Science Robotics
AAAS

Supplementary Materials for

Safety-assured high-speed navigation for MAVs

Yunfan Ren *et al.*

Corresponding author: Fu Zhang, fuzhang@hku.hk

Sci. Robot. **10**, eado6187 (2025)
DOI: 10.1126/scirobotics.ado6187

The PDF file includes:

Methods
Figs. S1 to S13
Table S1
Legends for movies S1 to S4
References (*66*–*75*)

Other Supplementary Material for this manuscript includes the following:

Movies S1 to S4

Supplementary Methods

System breakdown of SUPER

This section provides a detailed breakdown of the SUPER system with respect to weights, power consumption, and CPU usage (Fig. S1). As illustrated in Fig. S1A, the total hardware weight of SUPER is 1,512 g, comprising the airframe (510 g), battery (432 g), onboard computer (266 g), LiDAR sensor (255 g), and other avionics components (52 g). The power consumption of SUPER was measured during a typical flight mission in a park (see Fig. S2A), with fully onboard sensing, planning, and control on and an average speed of 2.84 m/s. In flight mission, SUPER is commanded to continuously traverse the waypoints following: $\mathbf{p}_1 \rightarrow \mathbf{p}_2 \rightarrow \mathbf{p}_3 \rightarrow \mathbf{p}_4 \rightarrow \mathbf{p}_1 \rightarrow \mathbf{p}_5 \rightarrow \mathbf{p}_3 \rightarrow \mathbf{p}_4 \rightarrow \mathbf{p}_1 \rightarrow \mathbf{p}_6 \rightarrow \mathbf{p}_3 \rightarrow \mathbf{p}_4$ as shown in Fig. S2B. With a 3300 mAh battery, the drone can perform autonomous navigation among a given sequence of waypoints until the battery is out. The total measured flight time is approximately 11 minutes and 24 seconds with a total flight distance of 1,942.56 m (see Fig. S2). The total average power consumption is around 432 W, with 87.9% (379.7 W) consumed by the actuators. The onboard computer is the second-largest power consumer, drawing around 43.3 W. The LiDAR sensor and other avionics components consume 6.3 W and 3.2 W, respectively.

The CPU usage of the onboard computer was also measured during the flight mission. The average CPU usage is around 13.9%, with a peak of 19.7%. The state estimation module is the primary consumer, taking up 11.1% on average. The three main software components, state estimation, planning, and control, are running in parallel as separate processes and communicate with each other through the publisher-subscriber mechanism of the Robot Operating System (ROS) (*66*). To further enhance computational efficiency, the processes within each module are parallelized using multiple threads. In the state estimation module, the nearest neighbor search during scan registration is parallelized across 16 threads. In the planning module, the updates to the spatio-temporal sliding map and the trajectory planning tasks, which include both frontend path search and backend trajectory optimization, are handled by two separate threads. Lastly, in the control module, the model predictive controller (MPC) operates on a single thread.

Benchmark comparison on LiDAR-based navigation methods

To compare the performance of the planning module of SUPER and state-of-the-art LiDAR-based navigation system Falco (*42*), Bubble planner (*13*) and IPC (*67*), this section provides a detailed benchmark comparison in simulation. The setup of the simulation test is the same as in the main results "Evaluation of Safety, Success Rate, and Efficiency" of the manuscript. We have tested two versions: Two versions of Falco were tested. The first, Falco without Gimbal, corresponds to the official open-source implementation of Falco, which does not include a gimbal structure as used in their real-world experiments. In this setup, when the drone accelerates or decelerates, the LiDAR field of view (FOV) and corresponding motion primitives are aligned with the UAV's body frame, as shown in Fig. S3A. The second version, Falco with Gimbal, involves a modification of the official Falco implementation by incorporating a gimbal structure in the simulation. This modification ensures that the LiDAR sensor's horizontal orientation remains independent of the drone's attitude, as illustrated in Fig. S3B. This gimbal configuration, which is also used in the real-world experiments of Falco, facilitates stable operation during high-speed flights.

The results are shown in Fig. S4, where the result of Bubble is directly drawn from the main

results. As can be seen, the Falco with Gimbal approach achieves an overall success rate of 57.04% and a maximum attainable average speed exceeding 10 m/s, which is in agreement with the performance presented in the original paper (*42*). In contrast, the Falco without Gimbal configuration, the default open-source Falco implementation, exhibits a lower overall success rate of only 40.28% and a maximum average speed below 6 m/s, even in the sparsest environment. Regarding IPC, it achieves a high success rate of 75%, outperforming both versions of Falco and the Bubble planner. Although it shows a relatively lower success rate in low traversability environments due to its optimistic planning strategy, the success rate increases as traversability improves. However, in terms of maximum achievable speed, IPC falls behind both Bubble and SUPER. These results demonstrate that SUPER achieves a higher success rate compared to Falco, Bubble and IPC, while attaining much higher average flight speeds. The comparison of computation times is illustrated in Fig. S4C. Falco, with its offline trajectory library and mapless planning framework, requires less computation time than both Bubble and SUPER. For IPC, it generates only one or two polyhedra in each replan cycle, leading to lower computation costs on the frontend. Additionally, its MPC-based planning and control framework takes less than 1 ms for the backend, resulting in much lower overall computational costs.

Implementation Details of Different Baseline Methods

We provide a concise comparison of the implementation details of different baseline methods in the "Evaluation of Safety, Success Rate, and Efficiency" section of the main text. Table S1 presents the comparison of these methods across four aspects. Firstly, the methods are categorized into three types based on their safety strategies: optimistic, safety-aware, and safety-assured. Bubble (*13*) is an optimistic method that treats unknown spaces as free. Raptor (*20*) is a safety-aware method that actively plans trajectories to enhance the visibility of unknown spaces but lacks safety guarantees. Faster (*23*) and our proposed SUPER are both safety-assured methods with theoretical safety guarantees. Secondly, the mapping approaches differ among the methods. Raptor employs an Euclidean signed distance field (ESDF)-based map (*31*), as Faster uses an occupancy grid map (OGM). In contrast, Bubble and SUPER adopt a point cloud-based method, with Bubble additionally utilizing a K-dimensional tree map for corridor generation. Generally, the point cloud-based mapping approach in Bubble and SUPER exhibits much lower computation time compared to Faster and Raptor. All four baselines adopt a frontend and backend style in trajectory generation, with different methods employed for each. Bubble, Faster, and SUPER utilize flight corridor-based frontends, as Raptor employs topological path search. For the backend, Faster uses a mixed-integer quadratic programming (MIQP)-based method, and Raptor uses path-guided trajectory optimization. However, both Faster's MIQP-based method and Raptor's path-guided optimization only optimize the spatial properties of the trajectory and have limited freedom to find feasible trajectories during high-speed flight, which necessitates temporal optimization. In contrast, both Bubble and SUPER incorporate spatio-temporal trajectory optimization based on MINCO (*29*), demonstrating superior performance in high-speed flights, as verified in the main text.

Evaluation of the Planning Module of Faster and SUPER

We evaluated the planning module of Faster (*23*) and SUPER in a simulation, using a challenging indoor corner environment where an obstacle is concealed behind the corner, as depicted in Fig. S6A-i. Additionally, we conducted an ablation study on SUPER, specifically SUPER with-

out OT, which is a variant of SUPER that disables switching time optimization. In each test, the drone commenced at $\mathbf{p_s} = [0, 0, 1.2]^{\mathrm{T}}$, with the goal set to $\mathbf{p_g} = [12.5, 7.5, 1.2]^{\mathrm{T}}$. All three methods achieved the goal safely. Both Faster and SUPER without OT switched to a backup trajectory twice when the drone detected the obstacle behind the corner. However, due to the computationally demanding MIQP-based trajectory optimization employed by Faster, it failed to plan a new trajectory transition from the backup trajectory and execute the backup trajectory till a stop. Consequently, the execution time of the backup trajectory was much longer for Faster compared to SUPER without OT, resulting in a total time of 6.2 s for Faster, exceeding the 4.9 s recorded for SUPER without OT. Furthermore, the full version of SUPER, incorporating the proposed backup trajectory switching time optimization approach, did not switch to the backup trajectory and avoid unnecessary braking maneuvers. SUPER achieved the shortest arrival time of 4.8 s, showcasing the high-speed capability of the proposed planning module.

Additionally, we conducted a controlled, quantitative analysis of the backup trajectory generation between Faster and SUPER. As illustrated in Fig. S6C, both planners were provided with the same exploratory trajectory and backup corridor. The resulting backup trajectories and switching times t_s are depicted by the green curve and orange circle, respectively. Faster employed a heuristic approach to determine the switch time (*23*), setting the switching time t_s to be 0.45 s later than the drone's current state. In comparison, SUPER simultaneously optimized the switch time and backup trajectory, aiming to maximize t_s while ensuring the backup trajectory remained within the backup corridor. This optimization process yielded a switching time of $t_s = 0.84$ s for SUPER. Consequently, SUPER delayed the switching time by 86% compared to Faster, allowing more time for the planning module to re-plan new exploratory and backup trajectories. As a result, the portion of executing the backup trajectory was reduced, as shown in Fig. S6B, enhancing the overall flight speed and smoothness.

To verify the performance of our planning module when integrated into the complete system, we reproduced the aforementioned simulation experiments in a real-world setting, as depicted in Fig. S6D-F. The drone started at $\mathbf{p_s} = [0, 0, 1.2]^{\mathrm{T}}$ and was assigned a goal at $\mathbf{p_g} = [12.5, -7.5, 1.2]^{\mathrm{T}}$, with a maximum speed limit of $\mathbf{v}_{\text{max}} = 8$ m/s, mirroring the simulation setup. As shown in Fig. S6E, at time $t_1 = 2.1$ s, the obstacle was occluded by the wall, and the generated exploratory trajectory $\mathcal{T}_e^1$ collided with the obstacle. However, the backup trajectory $\mathcal{T}_b^1$ was confined within the backup corridor, ensuring that in the worst-case scenario, the drone could safely come to a stop without colliding with any obstacle. At $t_2 = 2.2$ s, the drone perceived the obstacle for the first time (Fig. S6D, ii) and re-planned a new exploratory trajectory $\mathcal{T}_e^2$ to avoid it. Consequently, the initial backup trajectory $\mathcal{T}_b^1$ was not executed, and a new backup trajectory was generated instead. The velocity profile is shown in Fig. S6F, demonstrating that the drone did not execute any backup trajectory and successfully reached the goal at a high average speed, highlighting the high-speed and safe properties of SUPER.

Convex Decomposition in Configuration Space

Convex decomposition is a technique used to identify a convex shape around the given seed (such as a point or a line segment) that can effectively separate a convex region of space from a set of obstacles. In robotic navigation missions, performing convex decomposition in the configuration space and plan trajectories in the decomposed spaces (*12, 13, 23, 40*) is a popular way to efficiently achieve collision avoidance. Currently, two mainstream methods exist for generating free polyhedra in the configuration space (C-space). The first method, called

Point Inflation (*12, 23, 60*), involves inflating points and performing convex decomposition on the inflated point cloud (Fig. S7A, i). However, the point inflation exponentially increases the number of point clouds and negatively affects computational efficiency. Additionally, the inflated obstacle point clouds often fail to represent the configuration space accurately, leading to small and sharp polyhedra (see Fig. S7A, i) that may over-constrained the trajectory optimization problems. The second method, known as Plane Shrinkage (*40, 61*), generates the convex shape directly from the original point cloud and then shrinks it by the robot's radius to obtain the free polyhedron in C-space (Fig. S7A, ii). However, this approach focuses on maximizing the polyhedron before shrinking, which can lead to the resultant polyhedron being small and sharp after the shrinking process. Additionally, there is a risk that the resulting shrunken polyhedron may not encompass the seed, thereby limiting its applicability in scenarios where it is crucial for the seed to reside within the generated polyhedron.

To address those issues, we propose the Configuration-space Iterative Regional Inflation (CIRI) algorithm. Unlike previous approaches (*40, 58, 59, 68, 69*) that represent obstacles as points, we model obstacle points as spheres with a radius equal to the robot's radius. This approach allows us to generate free polyhedra in the C-space directly, offering several advantages. Firstly, our algorithm operates directly on the original point clouds without the need for inflation. This approach reduces the number of points involved, resulting in a substantial improvement in computational efficiency. Secondly, the accurate modeling of obstacles in the configuration space enables our method to find polyhedra with larger sizes compared with the shrinkage-based or inflation-based methods. Additionally, we propose a plane adjustment method that ensures the inclusion of the seed in the generated polyhedra, which is helpful when generating backup corridors.

The proposed CIRI builds upon the iterative process of the FIRI framework (*58*) (see Algorithm 1), offering improved computational efficiency and generating larger polyhedra. The algorithm takes a line segment seed $\mathbf{S} : \{\mathbf{s}_a, \mathbf{s}_b\}$ and the obstacle point cloud $\mathcal{O}$ as input, and begins with the initialization of the ellipsoid as a unit ball (line 2) and proceeds iteratively until the ellipsoid's volume converges. The iterative process comprises two steps. First, the ellipsoid is uniformly inflated until it tangentially intersects the obstacle point cloud, generating a set of hyperplanes that separate a convex region from the obstacles while ensuring the inclusion of the seed (line 7). Second, a Second-Order Cone Programming (SOCP)-based optimization is applied to identify the maximum-volume ellipsoid within the polyhedron formed by the obstacle-free half-spaces defined by those hyperplanes (line 8). Through iterative refinement, the hyperplanes and the ellipsoid are adjusted to progressively expand the volume of the inscribed ellipsoid, resulting in a larger polyhedron and an increased region of obstacle-free space.

Ellipsoid-based Convex Decomposition

In this step, our objective is to determine a free polyhedron comprising a collection of separating hyperplanes. These hyperplanes ensure that the given seed is devoid of obstacles. Each hyperplane is denoted by $\mathcal{H}(A, b)$, where $A\mathbf{x} \leq b, \mathbf{x} \in \mathbb{R}^3$ defines a half-space. The polyhedron $\mathcal{P}$ is defined by intersection of K half-spaces: $\mathcal{P} : \{\mathcal{H}_1, \ldots, \mathcal{H}_k\}$. To represent the ellipsoid, we use the image of the unit ball, which is known as the Löwner-John ellipsoid (*70*), defined as follows:

$$\mathbf{E}(\mathbf{C}, \mathbf{d}) = \{\mathbf{C}\tilde{\mathbf{x}} + \mathbf{d} | \ \|\tilde{\mathbf{x}}\| \leq 1\}, \tag{S1}$$

Algorithm 1: Configuration-space Iterative Regional Inflation (CIRI)

Notation: Identity matrix: $\mathbf{I} \in \mathbb{R}^{3\times3}$; Ellipsoid $\mathbf{E}$ centered at $\mathbf{d}$ with the shape matrix $\mathbf{C}$; Converge threshold: ε
Input : Obstacle point cloud: $\mathcal{O}$; Robot's radius: r; Seed: $\mathbf{S}$
Output : Polyhedron: $\mathcal{P}$

1 **Function** `CIRI` $(\mathcal{O}, r, \mathbf{S})$
2 $\quad \mathbf{E}.\mathbf{C} = \mathbf{I}$;
3 $\quad \mathbf{E}.\mathbf{d} = (\mathbf{s}_a + \mathbf{s}_b)/2$;
4 $\quad$ converge = **False**;
5 $\quad$ **while *not* *converge*** **do**
6 $\quad\quad \mathbf{C}_{last}$ = $\mathbf{E}.\mathbf{C}$;
7 $\quad\quad \mathcal{P}$ = `EllipsoidCvxDecomp` $(\mathcal{O}, r, \mathbf{S}, \mathbf{E})$;
8 $\quad\quad \mathbf{E}$ = `MaximumInscribeVolumeEllipsoid` $(\mathcal{P})$;
9 $\quad\quad$ **if** $[\mathbf{det}(\mathbf{E}.\mathbf{C}) - \mathbf{det}(\mathbf{C}_{last})]\,/\mathbf{det}(\mathbf{C}_{last}) < \varepsilon$ **then**
10 $\quad\quad\quad$ converge = **True**;
11 $\quad\quad$ **end**
12 $\quad$ **end**
13 $\quad$ `return` $(\mathcal{P})$;
14 **End Function**

where $\mathbf{C} \in \mathbb{R}^{3\times3}$ is a diagonal shape matrix defined by $\mathbf{C} = \mathbf{diag}(r_x, r_y, r_z)$ and $\mathbf{d} \in \mathbb{R}^3$ is the center of the ellipsoid. We also define its uniform expansion as:

$$\mathbf{E}_\alpha(\mathbf{C}, \mathbf{d}) = \{\mathbf{C}\tilde{\mathbf{x}} + \mathbf{d} |\ \|\tilde{\mathbf{x}}\| \le \alpha\}, \tag{S2}$$

where $\alpha \ge 1$ is the expansion factor. We use the same definition to describe a sphere, where $\mathbf{C} = \mathbf{diag}(r, r, r)$ and $\mathbf{d}$ is the sphere's center.

The input of the proposed method consists of the obstacle point cloud $\mathcal{O} \in \mathbb{R}^{3\times N}$ with N points, the ellipsoid in the world frame $\mathbf{E}^{\mathcal{W}}$, the seed $\mathbf{S}$ and the robot's radius r. The process is outlined in Algorithm 2 and visualized in Fig. S7C.

At first, the activating obstacle point clouds $\mathcal{O}_a$ are initialized with the full obstacle point clouds $\mathcal{O}$ (line 2). Then, we continued to generate hyperplanes to separate $\mathcal{O}_a$ until it became empty. In each iteration, we choose the nearest obstacle sphere, denoted as $\mathbf{E}_s^{\mathcal{W}}$, to the center of the ellipsoid $\mathbf{E}$ (line 4) and try to find the separating hyperplanes between the $\mathbf{E}_s^{\mathcal{W}}$ and the ellipsoid. The key to finding these hyperplanes involves locating the planes that intersect the obstacles and are tangent to uniformly expanded ellipsoid $\mathbf{E}_\alpha^{\mathcal{W}}$. Building upon the concept presented in (*59*), we simplify the problem by transforming it to the ellipsoid frame $\mathcal{E}$ and form it into a single least-distance programming problem, thus avoiding searching over different expansion factors α. In Eq. S1, we define the ellipsoid in world frame $\mathcal{W}$ as the image of the unit ball in the ellipsoid frame $\mathcal{E}$. Thus, we construct the inverse map and transform the problem to the ellipsoid frame $\mathcal{E}$ as shown in Fig. S7B. The intersecting point $\mathbf{p}_t^{\mathcal{W}}$'s image $\mathbf{p}_t^{\mathcal{E}}$ is the closest point to the origin of the $\mathcal{E}$ frame on $\mathbf{E}_s^{\mathcal{E}}$. Now, the problem is turned into a minimum distance optimization problem.

$$\begin{aligned} &\mathbf{p}_t^{\mathcal{E}} = \arg\min_{\mathbf{x}} \|\mathbf{x}\|_2, \\ &\text{s.t.} \quad \mathbf{x} \in \mathbf{E}_s^{\mathcal{E}}. \end{aligned} \tag{S3}$$

Algorithm 2: Ellipsoid-based Convex Decomposition

Params : Active obstacle point cloud $\mathcal{O}_a$;
Input : Obstacle point cloud: $\mathcal{O}$; Robot's radius: r;
Seed: $\mathbf{S}$; Ellipsoid: $\mathbf{E}^{\mathcal{W}}$
Output : Polyhedron: $\mathcal{P}$

1 **Function** `GeneratePolytope` $(\mathcal{O}, r, \mathbf{S}, \mathbf{E})$
2 $\mathcal{O}_a = \mathcal{O}$;
3 **while not** $\mathcal{O}_a$.`empty`() **do**
4 $\mathbf{E}_s^{\mathcal{W}}$= `FindCloestObstacle` $(\mathbf{E}^{\mathcal{W}}, \mathcal{O}_a)$;
5 $\mathcal{H}$ = `ComputeSeparatingPlanes` $(\mathbf{E}_s^{\mathcal{W}})$;
6 **foreach** $\mathbf{s} \in \mathbf{S}$ **do**
7 $\mathcal{H}$ = `CheckAndAdjustPlane` $(\mathcal{H}, \mathbf{s}, \mathbf{E}_s^{\mathcal{W}})$;
8 **end**
9 $\mathcal{O}_a = \mathcal{O}_a \setminus (\mathcal{H} \cap \mathcal{O}_a)$;
10 $\mathcal{P}$.`AddPlane`$(\mathcal{H})$;
11 **end**
12 `return` $(\mathcal{P})$;
13 **End Function**

This problem can be solved efficiently by finding the root of a 6-th-degree polynomial (*71*). We implement it with C++, and on an Intel i7 CPU, the typical solving time is less than $0.5\,\mathrm{us}$. After finding the tangent point $\mathbf{p}_t^{\mathcal{E}}$, we can find the half space $\mathcal{H}^{\mathcal{E}}(A^{\mathcal{E}}, b^{\mathcal{E}}) = (\mathbf{p}_t^{\mathcal{E}}/\|\mathbf{p}_t^{\mathcal{E}}\|, -\|\mathbf{p}_t^{\mathcal{E}}\|)$ and the expansion factor $\alpha = \|\mathbf{p}_t^{\mathcal{E}}\|$. Then, we can transform them back to get the $\mathcal{H}^{\mathcal{W}}$ in the world frame:

$$\mathcal{H}^{\mathcal{W}}(\mathbf{C}\mathbf{A}^{\mathcal{E}}, b^{\mathcal{E}} + [\mathbf{C}\mathbf{A}^{\mathcal{E}}] \cdot \mathbf{d}), \tag{S4}$$

where $\mathbf{C}, \mathbf{d}$ is the shape matrix and the center of $\mathbf{E}$ respectively.

The process above finds a separating hyperplane between the ellipsoid and the obstacle spheres. However, it is possible that the seed does not lie in the half-space defined by the generated hyperplanes. To address this issue, we propose a plane adjustment method to ensure that the generated polyhedron includes the seed (line 6). As depicted in Fig. S7D, we first verify if both $\mathbf{s}_a$ and $\mathbf{s}_b$ lie within the half-space defined by $\mathcal{H}$. Without loss of generality, if the seed point $\mathbf{s}_a$ is located outside $\mathcal{H}$, we adjust $\mathcal{H}$ to a new hyperplane $\mathcal{H}_{adj}$ that passes through point $\mathbf{s}_a$, which is tangent to the obstacle sphere, and points towards another seed point $\mathbf{s}_b$. This adjustment ensures that the new hyperplane $\mathcal{H}_{adj}$ contains both $\mathbf{s}_a$ and $\mathbf{s}_b$. We then set the hyperplane generated in this round as the new $\mathcal{H}$, denoted as $\mathcal{H}_{adj}$. After the plane adjustment process (if necessary), all obstacle points outside $\mathcal{H}$ are excluded from $\mathcal{O}_a$, and $\mathcal{H}$ is added to the polyhedron $\mathcal{P}$. This process is repeated until all obstacle points are excluded.

Maximum Volume Inscribed Ellipsoid

Given the polyhedron (a set of hyperplanes), this step involves updating the ellipsoid by determining the maximum volume inscribed ellipsoid (MVIE) within the given polyhedron. Assume the polyhedron consists of K hyperplanes $\mathcal{P} = \{\mathcal{H}_1, \ldots, \mathcal{H}_K\}$ and the ellipsoid is define by Eq. S1, the optimization problem can be formulated as:

$$\begin{aligned} \max_{\mathbf{C},\mathbf{d}} \quad & \det(\mathbf{C}), \\ \text{s.t.} \quad & \sup(A_i\mathbf{C}\tilde{\mathbf{x}}) + A_i\mathbf{d} \leq b_i, \forall i = 1, \ldots, K, \\ & \mathbf{C} \succeq 0, \end{aligned} \tag{S5}$$

where $\mathbf{C}$ and $\mathbf{d}$ are the shape matrix and the center of the ellipsoid, respectively. A_i and b_i are the parameters of the plane equation form of the hyperplane $\mathcal{H}(A, b) : A\mathbf{x} \leq b$. The optimization problem Eq. S5 is a well-studied convex optimization problem in computational geometry. In this paper, we utilize the formulation proposed in (*72*) to convert Eq. S5 as a SOCP-based method and can be efficiently solved following the Conic Augmented Lagrangian method (*73*).

Comparison on Convex Decomposition

We conducted a comparative analysis of convex decomposition methods in three distinct environments characterized by different obstacle shapes: Pillars, Forest, and Perlin (*74*) (see (Fig. S8)). For each obstacle type, we randomly generated 300 scenarios with varying densities of obstacles. To quantify the obstacle density of a scenario, we used the obstacle filling rate (OFR), which is calculated as the ratio of the total volume of obstacles to the total volume of the space. The 900 scenarios over different obstacle type and densities lead to an OFR ranging from 0.0005 to 0.6. For each scenario, we sample a point cloud map with a spatial resolution of 0.1 m, along with a random line seed that was ensured to be at least 0.2 m (equal to the robot's radius) away from the nearest obstacles to ensure the existence of a feasible polyhedron in the configuration space. The point cloud map and line seeds are then fed to different convex decomposition methods.

We compared three approaches that perform convex decomposition in the configuration space (see Fig. S7A). The first approach, Plane Shrinkage, generates a polyhedron on the original point cloud and then shrinks each hyperplane by the robot's radius. The second approach, Point Inflation, inflates the input point clouds by adding additional points within cubes centered at each input point, where the cube length is equal to the robot's radius. Convex decomposition is then performed on the inflated point cloud, ensuring that the resulting polyhedron is valid in the configuration space. The third approach, CIRI (Configuration-space Iterative Regional Inflation), models each input point as a sphere with a radius equal to the robot's radius, and then performs convex decomposition on these point spheres. For the first two approaches, we implemented baseline methods based on two state-of-the-art convex decomposition techniques: FIRI (*58*) and RILS (*40*), resulting in FIRI-Shrinkage, FIRI-Inflation, RILS-Shrinkage, and RILS-Inflation, respectively.

We evaluated each method in terms of seed containment rate (the ratio of polyhedra that contains the line seed), computation time, and volume of the resulting polyhedra (see Fig. S8B-D). Considering the seed containment rate (Fig. S8B), CIRI and the point inflation (either FIRI-inflation or RILS-inflation) achieved 100% seed containment rate in all tests. In contrast, the plane shrinkage-based methods (either FIRI-shrinkage or RILS-shrinkage) exhibited a much lower seed containment rate, as they can only ensure the seed is within the polyhedron before the shrinking process but not after. Regarding the volume of the generated polyhedra (Fig. S8C), CIRI consistently produced polyhedra with the largest average volume compared to the other four baselines, thanks to its precise modeling of the configuration space. In terms of computational time (Fig. S8D), CIRI and the two shrinkage-based baselines, FIRI-shrinkage and RILS-shrinkage, had lower computation times compared to the two inflation-based baselines.

This is because the point inflation introduces more points in the input point cloud, which substantially increases the computation time. Compared to the shrinkage-based methods, CIRI has slightly higher computation time due to the need to solve the tangent plane between a sphere and an ellipsoid, while the other two methods solve the point-ellipsoid tangent plane, which is computationally easier.

Since plane shrinkage-based methods, including FIRI-Shrinkage and RILS-Shrinkage, do not guarantee the containment of line seed, they are not suitable for SUPER. Comparing among the seed-containment-assured methods, CIRI can generate polyhedra with larger volumes while requiring lower computation time than point inflation methods (FIRI-inflation and RILS-inflation). In conclusion, CIRI effectively enhances corridor quality within less computational time, thereby further improving the overall performance of SUPER.

Theorem for Backup SFC Generation

Theorem 1. *Let $\mathcal{D}$ be an omni-directional and infinitely-dense depth image captured at the position $\mathbf{p}_c \in \mathbb{R}^3$. If a set $\mathcal{S} \subseteq \mathbb{R}^3$ satisfies the following conditions, it must lie in the known-free space:*

(a) $\mathcal{S}$ is a convex set;

(b) $\mathcal{S}$ contains no point from $\mathcal{D}$;

(c) $\mathbf{p}_c \subseteq \mathcal{S}$.

Proof. Assume that $\mathcal{S}$ is not entirely contained within the known-free space. This implies the existence of a point $\mathbf{p}$ with an occupancy state that is either occupied or unknown. In the first case, if $\mathbf{p}$ is in the occupied space, it must be captured in $\mathcal{D}$, which contradicts condition (b). In the second case, if $\mathbf{p}$ is in the unknown space, its depth must be greater than any point $\mathbf{d} \in \mathcal{D}$ with the same bearing direction. For the point $\mathbf{d}$ lies on the line connecting $\mathbf{p}$ and $\mathbf{p}_c$ (the bearing direction), the convexity of $\mathcal{S}$ implies that $\mathbf{d} \in \mathcal{S}$. This once again contradicts condition (b). Hence, in both cases, we arrive at a contradiction to condition (b), leading to the conclusion that $\mathcal{S}$ must be entirely contained within the known-free space. □

Trajectory Planning with Limited LiDAR FOV

When considering the LiDAR sensor's limited field of view (FOV), the backup corridor must intersect with the FOV to ensure that the intersecting space lies entirely within the known-free area. One most common type of LiDAR sensor has a 360-degree horizontal FOV and a limited vertical FOV, as shown in Fig. S11A. Assuming the vertical FOV angle is $\theta < \pi$ (Fig. S11B), the total FOV forms a non-convex shape. To ensure that the intersected backup corridor is convex, we must select a convex subset of the FOV and intersect it with the convex polytope generated by CIRI with this subset. The largest convex subset of the 360-degree horizontal FOV is the intersection of two half-spaces defined by hyperplanes that are tangent to the FOV's upper and lower bounds, as shown in Fig. S11A and B. The FOV has infinitely many such convex subsets, each uniquely determined by a one degree-of-freedom (DOF) heading direction, $\mathbf{h}$, as illustrated in Fig. S11C. To ensure that the resultant backup corridor contains the largest possible exploratory trajectory, we align $\mathbf{h}$ with the tangent direction of the exploratory trajectory (Fig. S11D).

To investigate the influence of the vertical FOV angle θ on planning performance, we conducted a series of experiments in a simulated environment with vertical FOV angles ranging from 30 degrees to 180 degrees (omni-directional FOV). The testing environment is similar to that used in the benchmark comparisons in the main manuscript. We selected two types of simulation environments with different traversability scores: one with a traversability of 3.1, indicating a dense environment, and another with a traversability of 6.5, representing a relatively sparse environment. For each traversability setting, we tested 10 maps, running 10 trials per map without limiting the maximum velocity. For each trial, we recorded the average speed, the ratio of executing the backup trajectory, and the final outcome (success or failure). The results are presented in Fig. S12.

Considering the success rate, SUPER achieved a 100% success rate in both environments, regardless of lower or higher traversability (Fig. S12A, i and Fig. S12B, i), highlighting its safety-assured properties. In the denser environment, the average speed increased as the vertical FOV became larger (Fig. S12A, ii), which is due to the fact that a larger FOV enables generation of larger volume backup corridors, reducing the need to execute the backup trajectory (Fig. S12A, iii), resulting in a higher average speed. In the sparser environment, the average flight speed also showed a similar trend, though the influence of FOV size was less significant (Fig. S12B, ii). This is because, in environments with fewer obstacles, even a small FOV is sufficient for planning safe, high-speed trajectories, and the ratio of executing the backup trajectory is less affected by the FOV size (Fig. S12B, iii).

On-manifold MPC with Rotor Drag Compensation

In our prior work (*65*), we have proposed an on-manifold model predictive control (OMMPC) framework for robotic systems and demonstrated indoor quadrotor aerobatics. However, the aerodynamic effects were neglected in the previous work, which affects the tracking accuracy when the vehicle performs high-speed flights in outdoor environments. To overcome this limitation, we extend our previous quadrotor model by incorporating rotor drag, which is the main aerodynamic effect causing tracking error (*64*). We derive the resultant OMMPC and implement it for all flight experiments in this paper, demonstrating sufficiently high tracking accuracy. The effectiveness of this improvement is also verified through comparative experiments.

Quadrotor dynamics

The quadrotor coordinate directions X, Y, and Z are defined as forward, right, and down, respectively. The quadrotor dynamics involving rotor drag (*64*) are formulated as follows:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v}, \\ \dot{\mathbf{v}} &= \mathbf{g} - a_T \mathbf{R}\mathbf{e}_3 - \frac{1}{m}\mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}}\mathbf{v}, \\ \dot{\mathbf{R}} &= \mathbf{R}\lfloor \boldsymbol{\omega}^{\mathcal{B}} \rfloor, \end{aligned} \tag{S6}$$

where m is the mass of the drone, $\mathbf{g}$ represents the gravity vector with a fixed length of 9.81 m/s^2, and a_T denotes the scalar thrust acceleration. The position and velocity in the inertial frame are denoted by $\mathbf{p} \in \mathbb{R}^3$ and $\mathbf{v} \in \mathbb{R}^3$, respectively. The attitude of the airframe is represented by $\mathbf{R} \in SO(3)$, and $\boldsymbol{\omega}$ denotes angular velocity in the body frame. The operation $\lfloor \cdot \rfloor$ converts a vector into a skew-symmetric matrix. The operation $\lfloor \cdot \rfloor$ converts a vector into a skew-symmetric matrix. $\mathbf{D}$ denotes the rotor drag coefficient matrix, which is parameterized as $\mathbf{D} = \mathbf{diag}(d_h, d_h, d_v)$.

As discussed in (*64*), the above quadrotor dynamic system with rotor drag is differentially flat. The flat output vector is typically selected as $\sigma = [\mathbf{p}, \psi]^{\mathrm{T}}$, where ψ represents the yaw angle. This differential flatness property allows us to plan the trajectory of the flat output $\sigma(t)$ instead of the entire state trajectory $\mathbf{s}(t)$ of the quadrotor UAV.

System linearization

As previously described in our previous work (*65*), OMMPC takes the quadrotor system into a compound manifold, and defines the system states and inputs as follows:

$$\begin{aligned} &\mathcal{M} = \mathbb{R}^3 \times \mathbb{R}^3 \times SO(3), \quad \dim(\mathcal{M}) = 9, \\ &\mathbf{x} = \begin{pmatrix} \mathbf{p}^{\mathcal{I}} & \mathbf{v}^{\mathcal{I}} & \mathbf{R} \end{pmatrix} \in \mathcal{M}, \quad \mathbf{u} = \begin{bmatrix} a_T & \boldsymbol{\omega}^{\mathcal{B}} \end{bmatrix} \in \mathbb{R}^4, \end{aligned} \tag{S7}$$

and the error between measured state $\mathbf{x}$ and reference state $\mathbf{x}_d$, lying on the manifold $\mathcal{M}$, is mapped into the local homeomorphic space (an open set in Euclidean space) around each point $\mathbf{x}_d$ using local coordinates. Particularly, the state error is defined as follows:

$$\begin{aligned} \delta\mathbf{x} &\triangleq \mathbf{x}_d \boxminus \mathbf{x} = \begin{bmatrix} \delta\mathbf{p}^{\mathrm{T}} & \delta\mathbf{v}^{\mathrm{T}} & \delta\mathbf{R}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^9, \\ \delta\mathbf{p} &\triangleq \mathbf{p}_d \boxminus \mathbf{p} = \mathbf{p}_d - \mathbf{p} \in \mathbb{R}^3, \\ \delta\mathbf{v} &\triangleq \mathbf{v}_d \boxminus \mathbf{v} = \mathbf{v}_d - \mathbf{v} \in \mathbb{R}^3, \\ \delta\mathbf{R} &\triangleq \mathbf{R}_d \boxminus \mathbf{R} = \mathrm{Log}(\mathbf{R}^{\mathrm{T}}\mathbf{R}_d) \in \mathbb{R}^3, \end{aligned} \tag{S8}$$

where $\boxminus$ is an encapsulated operator implemented to the mapping $\boxminus : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^n$ as mentioned in (*65*), and the input error is also defined as

$$\begin{aligned} \delta\mathbf{u} &\triangleq \mathbf{u}_d - \mathbf{u} = \begin{bmatrix} \delta a_T & \delta\boldsymbol{\omega}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^4, \\ \delta a_T &\triangleq a_{T_d} - a_T \in \mathbb{R}, \\ \delta\boldsymbol{\omega} &\triangleq \boldsymbol{\omega}_d - \boldsymbol{\omega} \in \mathbb{R}^3, \end{aligned} \tag{S9}$$

and the original quadrotor system can be equivalently expressed by an error system where the vehicle states are mapped to the local coordinates at each point along the reference trajectory. Follow the derivation in (*65*), the resultant linearized error system is given by

$$\delta\mathbf{x}_{k+1} = \mathbf{F}_{\mathbf{x}_k}\delta\mathbf{x}_k + \mathbf{F}_{\mathbf{u}_k}\delta\mathbf{u}_k, \tag{S10}$$

where

$$\begin{aligned} \mathbf{F}_{\mathbf{x}_k} &= \begin{bmatrix} \mathbf{I}_3 & \mathbf{I}_3\Delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 - \frac{\Delta t}{m}\mathbf{R}_k^d\mathbf{D}\mathbf{R}_k^{d^{\mathrm{T}}} & \frac{\Delta t}{m}\mathbf{R}_k^d\left(2\mathbf{I}_3 - \mathbf{D}\right)\left\lfloor a_T m\mathbf{e}_3 + \mathbf{D}\mathbf{R}_k^{d^{\mathrm{T}}}\mathbf{v} - \mathbf{R}_k^{d^{\mathrm{T}}}\mathbf{v}\right\rfloor \\ \mathbf{0} & \mathbf{0} & \mathrm{Exp}\left(-\boldsymbol{\omega}_k^d\Delta t\right) \end{bmatrix}, \\ \mathbf{F}_{\mathbf{u}_k} &= \Delta t \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\mathbf{R}_k^d\mathbf{e}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}\left(\boldsymbol{\omega}_k^d\Delta t\right)^{\mathrm{T}} \end{bmatrix}, \end{aligned} \tag{S11}$$

where

$$\mathbf{A}(\boldsymbol{\theta}) = \mathbf{I}_3 + \left(\frac{1-\cos\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|}\right)\frac{\lfloor\boldsymbol{\theta}\rfloor}{\|\boldsymbol{\theta}\|} + \left(1 - \frac{\sin\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|}\right)\frac{\lfloor\boldsymbol{\theta}\rfloor^2}{\|\boldsymbol{\theta}\|^2}. \tag{S12}$$

Therefore, the OMMPC can be given by the current error and reference trajectory, leading to an efficient QP solution:

$$\begin{aligned} \min_{\delta \mathbf{u}_k} & \sum_{k=0}^{N-1} \left(\|\delta \mathbf{x}_k\|_{\mathbf{Q}}^2 + \|\delta \mathbf{u}_k\|_{\mathbf{R}}^2 \right) + \|\delta \mathbf{x}_N\|_{\mathbf{P}}^2, \\ \text{s.t.} \quad & \delta \mathbf{x}_{k+1} = \mathbf{F}_{\mathbf{x}_k} \delta \mathbf{x}_k + \mathbf{F}_{\mathbf{u}_k} \delta \mathbf{u}_k, \\ & \delta \mathbf{u}_k \in \delta \mathbb{U}_k, \quad k = 0, \cdots, N-1, \end{aligned} \tag{S13}$$

where N is the predictive horizon, and positive-definite diagonal matrices $\mathbf{Q}, \mathbf{R}, \mathbf{P}$ denote the penalty of the stage state, stage input, and terminal state, respectively. $\mathbb{U}_k = \{\delta \mathbf{u} \in \mathbb{R}^4 | \mathbf{u}_{\min} - \mathbf{u}_k^d \leq \delta \mathbf{u} \leq \mathbf{u}_{\max} - \mathbf{u}_k^d\}$ denotes the constraints for the input error.

Experiment Results

To quantitatively assess the effectiveness of the proposed OMMPC incorporating the rotor-drag model, we conducted a comparative analysis with the common quadrotor model (*65*) using the PX4 SITL software with the Gazebo simulator (*75*). The simulator provides realistic dynamics and air resistance simulations. We generated an offline polynomial trajectory on the flat space denoted as $\sigma(t)$ with a maximum velocity of 15 m/s. The trajectory consisted of a straight line starting from $\mathbf{p}_0 = [0, 0, 0]^{\mathrm{T}}$ and ending at $\mathbf{p}_g = [120, 0, 0]^{\mathrm{T}}$ along the x-axis. Initially, we generated the desired state trajectory using differential flatness under the common quadrotor model and tracked the trajectory using the quadrotor OMMPC proposed in (*65*). As shown in Fig. S13A, the UAV exhibited significant attitude tracking errors, and the position tracking error was notably influenced by the velocity direction, reaching a maximum of 0.4 m. In contrast, by employing the quadrotor model with rotor drag described in equation Eq. (S6), the OMMPC effectively predicted and compensated for the drag, resulting in improved attitude-tracking performance and a reduced position-tracking error of 0.05 m.

Supplementary Figures and Tables

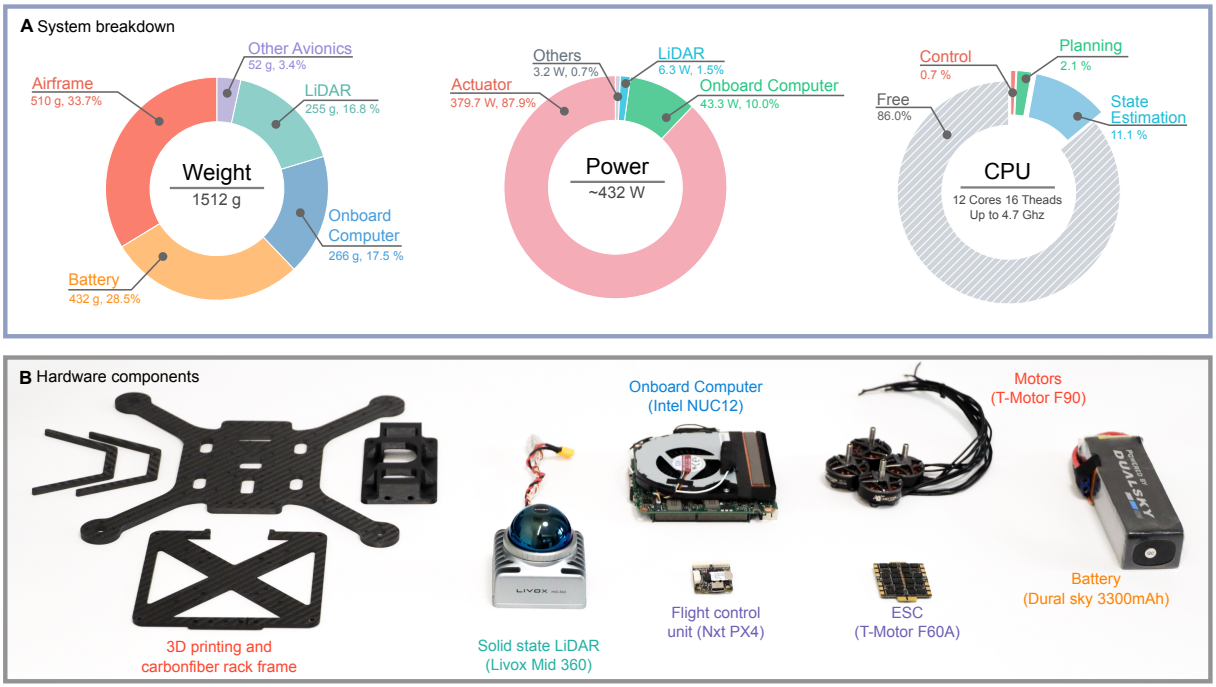


Fig. S1. System breakdown of SUPER (**A**) The weight, power, and CPU usage breakdown of SUPER. (**B**) The hardware components of SUPER.

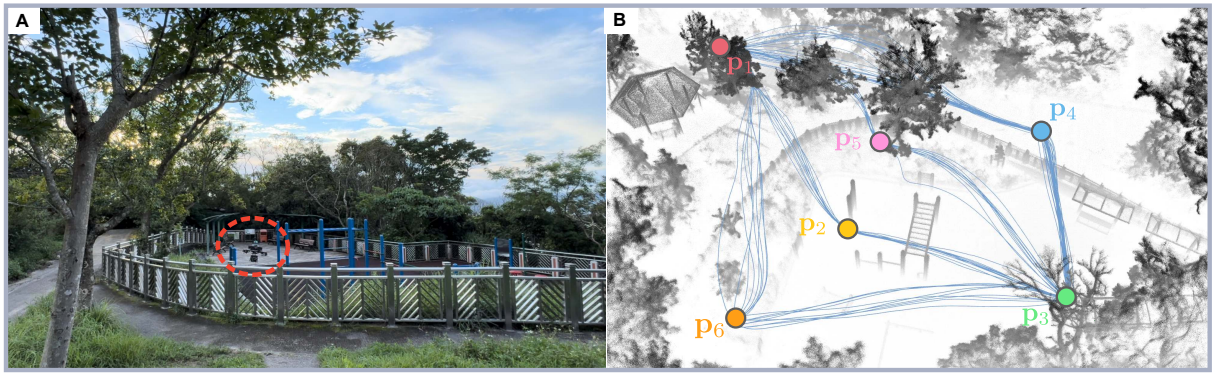


Fig. S2. Scenario and trajectory of the endurance test (**A**) The testing scenario. (**B**) The point cloud view and flying trajectory of the drone during the endurance test.

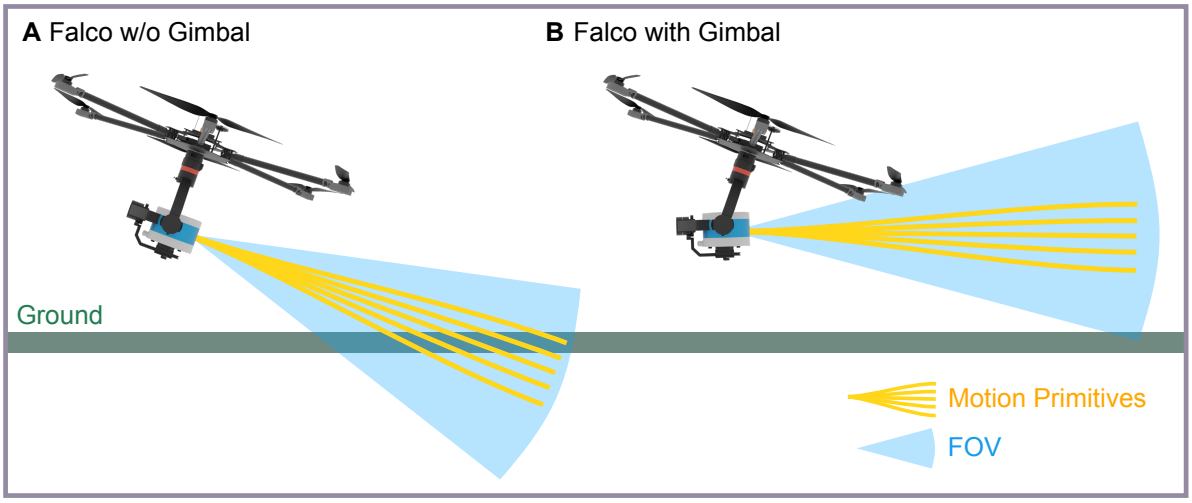


Fig. S3. Illustration of Falco with and without gimbal structure (**A**) The original version of Falco's open-sourced implementation lacked a gimbal structure. In this configuration, when the multirotor accelerates, the motion primitives are constrained to point downwards, prohibiting the drone from flying forward. (**B**) The modified version of Falco with a gimbal structure. The LiDAR sensor maintains a horizontal orientation independent of the drone's attitude.

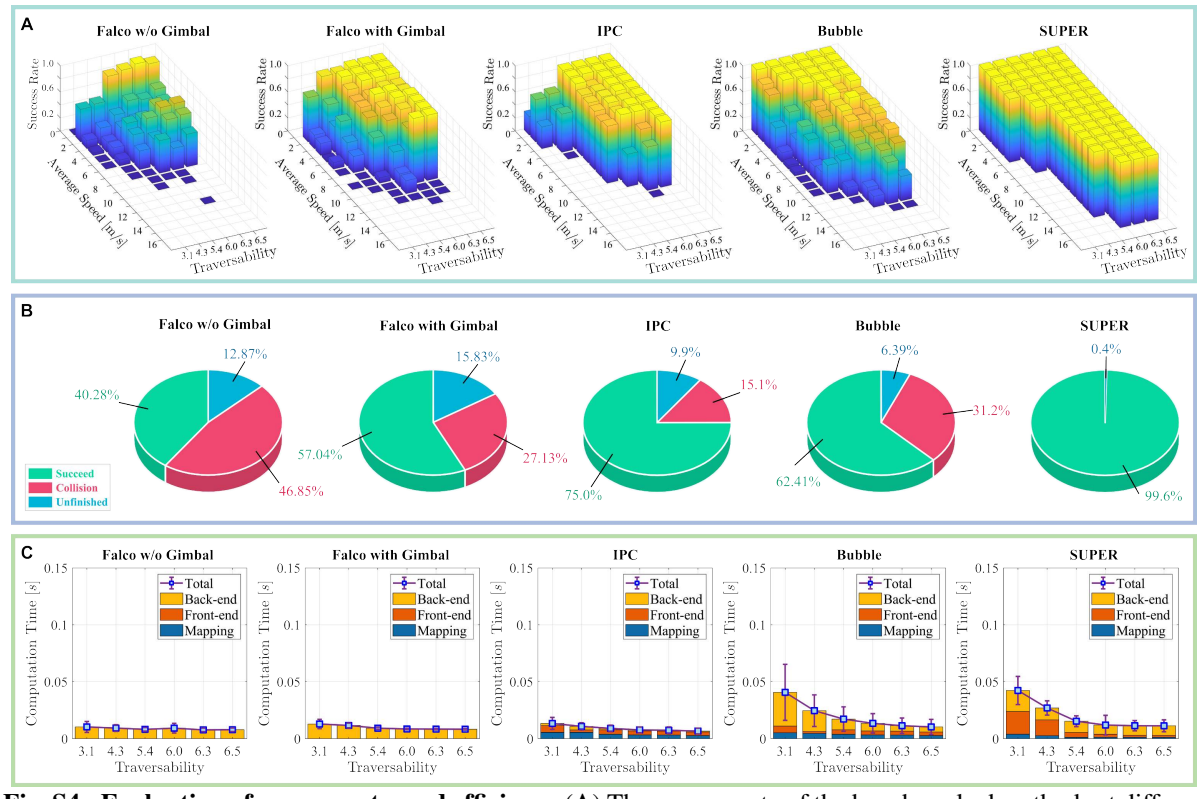


Fig. S4. Evaluation of success rate, and efficiency (**A**) The success rate of the benchmarked methods at different obstacle densities (measured as traversibility) and flight speeds. Empty columns indicate that the corresponding combination of speed and density was not achieved. (**B**) Flight results of the benchmarked methods in 1080 experiments. (**C**) Time consumption of the benchmarked methods, with squares representing the mean values and error bars indicating the standard deviations of the total computation time. Each mean and standard deviation is computed from 180 tests.

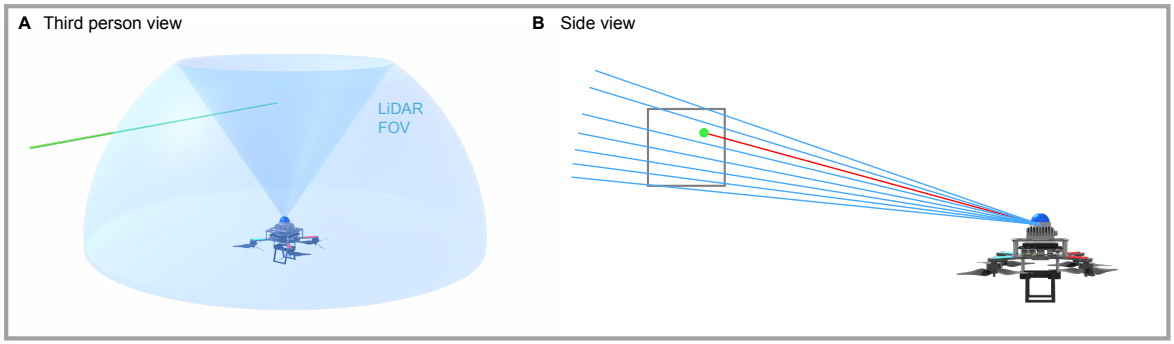


Fig. S5. Visual example of thin wire detection on OGM. (**A**) Third-person view of the MAV facing a thin wire (shown in green). (**B**) Side view of the detection process. The gray box represents the cell in the occupancy grid map (OGM) that contains the thin wire. The blue lines indicate the LiDAR beams that pass through the cell without hitting the wire, while the red line indicates the beam that hits the wire. Since most of the beams do not detect the wire, the cell is incorrectly classified as free space.

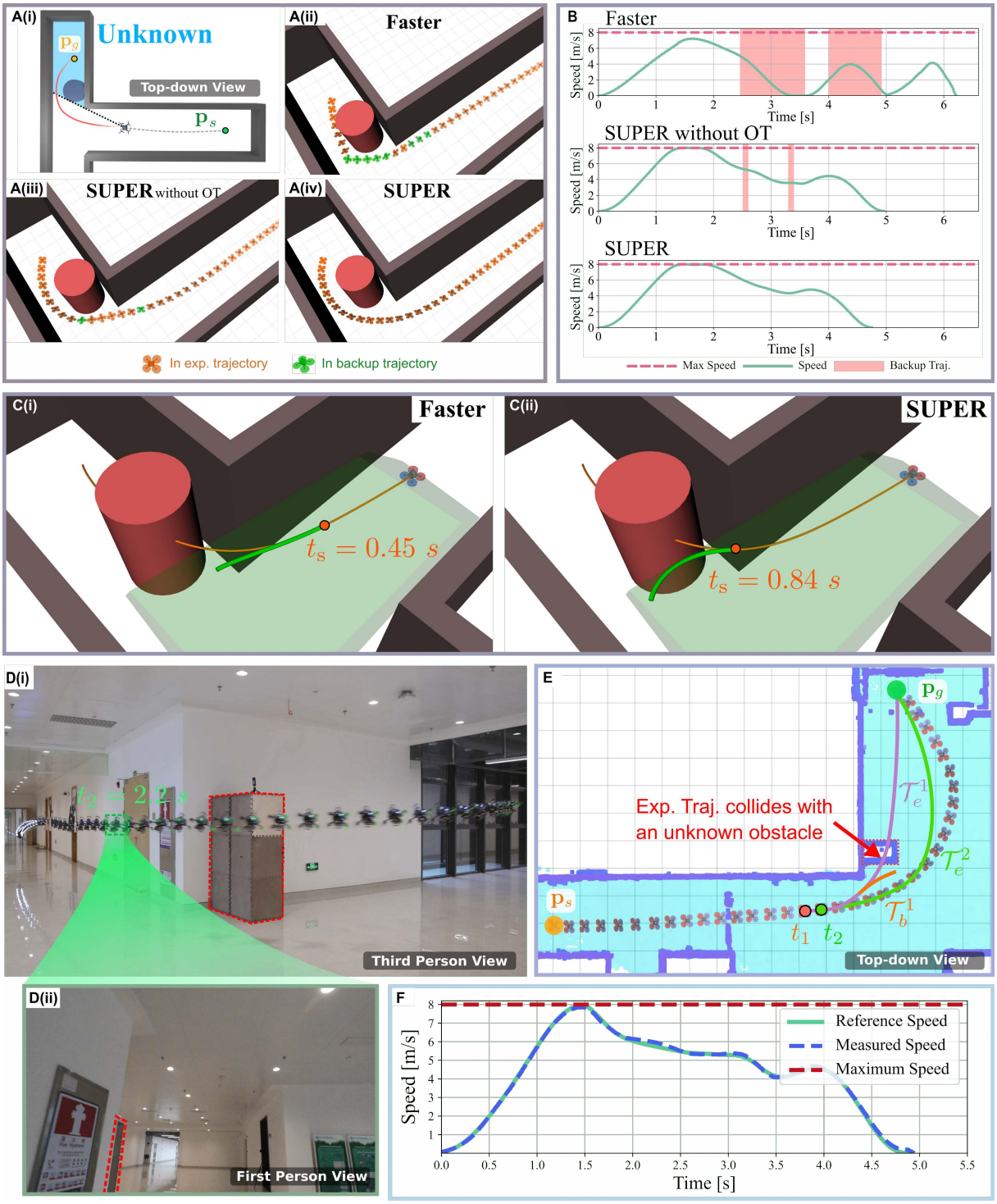


Fig. S6. Evaluation of planning module of Faster and SUPER (**A**) (i) A top-down view of the simulation environment, where a red cylinder is hidden behind the corner. (ii-iv) Comparison of the executed trajectories for the three methods: Faster (*23*), the proposed SUPER, and SUPER without switching time optimization (referred to as SUPER without OT). (**B**) Velocity profile and execution of backup trajectories in the simulation. (**C**) Switching time t_s of Faster and SUPER given the same exploratory trajectory and backup corridor. (**D**) (i) Real-world experiment of SUPER, with the hidden obstacle highlighted by a red dashed box. (ii) First-person view from the onboard camera at t_2. (**E**) Planned trajectories at time t_1 and t_2. $\mathcal{T}_e^1$ and $\mathcal{T}_b^1$ are the exploratory and backup trajectories at t_1, respectively, and $\mathcal{T}_e^2$ is the exploratory trajectory at t_2. The backup trajectory at t_2 is not displayed. (**F**) Velocity profile in the experiment.

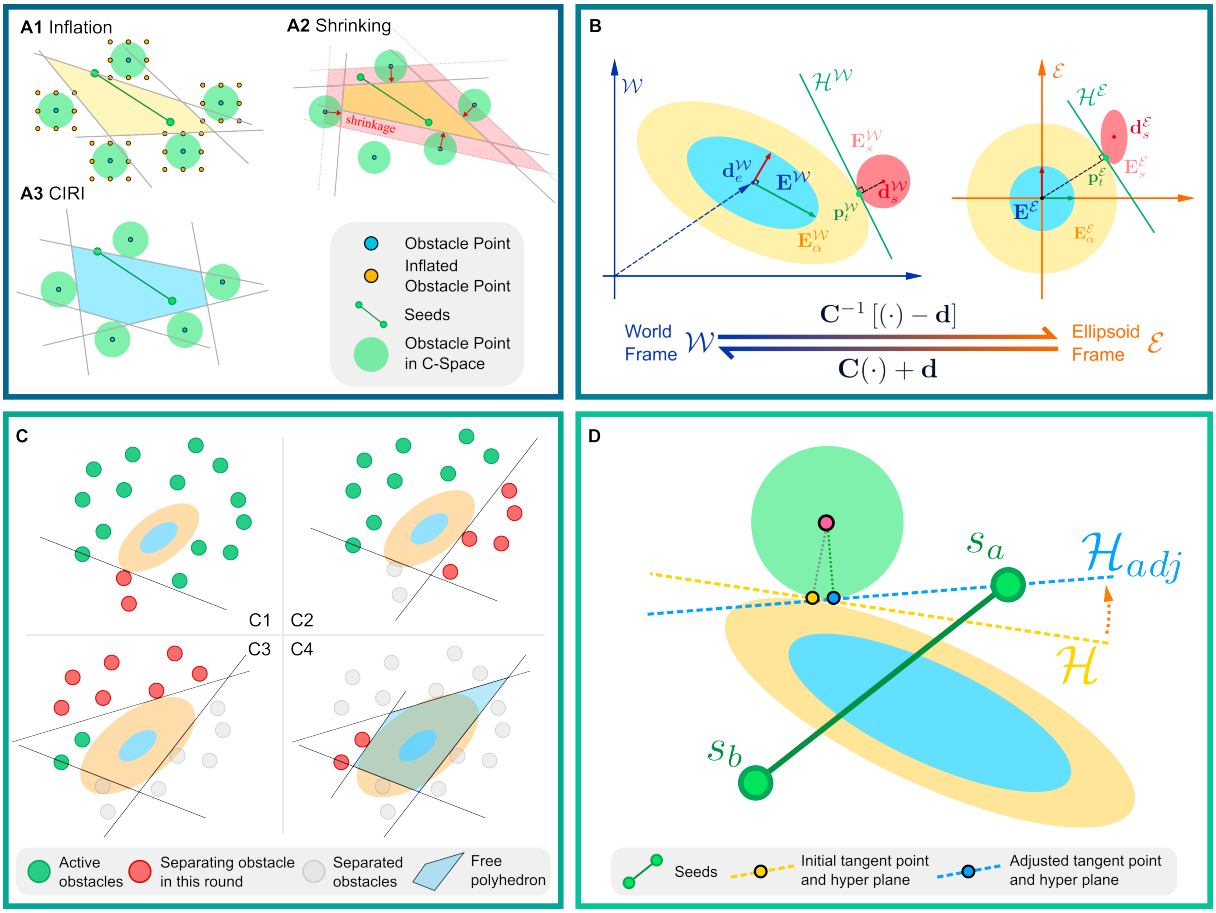


Fig. S7. Illustration of the proposed flight corridor generation algorithm (CIRI). (**A**) Comparison of three different methods for generating a safe flight corridor in configuration space. (**B**) Example illustrating the transformation between the ellipsoid frame $\mathcal{E}$ and the world frame $\mathcal{W}$. (**C**) Visual depiction of the process of convex decomposition in the C-space. (**D**) Example demonstrating the plane adjustment strategy to ensure that the given seeds lie within the generated polyhedron.

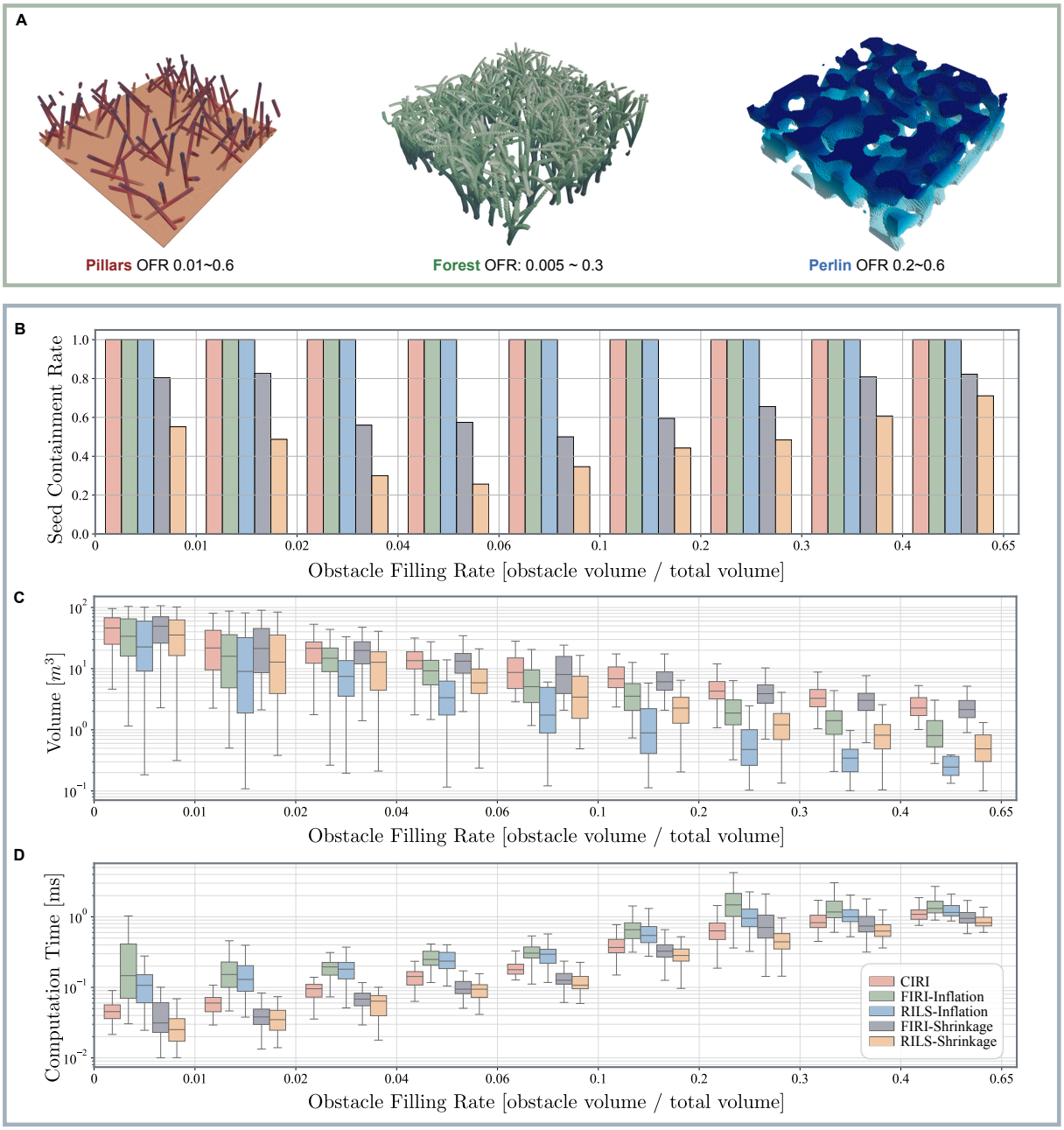


Fig. S8. Comparison of convex decomposition in configuration space. (**A**) Example benchmark environments with obstacle filling rates (OFR) ranging from 0.005 to 0.6. (**B**) Seed containment rate of generated polyhedra by different methods across OFR. (**C**) Boxplot showing the average polyhedron volumes generated by different methods across OFR, with each plot based on 100 tests. The central bar represents the median. (**D**) Boxplot of computation times for the various methods across OFR, with each plot derived from 100 tests. The central bar indicates the median.

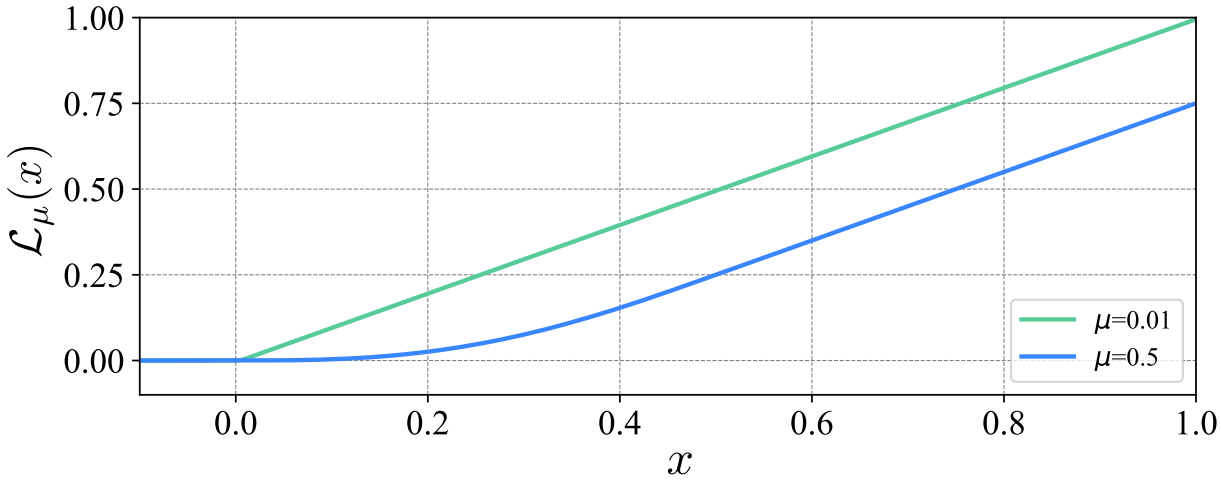


Fig. S9. The plot of the barrier function $\mathcal{L}_\mu$. The plot illustrates the function $\mathcal{L}_\mu$. As the parameter μ decreases from $\mu = 0.5$ to $\mu = 0.01$, the barrier function becomes progressively harder, with a sharper shape.

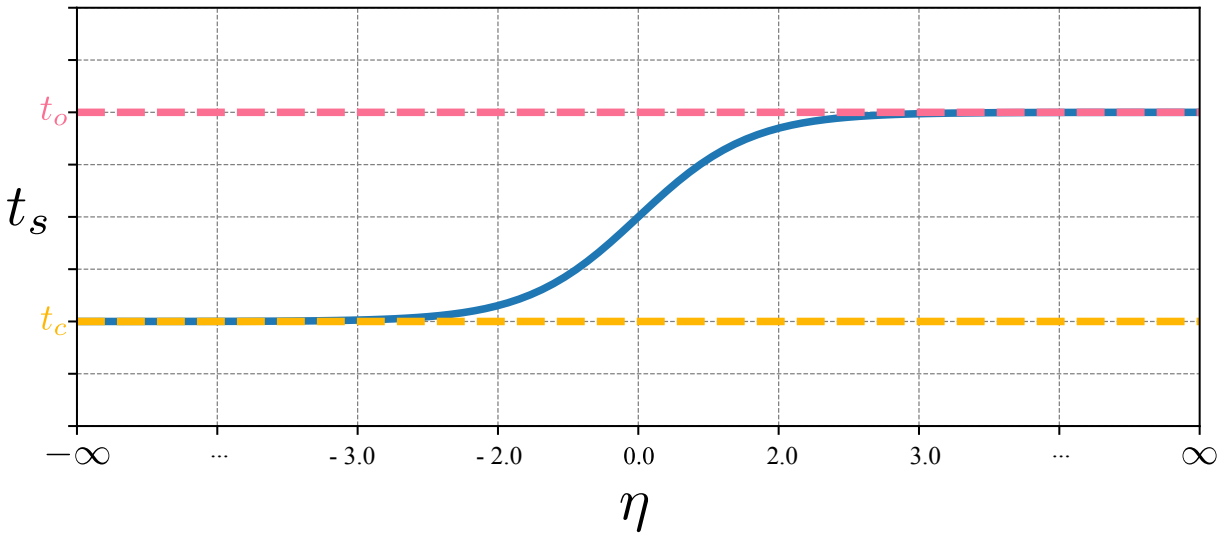


Fig. S10. The plot of the mapping from η to t_s. The plot illustrates a sigmoid-like function that maps $\eta \in \mathbb{R}$ to the interval (t_c, t_o).

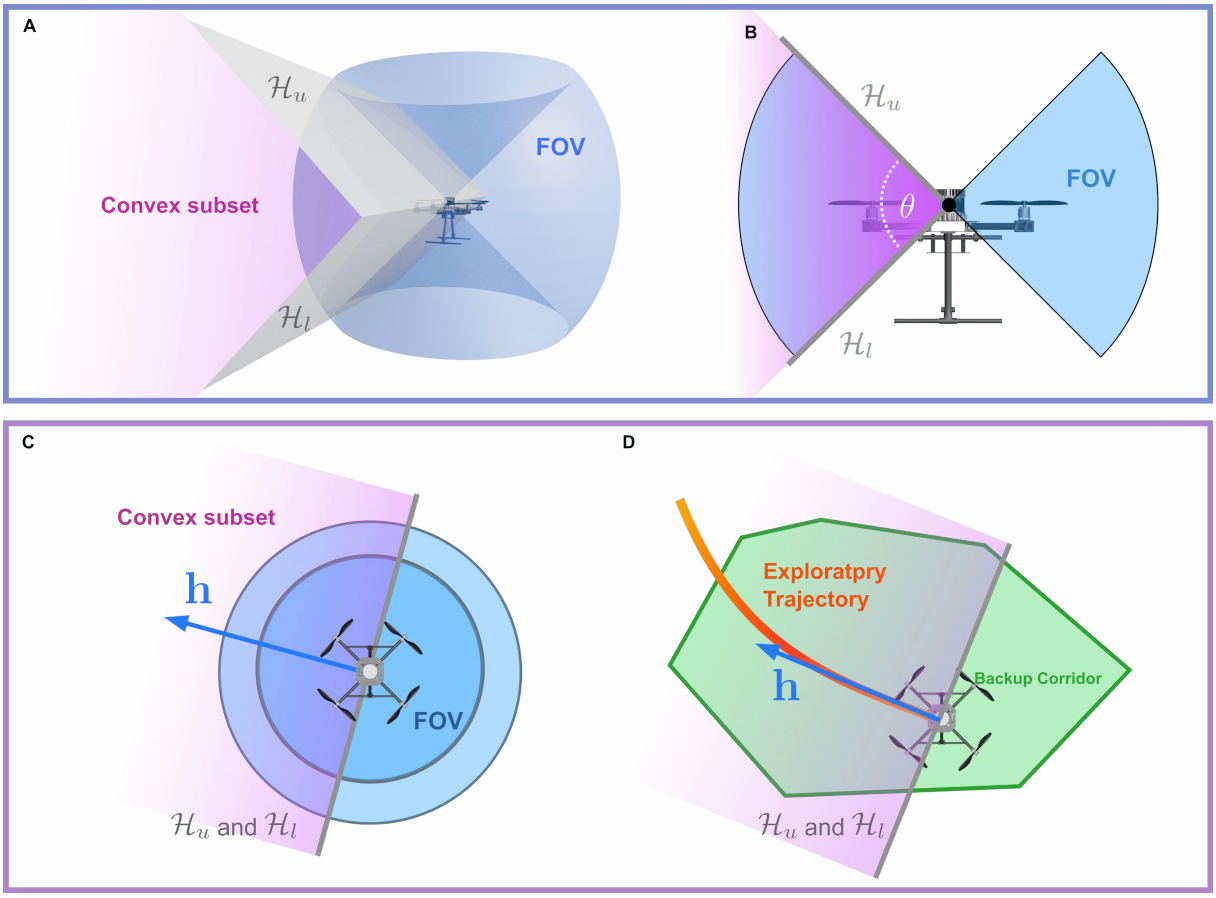


Fig. S11. Backup corridor generation of limited FOV (**A**) The largest convex subset of a non-convex FOV is formed by the intersection of two half-spaces defined by the hyperplanes tangent to the FOV's upper boundary $\mathcal{H}_u$ and lower boundary $\mathcal{H}_l$. (**B**) Side view of the hyperplanes $\mathcal{H}_u$ and $\mathcal{H}_l$ along with the vertical FOV angle, θ. (**C**) Top-down view showing the convex subset and the two hyperplanes, which are uniquely determined by the heading direction $\mathbf{h}$. (**D**) The heading direction is aligned with the exploratory trajectory to ensure that the generated backup corridor contains the initial portion of the exploratory trajectory.

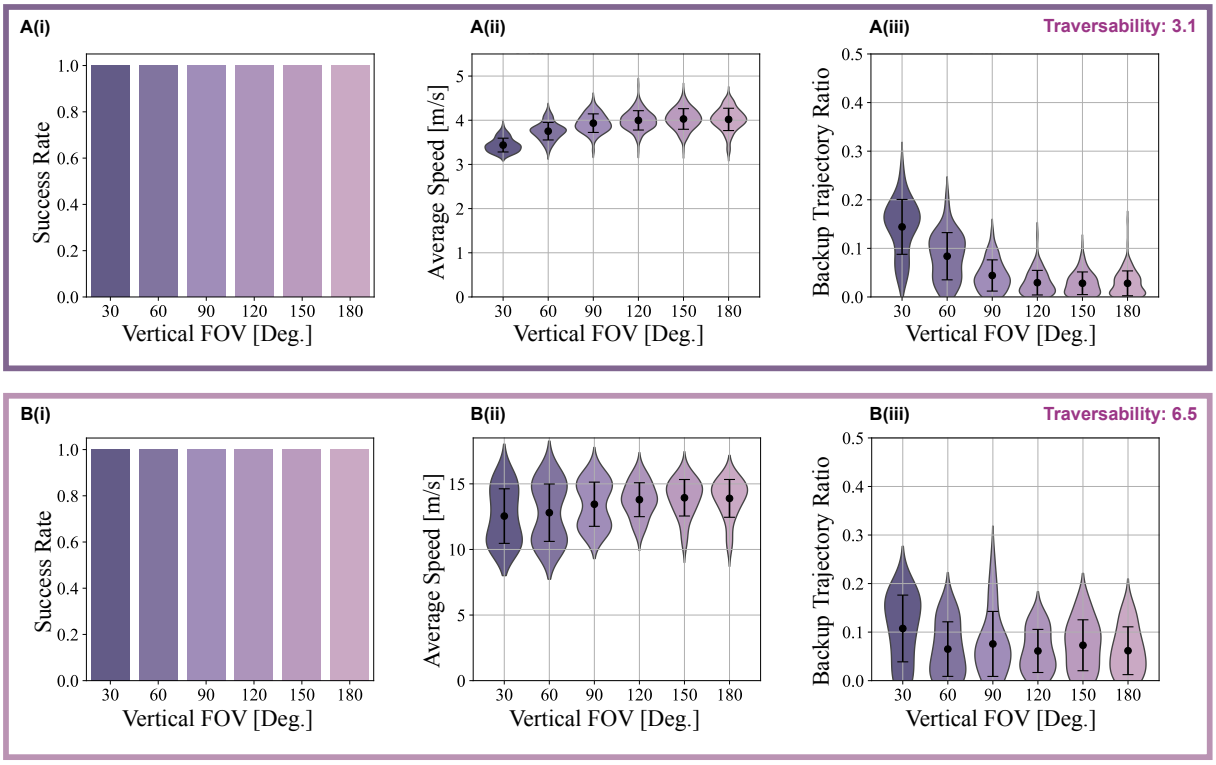


Fig. S12. Benchmark comparison with different vertical FOV (**A**) Benchmarking results on simulation maps with a traversability score of 3.1. (**B**) Benchmarking results on simulation maps with a traversability score of 6.5. For both (**A**) and (**B**), the distributions (ii-iii) illustrate the average flight speed and the ratio of executing the backup trajectory. Error bars represent the standard deviation, while the black points indicate the mean values. Each distribution is based on data from 180 tests.

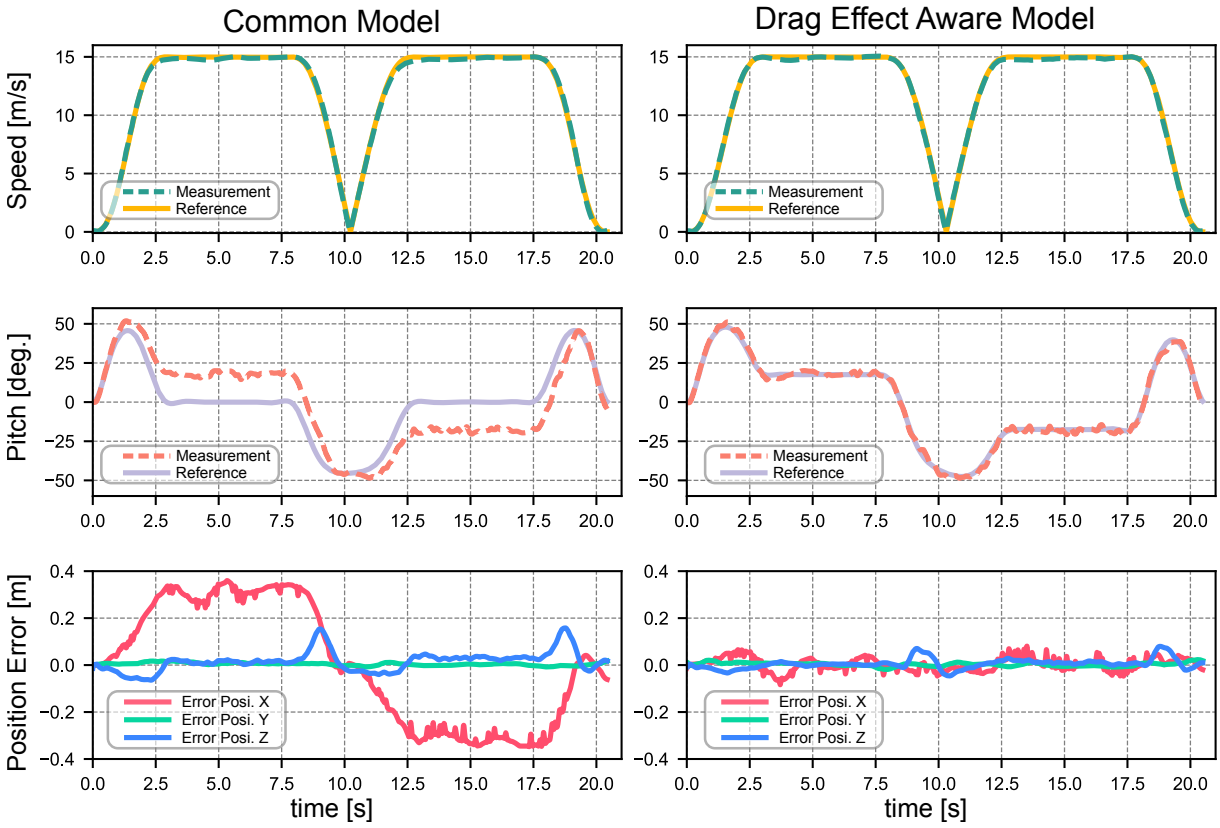


Fig. S13. Comparison of tracking performance with different drone models of MPC. The plot of speed, pitch angle, and position tracking error in each axis.

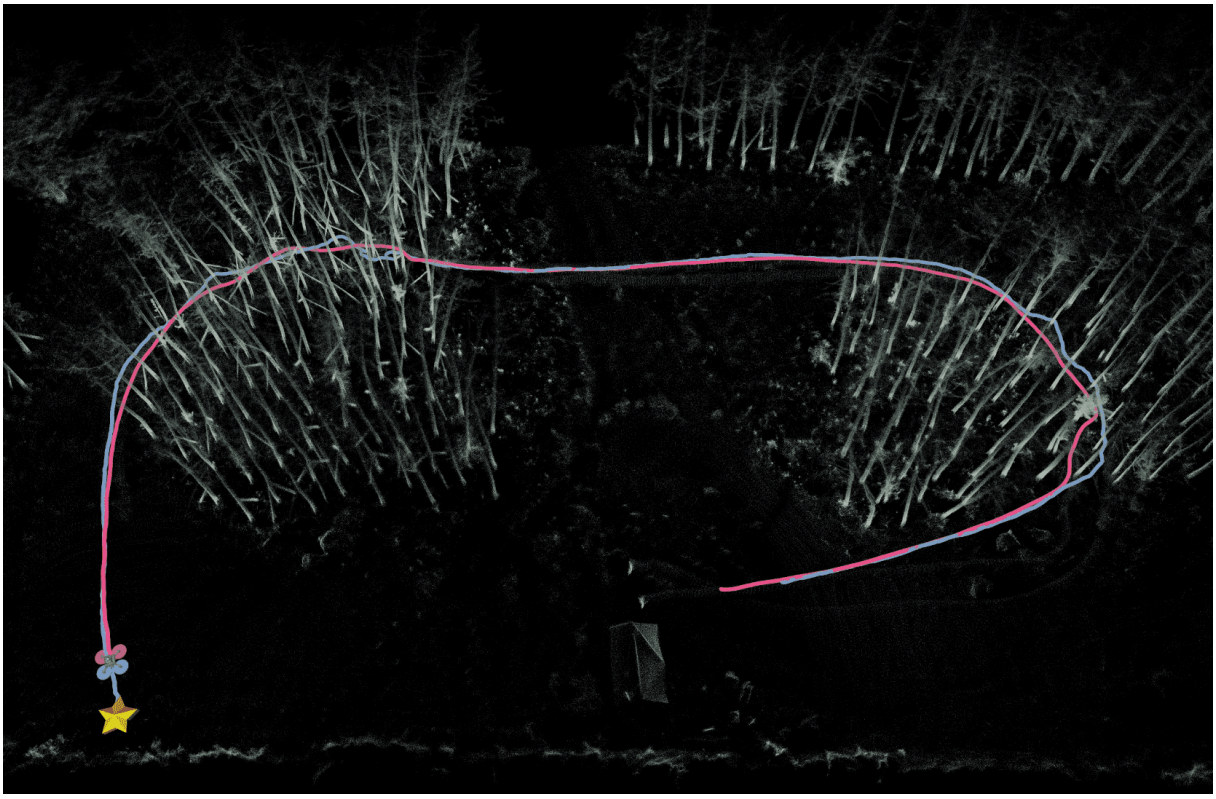
Table S1. Implementation details of different methods

	Type	Mapping	Frontend	Backend
Bubble (*13*)	Optimistic	Point Cloud + Kd-Tree	Sphere-shaped Corridor	Spatio-temporal Trajectory Optimization (STTP)
Raptor (*20*)	Safety-aware	ESDF	Topological Path Searching	Path Guided Trajectory Optimization (PGO)
Faster (*23*)	Safety-assured	OGM	Polyhedron-shaped Corridor	Mixed-Integer Qudratic Program (MIQP)
SUPER (ours)	Safety-assured	Point Cloud	Polyhedron-shaped Corridor	Spatio-temporal Trajectory Optimization (STTP)

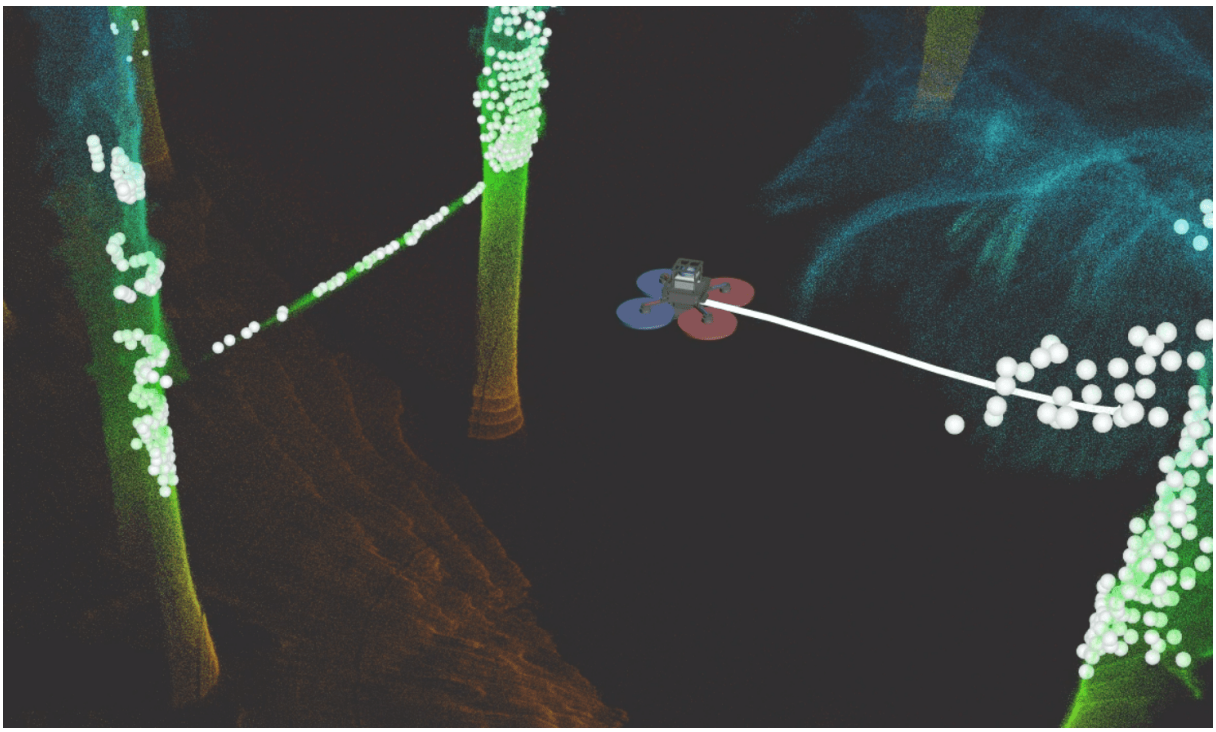
Supplementary Movies

Movie 1. An overview of the proposed SUPER system. SUPER demonstrates its ability to safely navigate through unknown, cluttered environments at high speeds, avoid thin obstacles like power lines, and perform robustly in various scenarios, including object tracking and autonomous exploration.

Movie S1. Safe High-speed Navigation in Unknown Environments.



Movie S2. Navigation in Cluttered Environments.



Movie S3. Demonstration of SUPER's Ability to Avoid Thin Obstacles.

Movie S4. Applications of SUPER in Object Tracking, Autonomous Exploration, and Waypoint Navigation Missions.

REFERENCES AND NOTES

1. J. Verbeke, J. D. Schutter, Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle. *Int. J. Micro Air Veh.* **10**, 3–11 (2018).

2. B. Mishra, D. Garg, P. Narang, V. Mishra, Drone-surveillance for search and rescue in natural disaster. *Comput. Commun.* **156**, 1–10 (2020).

3. S. M. S. M. Daud, M. Y. P. M. Yusof, C. C. Heo, L. S. Khoo, M. K. C. Singh, M. S. Mahmood, H. Nawawi, Applications of drone in disaster management: A scoping review. *Sci. Justice* **62**, 30–42 (2022).

4. B. Rabta, C. Wankmüller, G. Reiner, A drone fleet model for last-mile distribution in disaster relief operations. *Int. J. Disaster Risk Reduct.* **28**, 107–112 (2018).

5. Y. Song, A. Romero, M. Müller, V. Koltun, D. Scaramuzza, Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Sci. Robot.* **8**, eadg1462 (2023).

6. P. Foehn, A. Romero, D. Scaramuzza, Time-optimal planning for quadrotor waypoint flight. *Sci. Robot.* **6**, eabh1221 (2021).

7. A. Romero, R. Penicka, D. Scaramuzza, Time-optimal online replanning for agile quadrotor flight. *IEEE Robot. Autom. Lett.* **7**, 7730–7737 (2022).

8. A. Romero, S. Sun, P. Foehn, D. Scaramuzza, Model predictive contouring control for time-optimal quadrotor flight. *IEEE Trans. Robot.* **38**, 3340–3356 (2022).

9. E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, D. Scaramuzza, Champion-level drone racing using deep reinforcement learning. *Nature* **620**, 982–987 (2023).

10. J. Zhang, R. G. Chadha, V. Velivela, S. Singh, P-cal: Pre-computed alternative lanes for aggressive aerial collision avoidance, paper presented at the 12th International Conference on Field and Service Robotics (FSR), Tokyo, Japan, 31 August 2019.

11. J. Zhang, R. G. Chadha, V. Velivela, S. Singh, P-cap: Pre-computed alternative paths to enable

aggressive aerial maneuvers in cluttered environments, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018), pp. 8456–8463.

12. Y. Ren, S. Liang, F. Zhu, G. Lu, F. Zhang, Online whole-body motion planning for quadrotor using multi-resolution search, in *2023 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 1594–1600.

13. Y. Ren, F. Zhu, W. Liu, Z. Wang, Y. Lin, F. Gao, F. Zhang, Bubble planner: Planning highspeed smooth quadrotor trajectories using receding corridors, in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2022), pp. 6332–6339.

14. X. Zhou, Z. Wang, H. Ye, C. Xu, F. Gao, Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robot. Autom. Lett.* **6**, 478–485 (2020).

15. H. Ye, X. Zhou, Z. Wang, C. Xu, J. Chu, F. Gao, Tgk-planner: An efficient topology guided kinodynamic planner for autonomous quadrotors. *IEEE Robot. Autom. Lett.* **6**, 494–501 (2020).

16. A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, D. Scaramuzza, Learning high-speed flight in the wild. *Sci. Robot.* **6**, eabg5810 (2021).

17. H. D. Escobar-Alvarez, N. Johnson, T. Hebble, K. Klingebiel, S. A. Quintero, J. Regenstein, N. A. Browning, R-advance: Rapid adaptive prediction for vision-based autonomous navigation, control, and evasion. *J. Field Robot.* **35**, 91–100 (2018).

18. L. Quan, Z. Zhang, X. Zhong, C. Xu, F. Gao, Eva-planner: Environmental adaptive quadrotor planning, in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2021), pp. 398–404.

19. L. Wang, Y. Guo, Speed adaptive robot trajectory generation based on derivative property of b-spline curve. *IEEE Robot. Autom. Lett.* **8**, 1905–1911 (2023).

20. B. Zhou, J. Pan, F. Gao, S. Shen, Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Trans. Robot.* **37**, 1992–2009 (2021).

21. H. Oleynikova, Z. Taylor, R. Siegwart, J. Nieto, Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robot. Autom. Lett.* **3**, 1474–1481 (2018).

22. S. Liu, M. Watterson, S. Tang, V. Kumar, High speed navigation for quadrotors with limited onboard sensing, in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016), pp. 1484–1491.

23. J. Tordesillas, B. T. Lopez, M. Everett, J. P. How, Faster: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Transact. Robot.* **38**, 922–938 (2021).

24. E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, D. Scaramuzza, The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *Int. J. Robot. Res.* **36**, 142–149 (2017).

25. Livox Technology Company Limited, Livox Mid-360 User Manual (2023); https://livoxtech.com/mid-360.

26. Wikipedia, Lockheed Martin F-35 Lightning II stealth multirole combat aircraft (2024); https://en.wikipedia.org/wiki/Lockheed_Martin_F-35_Lightning_II.

27. DJI, DJI Mavic 3 (2024); https://dji.com/mavic-3.

28. J. Tordesillas, B. T. Lopez, 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.

29. Z. Wang, X. Zhou, C. Xu, F. Gao, Geometrically constrained trajectory optimization for multicopters. *IEEE Trans. Robot.* **38**, 3259–3278 (2022).

30. C. Nous, R. Meertens, C. De Wagter, G. De Croon, Performance evaluation in obstacle avoidance, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016), pp. 3614–3619.

31. L. Han, F. Gao, B. Zhou, S. Shen, Fiesta: Fast incremental Euclidean distance fields for online motion planning of aerial robots, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2019), pp. 4423–4430.

32. B. Tang, Y. Ren, F. Zhu, R. He, S. Liang, F. Kong, F. Zhang, Bubble explorer: Fast uav exploration in large-scale and cluttered 3D-environments using occlusion-free spheres. arXiv:2304.00852 [cs.RO] (2023).

33. P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, D. Scaramuzza, Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Sci. Robot.* **7**, eabl6259 (2022).

34. NVIDIA Corporation, Nvidia Jetson TX2 embedded AI computing device (2024); https://developer.nvidia.com/embedded/jetson-tx2.

35. X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, F. Gao, Swarm of micro flying robots in the wild. *Sci. Robot.* **7**, eabm5954 (2022).

36. Skydio 2 Plus Enterprise, An intelligent flying robot powered by AI (2024); https://skydio.com/skydio-2-plus-enterprise.

37. A. Harmat, M. Trentini, I. Sharf, Multi-camera tracking and mapping for unmanned aerial vehicles in unstructured environments. *J. Intell. Robot. Syst.* **78**, 291–317 (2015).

38. Intel, Intel RealSense Depth Camera D435i (2024); https://intelrealsense.com/depth-camera-d435i.

39. W. Xu, Y. Cai, D. He, J. Lin, F. Zhang, Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Trans. Robot.* **38**, 2053–2073 (2022).

40. S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, V. Kumar, Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robot. Autom. Lett.* **2**, 1688–1695 (2017).

41. F. Gao, W. Wu, W. Gao, S. Shen, Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments, *J. Field Robot.* **36**, 710–733 (2019).

42. J. Zhang, C. Hu, R. G. Chadha, S. Singh, Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *J. Field Robot.* **37**, 1300–1313 (2020).

43. D. Cheng, F. C. Ojeda, A. Prabhu, X. Liu, A. Zhu, P. C. Green, R. Ehsani, P. Chaudhari, V. Kumar, Treescope: An agricultural robotics dataset for lidar-based mapping of trees in forests and orchards, in *2024 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 14860–14866.

44. P. De Petris, H. Nguyen, M. Dharmadhikari, M. Kulkarni, N. Khedekar, F. Mascarich, K. Alexis, Rmf-owl: A collision-tolerant flying robot for autonomous subterranean exploration, in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)* (IEEE, 2022), pp. 536–543.

45. Y. Cai, F. Kong, Y. Ren, F. Zhu, J. Lin, F. Zhang, Occupancy grid mapping without raycasting for high-resolution LiDAR sensors. *IEEE Trans. Robot.* **40**, 172–192 (2023).

46. Ouster, Ouster OS1-128 sensor (2024); https://ouster.com/products/scanning-lidar/os1-sensor.

47. DJI, DJI Matrice 300 RTK (2024); https://enterprise.dji.com/zh-tw/matrice-300.

48. LiDAR USA, LiDAR USA surveyor 32 (2024); https://lidarusa.com/.

49. Intel, Product Family D400 Series Datasheet (2024); https://intelrealsense.com/download/21345/?tmstv=1697035582.

50. A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonom. Robot.* **34**, 189–206 (2013).

51. Y. Ren, Y. Cai, F. Zhu, S. Liang, F. Zhang, Rog-map: An efficient robocentric occupancy grid map for large-scene and high-resolution lidar-based motion planning. arXiv:2302.14819 [cs.RO]

(2023).

52. F. Kong, W. Xu, Y. Cai, F. Zhang, Avoiding dynamic small obstacles with onboard sensing and computation on aerial robots. *IEEE Robot. Autom. Lett.* **6**, 7869–7876 (2021).

53. H. Wu, Y. Li, W. Xu, F. Kong, F. Zhang, Moving event detection from LiDAR point streams. *Nat. Commun.* **15**, 345 (2024).

54. T-MOTOR, T-MOTOR F90 series racing drone motor specifications (2024); https://store.tmotor.com/goods-1064-F90.html.

55. Dronecode Foundation, PX4: Open source autopilot for drone developers (2024); https://px4.io.

56. J. Chen, K. Su, S. Shen, Real-time safe trajectory generation for quadrotor flight in cluttered environments, in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (IEEE, 2015), pp. 1678–1685.

57. P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**, 100–107 (1968).

58. Q. Wang, Z. Wang, C. Xu, F. Gao, Fast iterative region inflation for computing large 2-d/3-d convex regions of obstacle-free space. arXiv:2403.02977 [cs.RO] (2024).

59. R. Deits, R. Tedrake, Computing large convex regions of obstacle-free space through semidefinite programming, in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics* (Springer, 2015), pp. 109–124.

60. L. Yin, F. Zhu, Y. Ren, F. Kong, F. Zhang, Decentralized swarm trajectory generation for lidar-based aerial tracking in cluttered environments, in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2023), pp. 9285–9292.

61. J. Ji, N. Pan, C. Xu, F. Gao, Elastic tracker: A spatio-temporal trajectory planner for flexible aerial tracking, in *2022 International Conference on Robotics and Automation (ICRA)* (IEEE,

2022), pp. 47–53.

62. S. Liu, N. Atanasov, K. Mohta, V. Kumar, Search-based motion planning for quadrotors using linear quadratic minimum time control, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2017), pp. 2872–2879.

63. ZJU FAST Lab, LBFGS-Lite: A header-only L-BFGS unconstrained optimizer (Github, 2024); https://github.com/ZJU-FAST-Lab/LBFGS-Lite.

64. M. Faessler, A. Franchi, D. Scaramuzza, Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robot. Autom. Lett.* **3**, 620–626 (2017).

65. G. Lu, W. Xu, F. Zhang, On-manifold model predictive control for trajectory tracking on robotic systems. *IEEE Trans. Ind. Electron.* **70**, 9192–9202 (2022).

66. A. Koubaa, Ed., *Robot Operating System (ROS): The Complete Reference (Volume 1)*, vol. **625** of Studies in Computational Intelligence (Springer, 2017).

67. W. Liu, Y. Ren, F. Zhang, Integrated planning and control for quadrotor navigation in presence of suddenly appearing objects and disturbances. *IEEE Robot. Autom. Lett.* **9**, 899–906 (2023).

68. C. Toumieh, A. Lambert, Voxel-grid based convex decomposition of 3D space for safe corridor generation. *J. Intell. Robot. Syst.* **105**, 87 (2022).

69. X. Zhong, Y. Wu, D. Wang, Q. Wang, C. Xu, F. Gao, Generating large convex polytopes directly on point clouds. arXiv:2010.08744 [cs.RO] (2020).

70. C. D. Toth, J. O'Rourke, J. E. Goodman, *Handbook of Discrete and Computational Geometry* (CRC Press, 2017).

71. D. Eberly, Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid (Geometric Tools, 2013); https://geometrictools.com/Documentation/DistancePointEllipseEllipsoid.pdf.

72. Z. Wang, "A geometrical approach to multicopter motion planning," thesis, Zhejiang

University, Hangzhou, China (2022).

73. Y. Cui, D. Sun, K.-C. Toh, On the r-superlinear convergence of the KKT residuals generated by the augmented lagrangian method for convex composite conic programming. *Math. Program.* **178**, 381–415 (2019).

74. W. Wu, mockamap, https://github.com/HKUST-Aerial-Robotics/mockamap.

75. Dronecode Foundation, PX4 software in the loop simulation with Gazebo (2024); https://docs.px4.io/v1.12/en/simulation/gazebo.html.