]^{\mathrm{T}}$, $\mathbf{a} = [a_x, a_y, a_z]^{\mathrm{T}}$, $\mathbf{j} = [j_x, j_y, j_z]^{\mathrm{T}}$ and $\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} = [\hat{\nu}_x^i, \hat{\nu}_y^i, \hat{\nu}_z^i]^{\mathrm{T}}$respectively to simplify the formula. Then, we first derive the gradient to the coefficient matrix $\mathbf{C}$:

$$\begin{aligned}\frac{\partial E_{v_{jl}}}{\partial \mathbf{C}} &= 3W_v \frac{\delta_j}{L}(||\mathbf{v}||_2^2 - v_{max}^2)^2 \frac{\partial \mathbf{v}^{\mathrm{T}}\mathbf{v}}{\partial \mathbf{C}} \\ &= 3W_v \frac{\delta_j}{L}(||\mathbf{v}||_2^2 - v_{max}^2)^2 \begin{bmatrix} \mathbf{0}_{2(j-1)s\times 3} \\ 2\beta^{(1)}(\frac{l}{L}\delta_j)\mathbf{v}^{\mathrm{T}} \\ \mathbf{0}_{2(M-j)s\times 3} \end{bmatrix}, \end{aligned} \tag{25}$$

$$\begin{aligned}\frac{\partial E_{a_{jl}}}{\partial \mathbf{C}} &= 3W_a \frac{\delta_j}{L}(||\mathbf{a}||_2^2 - a_{max}^2)^2 \frac{\partial \mathbf{a}^{\mathrm{T}}\mathbf{a}}{\partial \mathbf{C}} \\ &= 3W_a \frac{\delta_j}{L}(||\mathbf{a}||_2^2 - a_{max}^2)^2 \begin{bmatrix} \mathbf{0}_{2(j-1)s\times 3} \\ 2\beta^{(2)}(\frac{l}{L}\delta_j)\mathbf{a}^{\mathrm{T}} \\ \mathbf{0}_{2(M-j)s\times 3} \end{bmatrix}, \end{aligned} \tag{26}$$

$$\begin{aligned}\frac{\partial E_{c_{jl}}}{\partial \mathbf{C}} &= W_c \sum_{i=1}^{K}\sum_{k=1}^{N_j} \frac{\partial [(\vec{n}_j^k)^{\mathrm{T}}(\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} - \widetilde{p}_j^k)]^3}{\partial \mathbf{C}} \frac{\delta_j}{L} \\ &= 3W_c \frac{\delta_j}{L} \sum_{i=1}^{K}\sum_{k=1}^{N_j} [(\vec{n}_j^k)^{\mathrm{T}}(\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} - \widetilde{p}_j^k)]^2 \frac{\partial (\vec{n}_j^k)^{\mathrm{T}}\hat{\boldsymbol{\nu}}^{\boldsymbol{i}}}{\partial \mathbf{C}}. \end{aligned} \tag{27}$$

Also, the gradient to the time allocation vector $\mathbf{T}$ is as follows:

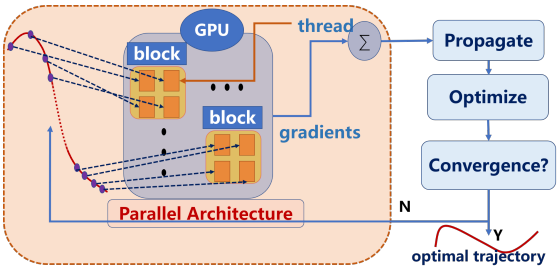
$$\frac{\partial E_{v_{jl}}}{\partial \mathbf{T}} = W_v \frac{(||\mathbf{v}||_2^2 - v_{max}^2)^2}{L} \begin{bmatrix} \mathbf{0}_{j-1} \\ (||\mathbf{v}||_2^2 - v_{max}^2) + \frac{6\delta_j l}{L}\mathbf{v}^{\mathrm{T}}\mathbf{a} \\ \mathbf{0}_{M-j} \end{bmatrix}, \tag{28}$$

$$\frac{\partial E_{a_{jl}}}{\partial \mathbf{T}} = W_a \frac{(||\mathbf{a}||_2^2 - a_{max}^2)^2}{L} \begin{bmatrix} \mathbf{0}_{j-1} \\ (||\mathbf{a}||_2^2 - a_{max}^2) + \frac{6\delta_j l}{L}\mathbf{a}^{\mathrm{T}}\mathbf{j} \\ \mathbf{0}_{M-j} \end{bmatrix}, \tag{29}$$

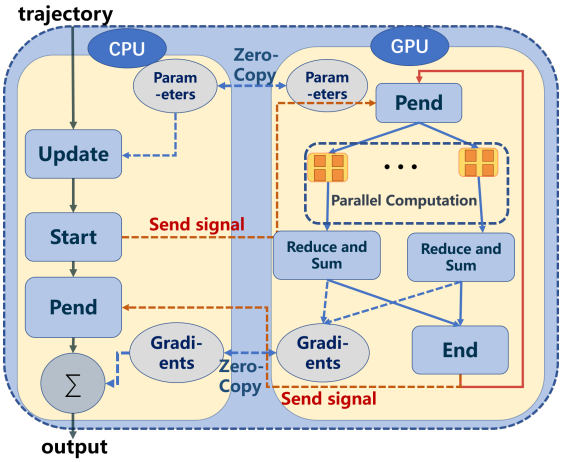
$$\begin{aligned}\frac{\partial E_{c_{jl}}}{\partial \mathbf{T}} &= W_c \sum_{i=1}^{K}\sum_{k=1}^{N_j} \frac{\partial [(\vec{n}_j^k)^{\mathrm{T}}(\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} - \widetilde{p}_j^k)]^3 \frac{\delta_j}{L}}{\partial \mathbf{T}} \\ &= W_c \sum_{i=1}^{K}\sum_{k=1}^{N_j} \begin{bmatrix} \mathbf{0}_{j-1} \\ \frac{[(\vec{n}_j^k)^{\mathrm{T}}(\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} - \widetilde{p}_j^k)]^3}{L} \\ \mathbf{0}_{M-j} \end{bmatrix} + \\ &\quad 3W_c \sum_{i=1}^{K}\sum_{k=1}^{N_j} \frac{\delta_j}{L}[(\vec{n}_j^k)^{\mathrm{T}}(\hat{\boldsymbol{\nu}}^{\boldsymbol{i}} - \widetilde{p}_j^k)]^2 \frac{\partial (\vec{n}_j^k)^{\mathrm{T}}\hat{\boldsymbol{\nu}}^{\boldsymbol{i}}}{\partial \mathbf{T}}. \end{aligned} \tag{30}$$

Furthermore, we obtain $\partial H/\partial \mathbf{C}$ and $\partial H/\partial \mathbf{T}$ by substituting (25)(26)(27) and (28)(29)(30) to (21) and (22) respectively. Based on the optimality condition proved in [7], the minimum control effort piecewise-polynomial trajectory $\mathbf{x}^*(t)$ is d times continuously differentiable at each T_j while the interval condition $\overline{p}_j \in \mathbb{R}^{3(\widetilde{d}+1)}$ is fixed, where $2s = \widetilde{d}+d+2$. We choose $d = 4, \widetilde{d} = 0$ in our work, which means snap is always continuous on the whole trajectory and waypoints $\mathbf{q} = (q_1, \cdots, q_{M-1}) \in \mathbb{R}^{3\times(M-1)}$ are fixed. Then, we inspect the relationships among $\mathbf{q}$, $\mathbf{C}$, and $\mathbf{T}$:

$$\mathbf{MC} = \mathbf{b}, \tag{31}$$



(a) The whole optimization process.



(b) The parallel architecture.

Fig. 4: a) is the flowchart of the entire optimization process. b) is the detailed procedure of the parallel architecture. The parallel architecture inputs are the parameters information of a trajectory, while the outputs are the gradients for the trajectory.

$$\mathbf{M} = \begin{bmatrix} \mathbf{F}_0 & \mathbf{0}_{s\times 2s} & \mathbf{0}_{s\times 2s} & \cdots & \mathbf{0}_{s\times 2s} \\ \mathbf{E}_1 & \mathbf{F}_1 & \mathbf{0}_{2s\times 2s} & \cdots & \mathbf{0}_{2s\times 2s} \\ \mathbf{0}_{2s\times 2s} & \mathbf{E}_2 & \mathbf{F}_2 & \cdots & \mathbf{0}_{2s\times 2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{2s\times 2s} & \mathbf{0}_{2s\times 2s} & \mathbf{0}_{2s\times 2s} & \cdots & \mathbf{F}_{M-1} \\ \mathbf{0}_{s\times 2s} & \mathbf{0}_{s\times 2s} & \mathbf{0}_{s\times 2s} & \cdots & \mathbf{E}_M \end{bmatrix},$$

$$\mathbf{b} = [\overline{p}_0^{\mathrm{T}}, q_1^{\mathrm{T}}, \mathbf{0}_{3\times(d+1)}, \cdots, q_{M-1}^{\mathrm{T}}, \mathbf{0}_{3\times(d+1)}, \overline{p}_f^{\mathrm{T}}]^{\mathrm{T}},$$

where $\mathbf{E}_j, \mathbf{F}_j$ are defined as:

$$\begin{aligned} &\forall j \in \{1, \cdots, M-1\} \\ &\mathbf{E}_j = (\beta(T_j), \beta(T_j), \cdots, \beta^{(d)}(T_j))^{\mathrm{T}}, \\ &\mathbf{F}_j = (\mathbf{0}_{2s}, -\beta(0), \cdots, -\beta^{(d)}(0))^{\mathrm{T}}, \\ &\mathbf{F}_0 = (\beta(0), \cdots, \beta^{(s-1)}(0))^{\mathrm{T}}, \\ &\mathbf{E}_M = (\beta(T_M), \cdots, \beta^{(s-1)}(T_{M-1}))^{\mathrm{T}}. \end{aligned} \tag{32}$$

Therefore, the trajectory is uniquely determined by $\mathbf{q}$ and $\mathbf{T}$. Then, the problem formulation is transformed to:

$$\min\ \hat{H}(\mathbf{q}, \mathbf{T}) = H(\mathbf{C}(\mathbf{q}, \mathbf{T}), \mathbf{T}), \tag{33}$$

$$s.t.\ q_j \in \mathcal{C}_j \cap \mathcal{C}_{j+1},\ \forall j \in \{1, \cdots, M-1\}, \tag{34}$$

$$\delta_j > 0\ \forall j \in \{1, 2, \cdots, M\}. \tag{35}$$

Moreover, gradients to $\mathbf{q}$ and $\mathbf{T}$ are as follows:

$$\frac{\partial \hat{H}(\mathbf{q}, \mathbf{T})}{\partial \mathbf{q}} = \frac{\partial H}{\partial \mathbf{C}} \frac{\partial \mathbf{C}}{\partial \mathbf{q}}, \tag{36}$$

$$\frac{\partial \hat{H}(\mathbf{q}, \mathbf{T})}{\partial \mathbf{T}} = \frac{\partial H}{\partial \mathbf{T}} + \frac{\partial H}{\partial \mathbf{C}} \frac{\partial \mathbf{C}}{\partial \mathbf{T}}, \tag{37}$$

where $\partial \mathbf{C}/\partial \mathbf{q}$ and $\partial \mathbf{C}/\partial \mathbf{T}$ can be computed with a linear-complexity way leveraging the result in [7]. Therefore, we accomplish the gradient computation for both $\partial \hat{H}/\partial \mathbf{q}$ and $\partial \hat{H}/\partial \mathbf{T}$. Then, we use a method proposed in [7] to eliminate the temporal constraints (35) and spatial constraints (34). The constrained optimization problem is turned into an unconstrained one that can be solved by quasi-Newton methods efficiently and robustly.

D. High-performance Implementation of SE(3) Planner

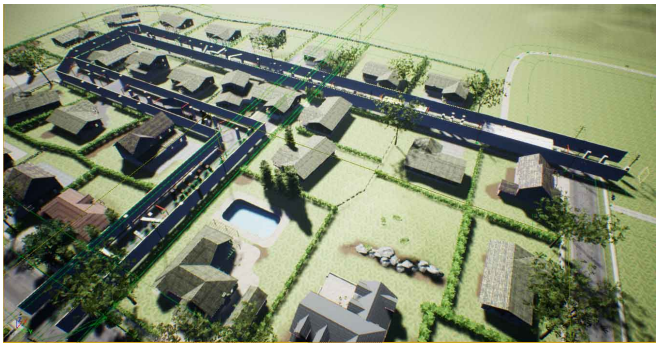
In the proposed planning approach, one necessary part is to calculate the gradient for each constraint point. In each iteration of quasi-Newton method, the gradient calculation for each piece of the trajectory costs constant time U which is related to L. Therefore, the time complexity for calculating gradients of the whole trajectory is $U \cdot M$. Despite the linear complexity, it is a huge cost for large-scale trajectory optimization such as global trajectory generation which is common in drone racing. Fortunately, the gradient calculation for each constraint point on the entire trajectory is independent. Thus, we calculate the gradients parallelly to solve the problem more efficiently with GPU acceleration tools. Then, the computation time for each iteration is reduced to $\frac{U}{L}$. Moreover, we adopt zero copy which maps host memory to device memory space directly to reduce the explicit data transmission time. The whole framework is shown in Fig.4. The procedures of the parallel architecture are detailed as follows:

1) Once the optimization problem is set up, we first launch our persistent kernel function which is always running on GPU to reduce the time of repeated activation of the device function.
2) In each iteration, we update the trajectory parameter information and send a signal from the CPU to the GPU to start the calculation process. The gradient g_{jl} of the l-th discretization point in the j-th piece of the trajectory is computed in the l-th thread of the j-th block independently. Moreover, We store the result in the local shared memory of each block.
3) On the GPU, we reduce and sum [22] the gradients stored in the local memory for each block to obtain the gradient $\sum_{l=0}^{L-1} g_{jl}$ for each piece of the trajectory in logarithmic time complexity.
4) We detect the parallel computation process continuously on the CPU. After the computation process is finished, the GPU sends the signal to the CPU. Then, we sum the gradient of each piece on the CPU to acquire the final gradients for the entire trajectory.

IV. SE(3) Drone Racing Simulation

One of our main goals is to provide lightweight but realistic simulation environments for SE(3) drone racing. To achieve this, we build them on AirSim [23], a highly integrated and complete simulation framework for multirotor which implements a lightweight physics engine, flight controller, and photorealistic scene. Nonetheless, drone racing utilizing AirSim is not out-of-the-box. For drone racing that focuses on motion planning, challenging scenarios, evaluation metrics, racing progress monitor, and the interface for the global map are all important to validate a planning module. In Sect.IV-A, we show our simulation for drone racing tracks and the establishment of the global point cloud. In Sect.IV-B, we discuss the evaluation criteria and rules of drone racing. In Sect.IV-C, we show map interfaces for users.

(a) Track(a).



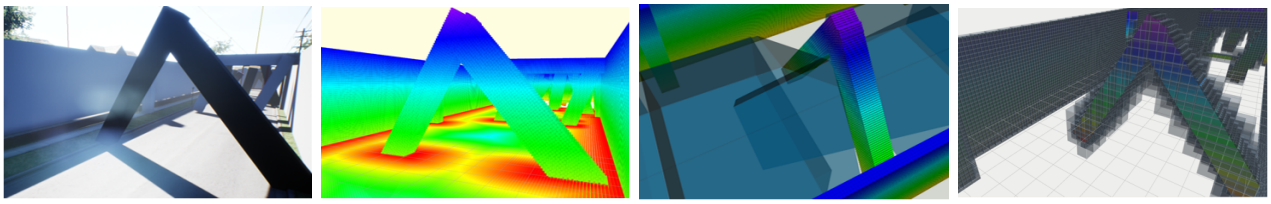
(b) Track(b).

(c) Track(c).

Fig. 5: A panoramic view of our racing environments.

A. Simulation Environments

Thanks to Unreal Engine (UE) Editor, we are able to build various drone racing scenes. As shown in Fig.5, we design extremely dense and complex drone racing tracks containing a set of custom UE actors. These actors detect race-related simulation events that are important to the racing evaluation, such as drone collisions. Furthermore, some gates with known poses and sizes are placed in the environment for racing score statistics and direction guides.



(a) (b) (c) (d)

Fig. 6: (a): real scene, (b): ESDF, (c): free space, (d): octree map.

Drone racing is a systematic problem that involves lots of aspects such as motion planning, flight control, state estimation, and map construction. However, we hope that our racing is a pure test of motion planning algorithms. The construction of maps is never our concern. As a result, a global map is known for planning. Although AirSim does not have the direct API to obtain the map information, the size and pose of each obstacle are known which are used to set up the global point cloud.

To discriminate performance of different motion planning algorithms, and show the significance of the SE(3) planning, there are some narrow gaps in track(a), as shown in Fig.3. Therefore, similar to our planner in Sect.III, the shape and attitude of the drone must be taken into consideration to complete the whole track, as shown in Fig.1b. For track(b), there are no gaps but extremely dense obstacles, which are difficult for obstacle avoidance. Moreover, to imitate traditional drone-racing scenes such as [1] and [2], we build track(c) which only consists of gates for drones to pass.

B. SE(3) *Drone Racing Evalution*

In drone racing, drones are required to pass a series of gates with known poses while avoiding various obstacles, and reach the finish line in the minimum possible time. To collect data for comparison, a monitor program is provided which records the racing time, collision information, the number of passed gates, and the position of the drone in real time. We present a metric to quantify each racing result. An evaluation score is calculated as follows:

$$S = \begin{cases} 100 - T_f + 4 * N_G - P_c & if\ collision \\ 100 - T_f + 4 * N_G & else \end{cases} \tag{38}$$

where S is the score, T_f is the time to finish the whole track, N_G is the number of passed gates, and $P_c = 30$ is the penalty of collision.

C. Map Interfaces

Although the global point cloud is provided as described in Sect.IV-A, different planners always require different data structures to represent environments, such as grid map, Euclidean Sign Distance Field (ESDF), and octree map. Therefore, we provide all of them to free our users from map construction. Moreover, some planners such as [14] require free space as the precondition to optimize the trajectory. We also render the optimal path search and the safe space generation interfaces for users, and all these map representations are shown in Fig.6. In addition, to track the planned trajectory, a controller which minimizes position, velocity and attitude errors is also offered, so that users only need to focus on the planner to enjoy the drone racing in our simulation environments.

TABLE I: Serial Computing and Parallel Architecture Comparison in Different Cases.

Cases	$1 \leq M < 31$			$31 \leq M < 65$			$65 \leq M \leq 100$		
L	$t_c(s)$	$t_g(s)$	$r_e(\%)$	$t_c(s)$	$t_g(s)$	$r_e(\%)$	$t_c(s)$	$t_g(s)$	$r_e(\%)$
16	1.23	**0.62**	**98**	7.14	**3.09**	**131**	14.94	**6.43**	**132**
32	2.45	**0.71**	**245**	12.56	**3.44**	**265**	28.94	**7.51**	**285**
48	3.31	**0.73**	**353**	17.22	**3.77**	**357**	35.94	**7.79**	**361**

TABLE II: Drone Racing Results.

Environments	$t_{racing}(s)$	$v_m(m/s)$	$a_m(m/s^2)$	collisions	score
track(a)	35.49	14.0	9.9	0	124.5
track(b)	33.01	18.03	13.94	0	151.0
track(c)	18.1	21.0	16.2	0	157.9

V. Evaluation

A. Benchmark

This section gives benchmarks for the efficiency of two different implementations of SE(3) planner based on both CPU and the proposed parallel architecture. We conduct the test under different settings on M and L, which are defined in Sect.III-B and Sect.III-C, respectively. The trajectory optimization is repeated for more than 200 times in each case. All the tests are conducted in the Linux environment on a computer equipped with an Intel Core i7-10700 CPU and a GeForce RTX 2060 GPU. We denote by t_c and t_g the optimization time of the planner operated on CPU and parallel architecture, respectively. Moreover, we define $r_e = t_c/t_g - 1$ to represent the efficiency improvement rate. From Tab.I, we conclude that the proposed parallel architecture is more efficient than the sequential one in all of these cases, especially for a long trajectory with highly dense collision evaluation. Moreover, in most scenarios, autonomous drone racing requires the global trajectory to be generated once and for all. Thus the trajectory always consists of a large number of pieces. Furthermore, as L increases, the characterization of the trajectory becomes more precise, thus improving its quality. In this case, our parallel architecture shows its superiority in drone racing with high-fidelity collision and feasibility detection.

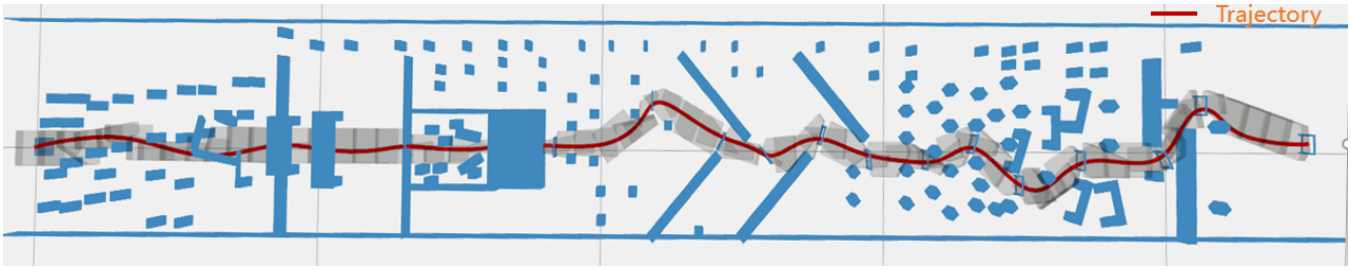


Fig. 7: Visualization of track(a) and the planned SE(3) trajectory of the drone.

B. SE(3) *Drone Racing Results*

We validate our SE(3) planner in our simulator with different drone racing tracks. The racing result is recorded and shown in Tab.II. The visualization of track(a) and the planned SE(3) trajectory is shown in Fig.7. The results demonstrate that our planner enables the drone's safe and high-speed flights through all tracks. Taking our results as the baseline, we hope that users can improve their own planner to achieve better performance.

VI. Conclusion

In this paper, we present an open-source SE(3) planning simulator with a strong benchmark baseline, focusing on high-quality and extremely aggressive SE(3) trajectory generation. We further propose a parallel architecture to speed up the trajectory generation. Our works are user-friendly through their comprehensive interfaces. All works are open-source with the goal of push forward the development of SE(3) motion planning for multirotor.

In the future, we plan to consider dynamic obstacle avoidance to make the planning baseline applicable to more general scenarios.

References

[1] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. Croon, S. Hwang, H. Shim, H. Kim, M. Park, T.-C. Au, and s.-j. Kim, "Challenges and implemented technologies used in autonomous drone racing," *Intelligent Service Robotics*, vol. 12, 04 2019.

[2] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," *arXiv preprint arXiv:2005.12813*, 2020.

[3] S. Li, M. M. Ozo, C. De Wagter, and G. C. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *Robotics and Autonomous Systems*, vol. 133, p. 103621, 2020.

[4] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The blackbird dataset: A large-scale dataset for uav perception in aggressive flight," in *International Symposium on Experimental Robotics*. Springer, 2018, pp. 130–139.

[5] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6713–6719.

[6] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor, "AirSim Drone Racing Lab," in *NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020, pp. 177–191.

[7] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically Constrained Trajectory Optimization for Multicopters," *arXiv preprint arXiv:2103.00190*, 2021.

[8] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6941–6948.

[9] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[10] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, p. 2032–2039, Jul 2018.

[11] A. L. Majdik, C. Till, and D. Scaramuzza, "The Zurich urban micro aerial vehicle dataset," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 269–273, 2017.

[12] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[13] M. van Nieuwstadt and R. M. Murray, "Real Time Trajectory Generation for Differentially Flat Systems," *IFAC Proceedings Volumes*, vol. 29, no. 1, pp. 2301–2306, 1996, 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July.

[14] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 344–351.

[15] 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.

[16] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2021.

[17] 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)*, 2017, pp. 5774–5781.

[18] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2017.

[19] T. Hirata and M. Kumon, "Optimal path planning method with attitude constraints for quadrotor helicopters," in *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*, 2014, pp. 377–381.

[20] 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.

[21] 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.

[22] M. Harris *et al.*, "Optimizing parallel reduction in CUDA," *Nvidia developer technology*, vol. 2, no. 4, pp. 1–39, 2007.

[23] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.