

Dynamic Free-Space Roadmap for Safe Quadrotor Motion Planning

Junlong Guo, Zhiren Xun, Shuang Geng, Yi Lin¹, Chao Xu, and Fei Gao

Abstract—Free-space-oriented roadmaps typically generate a series of convex geometric primitives, which constitute the safe region for motion planning. However, a static environment is assumed for this kind of roadmap. This assumption makes it unable to deal with dynamic obstacles and limits its applications. In this paper, we present a dynamic free-space roadmap, which provides feasible spaces and a navigation graph for safe quadrotor motion planning. Our roadmap is constructed by continuously seeding and extracting free regions in the environment. In order to adapt our map to environments with dynamic obstacles, we incrementally decompose the polyhedra intersecting with obstacles into obstacle-free regions, while the graph is also updated by our well-designed mechanism. Extensive simulations and real-world experiments demonstrate that our method is practically applicable and efficient.

I. INTRODUCTION

Mapping modules are of vital importance for safe motion planning. They often take raw sensor data as input then output abstract environment representation. However, voxel-based maps need to store cumbersome environment data, which is both memory and time-consuming. The resolution also limits it to represent the environment in a finer resolution [1, 2]. Point cloud maps do not directly support dynamic environments or efficient queries for motion planning. Therefore, it is necessary for a map representation to be able to consume less memory, handle dynamic obstacles, and be convenient for motion planning.

Free spaces, instead of obstacles, is useful information for robot navigation [3]. Polyhedron-shaped free-space roadmap is a natural idea that uses a set of polyhedra to represent the union of free space of the environment. It has the following advantages: **1)** Polyhedron could tightly approximate free configurations and naturally suits for non-convex and unstructured environments. **2)** Connectivity between each polyhedron represents topological structures of environments that have significant importance in motion planning. **3)** Storage requirement is significantly lower because polyhedrons can represent vast space by few parameters. **4)** Analytical expression of the polyhedron is convenient to encode free space information for trajectory optimization and can easily add safety requirements.

Although polyhedron-shaped free-space roadmap has the above advantages, existing methods cannot deal with dynamic environments, which are common scenes for aerial

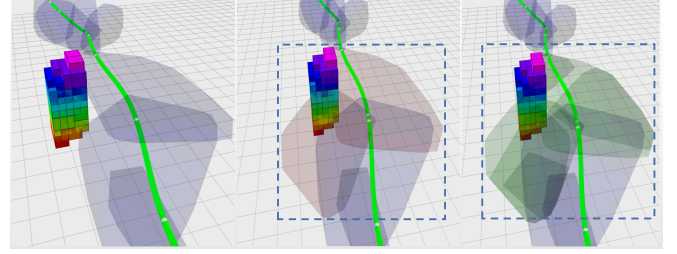


Fig. 1: Illustration of dynamic free-space roadmap. The polyhedron colored with red overlap dynamic obstacles. Green polyhedra are obtained by decomposing the red one.

robots. To our best knowledge, existing polyhedron-shaped free-space roadmaps have no efficient update strategy to deal with topological changes due to moving obstacles. Related works [4]–[8] all assume that the environment is static. To bridge this gap, this work introduces an efficient method that incrementally updates the free-space roadmap when there are moving obstacles on the map. Firstly, we use Iterative Regional Inflation by Semidefinite (IRIS) [9] to extract all free spaces in the map to generate polyhedra. Then a navigation graph will be built by checking connectivity between each polyhedron. After that, we build oriented bounding boxing [10] for every moving obstacle, thus the infeasible region is approximated tightly. Then we utilize the hyperplanes of the bounding boxes to decompose polyhedra overlapping the obstacles into some small free polyhedra. Finally, we take advantage of the prior information of the graph to update the graph efficiently.

Contributions of this paper are:

- 1) An efficient method is proposed that incrementally updates polyhedral maps in the dynamic environments yet preserves connectivity.
- 2) An incremental graph updating mechanism is designed.
- 3) An autonomous navigation system that uses the above methods is tested in simulation and real-world to validate that our methods are practical.

II. RELATED WORK

A. Free-Space Oriented Map Representation

There are many works focusing on polyhedronizing free regions in environments for efficient navigation and trajectory optimization. Deits and Tedrake [6] adopt IRIS which proposed by themselves to segment space into convex regions, then perform a mixed-integer optimization to obtain a collision free trajectory. Wang et al. [7] tightly approximates all free configurations by unions of polyhedra for encoding

All authors except Yi Lin¹ are with the State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China, and also with the Huzhou Institute of Zhejiang University, Huzhou 313000, China. Corresponding author: Fei Gao. ¹DJI, Shenzhen 510810, China. This work is supported by DJI. E-mails: {22060478, cxu, fgaoaa}@zju.edu.cn

Algorithm 1 MapPolyhedronization

```
1: Notation:sampling time: $s$ , maximum sampling times: $S$ ,  
   expected ratio of  $vol(\mathcal{P}_g)$  to  $vol(\mathcal{F})$ : $\rho_e$ ,  $vol(\cdot)$  means  
   volume.  
2: Input: $\mathcal{M}, S, \rho_e$ ,  
3: Output: $\mathcal{P}_g$   
4:  $\mathcal{F} \leftarrow GetFreeRegion(\mathcal{M})$   
5:  $\mathcal{O} \leftarrow GetObstacleRegion(\mathcal{M})$   
6:  $\mathcal{P}_g \leftarrow \emptyset, \rho \leftarrow 0, i \leftarrow 0, s \leftarrow 0$   
7: while  $\rho < \rho_e$  and  $s < S$  do  
8:    $q \leftarrow random(\mathcal{F}, \mathcal{P}_g)$   
9:    $s \leftarrow s + 1$   
10:   $bbox \leftarrow BoundingBox(q)$   
11:   $P_i \leftarrow IRIS(q, bbox, \mathcal{M})$   
12:  if  $IsGoodPoly(P_i)$  then  
13:     $i \leftarrow i + 1$   
14:     $\mathcal{P}_g \leftarrow \mathcal{P}_g \cup P_i$   
15:     $\rho \leftarrow UpdateRatio(\mathcal{P}_g, \mathcal{F})$   
16:  end if  
17: end while  
18: return  $\mathcal{P}_g$ 
```

free space information into multi-aerial robots trajectory optimization. The work mostly relevant to ours is [4]. The author extracts free spaces from noisy and partly incomplete visual SLAM data by voxel clustering and merging, then a topological navigable graph is constructed, which make global path planning easy and computationally inexpensive. In [5], the author also use IRIS to generate polyhedra to represent obstacle-free regions from raw point cloud data, then projecting original sparse and noisy points onto the surfaces of polyhedra to form a dense version of point cloud which provides abundant environment information for human users. This is a rather inspiring work which for the first time combines point regulation and space convexification to create dense point cloud and navigable space.

Nevertheless, these works still assume a static environment. In applications, there are more or less dynamic objects in the considered navigation scene.

B. Representation of Dynamic Environments

Dealing with dynamic environments is a challenging topic in autonomous navigation. There are some approaches which handle it at a low level of map abstraction. In [11], the author uses a probabilistic grid map to represent the environment and a hidden Markov model to occupancy state and state transition probabilities of the grid. The occupancy state of each grid will be updated when observations become available. Some other approaches explicitly model the dynamic objects instead. For example Anguelov [12] and Biswas [13] compute the shape of dynamic models, use expectation maximization to identify dynamic parts of the maps that are created at different timestamps. In [14], two probability occupancy grid map are maintained accordingly for static parts and dynamic parts of the environment, these two grid maps will be updated by Bayesian rules once

observations come from sensors. Lau et al. [15] develop algorithms that incrementally update grids which affected by changes. Meanwhile, Euclidean distance maps and generalized Voronoi diagrams can also be incrementally updated by their algorithms. Some other methods explicitly model aspects of the environment dynamics. For example, Wang et al. [16] and Eppenberger et al. [17] identify dynamic objects from environment based on the fact that the positions of dynamic objects change overtime. However these methods focus on tracking dynamic objects rather than represents the environments' dynamic parts.

III. METHODOLOGY

A. Map Polyhedronization

We combine IRIS [9] and RILS [18] to segment the environment into convex regions. IRIS proposed by Deits and Tedrake alternates two convex optimizations to generate a polyhedron by a given query seed: First a Quadratic Program (QP) generates a convex region represented by a set of hyperplanes. Second a Semidefinite Program (SDP) finds a maximum volume inscribe ellipsoid (MVIE) in the convex region that constructed in first step. RILS is similar to a non-iterative version of IRIS but more efficient because of a bounding box is applied to reduce obstacles need to considered. Like RILS does, we also use bounding box in IRIS to relieve computational burden.

Let \mathcal{M} denote the global occupied grid map. Note that the map \mathcal{M} will only be used for map polyhedronization and not needed in subsequently motion planning. We denote by $\mathcal{F} \subseteq \mathcal{M}$ the free configurations in map \mathcal{M} and by \mathcal{P}_g the union of polyhedra where $\mathcal{P}_g \subseteq \mathcal{F}$ and

$$\mathcal{P}_g = \bigcup_{i=0}^N P_i, \quad (1)$$

$$P_i = \{x \in \mathbb{R}^3 | A_i x \preceq b_i\}, \quad (2)$$

The map polyhedronization algorithm is detailed as Algorithm 1. Firstly, we randomly sample a query point $\mathbf{q} = [x_p, y_p, z_p] \in \mathbb{R}^3$ that \mathcal{F} while outside \mathcal{P}_g . Then IRIS takes the query point and the bounding box around it as input to generate a polyhedron P_i in H-representation from \mathcal{M} . Each elements in \mathcal{P}_g is managed in a hierarchical structure. In order to approximates the map more tightly, Algorithm 1 will not stop until the ratio ρ of \mathcal{P}_g to \mathcal{F} or sampling times s over expected thresholds.

Note that the purpose of polyhedronizing a map is to obtain the set of polyhedra which approximates all free spaces, thereby any other algorithms that generate a polyhedron by a query point can replace IRIS in Algorithm 1.

B. Polyhedron Decomposition and Restoration

1) *Preliminaries:* After map polyhedronization, we need to consider how to deal with the situation where obstacles overlap the polyhedra. Indeed the first thing is to distinguish point clouds which are inside the polyhedra. Therefore axis-aligned bounding box of these polyhedra are constructed, and

Algorithm 2 PolyhedronDecomposition

```

1: Notation:  $\mathcal{P} = \{P_1, P_2, \dots, P_m\}, P_i \in \mathcal{P}_g$ 
2: Input:  $\mathcal{P}, obb$ 
3: function POLYHEDRONDECOMPOSITION( $\mathcal{P}, obb$ )
4:   for  $P_i$  in  $\mathcal{P}$  do
5:     if  $P_i.state == \text{complete}$  then
6:        $Decomposition(P_i, obb)$ 
7:     else if  $P_i.state == \text{decomposed}$  then
8:        $\mathcal{P}_i \leftarrow ObtainCollisionsons(P_i)$ 
9:       POLYHEDRONDECOMPOSITION( $\mathcal{P}_i, obb$ )
10:    end if
11:  end for
12: end function
13:
14: Input:  $P, obb$ 
15: function DECOMPOSITION( $P, obb$ )
16:    $P.state \leftarrow \text{decomposed}$ 
17:    $\mathcal{P}_t \leftarrow \emptyset$ 
18:   for  $H_i$  in  $obb$  do
19:      $P_i \leftarrow Intersection(P, H_k)$ 
20:     if  $IsGoodPoly(P_i)$  and  $DeRedun(P_i, \mathcal{P}_t)$  then
21:        $\mathcal{P}_t \leftarrow \mathcal{P}_t \cup P_i$ 
22:        $P.addson(P_i)$ 
23:     end if
24:   end for
25: end function

```

managed by a multi-level segment tree *SegTree* which is used for efficient stabbing queries [19]. For a given point $\mathbf{p} = [x_p, y_p, z_p] \in \mathbb{R}^3$, the query time complexity is $O(\log^3 n + k)$ and k bounding boxes enclose the point \mathbf{p} can be obtained at the same time. We denote by \mathcal{P}_i the polyhedra obtained by querying the *SegTree* and enclosing the query point \mathbf{p} .

In order to separate obstacles from polyhedra, an oriented bounding box obb_i are generated for each obstacle O_i , where

$$obb_i = \{x \in \mathbb{R}^3 | A_i x \preceq b_i\}. \quad (3)$$

Any polyhedron intersecting the obstacle O_i is denoted by \mathcal{P}_i . Similar to dilating obstacles in grid map, we expand the oriented bounding box by radius r of the aerial robot to maintaining a configuration space.

2) *Polyhedron Decomposition and Restoration:* Benefiting from the convex sets property: *the intersection of any collection of convex sets is convex*. It is obvious that any polyhedron determined by halfspace intersection is also a convex set. In viewing of this, a natural idea is to utilize hyperplanes of the bounding box to decompose the polyhedron to exclude the obstacle enclosed by oriented bounding box. The pipeline for Polyhedron Decomposition algorithm is detailed in Algorithm 2.

We use label **complete** and **decomposed** to distinguish whether a polyhedron is a complete one or a decomposed one. For every polyhedron in \mathcal{P} that will be decomposed by obb , we first check the label of the polyhedron. Polyhedron that labeled as **complete** will be decomposed by each hyper-

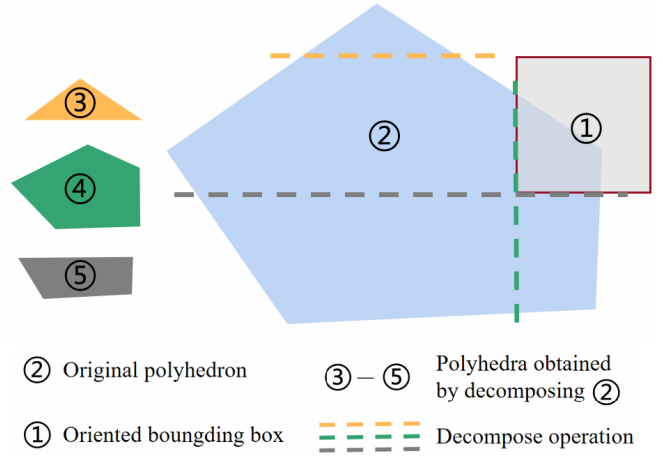


Fig. 2: Polyhedron ② decomposed into polyhedra ③-⑤ by oriented bounding box ①, however polyhedron ③ is redundant since it insides ④.

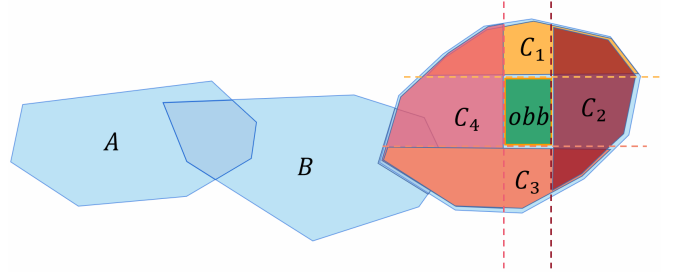


Fig. 3: Illustration of avoiding unnecessary connectivity checks by prior knowledge.

planes of obb . As illustrated in Fig. 2, redundant polyhedra in decomposition procedure will be abandoned. While for **decomposed** polyhedron, all elements in the next level of the hierarchy that intersecting with obb will be decomposed again.

Corresponding to polyhedron decomposition, polyhedron restoration is also a necessary part. Any polyhedron \mathcal{P}_i decomposed by the dynamic obstacle O_i should be restored as before. For the situation that the obstacle O_i moves to another place, we first restored all elements in \mathcal{P}_i to their original shape. Afterward, polyhedra will be decomposed by obstacles that still intersecting them. Algorithm 3 describes the process in detail.

Algorithm 3 PolyhedronRestoration

```

1: Notation:  $obb_j \cap obb_i = \emptyset$ , and  $p_{ij}$  will be treated as a vector as input to Algorithms 2 for simplicity.
2: Input:  $O_i, obb_i, \mathcal{P}_i$ 
3: for  $p_{ij}$  in  $\mathcal{P}_i$  do
4:    $Restoration(p_{ij})$ 
5:   for each  $obb_j$  that intetsecting with  $p_{ij}$  do
6:      $PolyhedronDecomposition(p_{ij}, obb_j)$ 
7:   end for
8: end for
9: return  $\mathcal{P}_g$ 

```

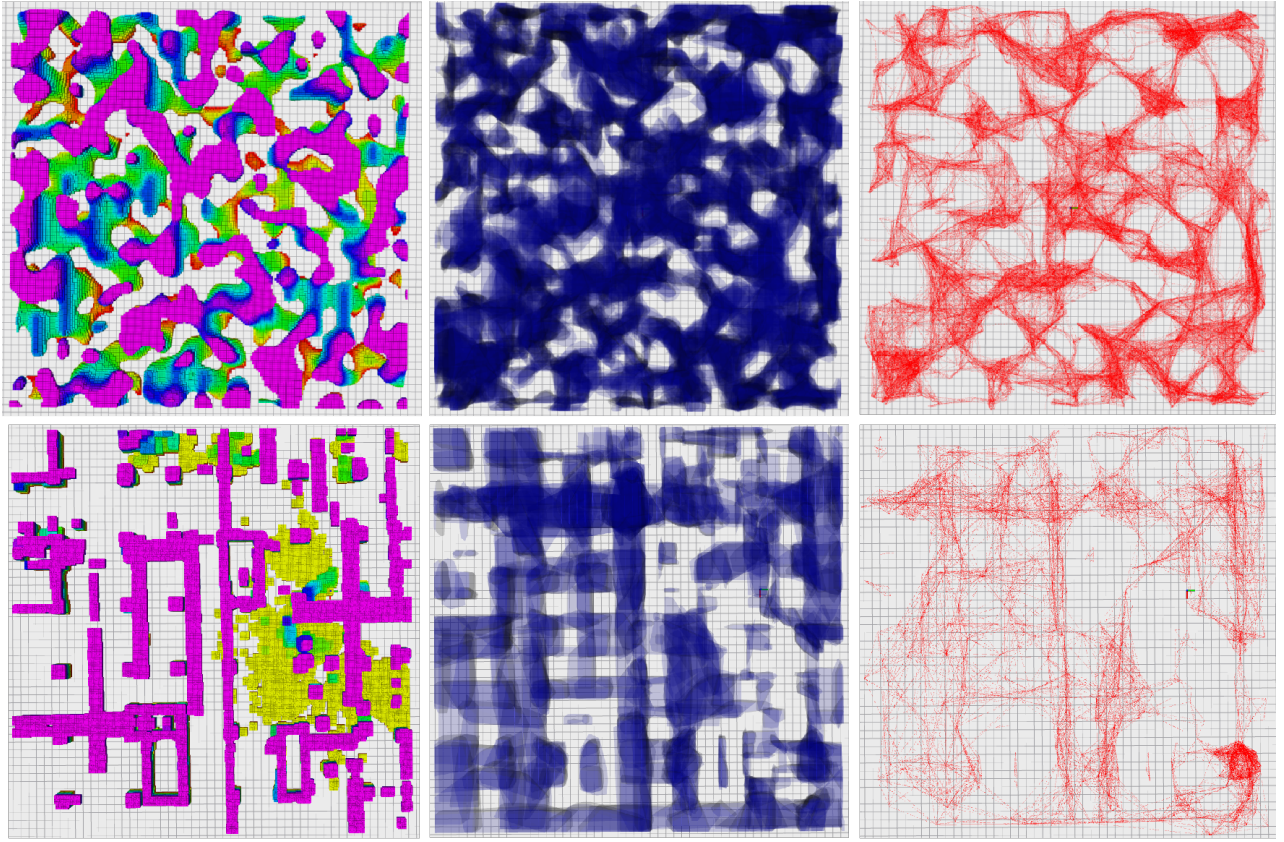


Fig. 4: The detailed information of cluster and sparse scenario. the grid map, free-space roadmap and navigation graph of sparse scenario are presented on the top while the bottom corresponds to the sparse.

3) *Graph Construct and Update*: Connectivity between each polyhedron can be naturally used for path searching. A global graph can be constructed base on it for search-based planning methods. We interpret the geometric center of the polyhedron as room and the geometric center of the intersection between polyhedra as doors. The rooms are vertices of the graph thus going from one room to another adjacent room will go through doors. Thanks to convex set property, edges that connect any two vertices in graph are collision free. The graph will be updated after every polyhedron decomposition and restoration. In order to update the graph efficiently, we take full advantage of the prior information between the convex hulls to avoid unnecessary connectivity checks. As illustrated in Fig. 3, there is no need to check the connectivity between Polyhedron C_1 and C_3 , C_2 and C_4 since they are naturally disjoint. The connectivity between $C_1 - C_4$ can be established quickly. Moreover, we do not need to check the connectivity between polyhedron A and each of $C_1 - C_4$ by utilizing the prior information of the graph.

IV. EXPERIMENTS AND RESULTS

To validate the applicability and evaluate the efficiency of our algorithms, simulations and real-world experiments are both conducted. For feasibility validation, we conduct motion planning in simulations and real-world experiments

to demonstrate that our roadmap can provide safe feasible spaces for our planners in real-time. While for efficiency evaluation, we mainly focus on roadmap update efficiency. We evaluate the map update efficiency by computing time for polyhedron decomposition t_d , the computing time for polyhedron restoration t_r and the computing time for graph update t_g .

A. Simulation Experiments

In simulation experiments, we validate our algorithm in both clustered and sparse scenarios to demonstrate that our algorithm is feasible and efficient. The size of two scenarios are set to the same. There are four dynamic obstacles in both scenarios. The detailed information of two scenarios is shown in Fig. 4. In clustered scenario, both the map and the dynamic obstacles are random generated. The detailed specifications of map are illustrated in Tab. I. While in sparse scenario, the map is constructed according to a underground garage in real world. The dynamic obstacles are also random generated. In clustered environment, there are totally 2613 polyhedra generated, and 37753 edges in the graph. While in sparse scenario, there are only 651 polyhedra, and 5468 edges in the graph.

We conduct trajectory planning in both scenarios to validate the feasibility of our algorithm. We randomly select the start and goal point in free spaces, then A^* is applied to

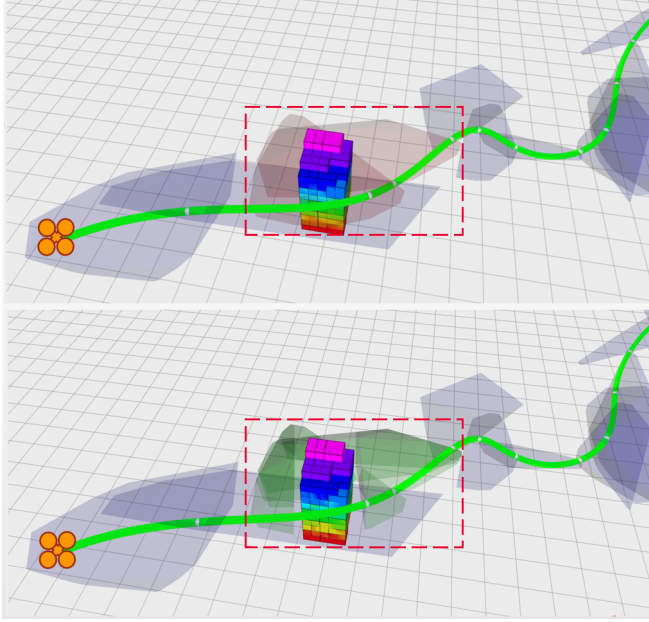


Fig. 5: Feasibility validation. Once there is an obstacle occur in convex hulls (red color) we want to pass through, the convex hulls (red color) will be decomposed into some small obstacle-free convex hulls (green color).

search a series of polyhedra connecting the start and goal in the graph. Finally, A collision-free trajectory is generated by a global trajectory optimizer [7]. The re-planning frequency is set to $10Hz$ and triggered whenever the polyhedra the vehicle is going through has been decomposed or restored in last $0.2s$. Results illustrated in Fig. 5 demonstrate that when dynamic obstacles occur in the polyhedra that the quadrotor are going through, these polyhedra is decomposed efficiently and the graph is updated at the same time. After that A^* finds another series of free polyhedra connecting the current point and the goal. The results validate that the roadmap is able to reflect changes in the environment for real-time safe motion planning.

We also evaluate the efficiency of our algorithm by the three metrics in both scenarios. Results are shown in both Tab. II and Tab. III. In clustered scenario, the average time of polyhedron decomposition and restoration are $2.784ms$ and 0.482 , and the average time of graph update is $25.017ms$. While in sparse scenario, the average time of polyhedron decomposition and restoration are $2.777ms$ and $0.3632ms$, and the average time of graph update is $16.823ms$. Comparing results in these two scenarios, we can see that the average time of t_d and t_r have negligible difference in two scenarios. However the average time of graph update t_g varies a lot. This is reasonable since it requires more convex

TABLE I: Specifications of Two Maps.

	Map Size	Resolution
Cluster Scenario	$50m \times 50m \times 5m$	$0.2m \times 0.2m \times 0.2m$
Sparse Scenario	$50m \times 50m \times 5m$	$0.2m \times 0.2m \times 0.2m$

TABLE II: Time Metrics of Cluster Scenario Simulation.

	Times	Total Time	Average Time
Decomposition	853	$2.375s$	$2.784ms$
Restoration	809	$0.390s$	$0.482ms$
Graph Update	801	$20.039s$	$25.017ms$

TABLE III: Time Metrics of Sparse Scenario Simulation.

	Times	Total Time	Average Time
Decomposition	1140	$3.166s$	$2.777ms$
Restoration	1527	$0.55458s$	$0.3632ms$
Graph Update	1115	$18.763s$	$16.823ms$

hulls to approximate the environment in clustered scenario than a sparse one in same size space. In a space with fixed size, the greater the number of convex hulls is, the more complex the connections between the convex hulls become, thus resulting in a slower graph update efficiency. That is why we utilize the prior information of the graph to avoid unnecessary connectivity checks. Even the update time of the map varies in this two scenarios, it still meets the real-time requirements.

B. Real-World Experiments

For real-world experiments, we integrate our roadmap into a quadrotor platform and use FAST-LIO [20] to pre-build the environment. We use the combination of foam pillars and a ground vehicle controlled by human operators as dynamic obstacles. The location of the quadrotor and dynamic obstacles is provided by motion capture. The quadrotor platform and the obstacles are shown in Fig. 6. The experiments was divided into two parts, that are to validate the feasibility and evaluate the efficiency of our algorithm. In the first part, the start point is fixed, then we manually select a goal point. The quadrotor searches a series of collision-free polyhedra connecting the start point and goal point, then accomplishes the flight. The Max velocity of the quadrotor is set to be $1m/s$. In the second part, we mainly focus on evaluate the efficiency, so there are only the obstacle move freely in the environment and the Max velocity is $6.5m/s$.

TABLE IV: Time Metrics of Real-World Experiments.

	Times	Total Time	Average Time
Decomposition	1070	$1.6806s$	$1.569ms$
Restoration	1069	$0.151254s$	$0.1414ms$
Graph Update	1070	$3.735s$	$3.491ms$

In the first experiments, we set the start point and goal point as the two ends of the map. We try to manipulate the obstacle to block the flight of quadrotor. Since the roadmap is efficiently updated and the connectivity is preserved, the quadrotor quickly finds a series of polyhedra connecting start and goal when obstacle moves. Finally the quadrotor successfully avoid the dynamic obstacle and reach the goal point as shown in Fig. 6. The results of second experiments are shown in Tab. IV, these three metrics reflect that the

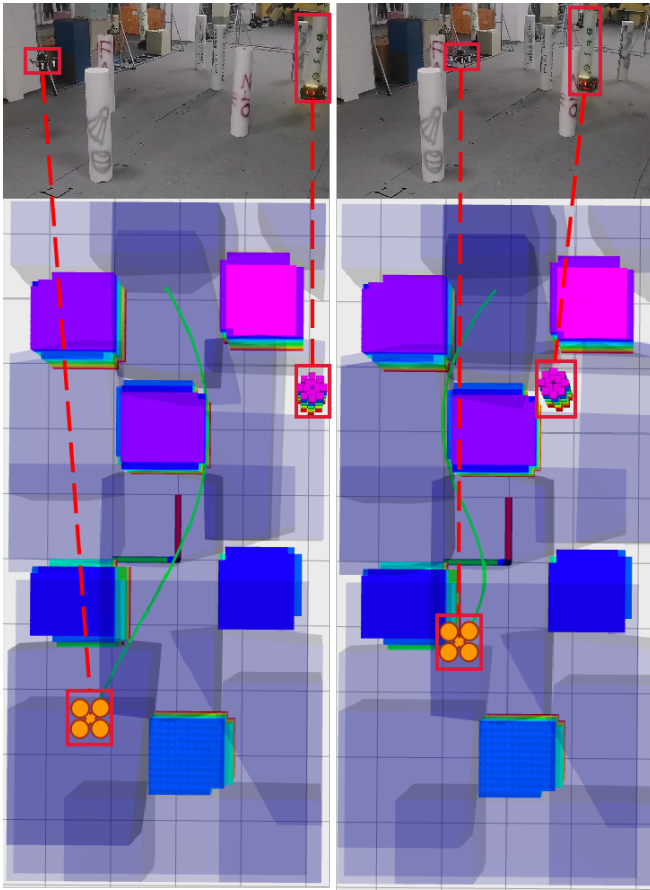


Fig. 6: Left figures show that A star find a path that through the right side of the middle pillar. As the obstacle moves, the trajectory was blocked, re-plann mechaand the algorithm finds another collision-free path.

roadmap is updated efficiently. Comparing to experiments in simulations, t_d , t_r , t_g in real-world environment is much lower because the map is smaller and sparser. Meanwhile, there are only one dynamic obstacle, hence there will only be two levels in the hierarchy at most.

V. CONCLUSION

In this paper, we present dynamic free-space roadmap, a novel map representation that provides both safe feasible space and navigation graph. The roadmap can be incrementally updated by using our polyhedron decomposition and restoration method. We adopt the hierarchy structure to manage the map, making our map convenient to maintain. By utilizing the prior information of the graph, unnecessary connectivity checks are avoided thus the graph can be updated efficiently. We also conduct simulations and real-world experiments to demonstrate that our method is feasible and efficient. In the future, we will design a path search method and trajectory planner that fit our roadmap. A prediction module for dynamic obstacles will also be incorporated into our quadrotor to achieve autonomous flights in dynamic environments.

REFERENCES

- [1] Y. Roth-Tabak and R. Jain, "Building an environment model using depth information," *Computer*, vol. 22, no. 6, pp. 85–90, 1989.
- [2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.
- [3] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4423–4430.
- [4] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3818–3825.
- [5] Z. Chen and L. Liu, "Creating navigable space from sparse noisy map points," *arXiv preprint arXiv:1903.01503*, 2019.
- [6] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 42–49.
- [7] Z. Wang, C. Xu, and F. Gao, "Robust trajectory planning for spatial-temporal multi-drone coordination in large scenes," *arXiv preprint arXiv:2109.08403*, 2021.
- [8] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.
- [9] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 109–124.
- [10] D. M. Advisor, M. C. Lin, F. P. Brooks, and S. Gottschalk, "Collision queries using oriented bounding boxes," *The University of North Carolina at Chapel Hill*, 2000.
- [11] F. H. Ajeil, I. K. Ibraheem, A. T. Azar, and A. J. Humaidi, "Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments," *Sensors*, vol. 20, no. 7, p. 1880, 2020.
- [12] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, and S. Thrun, "Learning hierarchical object maps of non-stationary environments with mobile robots," *arXiv preprint arXiv:1301.0551*, 2012.
- [13] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, "Towards object mapping in non-stationary environments with mobile robots," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 1. IEEE, 2002, pp. 1014–1019.
- [14] D. F. Wolf and G. S. Sukhatme, "Mobile robot simultaneous localization and mapping in dynamic environments," *Autonomous Robots*, vol. 19, no. 1, pp. 53–65, 2005.
- [15] B. Lau, C. Sprunk, and W. Burgard, "Efficient grid-based spatial representations for robot navigation in dynamic environments," *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1116–1130, 2013.
- [16] Y. Wang, J. Ji, Q. Wang, C. Xu, and F. Gao, "Autonomous flights in dynamic environments with onboard vision," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1966–1973.
- [17] T. Eppenberger, G. Cesari, M. Dymczyk, R. Siegwart, and R. Dubé, "Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 528–10 535.
- [18] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.
- [19] M. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry, Algorithms And Applications*. Computational Geometry, Algorithms And Applications, 2008.
- [20] W. Xu and F. Zhang, "Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317–3324, 2021.