Plane-Based Spatial Partitioning using Depth Sensors: Computationally Efficient Local Trajectory Planning for Aerial Vehicles over Obstacles

Ting-Hao Wang and Mark W. Mueller

***Abstract*—The agility of quadcopters enables diverse autonomous applications, but rapid trajectory planning in cluttered environments remains challenging due to strict payload constraints on sensing and computation. This paper presents a computationally efficient, memoryless local path planner for navigating quadcopters over obstacles using limited onboard resources. Operating at 20 Hz in a receding horizon fashion, the planner relies solely on the current vehicle state and the latest depth measurements. We introduce a plane-based spatial partitioning method that accelerates trajectory collision checking and selects optimal motion primitives to maximize mission velocity. Our algorithm reduces collision-checking duration to approximately 25% of the prior pyramid-based method while maintaining comparable mission completion behavior. The system is validated through simulation and physical experiments, demonstrating safe and efficient navigation toward designated goals above obstacles.**

***Index Terms*—path-planning, collision-avoidance**

I. Introduction

The agility of aerial vehicles enables applications like industrial inspection, which demand autonomous navigation in cluttered environments. Traditional map-based approaches (e.g., [1]) rely on previously-known global maps, making them infeasible for unseen or dynamic settings. To address this, incremental map-based planners [2]–[4] fuse real-time sensor data into a continuously updated map within a receding horizon framework. However, these methods assume instantaneous and accurate map queries, which is difficult to satisfy given the noise, incompleteness, and latency inherent in real-time onboard perception.

Recent advancements also explore optimization-based planners, utilizing flight corridors [5], [6], alternating minimization [7], or gradient-based refinement [8], [9]. While effective at finding high-quality trajectories, the iterative optimization loops and map maintenance impose significant computational overhead. This demand often exceeds the stringent constraints of weight-restricted aerial platforms.

To circumvent these latencies, memoryless, sampling-based strategies rapidly evaluate a set of feasible motion primitives [10] using single-frame perception, such as depth images [11]. By eliminating the need to maintain a global map, these planners relieve computational burdens and mitigate mapping errors like odometry drift. Because local search prioritizes immediate safety over global mission completion, it is typically integrated with a high-level global planner (Fig. 1(a)). While end-to-end learning offers an alternative [12], it requires exhaustive training and rigorous sim-to-real justification. In contrast, modular integration of a lightweight local planner [13] provides a transparent, robust solution for resource-constrained hardware.

Authors are with the High Performance Robotics Laboratory (HiPeRLab) at the Department of Mechanical Engineering, UC Berkeley.

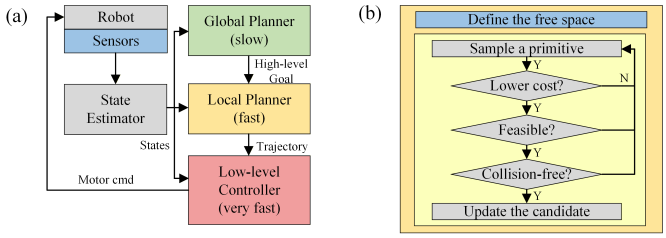

Fig. 1. (a) System architecture integrating global and local planners. (b) The proposed Plane-Based Spatial Partitioning logic, which defines the free space and iteratively evaluates sampled motion primitives.

Building on the memoryless planning principles of [14], this paper presents a local planner tailored for environments characterized by a 2.5-dimensional obstacle distribution, such as those found in agricultural sensing or low-altitude monitoring [15], [16]. Because the airspace above obstacles in these scenarios is typically unobstructed, our planner capitalizes on this structural property by prioritizing upward evasion maneuvers, planning trajectories over obstacles rather than navigating through narrow gaps.

To execute this efficiently, we propose the Plane-Based Spatial Partitioning (PBSP) planner. Instead of computing free space as the union of multiple volumetric pyramids [14], PBSP utilizes intersecting planes to divide the space into free and occupied regions. This spatial partitioning drastically reduces collision-checking durations, enabling the evaluation of a denser set of motion primitives per control cycle and making it ideal for severely computationally constrained vehicles.

The main contributions of this paper are: 1) a memoryless local planner optimized for 2.5D environments, featuring a novel plane-based spatial partitioning method for rapid collision checking; and 2) comprehensive validation through simulation and hardware experiments, demonstrating more computationally efficient navigation than state-of-the-art methods while ensuring robust and safe navigation.

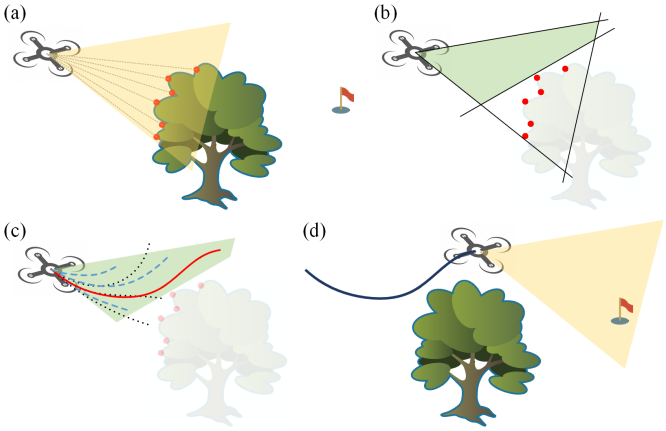

Fig. 2. The PBSP planner pipeline executes by (a) mapping obstacle distributions with depth data within the camera's field of view; (b) defining the safe free space via intersecting dividing planes; (c) sampling motion primitives to identify the feasible, minimum-cost trajectory; and (d) iterating this process in a receding-horizon fashion until the designated goal is reached.

II. Algorithm Description

The memoryless PBSP planner navigates quadcopters toward a high-level goal by generating collision-free trajectories, operating in a receding horizon fashion directly from streaming depth images. As outlined in Fig. 1(b), the algorithm first defines a free space devoid of depth points using dividing planes. It then continuously samples motion primitives, sequentially applying cost, kinematic feasibility, and collision checks. Ordering these checks by ascending computational cost ensures that resource-intensive collision checks are only performed on candidates satisfying preliminary criteria, maximizing the number of evaluated trajectories per control cycle.

Fig. 2 illustrates the operational pipeline of the PBSP planner in detail. In Fig. 2(a), the vehicle is situated in a complex environment with the target goal positioned ahead. The obstacle distribution is represented by the red depth points, while the yellow region denotes the camera's field of view (FOV). As shown in Fig. 2(b), the algorithm processes these points to define the free space using dividing planes; this collision-free volume is highlighted in green and contains no depth points. Subsequently, in Fig. 2(c), parameterized motion primitives are uniformly sampled within a predefined search space. Primitives that satisfy all cost and feasibility constraints are identified as candidates and marked with blue dashed lines, whereas the eliminated ones are denoted by black dotted lines. From this candidate set, the trajectory with the minimum cost, depicted in red, is selected as the output trajectory. This procedure iterates until the goal is reached, as shown in Fig. 2(d).

A. System Modeling Definition

Following [14] and [10], the odometry frame O is an inertial frame where $-{}^{O}z$ aligns with gravity and $+{}^{O}x$ aligns toward the goal. The vehicle's center of mass trajectory $\mathbf{s}(t) \in \mathbb{R}^3$

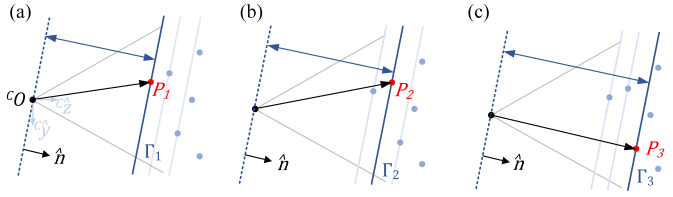

Fig. 3. Determining Γ_t. The depth point P_i with the minimum distance to ${}^{C}O$ along normal $\hat{n}$ is selected to maximize free space.

over duration T is parameterized as a 5th-order minimum-jerk polynomial [10]:

$$\mathbf{s}(t) = \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + \frac{\ddot{\mathbf{s}}(0)}{2}t^2 + \dot{\mathbf{s}}(0)t + \mathbf{s}(0) \quad (1)$$

The coefficients α, β, and γ are uniquely determined by the duration T and the boundary states. To streamline collision checking against depth data, we define a corresponding trajectory $\mathbf{s}_c(t)$ in the camera-fixed frame C:

$$\mathbf{s_c}(t) = \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + \frac{\ddot{\mathbf{s}}_\mathbf{c}(0)}{2}t^2 + \dot{\mathbf{s}}_\mathbf{c}(0)t + \mathbf{s_c}(0) \quad (2)$$

Because the initial states of $\mathbf{s}_c(t)$ match the current known vehicle states, finding trajectory extrema relative to partitioning planes simplifies to analytically solving a 4th-order polynomial.

B. Defining Free Space

To bypass computationally expensive point-wise collision checking, PBSP defines the safe free space as the intersection of obstacle-free half-spaces. We employ two primary dividing planes. The first, Γ_v, is a vertical plane (normal aligned with $+{}^{O}x$), ensuring invariance to vehicle roll and pitch. The second, Γ_t, is pitched 45 degrees relative to Γ_v. Under the assumption that 2.5D environments feature unobstructed upper airspace, this configuration naturally prioritizes upward evasion maneuvers, enabling the vehicle to ascend over obstacles while maintaining forward progress. To maximize the available free space, Γ_t is pushed forward along its normal $\hat{n}$ until it intersects the nearest depth point P_t (Fig. 3), yielding the following formulation:

$$\underset{}{\text{argmax}} \ \|{}^{C}O\Gamma_t\|_2 = \underset{P_i}{\text{argmin}} \ \hat{n} \cdot \overrightarrow{{}^{C}OP_i} \quad (3)$$

While iterating over every depth pixel, this inner loop requires only a single dot product and scalar comparison, representing the theoretical lower bound of computational complexity for full-resolution processing.

Three additional planes, derived from the camera FOV, bound the lateral and bottom limits. Conservatively, Γ_v is capped at the highest observed depth point to prevent navigation into unobserved regions above tall obstacles. A typical scenario demonstrating this necessity is when the vehicle flies directly toward a tall wall where the upper boundary is unobserved (Fig. 4). Finally, each plane's reference point is

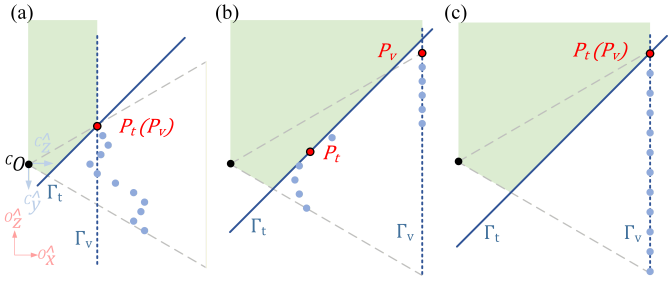

Fig. 4. Planar examples of the free space (green) defined by the intersection of Γ_v, Γ_t, and the camera FOV boundaries.

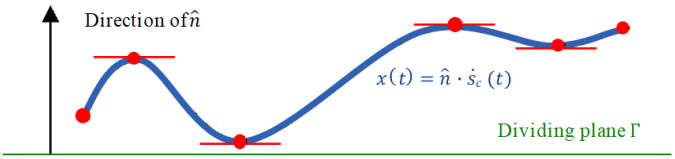

Fig. 5. Characteristic points for collision checking are identified by solving $x(t) = 0$ alongside the boundary points.

shifted inward by the vehicle's radius to account for vehicle physical dimensions.

C. Trajectory Sampling

Motion primitives are generated by uniformly sampling the duration T and endpoint position $\mathbf{s}(T)$ within a predefined search space in front of the vehicle. To ensure safety, terminal velocity $\dot{\mathbf{s}}(T)$ and acceleration $\ddot{\mathbf{s}}(T)$ are set to $\mathbf{0}$. This boundary condition guarantees that if the planner is unable to identify a valid trajectory in subsequent cycles, the vehicle simply continues tracking its current path to a safe stop.

D. Cost, Velocity, and Input Feasibility Checks

Sampled primitives are prioritized by a user-defined cost function balancing average velocity and goal proximity [17]:

$$\underset{\mathbf{s}(T),\ T}{\operatorname{argmin}} \ -\frac{\|\mathbf{s}(T)-\mathbf{s}(0)\|_2}{T} + w\,\|\mathbf{s}_G-\mathbf{s}(T)\|_2 \qquad (4)$$

where $\mathbf{s}_G$ is the goal and w is a tuning weight. While minimizing this local-horizon cost does not guarantee a globally optimal solution, it effectively prevents stagnation by encouraging forward progress based on the vehicle's current partial perception.

To ensure trackability, per-axis velocity constraints (v_{max}) are enforced. The velocity profile is a 4th-order polynomial:

$$\dot{\mathbf{s}}_\mathbf{c}(t) = \frac{\alpha}{24}t^4 + \frac{\beta}{6}t^3 + \frac{\gamma}{2}t^2 + \ddot{\mathbf{s}}_\mathbf{c}(0)t + \dot{\mathbf{s}}_\mathbf{c}(0) \qquad (5)$$

Its extrema are found analytically via the roots of its derivative (a 3rd-order polynomial). If the velocity at any root or boundary exceeds v_{max}, or if established input feasibility limits [10] are violated, the primitive is immediately discarded, sparing the planner from expensive collision checks.

E. Collision Checking Using Predefined Free Space

Evaluating continuous trajectory points is computationally impractical. Instead, we evaluate at most six characteristic points per defining plane. For a plane with normal $\hat{n}$, the trajectory's distance to the plane reaches a local extremum only when its velocity projected along $\hat{n}$ vanishes (Fig. 5). Therefore, we identify these extrema by solving for the roots of the scalar function $x(t)$:

$$x(t) = \hat{n} \cdot \dot{\mathbf{s}}_\mathbf{c}(t) = 0 \qquad (6)$$

Analytically solving the 4th-order polynomial $x(t) = 0$ yields at most four internal roots. By evaluating the half-space constraints exclusively at these characteristic roots and the trajectory boundaries ($t = 0, T$), we mathematically guarantee safety. Specifically, the sign of the dot product between the plane's normal and the vector from its reference point to the characteristic point dictates whether the point resides within the safe free space. If all evaluated points lie within a given half-space, the convexity of the polynomial ensures that the entire trajectory segment remains strictly bounded on the safe side of the plane. A candidate is categorized as collision-free only if it passes this check across all five defining planes. The valid candidate minimizing the cost function is then dispatched to the controller, and the planning cycle iterates.

III. Algorithm Performance

To evaluate the efficacy and computational efficiency of the proposed algorithm, we designed a two-phase simulation study. First, we conducted open-loop benchmarks on individual depth images to quantify the computational overhead. Second, we performed closed-loop flight simulations to validate the robustness of the entire planning pipeline in ensuring mission completion.

A. Computational Complexity and Relevant Work Comparison

To contextualize the experimental results, a general complexity analysis comparing relevant works is presented in Table I. Standard *k-d tree* approaches [18]–[21] scale logarithmically with the input image size n and the number of points checked per trajectory m during collision checking ($\mathcal{O}(m \log n)$). In contrast, both RAPPIDS and the proposed PBSP achieve constant-time ($\mathcal{O}(1)$) collision checks following a linear ($\mathcal{O}(n)$) pre-construction phase. This ensures that the planner maintains high-frequency performance regardless of environmental clutter. Furthermore, PBSP achieves a lower constant factor in practice due to its streamlined algebraic checks against planar boundaries rather than pyramidal intersections.

A computational comparison with relevant research is provided in Table II, detailing the time required to construct the spatial representation (tree/free space), the average collision-checking duration per trajectory, the total computation time for N trajectories, and the number of trajectories evaluated

TABLE I
COMPUTATIONAL COMPLEXITY COMPARISON

Operation	*k-d tree*	*RAPPIDS*	*PBSP (Ours)*
Pre-construction	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Collision check	$\mathcal{O}(m \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

TABLE II
NORMALIZED COLLISION CHECKING PERFORMANCE COMPARISON

	Lopez [18]	**Florence [19]**	**RAPPIDS [14]**	**PBSP (Ours)**
Build [ms]	27.89	23.10	–	**0.15**
Col. check [μs]	19.92	30.80	4.62	**0.81**
Dur. N traj. [ms]	$31 + 0.0249N$	$23 + 0.0308N$	$0.00736N$	$\mathbf{0.58 + 0.00083N}$
Eval. traj. in 30 ms	0	224	4076	**35446**

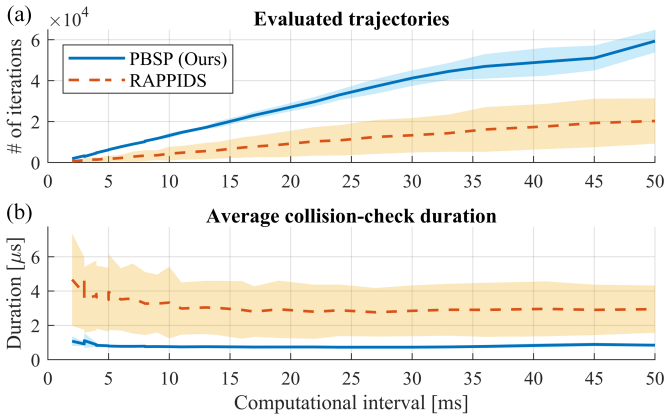

Fig. 6. (a) The number of trajectories evaluated increases monotonically as the computational interval extends. (b) The average duration required to evaluate a single sampled trajectory across varying computational periods.

within a 30 ms control budget. While fixed-time trajectory optimization approaches [20], [21] offer alternative solutions for local planning, their computational structures do not rely on evaluating discrete trajectory samples. Therefore, to ensure a direct and fair comparison of collision-checking efficiency, we focus our quantitative analysis on sampling-based methods [14], [18], [19]. To guarantee a meaningful evaluation across varying hardware platforms and sensor resolutions, the reported runtimes from these prior works have been extrapolated to match the performance of our benchmark CPU (Intel Core i7-12700H) and our operational depth image resolution of 840×480 pixels. Assuming map-building operations scale linearly with the number of processed pixels, the time required to build the spatial representation was multiplied by the relative pixel count ratio and scaled inversely by the single-thread CPU PassMark score [22]. The collision-checking duration per trajectory was scaled solely by the CPU performance ratio. Under these scaling assumptions, the normalized data suggest that processing modern high-resolution depth images causes the computation overhead to severely bottleneck the system. For instance, [18] exhausts the entire 30 ms budget simply by building the spatial tree, resulting in zero evaluated trajectories. In contrast, the proposed PBSP method seamlessly handles high-resolution data and significantly outperforms prior works, achieving an evaluation rate that is orders of magnitude higher than the baseline methods.

B. Open-loop Single-image Tests

Given that this work extends the architecture of the RAPPIDS planner [14], we specifically perform a direct comparative simulation between our proposed PBSP and the baseline RAPPIDS algorithm. Both frameworks process depth images to generate feasible trajectories within a fixed time budget; the fundamental distinction lies in their approach to free space definition and collision checking. To benchmark computational efficiency, we quantified the number of motion primitives each planner could evaluate within the allocated timeframe, using four distinct depth images drawn from simulation and real flight data. A more efficient collision-checking kernel reduces the evaluation time per primitive, enabling the planner to sample a denser set of trajectories within the same control cycle. By exploring the sampled space more thoroughly, the planner increases the probability of discovering a lower-cost solution, yielding smoother and more optimal flight behavior.

Fig. 6(a) illustrates that the number of sampled trajectories scales linearly with the available computational budget. Fig. 6(b) presents the average processing time required to validate a single trajectory across durations from 2 to 50 ms. On average, the PBSP planner requires a computational duration approximately 25% of that required by RAPPIDS to evaluate a single trajectory. This reduction is the primary driver allowing PBSP to evaluate significantly more primitives within the same control cycle.

C. Closed-loop Flights in Simulated Environments

To validate the robustness of the planning pipeline, we evaluated the closed-loop performance of PBSP against RAPPIDS in three distinct Gazebo environments that follow the 2.5D assumption: flat ground, bricks (stacked thin obstacles), and forest (randomly distributed trees). Using RotorS [23] for vehicle dynamics and a simulated RealSense depth camera, we conducted five flight trials per scene. In each trial, the vehicle was initialized 2 m above the ground with a goal 30 m ahead. We report the average forward velocity (v_x) as a metric for trajectory efficiency, noting that higher velocities imply smoother, more optimal path selection.

Fig. 7 presents quantitative performance metrics alongside environment snapshots and example flight trajectories overlaid with perceived depth point clouds. Specifically, we utilize the average forward velocity (v_x) across five independent flights per scene as the primary metric for assessing mission performance. In the flat ground and forest scenarios, both planners successfully completed all missions, with PBSP achieving comparable average velocities (flat: $\sim$2.4 m/s versus 2.1 m/s; forest: $\sim$2.2 m/s versus 1.9 m/s). The critical distinction emerges in the brick environment. While RAPPIDS frequently became trapped in front of vertical structures under the tested configuration, PBSP successfully navigated the field by generating upward trajectories over the obstacles, achieving an average velocity of 1.44 m/s. These behavioral improvements

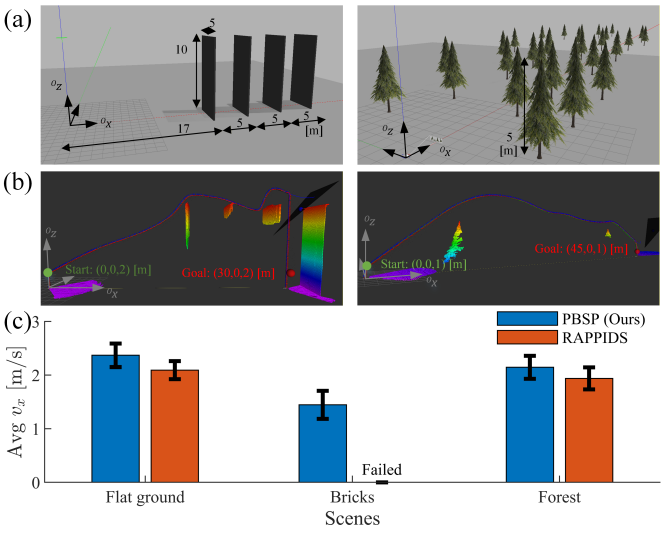

Fig. 7. Closed-loop simulation results and environment visualizations. (a) Snapshots of the brick and forest scenarios. (b) Example PBSP flight trajectories overlaid with perceived depth point clouds. (c) Comparison of the average forward velocity (v_x) across three simulated scenes. Each bar represents the mean of five independent flights, with error bars indicating the standard deviation. PBSP achieves comparable velocities in the flat and forest scenes, and successfully navigates the challenging brick environment where RAPPIDS fails.

correlate with the computational efficiency demonstrated in the open-loop tests, combined with the planner's strategy to prioritize upward maneuvers. By reducing the collision-checking duration, the PBSP planner can evaluate a denser set of candidate motion primitives. This computational advantage not only refines flight trajectories but also effectively leverages the 2.5D environmental structure to explore upward evasion maneuvers, making the system highly suitable for deployment on resource-constrained hardware.

In summary, results from both open-loop benchmarks and closed-loop simulations validate the efficacy of the proposed PBSP planner. While prior volumetric planners remain necessary for fully enclosed 3D structures like tunnels, PBSP deliberately constrains the problem formulation to environments lacking such complex overhangs. By exploiting these specific structural characteristics, PBSP streamlines its collision-checking kernel to achieve highly efficient computation. This efficiency enables the evaluation of a significantly denser trajectory set that prioritizes upward evasion maneuvers within the same control cycle, allowing the planner to successfully navigate its targeted environments with reduced overhead.

IV. Experimental Validation

To validate the feasibility of the proposed PBSP planner in real-world scenarios and verify the fidelity of the simulation environment, we implemented the complete planning pipeline on a custom-built quadrotor platform.

A. System Overview

The custom experimental platform, shown in Fig. 8(a), is a 1.5 kg quadcopter (including a 4S 5000 mAh LiPo battery)

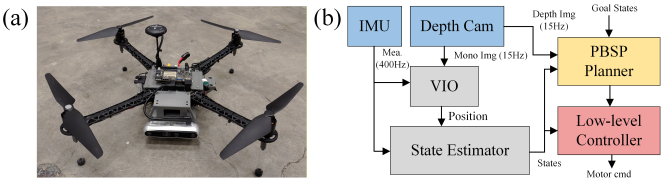

Fig. 8. (a) The custom experimental vehicle, equipped with an Intel RealSense D455 depth camera and a Qualcomm RB5 computing unit. (b) System block diagram detailing the hardware and software pipeline. The vehicle state is assisted via VIO by fusing inertial sensor data with sparse visual features, providing robust estimates that drive both the PBSP planner and the low-level flight controller.

capable of approximately 15 minutes of flight time. The frame has a diagonal motor-to-motor distance of 500 mm and utilizes 254 mm propellers. The onboard computing unit is a Qualcomm RB5, which executes the PBSP planner to generate collision-free trajectories at 20 Hz.

Perception and state estimation are powered by a forward-looking Intel RealSense D455 camera. The camera provides depth images at a resolution of 840×480 pixels for obstacle avoidance and monocular images at 15 Hz for visual-inertial odometry (VIO). As illustrated in Fig. 8(b), the VIO module utilizes OpenVINS [24] to fuse visual features with 400 Hz IMU data, ensuring robust real-time state estimation. A Pixracer flight controller handles the low-level attitude and position control, tracking the trajectory commands generated by the PBSP planner. Additionally, a downward-facing range sensor is integrated to enhance altitude estimation accuracy.

B. Flight Validation and Simulation Correspondence

To evaluate real-world performance and verify sim-to-real fidelity, we conducted outdoor flight tests in an open, flat environment free of obstacles. The primary objective of these baseline flights was to validate the system pipeline on physical hardware and ensure that the vehicle's dynamics align with the simulated predictions. In each trial, the vehicle was commanded to reach a goal 20 m ahead with a target average velocity of 2 m/s. We executed three consecutive flights under identical conditions, with the resulting mean and standard deviation of the trajectories presented in Fig. 9(a) alongside results from three corresponding simulation trials. A visual demonstration of the flight is shown in Fig. 9(b).

The results demonstrate a reasonable correspondence between simulated and experimental behaviors. In the forward x-axis progression, both trajectories exhibit a consistent slope, confirming that the physical system successfully maintains the target velocity of 2 m/s and aligns with simulation predictions. Lateral deviation in the y-axis remains minimal in both cases. An offset is observed in the z-axis upon reaching the goal; this is attributed to the conservative nature of the collision-checking kernel of the PBSP planner when encountering real-world sensor noise. Specifically, transient artifacts in the outdoor depth images are treated as potential obstacles, leading the planner to maintain a safer altitude rather than descending directly to the goal. Despite this variance in sensor input,

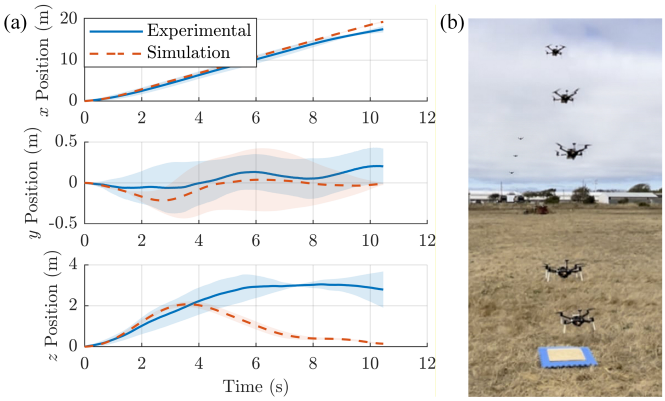

Fig. 9. (a) Comparison of experimental (solid blue) and simulated (dashed orange) trajectories for a 20 m goal with a target velocity of 2 m/s on flat ground. The shaded regions represent the standard deviation across trials. (b) A photographic snapshot of the corresponding flight trajectory.

the overall dynamic behavior closely follows the simulation trends. This deployment confirms that the PBSP planner is computationally feasible for onboard processing and that the simulation results in Section III serve as a valid predictor of the system's behavioral logic and computational timing.

V. Conclusion

This paper presented the Plane-Based Spatial Partitioning (PBSP) planner, a computationally efficient, memoryless local path planner designed to address the strict payload and computational constraints of agile quadcopters. Capitalizing on the structural properties of 2.5D environments, where the airspace above obstacles is typically unobstructed, the proposed method utilizes intersecting planes to define safe free space directly from streaming depth images, thereby enabling highly efficient trajectory collision checking.

As our primary contribution, the spatial partitioning technique reduces the collision-checking duration to approximately 25% of that required by prior pyramid-based algorithm. This reduction in computational overhead allows the planner to evaluate a denser set of motion primitives at 20 Hz without maintaining a global map. Consequently, PBSP maintains comparable average flight velocities while exhibiting behavioral robustness, successfully executing upward evasion maneuvers in cluttered scenarios where baseline methods tend to stagnate.

Extensive validation across both simulation and real-world hardware experiments demonstrated that the PBSP pipeline provides a robust, real-time solution for safe autonomous navigation. By eliminating the latencies inherently associated with iterative optimization and map maintenance, the proposed planner significantly broadens the feasibility of deploying autonomous UAVs on severely resource-restricted hardware.

References

[1] L. Quan, L. Yin, T. Zhang, M. Wang, R. Wang, S. Zhong, X. Zhou, Y. Cao, C. Xu, and F. Gao, "Robust and efficient trajectory planning for formation flight in dense environments," *Transactions on Robotics*, vol. 39, no. 1, pp. 1–18, 2023.

[2] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2452–2458.

[3] Z. Xu, C. Suzuki, X. Zhan, and K. Shimada, "Heuristic-based incremental probabilistic roadmap for efficient uav exploration in dynamic environments," *International Conference on Robotics and Automation (ICRA)*, pp. 11 832–11 838, 2024.

[4] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," in *International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3235–3240.

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

[6] X. Peng, X. Wang, J. Shen, and Y. Lv, "Multi-robot trajectory planning using safety-enhanced a* and piecewise bézier curves," in *International Conference on Unmanned Systems (ICUS)*. IEEE, 2025, pp. 1529–1534.

[7] V. K. Adajania, S. Zhou, A. K. Singh, and A. P. Schoellig, "Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments," in *International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1421–1427.

[8] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2021.

[9] Z. Xu, Y. Xiu, X. Zhan, B. Chen, and K. Shimada, "Vision-aided uav navigation and dynamic obstacle avoidance using gradient-based b-spline trajectory optimization," in *International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1–7.

[10] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[11] J. Lee, X. Wu, S. J. Lee, and M. W. Mueller, "Autonomous flight through cluttered outdoor environments using a memoryless planner," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 1131–1138.

[12] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[13] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[14] N. Bucki, J. Lee, and M. W. Mueller, "Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation," *Robotics and Automation Letters*, vol. 5, no. 3, pp. 4626–4633, 2020.

[15] K. Anderson and K. J. Gaston, "Lightweight unmanned aerial vehicles will revolutionize spatial ecology," *Frontiers in Ecology and the Environment*, vol. 11, no. 3, pp. 138–146, 2013.

[16] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (uavs)," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 4, pp. 281–340, 2015.

[17] X. Wu, S. Chen, K. Sreenath, and M. W. Mueller, "Perception-aware receding horizon trajectory planning for multicopters with visual-inertial odometry," *IEEE Access*, vol. 10, pp. 87 911–87 922, 2022.

[18] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5759–5765.

[19] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *Algorithmic Foundations of Robotics XII*, ser. Springer Tracts in Advanced Robotics, 2020, pp. 471–487.

[20] F. Kong, W. Xu, Y. Cai, and F. Zhang, "Avoiding dynamic small obstacles with onboard sensing and computation on aerial robots," *Robotics and Automation Letters*, vol. 6, no. 2, pp. 786–793, 2021.

[21] J. Ji, Z. Wang, Y. Wang, C. Xu, and F. Gao, "Mapless-planner: A robust and fast planning framework for aggressive autonomous flight without map fusion," in *International Conference on Robotics and Automation (ICRA)*, 2021, pp. 5319–5325.

[22] PassMark Software, "CPU benchmarks," https://www.cpubenchmark.net/singleThread.html, 2026, accessed: Feb., 2026.

[23] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS): The Complete Reference*. Springer, 2016, pp. 595–625.

[24] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4666–4672.