2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Obstacle-Aware Topological Planning over Polyhedral Representation for Quadrotors

Junjie Gao, Fenghua He, Wei Zhang, and Yu Yao

***Abstract*— In this paper, we propose a novel mapping-planning framework for autonomous quadrotor navigation. First, a polyhedron-based mapping algorithm is presented to fully exploit the information of the onboard sensor data. Polyhedra are generated to approximate the segmented clusters of occupied voxels. Then, customized data structures are designed to extract information for motion planning in real time. With complete knowledge of the shape, position, and number of the observed obstacles, we can conveniently generate smooth trajectories with sufficient obstacle clearance along the most desired direction. Before searching for the initial path, a local topological graph is constructed to keep the path expanding in the most favorable topology class. The following path search is segmented based on the graph vertices, which allows fast convergence. The refined trajectory is obtained after smoothing, and large deviations are penalized in the formulated optimization problem to preserve the original clearance. Finally, we analyze and validate the proposed framework through extensive simulations and real-world quadrotor flights.**

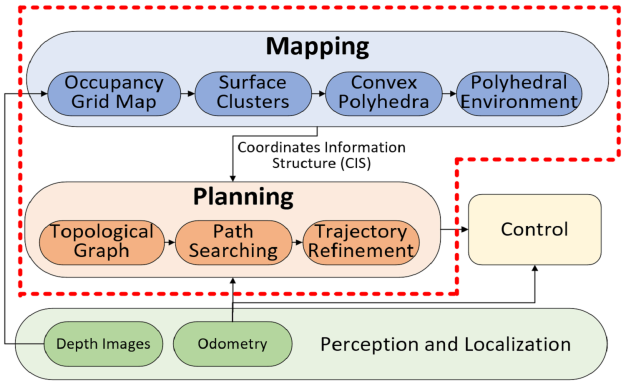


Fig. 1. Overview of the proposed mapping-planning framework.

I. INTRODUCTION

Robust and efficient motion planning algorithms are crucial for quadrotors to achieve autonomous flight in complex environments. As the bridge between the perception and planning modules, environment representation has a huge impact on the quality of the generated trajectories. Various algorithms have been proposed to build maps for navigation, and each corresponds to different planning methods.

As the most commonly used type, volumetric maps discretize the space into voxels of equal size. Such maps can distinguish safe regions from obstacles by storing the accumulative occupancy probabilities. This environment representation is adopted by multiple search-based and sampling-based planning methods like [1]-[3]. However, the computational overhead increases exponentially for more precise descriptions. Also, the occupancy status alone is insufficient to generate high-quality trajectories. Maps like Truncated Signed Distance Fields (TSDFs) and Euclidean Signed Distance Fields (ESDFs) provide extra distance information of each voxel, and are widely used for gradient-based trajectory optimizations [4]-[6]. Although [7] and [8] enable online incremental construction of ESDF maps, updating and maintaining the distance fields still require intensive computation. Besides, the distance gradient can be misleading and cause local minima [9]. Recently, some studies have focused on the problem of constructing three-dimensional topological graphs for navigation [10]-[12]. Such sparse graphs are resistant to sensor noises, and the topology information they provide can greatly accelerate the path search. In addition, the planner can avoid local minima by selecting from different homotopy classes. However, they all require offline calculation due to the complex construction process.

In this paper, we propose a novel mapping-planning framework (shown in Fig. 1) to navigate the online quadrotor flight. For the mapping module, the environment information is fully extracted and utilized by generating polyhedra to represent observed obstacles. Such an environment is directly derived from the occupancy grid map, therefore the construction is computationally inexpensive. Since the geometry information of the obstacle is completely contained in the polyhedra, various information can be provided for motion planning. For the planning module, we first efficiently build a local topological graph covering the potential searching area. The graph is then utilized to guide the segmented motion primitives-based path search. Finally, a polynomial-based optimization method is adopted to obtain safe and smooth trajectories. The contributions of this paper are summarized as follows.

- A novel mapping algorithm, which constructs a polyhedral environment online and provides comprehensive obstacle information to facilitate motion planning.
- A topological planner, which adopts a segmented searching strategy for acceleration and efficiently generates trajectories with sufficient obstacle clearance.
- Extensive simulations and real-world experiments are conducted to validate the proposed mapping-planning framework.

The authors are with the School of Astronautics, Harbin Institute of Technology, 92 West Dazhi Street Harbin, China. gaojunjie@stu.hit.edu.cn

II. RELATED WORK

A. Environment Representation

The mapping approaches can be roughly classified into two categories: cell decomposition and roadmap. For three-dimensional environments, regular cells are discrete, non-overlapping cubes. For example, octomap [13] uses a hierarchical structure to store the occupancy probability and is resistant to sensor noises. But it fails to provide more information other than the occupancy status. Geometric shapes are also adopted for decomposition. Wahrmann et al. defined and built swept-sphere-volumes (SSVs) to approximate obstacles [14]. However, fixed shapes are inaccurate for planning in complex environments. In [15], Majeed et al. modeled the obstacles as convex polyhedrons with six faces, and the search space was circumscribed in the form of a half cylinder. A roadmap is a union of curves such that all start and goal points in the free space can be connected by a path. There are various types of roadmaps, such as the visibility graph [16], the Voronoi graph [17], and the topological graph [10]. A recent work proposed by Fan et al. developed a dynamic updating method to generate visibility graphs [18]. Polygons were generated to approximate obstacles, and the three-dimensional cases were implemented by dividing the space into two-dimensional layers.

B. Kinodynamic Motion Planning

The existing methods can be divided into search-based and sampling-based. For search-based methods, a lattice of states repeated at regular intervals are constructed and applied for searching. Liu et al. generated motion primitives based on the differential flatness of quadrotor systems and achieved online planning [19]. Zhou et al. designed a novel heuristic function for motion primitives by formulating an optimal control problem [20]. In [21], motion primitives without safe ends were eliminated and a gain function was designed for quadrotor exploration. However, the above methods are bothered by the high computational complexity when planning in an obstacle-dense environment. Building a sparse topological graph can greatly accelerate the graph search, but it is performed offline due to the high calculation cost [10]. Similar to lattice-based approaches, sampling-based planning methods connect discrete state samples using collections of feasible motions. Proposed in [22], Kinodynamic RRT* algorithm computed state transitions with a time and energy cost function. To decrease the time cost of the random sampling, Ye et al. presented a topology-guided kinodynamic planner. They employed a biased sampling strategy based on the built topology graph [23]. However, only a few homotopy classes were extracted for sampling.

III. POLYHEDRAL ENVIRONMENT CONSTRUCTION

In this section, we present the proposed polyhedral environment construction algorithm. Our method starts with an occupancy grid map built from the onboard sensor data. The original voxels are abandoned afterward to reduce the storage requirements. The detailed procedures are described as follows:

A. Obstacle Surface Clusters Extraction

We define the outermost layer of voxels of an occupied voxel cluster as the obstacle surface. We first extract the surface voxels since they can preserve the geometry information of obstacles, whilst the voxel number is greatly reduced compared to the original cluster. This is efficiently implemented based on the following property of the grid map: an occupied voxel does not belong to the surface if and only if it has six direct neighbors. Then the obtained surface is segmented into different clusters based on the Euclidean distance [25] and each surface cluster is corresponding to an observed obstacle.

B. Convex Polyhedron Generation

In this step, we construct a convex polyhedron for each surface cluster to extract the preliminary geometry information. We solve the problem by constructing a three-dimensional convex hull that completely surrounds the cluster. The Quickhull algorithm [24] is adopted due to its high efficiency. The generated polyhedron is then stored as triangle meshes using the face-list data structure. Note that the merged surfaces and edges are excluded. The volume of each polyhedron is also calculated for future processing.

C. Concave Polyhedron Approximation

In this step, we further fit the obstacles with concave polyhedra if the previous approximation is overly conservative and compresses the solution space. A convex polyhedron will shrink if it satisfies:

$$V_{conv}/V_s < V_{thresh} \tag{1}$$

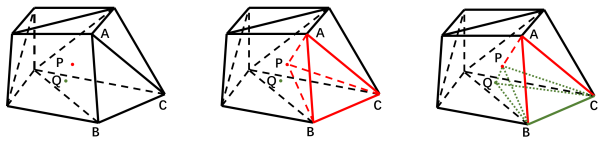
where V_{conv} and V_s represent the volume of the convex polyhedron and the surface cluster, respectively. V_{thresh} determines the accuracy of the polyhedral environment.

The concave polyhedron is generated based on the original convex one as shown in Alg. 1. Note that Φ_t is represented by Φ_v and Φ_i, and is written as a separate variable just for the convenience of description. The process can be divided into curvature calculation and polyhedron update. Surface normals are calculated using PCL's *NormalEstimation* [25] (line 2). To orient all normal vectors $\overrightarrow{n_i}$ consistently away from the polyhedron surface, the viewpoint $\boldsymbol{v}_p$ is set to be the cluster centroid. In this way, the following relationship holds:

$$\overrightarrow{n_i} \cdot (\boldsymbol{v}_p - \boldsymbol{p}_i) < 0 \tag{2}$$

where p_i is the voxel in the surface cluster. The two principal curvatures κ_1 and κ_2 can be applied to characterize the local shape of the surface. Consequently, we use κ_1 and κ_2 to locate the dents on the obstacle surface. Following the CAN method [26], we adopt the least square fitting algorithm to calculate curvatures (line 4). A group of normal curvatures κ is first calculated using the normal vector of each surface voxel and its neighbors. Then the principal curvatures are obtained by formulating an optimization problem based on the Euler formula

$$\kappa = \kappa_1 \, cos^2\theta + \kappa_2 \, sin^2\theta \tag{3}$$



(a) (b) (c)

Fig. 2. Illustration of the polyhedron update. (a) The interior vertices P and Q are obtained by computing the principal curvatures. (b) (c) Collision checks are conducted to find planes that ensure safety while shrinking: triangle ABC and PBC. The polyhedron is updated by eliminating the largest shrinking volume: tetrahedron $APBC$ and $PQBC$.

where κ is the normal curvature in the direction making an angle θ with the first principal direction. As presented in table I, κ_{th1}, κ_{th2}, κ_{th3} should satisfy

$$\kappa_{th1} > \kappa_{th2} > 0 > \kappa_{th3} \tag{4}$$

In this way, interior voxels are stored in Φ_{conv} and Φ_{conc}, respectively (lines 5-8).

Algorithm 1: Concave Polyhedron Approximation

1 **Notion:** Vertex Set Φ_v, Index Set Φ_i, Triangle Mesh Set Φ_t, Surface Cluster Set Φ_s, Potential Triangle Set Φ_p, Convex/Concave Vertex Set Φ_{conv}/Φ_{conc};
2 $\Phi_n \leftarrow$ **CalculateNormal**(Φ_s);
3 **for** p_i *in* Φ_s **do**
4 $\kappa_1, \kappa_2 \leftarrow$ **CalculatePrincipalCurvature**(p_i);
5 **if** $\kappa_1 > \kappa_{th1}$ *and* $\kappa_2 > \kappa_{th2}$ **then**
6 Φ_{conv}.**insert**(p_i);
7 **else if** $\kappa_2 < \kappa_{th3}$ **then**
8 Φ_{conc}.**insert**(p_i);
9 **while** ¬ **FitTight**(Φ_v,Φ_i,V_{thresh}) **do**
10 **for** p_{conv} *in* Φ_{conv} **do**
11 **if** ¬ **NearExistingVertices**(p_{conv},Φ_v) **then**
12 **for** τ_i *in* Φ_t **do**
13 **if SafeTetrahedron**(τ_i,p_{conv}) **then**
14 Φ_p.**insert**(τ_i);
15 $\tau_n \leftarrow$ **GetLargestVolume**(Φ_p,p_{conv});
16 **ShrinkPolyhedron**(τ_n,Φ_v,Φ_i,Φ_n);
17 **UpdateWithConcavePoints**(Φ_{conc},Φ_v,Φ_i) ;

The second step is to shrink the polyhedron until its volume no longer satisfies the condition of (1) (line 9). Interior vertices with larger κ_1 are selected first (line 10). We iterate through Φ_t and add the triangle τ_i to Φ_{mesh} if the intersection between t_i and the current surface cluster is empty (lines 13-14). Here t_i denotes the tetrahedron formed by p_{conv} and three vertices of τ_i. To reduce the time cost, we use KD-tree to narrow down the checking range to voxels inside C_i, where C_i is the circumsphere of t_i. Then, we select τ_n that allows the largest shrinking volume from Φ_p (line 15). Finally, the polyhedron contracts around p_{conv}. Φ_v and Φ_i are updated by adding one more vertex p_{conv} and replacing τ_n with three new triangles. Normal vectors of the newly added planes are also calculated and inserted into Φ_n (line 16). The shrinking process for Φ_{conv} and Φ_{conc} are the same and the latter is omitted for brevity (line 17). Note that voxels in Φ_{conc} are not used until Φ_{conv} is empty to ensure that Φ_p is completely enclosed by the new polyhedron. The update process of a convex polyhedron is shown in Fig. 2.

TABLE I
CORRESPONDENCE BETWEEN SURFACE SHAPE AND PRINCIPAL CURVATURES

	$\kappa_1 < 0$	$\kappa_1 > 0$
$\kappa_2 < 0$	Concave Ellipsoid	Hyperboloid Surface
$\kappa_2 > 0$	Hyperboloid Surface	Convex Ellipsoid

D. Data Structures Design

Now that polyhedra have replaced occupied voxels to represent the observed obstacles, we can obtain diverse environment information by dealing with three-dimensional geometries. We present our Sphere Bounding Volume Hierarchy (SBVH) for real-time information queries and Coordinates Information Structure (CIS) for topology and obstacle distance extraction. Note that data structures can be customized to accommodate different planning methods.

The CIS (described in Table II) of the queried coordinates $\boldsymbol{p}_i$ is obtained by the following process. By searching the SBVH, we first insert the polyhedron into the nearest polyhedron set Φ_{poly} if it satisfies

$$\|\boldsymbol{p}_i - \boldsymbol{o}_j\| - r_j < d_{thresh} \tag{5}$$

where $\boldsymbol{o}_j$ refers to the centroid coordinates of the polyhedron, and r_j is the radius of its SBVH root node. The indices of the polyhedra in Φ_{poly} are stored in Φ_{idx}. The maximum horizontal distance between two vertices of each polyhedron in Φ_{poly} is stored in Φ_r to approximate the circumradius.

Then, we find the nearest surface position to p_i of each polyhedron in Φ_{poly}. With the nearest triangle mesh provided by SBVH, The desired position p_e can be obtained by projecting p_i onto the plane containing the triangle. As shown in Fig. 3, the projection $p_i^{'}$ lies in one of the seven regions ($R1 \sim R7$) and each case corresponds to a different p_e. To distinguish the region to which $p_i^{'}$ belongs, we compute its position in the barycentric coordinate system. The problem is solved if $p_i^{'}$ lies in R_a, R_e, or R_g. Otherwise (as the case in Fig. 3), we further calculate the angle between vectors $\boldsymbol{p}_a\boldsymbol{p}_c$ and $\boldsymbol{p}_a\boldsymbol{p}_i^{'}$ to locate p_e. In this way, the unsigned nearest distance is calculated by

$$d_i = \sqrt{\|\boldsymbol{p}_i\boldsymbol{p}_i^{'}\|^2 + \|\boldsymbol{p}_e\boldsymbol{p}_i^{'}\|^2} \tag{6}$$

d_i is set to negative if

$$\boldsymbol{e}_i \cdot (\boldsymbol{p}_i - \boldsymbol{p}_e) < 0 \tag{7}$$

where $\boldsymbol{e}_i$ is the surface normal at $\boldsymbol{p}_e$. If $p_i^{'}$ lies in $R1$, e_i is given by Section III. Otherwise, the pseudo normals [27] are used instead. Finally, the nearest coordinates and the

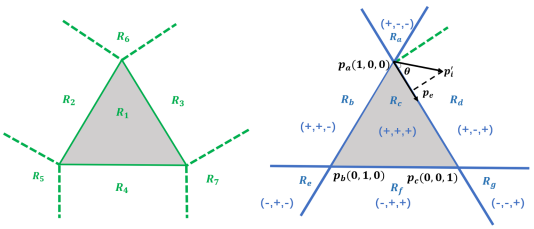


Fig. 3. The plane is divided in two ways to search for p_e. Left: the plane is divided based on the position of $p_i^{'}$. Right: position of $p_i^{'}$ in the barycentric coordinate system is mapped to $R_1 \sim R_7$. In this case, $\theta < 90°$ and $p_i^{'}$ is mapped to $R3$. Therefore, p_e lies on the edge $p_a p_c$

corresponding distance are placed in the CIS, which is then utilized by the planning module.

TABLE II
COORDINATES INFORMATION STRUCTURE

Name	Meaning
the nearest position set Φ_{coord}	coordinates of the nearest voxel of each polyhedron in Φ_{poly}
the nearest distance set Φ_d	distance to the coordinates in Φ_{coord}
polyhedron index set Φ_{idx}	indices of Φ_{poly}
polyhedron circumradius set Φ_r	circumradius of the horizontal cross section of each polyhedron Φ_{poly}

IV. MOTION PRIMITIVES-BASED TOPOLOGICAL PLANNING

In this section, we present the proposed topological planner to obtain safe and dynamically feasible trajectories. The obstacle clearance is also maximized based on the extracted environment information. The detailed algorithms are described as follows.

A. Local Topological Graph Generation

As shown in Alg. 2, the breadth-first search (BFS) framework is adopted for local topological graph construction. For each iteration, we first try to build an edge to connect n_e and n_i. (line 5). For cases where a collision exists, an expansion starts with finding the collision position and calculating its CIS. Q_n is obtained by extending the polyhedron centroid outward along the direction perpendicular to $n_i n_e$ by a distance, and the distance is obtained from the Φ_r of a CIS (line 6). New edges are then built between n_i and each node in Q_n. If the collision still occurs, such a neighbor will be updated using the CIS of the new collision position (lines 7-8). The new neighbor is obtained by expanding the nearest position in Φ_{coord} outward until it lies out of the polyhedron. Safe edges are then generated and added to the graph (line 9). For another case, the edge directly connecting n_i and n_e is added to the graph (line 12). The final graph consists of topologically distinct paths from n_s to n_e and is recorded using the coordinates of its vertices. Fig. 4 shows the construction process of the local topological graph from a two-dimensional perspective.

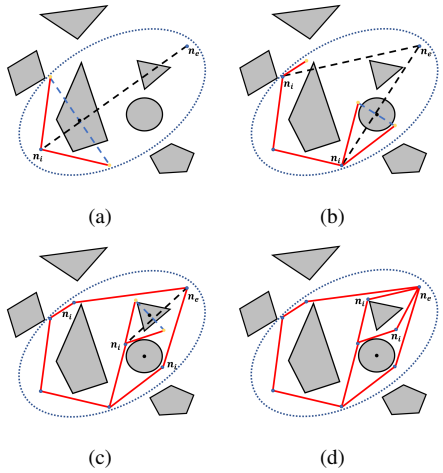


Fig. 4. Illustration of constructing a local topological graph. (a) (c) Expanding safe neighbors (yellow dots) based on Φ_r if edge $n_i n_e$ is not safe. (b) Expanding one neighbor based on Φ_{coord} if edge $n_i n_e$ collides with the same obstacle again (for n_i on the upper side). (d) No expansion if edge $n_i n_e$ does not intersect with any polygons. The generated graph consists of edges (red line segments) and vertices (blue dots).

Algorithm 2: Local Topological Graph Generation

1 **Notion:** Topological graph G, Node queue Q_n, Start node n_s, End node n_e;
2 Q_n.**Enqueue**(n_s);
3 **while** $\neg Q_n$.**empty() do**
4 n_i = Q_n.**Dequeue**();
5 **if** $\neg$ **CheckEdgeSafety**(n_i,n_e) **then**
6 $\Phi_n \leftarrow$ **QueryandExpand**(n_i,n_e);
7 **if** $\neg$ **CheckEdgeSafety**(n_i,Φ_n) **then**
8 **QueryandUpdate**(n_i,Φ_n);
9 $G \leftarrow$ **UpdateGraph**(n_i,Φ_n);
10 Q_n.**Enqueue**(Φ_n);
11 **else**
12 $G \leftarrow$ **UpdateGraph**(n_i,n_e);

B. Motion Primitives-based Path Search

The concept of "segment goal" is proposed in [28], which generates virtual intermediate goals to accelerate planning algorithms. In our case, the vertices obtained from the topological graph naturally serve as intermediate goals to guide the path search. If x and y coordinates of the current state are close enough, the current goal will be switched to the next vertex.

Following [19], we approximate the quadrotor system as a second-order integrator. Thus the system state vector is $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{z}]^T$ and the control input vector is $\boldsymbol{u} = [\boldsymbol{a_x}, \boldsymbol{a_y}, \boldsymbol{a_z}]^T$. The motion primitives are generated by uniformly discretizing the control space. Aimed to find a path with low control cost, low time cost, and high obstacle clearance, the cost of a motion primitive with an interval τ is defined as:

$$f = \lambda_c \int_0^\tau (\|\boldsymbol{u}_i\|^2 + w) d\tau + \lambda_o \sum_i \| \sum_j \boldsymbol{p}_j \boldsymbol{p}_i \|^2 \qquad (8)$$

where w is the time weight to minimize the path duration. $\boldsymbol{p}_i$ are the intermediate states of the primitive, and their coordinates are queried to obtain CISs. The sum of the vectors $\boldsymbol{p}_j\boldsymbol{p}_i$ refers to the repulsive force from the nearby obstacles, which requires primitives to be close to the medial axis between obstacles. The second term is set to infinity if the nearest distance set of a CIS contains negative values.

C. Polynomial-based Trajectory Refinement

We adopt a three-dimensional polynomial $p(t)$ with m segments, and each segment $p_i(t)$ of order n to represent the trajectory. Considering that the obtained path is only segment-wise C^2 continuous, we formulate an optimization problem to improve its smoothness. However, extensive deviation from the initial path is not desired during the optimization to maintain the obstacle clearance. Therefore, the cost function is designed as:

$$\mathrm{J} = \lambda_\mathrm{s}\mathrm{J}_\mathrm{s} + \lambda_\mathrm{b}\mathrm{J}_\mathrm{b} \tag{9}$$

where J_s is the smoothness cost as the k^{th}-order squared derivative integral of the trajectory. J_b is the bounding cost to punish the deviation. The optimization problem can be described as:

$$\begin{aligned} \min \mathrm{J} = &\sum_{i=0}^{m-1} \int_0^{\tau_i} \|p_i^{(k)}(t)\|^2 dt + \int_0^T \|p(t) - p_o(t)\|^2 dt \\ s.t.\ &x(0) = x_0, \quad x(T) = x_T \\ &p_i^{(j)}(\tau_i) = p_{i+1}^{(j)}(0), i \in \{0, 1 \dots m-2\}, j \in \{0, 1, 2\} \\ &p_i(\tau_i) = p_o(\tau_i), \qquad i \in \{0, 1 \dots m-2\} \end{aligned} \tag{10}$$

where $p_o(t)$ is the original path. τ_i and T are the duration of the i^{th} segment and the whole trajectory, respectively. The equality constraints are formulated as above to restrict the start and end states, the continuity between two consecutive segments, and the intermediate positions. Since the costs are independent, J is obtained by accumulating over axis x, y, z. The above optimization is solved by reformulating an unconstrained quadratic programming problem as described in [29].

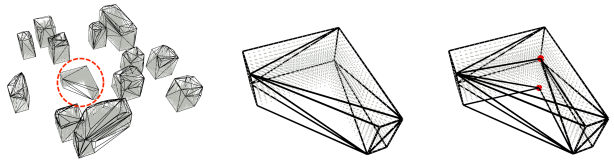
V. Experimental Results and Analysis

A. Simulation Experiments

The following simulation experiments are implemented in ROS with C++ running on a 2.7 GHz Intel i5-11400H processor. The resolution of the occupancy grid map is set to $0.1m$.

1) *Polyhedral Generation Module:* We first validate the mapping algorithm in a 15 × 15 × 5m environment, which is larger than the actual sensing range. Rectangular obstacles with heights of 1.0 to 1.5m are randomly generated in the environment. As shown in Fig. 5(a), convex polyhedra are built for each obstacle cluster. The timing information of each step is presented in Table III. We run 20 tests for each group with different obstacle numbers and calculate the average time cost, respectively. Since convex polyhedra are sufficient to approximate most clusters, the cost of the shrinking process is not included here. The result shows that although the cost increases dramatically when more voxels require to be clustered, the overall computational time is always below 100ms even when the obstacles are extremely dense. As a result, the update frequency can be maintained at over 10Hz.

With V_{thresh} set to 0.7 in (1), a few convex polyhedra are overly conservative and need a tighter approximation. As shown in Fig. 5(a), the underfitting cluster is circled in red and zoomed in for illustration. For the cluster in Fig. 5(b), V_{conv}/V_s is well below 0.7 due to the concave obstacle shape. By setting κ_{th3} to -0.2 in (4), p_{conc} are found and visualized by red dots in Fig. 5(c). Then a concave polyhedron is generated by replacing two original triangles with six new triangles closer to the centroid. V_{conv}/V_s grows from 0.58 to 0.88 after shrinking. The run time is 17.68ms for 2652 grids in this case, which is sufficient for online updates.



(a) (b) (c)

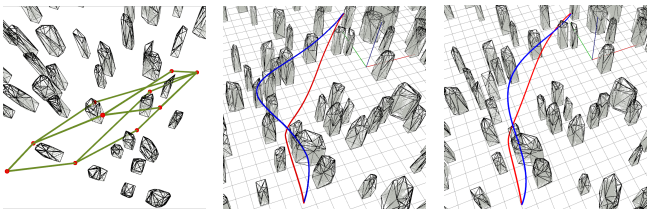
Fig. 5. Visualization of the polyhedral environment in the simulation

TABLE III
POLYHEDRAL ENVIRONMENT CONSTRUCTION RUN TIME

Obstacle Number /Voxel Number	Total Time (ms)	Surface Extraction (ms)	Euclidean Clustering (ms)	Quick Hull (ms)
15/16085	22.53	16.63	5.44	0.46
20/22019	28.82	21.36	6.89	0.57
30/33536	44.20	32.96	10.41	0.83
50/42572	62.24	45.23	15.98	1.03

2) *Topological Graph-guided Path Searching Module:* As shown in Fig. 6(a), a local topological graph is created between the start and the target position, containing paths (green line segments) from different homotopy classes. The graph is recorded by its vertices (red dots) and is utilized to guide the path search. The building time is always below 5ms, which is acceptable for rapid online planning. The shortest path in the graph is selected and the corresponding vertices are utilized as the "segment goals". We compare our method against a state-of-the-art motion primitives-based path searching algorithm proposed in [19]. Both taking the quadrotor acceleration as the system input, Fig. 6(b) shows that our method is capable of generating a smoother initial path (red curve) with more obstacle clearance. For a fair comparison, we apply the same optimization algorithm for two initial paths. As shown in Fig. 6(c), our trajectory (red curve) is much smoother and shorter, meanwhile maintaining sufficient clearance when traversing between obstacles.

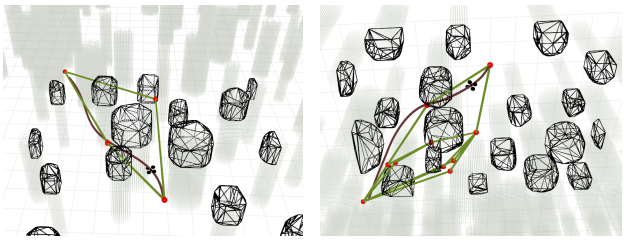
2) *Benchmark Comparisons:* We compare our path-searching and trajectory refinement methods with algorithms



(a) (b) (c)

Fig. 6. (a) The constructed local topological graph. (b) The comparison of the front-end trajectories in the simulation. (c) The comparison of the back-end trajectories in the simulation.

proposed by Zhou et al. [20] and Liu et al. [19] in a 40 $\times$ 40 $\times$ 5m map (shown in Fig. 7). The maximum velocity and acceleration are set as $3m/s$ and $2m/s^2$, respectively. The simulations are conducted with 150 and 300 random obstacles, and Table IV demonstrates the results. For path search, three methods all generate motion primitives for kinodynamic planning. However, with the knowledge of environment topology, the proposed method tends to expand neighbors in the most desired homotopy class and thus leads to trajectories with the shortest length. Moreover, our method generates initial paths with sufficient obstacle clearance, which results in the highest success rate in obstacle-dense environments. For trajectory refinement, although we formulate a quadratic programming problem like [19] as well, the optimized trajectories still enjoy enough clearance due to the designed bounding cost. In addition, our method achieves considerable efficiency improvement for two reasons: Topological vertices guide the segmented graph search into safe areas, thus saving the cost of aimless expansions. Besides, an analytical solution exists for the formulated optimization problem, and its computational effort is negligible.



(a) (b)

Fig. 7. (a) Flight simulation in the environment with 150 obstacles. (b) Flight simulation in the environment with 300 obstacles. Brown curves are the executed trajectories.

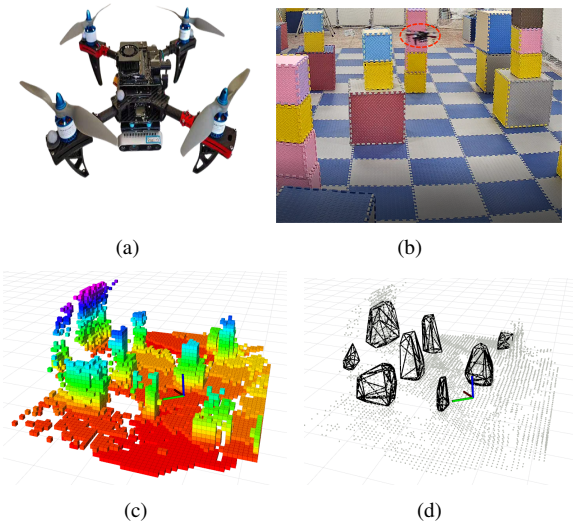
TABLE IV

BENCHMARK COMPARISON RESULTS

Method	150 Obstacles			300 Obstacles		
	Comp. Time (ms)	Traj. Len. (m)	Succ. Rate (%)	Comp. Time (ms)	Traj. Len. (m)	Succ. Rate (%)
Proposed	**20.67**	**29.97**	**100.00**	**29.60**	**32.40**	**89.17**
Zhou. [20]	26.58	31.03	**100.00**	41.14	34.32	87.93
Liu. [19]	22.16	31.98	86.88	35.28	35.94	60.10

B. Real-world Experiments

The custom-built quadrotor platform (shown in Fig. 8(a)) is equipped with a JETSON Xavier NX for onboard computing, a Pixhawk flight controller for stabilization, and a forward-facing RealSense D435i to provide depth images. The localization information is obtained from the NOKOV motion capture system. Real-world flight experiments are conducted in an unknown indoor environment as shown in Fig. 8(b). The original occupancy grid map and the corresponding polyhedron map are shown in Fig. 8(c) and Fig. 8(d), respectively. We abandon the occupied voxels below 0.15m and above 2.3m to segment the floor and the ceiling. The height constraint is posed during the path search. The collision check is implemented by regarding the quadrotor as a sphere with a radius of 0.2m.



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

Fig. 8. (a) Hardware setting of our custom-built quadrotor. (b) A snapshot of the quadrotor crossing the unknown environment, and the average flight speed is $1.5m/s$ (c) The original occupancy grid map. (d) The polyhedral environment within the sensing range. The surface noise of the original map is covered by the generated planes, resulting in smoother trajectories.

VI. CONCLUSIONS

In this paper, we propose a novel mapping-planning framework to generate high-quality trajectories online. Based on a pre-built occupancy grid map, convex and concave polyhedra are generated to extract environment information. Served as the bridge between the two modules, data structures are designed to obtain the nearby obstacle information of the queried position. To acquire better initial paths, we construct a local topological graph to guide the motion primitives-based graph search. With the path sticking to the obstacle medial axis, we formulate a quadratic programming problem to obtain smoother trajectories with adequate obstacle clearance. The experimental results demonstrate that our method outperforms the selected benchmarks in terms of computational efficiency and trajectory quality. Real-world autonomous flights are performed to verify the robustness and efficiency of the proposed framework.

REFERENCES

[1] G H. Qin, Z. Meng, W. Meng, X. Chen, H. Sun, F. Lin, and M. H. Ang, "Autonomous exploration and mapping system using heterogeneous UAVs and UGVs in GPS-denied environments," IEEE Trans. Veh. Technol., vol. 68, no. 2, pp. 1339–1350, Feb. 2019.

[2] L. Yao and Q. Jia, "The Sampling Task Planning of On-board Manipulator Based on Planning Knowledge Base," 2019 IEEE 2nd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), 2019, pp. 366-369, doi: 10.1109/AUTEEE48671.2019.9033237.

[3] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Oct. 2016, pp. 5332–5339.

[4] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS), Daejeon, Korea, Oct. 2016, pp. 5332–5339.

[5] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS), Sept 2017, pp. 3681–3688.

[6] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer," in Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS). IEEE, 2017, pp. 215–222.

[7] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1366-1373, doi: 10.1109/IROS.2017.8202315.

[8] L. Han, F. Gao, B. Zhou and S. Shen, "FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 4423-4430, doi: 10.1109/IROS40897.2019.8968199.

[9] B. Zhou, F. Gao, J. Pan and S. Shen, "Robust Real-time UAV Replanning Using Guided Gradient-based Optimization and Topological Paths," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1208-1214, doi: 10.1109/ICRA40945.2020.9196996.

[10] H. Oleynikova, Z. Taylor, R. Siegwart and J. Nieto, "Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1-9, doi: 10.1109/IROS.2018.8594152.

[11] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider and R. Siegwart, "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps," 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 3818-3825, doi: 10.1109/ICRA.2018.8460641.

[12] M. Collins and N. Michael, "Efficient Planning for High-Speed MAV Flight in Unknown Environments Using Online Sparse Topological Graphs," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 11450-11456, doi: 10.1109/ICRA40945.2020.9197167.

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," Auto. Robots, vol. 34, no. 3, pp. 189–206, Apr. 2013.

[14] D. Wahrmann, A. -C. Hildebrandt, R. Wittmann, F. Sygulla, D. Rixen and T. Buschmann, "Fast object approximation for real-time 3D obstacle avoidance with biped robots," 2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2016, pp. 38-45, doi: 10.1109/AIM.2016.7576740.

[15] Majeed, A.; Lee, S. A Fast Global Flight Path Planning Algorithm Based on Space Circumscription and Sparse Visibility Graph for Unmanned Aerial Vehicle. Electronics 2018, 7, 375. https://doi.org/10.3390/electronics7120375

[16] B. Oommen, S. Iyengar, N. Rao and R. Kashyap, "Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case," in IEEE Journal on Robotics and Automation, vol. 3, no. 6, pp. 672-681, December 1987, doi: 10.1109/JRA.1987.1087133.

[17] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," Artificial Intelligence, vol. 99, no. 1, pp. 21–71, 1998.

[18] Yang, Fan, et al. "FAR planner: Fast, attemptable route planner using dynamic visibility update." arXiv preprint arXiv:2110.09460 (2021).

[19] S. Liu, N. Atanasov, K. Mohta and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 2872-2879, doi: 10.1109/IROS.2017.8206119.

[20] B. Zhou, F. Gao, L. Wang, C. Liu and S. Shen, "Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight," in IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3529-3536, Oct. 2019, doi: 10.1109/LRA.2019.2927938.

[21] M. Dharmadhikari et al., "Motion Primitives-based Path Planning for Fast and Agile Exploration using Aerial Robots," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 179-185, doi: 10.1109/ICRA40945.2020.9196964.

[22] D. J. Webb and J. van denBerg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in Proc. IEEE Intl. Conf. Robot. Autom., May 2013, pp. 5054–5061.

[23] H. Ye, X. Zhou, Z. Wang, C. Xu, J. Chu and F. Gao, "TGK-Planner: An Efficient Topology Guided Kinodynamic Planner for Autonomous Quadrotors," in IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 494-501, April 2021, doi: 10.1109/LRA.2020.3047798.

[24] Barber, C. Bradford, David P. Dobkin, and Hannu Huhdanpaa. "The quickhull algorithm for convex hulls." ACM Transactions on Mathematical Software (TOMS) 22.4 (1996): 469-483.

[25] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in IEEE International Conference on Robotics and Automation, 2011, pp. 1–4.

[26] Zhang, Xiaopeng, et al. "Robust curvature estimation and geometry analysis of 3d point cloud surfaces." J. Inf. Comput. Sci 6.5 (2009): 1983-1990.

[27] Bærentzen, J. A., and H. Aanęs. "Generating Signed Distance Fields From Triangle Meshes. 2002." Informatics and Mathematical Modeling, Technical University of Denmark, DTU 20.

[28] Vaish, Shikhar, and Sunita Singhal. "Segmented Approach to Path Planning." 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC). IEEE, 2020.

[29] Richter, Charles, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments." Robotics research. Springer, Cham, 2016. 649-666.