Into the Unknown: UAV Navigation with F-Star (F*)

Tobias Fischer*†, Zubair Shaik*, Andreas Mitschele-Thiel*, and Florian Klingler*†
*Dept. of Computer Science and Automation, TU Ilmenau, Germany
†Chair of Computer Networks, Dept. of Computer Science, University of Bamberg, Germany
{zubair.shaik, andreas.mitschele-thiel}@tu-ilmenau.de
{tobias.fischer, florian.klingler}@uni-bamberg.de | www.uni-bamberg.de/comnet

***Abstract*—With increasing demand for flexible aerial systems, Unmanned Aerial Vehicles (UAVs) have become central to research in areas like emergency response, mobile networking, and surveillance. Multirotor UAVs enable seamless movement in all directions, but introduce challenges in localization, mapping, and real-time path planning. Modern missions require more than basic collision avoidance or shortest-path computation; they demand energy-efficient, safe, and adaptive navigation. In networking applications, additional constraints include restricted airspaces, frequency zoning, radio silence mandates, and uplink visibility. Although UAV path planning has been studied extensively, many existing methods are computationally intensive. To address this, we propose F-Star (F*), a framework that integrates adaptive spatial partitioning, weighted graph search, and trajectory smoothing for efficient and safe UAV navigation, while providing extensibility for multi-objective planning. A preliminary study in realistic scenarios demonstrates our proposed solutions effectiveness in dynamic obstacle avoidance with low computational cost and extensibility for multi-objective planning domains.**

***Index Terms*—Navigation, Path planning, SLAM, Autonomous vehicles, Obstacle avoidance**

I. Introduction

Unmanned Aerial Vehicles (UAVs) gained a lot of interest in the past for various application domains ranging from automatic delivery services in context of logistics, up to surveillance and videography purposes. Among those, a very promising use case for UAV based systems is networking support, e.g., using a UAV to carry networking equipment like small cell basestations for supporting communication systems in remote areas or disaster sites. For operation, these UAVs can use terrestrial, or even satellite links, making it a viable option due to the increasing popularity of Low Earth Orbit based satellite systems. Furthermore, the simultaneous deployment of multiple UAVs as mobile network nodes necessitates reliable inter-node communication visibility.

To operate autonomously, UAVs rely on multiple algorithmic components including collision avoidance, sensor data interpretation, and fault detection. Yet, one of the most critical elements in ensuring robust autonomous behavior is path planning – computing a feasible and safe trajectory for the UAV from a source location to one or multiple destination positions.

Many path planning algorithms simply aim to avoid static or dynamic obstacles, prioritizing shortest-path efficiency [1]. In the context of aerial robotics, this approach is often insufficient, as flight introduces additional constraints such as aerodynamic disturbances, limited energy availability, and three-dimensional mobility, significantly increasing computational complexity [2].

Algorithms designed without consideration for air turbulence in proximity to obstacles or the drone's inertia may yield unrealistic trajectories, possibly posing a risk in physical deployment. To address this, smooth and obstacle-buffered trajectories are preferable, as they minimize jerk, extend flight time, and improve robustness against environmental disturbances. Additionally, UAVs in the context of network mobility – e.g., mobile base stations – demand multi-objective optimization, including restricted aerospace zones, communication-related constraints such as uplink visibility and allocated frequency regions, and dynamic interference sources. These factors transform the path planning problem into a high-dimensional decision-making challenge.

As a first step toward a comprehensive multi-objective planning framework, this paper presents the core architecture and validation of a static and dynamic path planning system for obstacle avoidance. We target a modular and extensible system, enabling future integrations for addressing the complex challenges introduced by multi-objective UAV operations in network-aware aerial systems.

II. Related Work and Fundamentals

Ait Saadi et al. [1] explored different path planning solutions for UAV applications. We evaluated their results for our application of multi-objective planning in aerial networked robotics, specifically mobile base stations. Their findings conclude, that *A-Star (A*)* provides good solutions with fast convergence but is ill-suited for multi-objective planning, while *Bi-Level Programming (BLP)* seems to be the optimal approach for such complex scenarios [1].

Liu et al. [3] claim, that their *BLP* based algorithm can be applied to different kind of threats, by expressing those as stochastic models. Due to concerns regarding the complexity and suitability of *BLP* for our problem, we instead research adapting the *A** algorithm, as it serves as a strong baseline, offering adequate performance and scalability [1].

Details on mapping structures for UAV path planning are presented in Section III. Graph-based pathfinding algorithms are discussed in Section IV, followed by trajectory smoothing techniques in Section V. Our proposed approach, *F-Star (F*)*, is introduced in Section VI, highlighting key deviations from existing methods. Finally, we evaluate our solution in synthetic tests and simulations, which is outlined in Section VII.

50th IEEE Conference on Local Computer Networks (LCN), Special Track on Autonomous Intelligence for Low-Altitude Economy

III. Mapping in robotics

UAV map representations can be broadly categorized into two types:

- *Metric maps* use geometric information to place objects within a defined coordinate space – in this case, the three-dimensional volume in which a drone operates.
- *Topological maps*, by contrast, capture only the connectivity or adjacency between discrete locations, forming a graph of navigable routes without precise geometric detail.

While topological maps are lightweight and efficient for route computation, they lack spatial granularity, making them unsuitable for tasks like dynamic obstacle avoidance. To address this trade-off, we adopt a common hybrid approach, combining high-resolution metric data for obstacle representation with graph-based structures for efficient navigation planning [4].

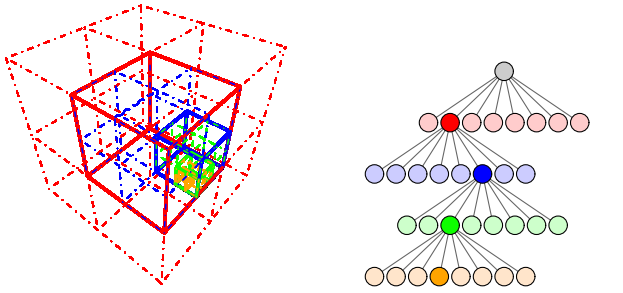
A. Occupancy Grid Maps

A foundational method for representing spatial environments is the occupancy grid map, which discretizes 3D space into uniform cells arranged in a regular grid. These volumetric units, known as voxels (volume pixels), are the three-dimensional analog of image pixels and can be resized to control spatial resolution. While uniform resolution simplifies processing, storing different levels of detail within a single map would allow for efficient representation of both dense and sparse regions, albeit at the cost of increased complexity in map maintenance and traversal [5]. The most basic implementation stores a binary state – occupied or free – for each voxel. However, due to sensor noise and uncertainty, probabilistic representations are preferred in physical deployments.

In a probabilistic occupancy grid, each cell is initialized with an intermediate occupancy probability value, indicating unknown state. Subsequent sensor readings update this value: detections of obstacles increment the occupancy probability, while observations suggesting free space decrement it. [6]

One limitation of occupancy grids is the inefficiency of Nearest-Neighbour Search (NNS). To locate the nearest obstacle, the system must perform an expanding search in all directions, a computationally expensive operation, especially in 3D environments. Adaptive resolution can improve efficiency, but non-uniform voxel sizes introduce irregular neighbor relationships that complicate traversal and path planning. Nevertheless, occupancy grids remain popular in robotics due to their intuitive structure and compatibility with topological representations.

Graph-based planners can be overlaid by treating neighboring voxels as connected nodes, with edge weights proportional to Euclidean distances. Typical voxel neighborhood definitions in three dimensions include shared faces (6-connectivity), edges (18-connectivity), or corners (26-connectivity). While face-only connectivity enhances safety by avoiding diagonal shortcuts, including edges and corners expands the possible solution set and can yield shorter paths.



(a) Visual Octree representation (b) Logical Octree representation

Figure 1: Representation of an Octree visually (left), with its corresponding graph structure (right).

B. Octrees

Whereas occupancy grids represent a bottom-up discretization of space, octrees offer a top-down, hierarchical approach to space partitioning. Octrees, as visualized in Figure 1a, recursively divide 3D space by evenly splitting a cube into eight smaller sub-cubes, allowing for efficient representation of heterogeneous environments. This structure extends the 2D quadtree concept – where each square region can be divided by splitting it evenly among both axes – into three dimensions.

A common approach to octree implementation uses a graph structure internally, where every cube is represented by a graph node, each having either zero or eight children referencing the containing sub-cubes.

As subdivision is applied only in regions that require higher detail, memory usage for uniform or unexplored areas is decreased significantly in comparison to regular grid maps. This adaptive resolution is especially valuable in outdoor UAV applications, where large volumes of mostly free space are punctuated by small obstacles.

Efficient Nearest-Neighbour Search (NNS) in octrees can be achieved by recursively iterating over all nodes through their child relationships. Using heuristics improves this process by prioritizing traversal toward nodes closest to the query point, and terminating early when an occupied leaf node is encountered.

IV. Graph-Based Path Planning

Given is a topological map as a graph $G = (V, E, c)$, where:

- V is a set of vertices,
- $E \subseteq V \times V$ a set of connecting edges and
- $c : E \to \mathbb{R}_0^+$ a cost-function assigning non-negative traversal cost to each edge.

Two commonly used distance functions in pathfinding are the Manhattan distance (1) and the Euclidean distance (2), both defined for n-dimensional spaces:

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{1}$$

$$d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

A simple path $P_{v,w}$ in G with $v, w \in V$ and $v \neq w$, is defined as a sequence of distinct vertices $(v_0, v_1, \ldots, v_k)$, such that $v_0 = v$, $v_k = w$, and $\forall\ i \in \{1, 2, \ldots, k\} : (v_{i-1}, v_i) \in E$. Consequently, let $\mathbb{P}_{v,w}$ denote the set of all valid paths from v to w in G. The path distance between two nodes v_x, v_y on a path $P_{v,w}$ is defined as the sum of the edge costs along that sub-path and denoted as

$$d_{P_{v,w}}(v_x, v_y) = \sum_{(u,v) \in P_{v_x, v_y}} c(u, v) \tag{3}$$

Therefore, the length of the full path is $l(P_{v,w}) = d_{P_{v,w}}(v, w)$. Graph-based path planning defines the process of determining the minimum of $l(P_{s,t})$ for all $P_{s,t} \in \mathbb{P}_{s,t}$ on a graph G, with $s, t \in V$, finding the shortest path between the two. Although edge cost often reflects geometric distance, we define the shortest path as the least-cost path between two nodes $s, t \in V$:

$$\text{minpath}(s, t) = \arg \min_{P_{s,t} \in \mathbb{P}_{s,t}} l(P_{s,t}) \tag{4}$$

A. Dijkstra's Algorithm and A*

Dijkstra's Algorithm is one of the foundational approaches to shortest-path search. Let us define a cost function for traversal as $c(n, m) = f(m) + d(n, m)$, where $d(n, m)$ is the edge weight (e.g., Euclidean distance), and $f(m)$ is a path cost estimate from s to m. Further, let $g(n)$ denote the currently known shortest-path cost from n to t: $g(n) = \text{minpath}(n, t)$.

In the original formulation by Dijkstra [7], a uniform-cost search with $f(n) = g(n)$ is executed with a reversed direction, expanding from the goal node t rather than the start node s, which yields all shortest paths to t from any $v \in V$. This has advantages in robotic scenarios where path deviation occurs: rather than recomputing the entire path, the robot can resume from the nearest segment of a precomputed tree.

The *A** algorithm enhances Dijkstra's method by introducing a heuristic estimate: $f(n) = g(n) + h(n)$, where $h(n)$ estimates the remaining cost from n to the goal t – commonly $h(n) = d(n, t)$ [8]. This converts the search into a best-first traversal, allowing early termination once t is reached. According to Hart et al. [8], if $h(n)$ is admissible (i.e., never overestimates the true cost), *A** guarantees an optimal solution.

B. Safety focused A*

Yu et al. [9] modified *A** for safety-critical applications by adapting the heuristic to penalize proximity to obstacles. Their 2D formulation extends naturally to 3D.

Let $d_{\text{obst}}(n)$ denote the distance from node n to the nearest obstacle, computed via a nearest-neighbor query in the map. Following the recommendations provided by Yu et al. [9], a safety-aware heuristic can be written as:

$$h(n) = d(n, t) + \frac{100\%}{5 \cdot d_{\text{obst}}(n) + 1} \tag{5}$$

This formulation increases the cost of nodes near obstacles, effectively biasing the planner away from dangerous regions without the necessity to prune the graph.

V. Trajectory Smoothing

For some UAV systems – e.g Multirotor drones, movement in any direction is theoretically possible at any time. However, to optimize energy efficiency and improve fluidity, it is beneficial to smooth the computed path to leverage the system's inertia and minimize abrupt direction changes.

A. Splines

Linear Interpolation (LERP) is a method to generate intermediate points between two given positions, A and B, in a vector space $\mathbb{R}^n$, based on a parameter $t \in [0, 1]$. The resulting point lies on the line segment connecting A and B:

$$\text{Lerp}_{A,B}(t) = (1 - t) \cdot A + t \cdot B = A + t \cdot (B - A) \tag{6}$$

This linear interpolation forms the basis for constructing more complex curves such as Bézier curves through recursive applications.

B. Bézier Curves

Bézier curves provide a way to generate smooth trajectories from a sequence of control points. A quadratic Bézier curve, for example, interpolates three points A, B, and C by recursively applying LERP:

$$\begin{aligned} \text{Lerp}_{A,B,C}(t) &= (1 - t) \cdot \text{Lerp}_{A,B}(t) + t \cdot \text{Lerp}_{B,C}(t) \\ &= (1 - t)^2 \cdot A + 2t \cdot (1 - t) \cdot B + t^2 \cdot C \end{aligned} \tag{7}$$

This recursive construction generalizes into the Bernstein form for Bézier curves of arbitrary degree n, using De Casteljau's algorithm:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} \cdot (1 - t)^{n-i} \cdot t^i p_i \tag{8}$$

With p_i being the i-th control point in order (for A, B, C : $p_0 = A, p_1 = B, p_2 = C$). The binomial term is known as the Bernstein basis polynomial and controls the influence of each control point in relation to the input parameter t. Bézier splines of degree n generate smooth curves between n control points, visualized in Figure 2.

To retain local control and avoid over-smoothing with high-degree curves, it is common to use piecewise cubic Bézier curves (degree 3) with shared endpoints or tangent blending, forming what is known as composite Bézier splines [10].

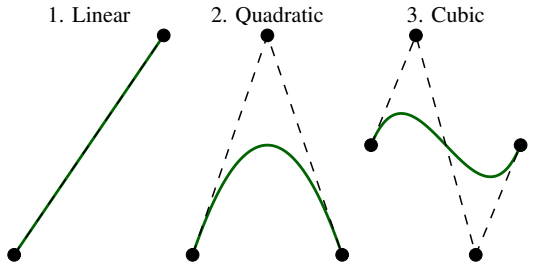


Figure 2: Visualization of different order Bézier splines. More control points result in a reduction in local control, meaning each point has less influence on the final curve.

C. Catmull-Rom Splines

An alternative composite parametric curve are Catmull-Rom splines, that interpolate the given control points directly, passing through each one. They are defined using four sequential control points A, B, C, D, and interpolate between B and C. Missing points can be extrapolated by mirroring an existing point to the other side of an adjacent control point. The matrix formulation is:

$$f(t) = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}^T \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\alpha & 0 & \alpha & 0 \\ 2\alpha & \alpha-3 & 3-2\alpha & -\alpha \\ -\alpha & 2-\alpha & \alpha-2 & \alpha \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \quad (9)$$

Yuksel et al. [11] analyzed the behavior of Catmull-Rom splines, when adjusting the spline tension α between 0 (uniform), 0.5 (centripetal), and 1 (chordal). This parameter influences how tightly the curve follows the given control points, affecting both smoothness and the likelihood of overshooting. Yuksel et al. [11] observed, that centripetal splines offer good smoothing results without deviating too far from the given curve, an observation we could confirm during initial experimentation.

D. Moving Averages

Moving or rolling average smoothing is a simple yet effective algorithm often used in signal processing. It operates by averaging over a sliding window centered on each point in the input sequence [12]. In our context, the input is a list of waypoints forming a path, and the window defines a subset of these points.

VI. Our F* approach: Single- and Multi-Objective Planning

Inspired by the ideas from a safety focused *A** approach [9], in this section we propose our *F** path finding approach, which takes advantage of an improved cost function in favor of removing nodes for optimized path planning.

A. Optimized Octree

To maintain neighborhood relations in the octree structure, each node stores its neighbors directly, avoiding costly re-computation during graph conversion. When a node splits, its children are interconnected and linked to the original neighbors that satisfy the 26-connectivity criterion. For consolidation of child nodes to one parent cell – e.g. when all sub-cubes are marked as free or occupied – all neighbors of all children are added to the parent node. This recursive approach supports dynamic updates efficiently, including root extension, without reprocessing previous obstacles. During graph conversion, only newly added, non-obstructed nodes are iterated over and converted to graph vertices using the stored neighborhood data.

B. Guiding the graph search

We use the squared euclidean distance for all distance calculations during mapping, planning, and smoothing:

$$(d_{\text{Euclidean}}(x,y))^2 = \sum_{i=1}^{n} (x_i - y_i)^2 \quad (10)$$

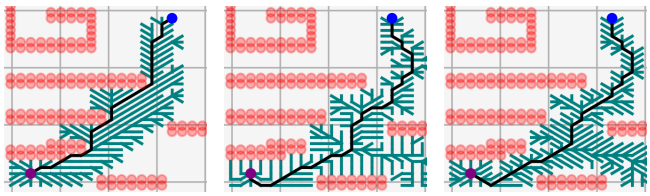
To establish a baseline for safe trajectory planning, we first defined a purely safety-oriented objective that maximizes distance from obstacles. Instead of modifying $g(n)$ in an additive way, we employ a multiplicative strategy: $f(n) = g(n) \cdot h(n)$. Let's define a safeness-estimation function $s(n) = d_{obst}(n)^{-2}$, using the inverse square of the distance to the nearest obstacle. Substituting $s(n)$ as our heuristic yields:

$$f(n) = g(n) \cdot s(n) = g(n) \cdot d_{obst}(n)^{-2} = \frac{g(n)}{d_{obst}(n)^2} \quad (11)$$

This formulation is similar to potential field methods, where obstacles exert repelling forces [13]. However, the safest paths tend to be unnecessarily long, which is undesirable for battery-limited UAVs. To address this, we balance safety with goal-seeking by combining both objectives:

$$f(n) = g(n) \cdot (s(n) + d(n,t)) \quad (12)$$

Initial testing validated our concepts, as visualized in Figure 3. The leftmost graph shows *A**, the middle one displays the initial safe algorithm, while the result of *F** is on the right.



(a) A* (b) safest path (c) F*

Figure 3: Examples for different path shapes: *A** (left), *safest path* (middle), and our approach *F** (right). Whilst both our approaches show similar performance (i.e., safe distances), from a computational perspective *F** is more efficient.

C. Multi-Objective F*

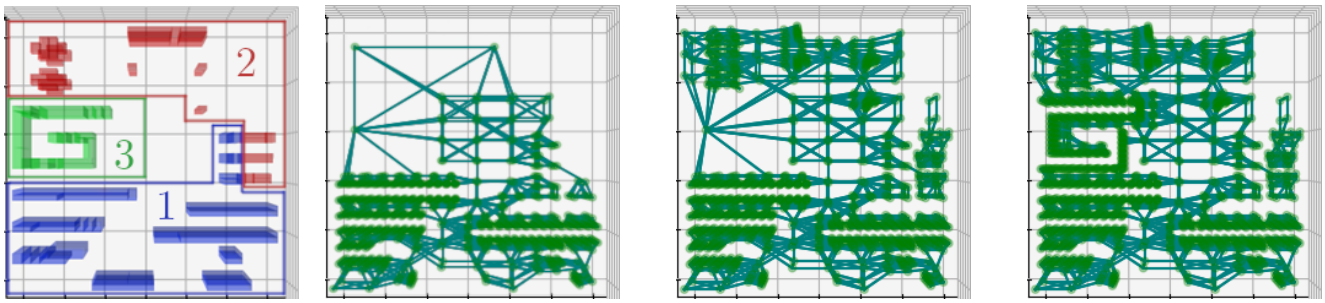
Initial extension concepts for our *F** formulation show the possibility of replacing $s(n)$ with a weighted multi-objective function. Let $s_i(n)$ denote the i-th objective function and w_i its corresponding weight. The combined safety function becomes:

$$s(n) = \sum_{i=0}^{n} w_i \cdot s_i(n)$$
$$\text{subject to } 0 \le w_i \le 1, \quad \sum_{i=0}^{n} w_i = 1 \quad (13)$$

This formulation enables flexible multi-objective planning, though development is ongoing and results are preliminary.

D. Variable trajectory smoothing

While Catmull-Rom splines showed excellent smoothing results for most paths during initial testing, empirical tests revealed that simple moving averages with small window sizes (e.g., 3–5) provided a favorable trade-off between smoothing and computation time. We opted to dynamically calculate the window size as one sixth the number of points in the input path, with a minimum bound of 2.



(a) Three-Stage Test Map (b) Phase 1 (c) Phase 2 (d) Phase 3

Figure 4: Simplified visualization of our Test Map, and the octree expansion after each phase of map exploration.

VII. Evaluation

To evaluate the behavior and performance of the presented planning strategies under stable and repeatable conditions, we constructed a synthetic test map, enabling systematic exploration and benchmarking. This allowed us to validate the core functionality of our algorithms and provide a consistent basis for performance comparisons.

A. Experimental Setup and Analysis

The test map as shown in Figure 4a was designed to gradually expand in three phases, enabling an incremental evaluation of map growth and its impact on planning performance. This structure helps isolate the performance contributions of partial updates, spatial complexity, and resolution of the underlying data structure. As the system explores the map, the octree-based data representation adapts accordingly, building a sparse graph network. This evolution is visualized in a slightly simplified manner in Figure 4. Each planning iteration compares the performance of the Grid-based and octree-based maps across several metrics, including pathfinding runtime, graph size, path length, and Nearest-Neighbour Search (NNS) performance.

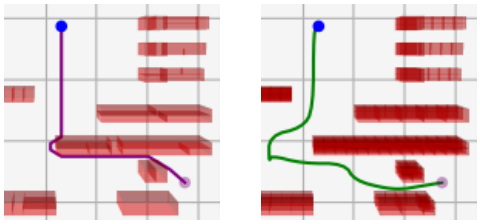
All of these tests were conducted on an Intel® Core™ i9-13900KF (single-threaded) with 32GB of RAM inside a Python 3.10 virtual environment. For validation purposes, we show results for our test scenario in Table I:

1) The first phase initializes the basic structure of the environment. Octree-based planning yields substantial gains in runtime and memory efficiency, at the cost of slight accuracy loss in NNS. However, this distance deviation is smaller than 50% of the voxel size, leaving the results accurate enough for most practical use cases. Notably, $F*$ exhibits a substantial performance boost when paired with octree representations. Although $A*$ also benefits from octree usage, the improvement is less pronounced compared to $F*$.
2) In phase two, with further map expansion, the octree continues to scale well, maintaining efficient pathfinding runtimes. Despite the growing graph size, the octree-based approach preserves its speed advantage due to its hierarchical representation. The performance gap between $A*$ and $F*$ remains consistent with observations from the first Phase.
3) In the final stage at phase three, the environment reaches its maximum planned complexity. At this point, the grid-map demonstrates better graph conversion performance due to minimal changes in the precomputed nodes. However, the octree continues to outperform in both pathfinding and NNS. As complexity increases, $F*$ loses its performance advantage, but the absolute difference compared to $A*$ remains within a few milliseconds.

Table I: Full Overview of our experimental test results

Category	Metric	Grid-Map			Octree		
		Phase 1	Phase 2	Phase 3	Phase 1	Phase 2	Phase 3
Map Generation	Runtime	76.7	0.08	0.003	49.4	0.01	0.09
Nearest-Neighbour	Runtime	4.86	6.18	6.18	0.17	0.19	0.19
	Accuracy [%]	100	100	100	90.5	84.1	84.1
Graph Conversion	Runtime	759	1916	8.29	153	125	101
	Node-Count	3730	6553	6248	715	951	1278
A* Pathfinding	Runtime	9.37	15.4	15.6	2.55	3.26	4.60
	Path Length	42.5	38.4	42.8	46.7	49.9	54.0
F* Pathfinding	Runtime	58.5	66.1	56.4	3.09	3.88	6.41
	Path Length	49.8	45.4	47.8	48.2	50.6	56.1

All runtimes in milliseconds, and path lengths in meters.



(a) Unsmoothed $A*$ in a Grid-Map (b) $F*$ in an Octree with moving average smoothing

Figure 5: Comparison of base $A*$ in a Grid-Map and smoothed $F*$ in an octree. Where $A*$ tightly hugs the wall to reduce path length, $F*$ keeps a safe distance, allowing for trajectory smoothing with minimal risk of collisions.

B. Learned Lessons

The hierarchical structure of octrees, combined with our pathfinding and Nearest-Neighbour Search (NNS) optimizations applied during map construction, shows to be highly effective. Based on our experimental results, we highlight the following key findings:

- Grid maps incur significantly higher conversion and update costs, correlating directly with node count. In contrast, octrees exhibit efficient scalability with increasing environmental complexity.
- Across all test scenarios and iterations, octree-based planning consistently achieves significantly lower NNS and pathfinding runtimes for both *A-Star (A*)* and *F-Star (F*)*, often by an order of magnitude.
- Reduced graph complexity allows $F*$ to approach $A*$ runtimes, making it well-suited for performance-sensitive tasks when combined with hierarchical representations.
- Trajectory smoothing on paths generated by $A*$ can result in collisions with obstacles, as the applied smoothing algorithms are not obstacle-aware, and $A*$ frequently selects paths that closely hug obstacles. In comparison, we observe that $F*$ generates highly suitable paths for our application, as visualized in Figure 5.

C. Simulation-Based Validation

To validate the practical feasibility of our planner, we also ran some simulated experiments, conducted in a previously unseen environment using ©Microsoft AirSim v1.8.1.

The simulation setup employed the default configuration parameters, with the addition of a multirotor UAV equipped with two LiDAR sensors, one providing 360-degree horizontal scanning and the other directed downward for floor mapping. We observed that variations in sensor field of view had only marginal effects on pathfinding, provided they were selected appropriately relative to the mapping resolution.

The drone successfully navigated the unknown environment while continuously building and updating the internal map representation in near real-time. During the mission, the planner recalculated the path multiple times in response to newly discovered obstacles in the environment. To execute the re-planning step, the drone briefly paused, ensuring that it never proceeded blindly into unmapped or unsafe areas.

VIII. Conclusion

This work explored real-time path planning and spatial environment representation for UAVs, specifically integrating an octree structure with partial updating capabilities, and a safety-optimized pathfinding algorithm.

Our conducted tests and simulation experiments demonstrate the significant performance gains achieved through octree-implementations in comparison to traditional voxel grid maps. Its ability to represent spatial relationships hierarchically allows for scalable resolution control, tuned to the capabilities of the onboard sensors.

In terms of planning algorithms, we compared our $F*$ solution to the commonly used $A*$ algorithm. $F*$ benefits greatly from the lesser complexity graphs generated by our octree map representation, resulting in performance close to that of $A*$, while providing preferable paths for our applications.

The use of a moving average for trajectory smoothing proved highly effective, allowing for momentum preservation without significantly deviating from the given path. While Bézier curves lack the local control necessary to guarantee that our safety requirements could still be met, Catmull-Rom splines provide a valid alternative, sacrificing some performance for improved smoothing capabilities.

References

[1] A. Ait Saadi, A. Soukane, Y. Meraihi, A. Benmessaoud Gabis, S. Mirjalili, and A. Ramdane-Cherif, "UAV Path Planning Using Optimization Approaches: A Survey," *Archives of Computational Methods in Engineering*, vol. 29, no. 6, pp. 4233–4284, 2022.

[2] M. Jones, S. Djahel, and K. Welsh, "Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey," vol. 55, no. 11, Feb. 2023.

[3] W. Liu, Z. Zheng, and K.-Y. Cai, "Bi-level programming based real-time path planning for unmanned aerial vehicles," *Knowledge-Based Systems*, vol. 44, pp. 34–47, 2013.

[4] F. Blöchliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps," *CoRR*, vol. abs/1709.05533, 2017.

[5] R. Schnabel and R. Klein, "Octree-based Point-Cloud Compression," in *Symposium on Point-Based Graphics*, M. Botsch, B. Chen, M. Pauly, and M. Zwicker, Eds., The Eurographics Association, 2006.

[6] S. Thrun and A. Bücken, "Integrating Grid-Based and Topological Maps for Mobile Robot Navigation," in *AAAI/IAAI, Vol. 2*, 1996.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] J. Yu, J. Hou, and G. Chen, "Improved Safety-First A-Star Algorithm for Autonomous Vehicles," in *2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2020, pp. 706–710.

[10] P. Luan and N. Truong Thinh, "C2 Piecewise Cubic Bezier Curve Based Smoothing Path for Mobile Robot," *International Journal of Mechanical Engineering and Robotics Research*, pp. 519–525, Jan. 2021.

[11] C. Yuksel, S. Schaefer, and J. Keyser, "Parameterization and Applications of Catmull-Rom Curves," *Computer Aided Design*, vol. 43, no. 7, pp. 747–755, 2011.

[12] R. Hyndman, "Moving Averages," in Springer-Verlag, Jan. 2010, pp. 866–869.

[13] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-D path planning in a dynamic environment using an octree and an artificial potential field," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 2, 1995, 474–481 vol.2.