DSVP: Dual-Stage Viewpoint Planner for Rapid Exploration by Dynamic Expansion

Hongbiao Zhu[1,2], Chao Cao[1], Yukun Xia[1], Sebastian Scherer[1], Ji Zhang[1], and Weidong Wang[2]

***Abstract*— We present a method for efficiently exploring highly convoluted environments. The method incorporates two planning stages - an exploration stage for extending the boundary of the map, and a relocation stage for explicitly transiting the robot to different sub-areas in the environment. The exploration stage develops a local Rapidly-exploring Random Tree (RRT) in the free space of the environment, and the relocation stage maintains a global graph through the mapped environment, both are dynamically expanded over replanning steps. The method is compared to existing state-of-the-art methods in various challenging simulation and real environments. Experiment comparisons show that our method is twice as efficient in exploring spaces using less processing than the existing methods. Further, we release a benchmark environment to evaluate exploration algorithms as well as facilitate development of autonomous navigation systems. The benchmark environment and our method are open-sourced.**

I. Introduction

Autonomous exploration tackles the problem of deploying robots in environments unknown a priori for information gathering. This problem is essential for fulfilling tasks such as search, rescue, and survey. Yet, it remains challenging due to the complex structural setting and geometric layout in the environment. Very often, the environment to be explored is convoluted, consisting of branches connected at intersections. The robot needs to transit between sub-areas in order to efficiently explore the environment.

The paper puts forward a method capable of efficiently exploring environments at a high-degree of convolution. The method incorporates two planning stages - an exploration stage in charge of extending the boundary of the map, and a relocation stage for explicitly transiting the robot to different sub-areas in the environment (see Fig. 1). The exploration stage uses a local Rapidly-exploring Random Tree (RRT) [1] to span the space in the surroundings of the robot, searching for the branch on the RRT leading to the highest collective reward for the robot to execute. The relocation stage involves a global graph built along the course of the exploration, keeping a record of fully and partially covered areas. During deployment, the robot transitions back-and-forth between the two stages to explore all areas in the environment.

Our method draws inspiration from a well-known exploration algorithm framework [2]. Such a framework expands a RRT in the free space and considers nodes on the RRT as viewpoints. Sensor coverage is estimated from each viewpoint. Computing the reward of each branch accounts for the coverage of all underlying viewpoints on the branch. Our method extends the framework in mainly two aspects.

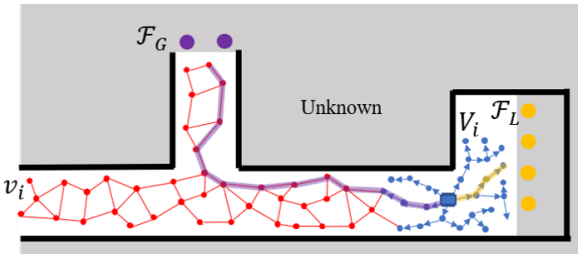


Fig. 1: Illustration of our method. The grey area stands for the unknown space. The black solid lines are the obstacles , i.e. the occupied space. The blue solid rectangle is the robot. The purple dots in the unknown space are the global frontiers $\mathcal{F}_G$, while the yellow dots are the local frontiers $\mathcal{F}_L$. The blue dots, local viewpoints V_i, and blue arrows form the local tree. The red dots and red lines make up the global graph. Those red dots are the global vertices v_i, which are viewpoints as well. The yellow and purple semi-transparent lines are exploration and relocation paths.

- *Dynamic Expansion*: Both stages dynamically expand the RRT and graph, respectively, over replanning steps. Nodes on the RRT that are occluded or out of the planning horizon are trimmed off. Then, new nodes are sampled in the free space. This way, useful viewpoints are kept, while newly sampled viewpoints further enforce the solution. Further, much computation is saved not re-building the entire RRT at each replanning step.
- *Hybrid Frontiers*: The method uses a combination of frontiers extracted in the sensor range associated with the RRT nodes as well as frontiers extracted in the surroundings of the robot. Due to the randomness of RRT, using any single type of frontiers or none often results in certain areas in the environment being overlooked. Our method uses both types of frontiers to guide the expansion of the RRT, ensuring complete coverage.

In addition to the above theoretical contributions, we release a benchmark environment[1] containing representative simulation environments, fundamental navigation modules, e.g. collision avoidance, terrain traversability analysis, waypoint following, and visualization tools for benchmarking exploration algorithms. The environment is also meant to facilitate development of autonomous navigation systems.

Our exploration method is evaluated in the benchmark environment and physical experiment where the robot explores an area containing multiple buildings on the university campus. In all evaluated environments, our method signifi-

[1]H. Zhu, C. Cao, Y. Xia, S. Scherer, and J. Zhang are with the Robotics Institute at Carnegie Mellon University, Pittsburgh PA.

[2]H. Zhu and W. Wang are with the Robotics Institute at Harbin Institute of Technology, Harbin China.

[1]Benchmark environment: `https://www.cmu-exploration.com`

cantly outperforms the state-of-the-art methods in terms of exploration efficiency. Our method is open-sourced[2] and our experiment results are available in a public video[3].

II. Releated Work

In recent years, numerous techniques have been developed to solve the autonomous exploration problem, such as frontiers-based algorithm [3]–[9], next-best-view algorithms [2], [10]–[13] and algorithms based on machine learning [14]–[17]. The previous two types of algorithm are commonly used in most exploration methods while machine learning based approaches have emerged recently.

Frontier-based approach is among the most effective ways in exploration. In frontier-based algorithms, one important issue to solve is the sequence to visit frontiers. Method in [3] selects the closest frontier as the goal, often causing repeated visits during the exploration. Approach [4] makes improvements by using a repetitive rechecking method and segmenting the environment into small pieces. Since the segmentation is adapted to structured indoor environments, the method only works well indoors. Traveling Salesman Problem (TSP) is later employed in [5], [18] to get the sequence of visiting all frontiers. However, as more frontiers are generated along the exploration, the TSP becomes larger and heavier to solve. In [19], instead of taking frontiers as goals, viewpoints that can see all frontiers are generated and a generalized TSP is used to obtain the best sequence to visit the viewpoints.

In contrast, next-best-view approaches do not use frontiers as the direct guidance or goals, but use randomly sampled viewpoints in the free space. Next Best View Planner (NBVP) [2] is considered the state-of-art in this category. It generates viewpoints with RRTs and then computes the volumetric gain of each viewpoint. Yet, its major limitation is that it only focuses on the process of extending the map boundary. The method is limited in transiting to different areas in the environment for further exploration, after one area is fully explored. A method named Graph-Based exploration Planner (GBP) [13] develops a global Rapidly-exploring Random Graph (RRG) [20] to re-position the robot to unexplored areas. Another method named Motion-primitive-Based exploration Planner (MBP) [12] develops the local RRT using motion primitives. The resulting paths are smoother and span in constrained directions. However, all the three methods expand a new RRT or RRG at each replanning step in the exploration mode. A large number of useful nodes are removed and re-sampled, wasting computation. Further, since RRT and RRG expand randomly in the free space, areas that are not spanned by the RRT or RRG are ignored. Consequently, the methods are prone to overlook areas, especially for areas with small openings, and produce an incomplete coverage.

Our method extends the state-of-the-art methods by dynamically maintaining and expanding the RRT during the exploration and guiding the expansion of the RRT with hybrid frontiers. We compare our method with NBVP [2], MBP [12], and GPB [13] in various simulation and real-world environments. We conclude that our method produces more complete coverage and the exploration efficiency is more than twice of the state-of-the-art while the processing load is less.

[2]DSVP code: `https://github.com/HongbiaoZ/dsv_planner`

[3]Representative result: `https://youtu.be/1yLLIZIIsDk`

III. Methodology

Define $\mathcal{S} \subset \mathbb{R}^3$ as the space to be explored. Let $\mathcal{S}_{free} \subset \mathcal{S}$ be the known free space, $\mathcal{S}_{occ} \subset \mathcal{S}$ be the known occupied space and $\mathcal{S}_{unk} \subset \mathcal{S}$ be the current unknown space. As shown in Fig. 1, at the exploration stage of our method, a dynamically-expanded RRT is used to create the local random tree, the nodes of which are viewpoints. The best branch is then obtained and taken as the trajectory by computing the reward of each branch in the tree. In this stage, frontiers that are within the field of view of the robot as well as the RRT nodes are extracted as local frontiers. At the relocation stage, global frontiers, which are made up of the local frontiers that are not cleared until the latest update, assist the planner to choose the best one from all remaining viewpoints in the global graph.

A. Exploration Stage

Our method uses dynamically-expanded RRT at the exploration stage to generate viewpoints around the robot in each iteration. Fig. 2 shows the process of dynamic expansion. As shown in Fig. 2a, define $\mathcal{H} \subset \mathbb{R}^3$, the green square, as the planning horizon and set the current position $\mathcal{P}_R$, point A, as the root of the tree. Then, a RRT is constructed at the first iteration, before which there is no previous RRT. All nodes on the tree are viewpoints, defined as V. we use octomap [21] as the underlying occupancy map, with which the collision check of viewpoints and edges between viewpoints are performed to ensure they are in the free space. When the robot moves to point B in Fig. 2b after an iteration, the previous tree is reconstructed in two steps. First, the tree is pruned by deleting all nodes that are occluded or out of the current planning horizon $\mathcal{H}$, such as the light blue nodes in Fig. 2b. Next, we update the tree structure so that $\mathcal{P}_R$, point B, becomes the root of the new tree. New viewpoints, orange nodes in Fig. 2b, are randomly sampled and added to the new tree. With the dynamic expansion, only a small fraction of nodes are re-generated in each iteration, which results in less computation compared to completely constructing a new tree. In addition, we prune nodes that are in collision due to dynamic obstacles and thin structures previously overlooked by the sensor.

We use local frontiers $\mathcal{F}_L$ to bias the tree construction during the exploration stage so that the tree expands towards unknown areas along the previous direction of exploration. Local frontiers in this paper must meet the following conditions in Eq. (1)-(3).

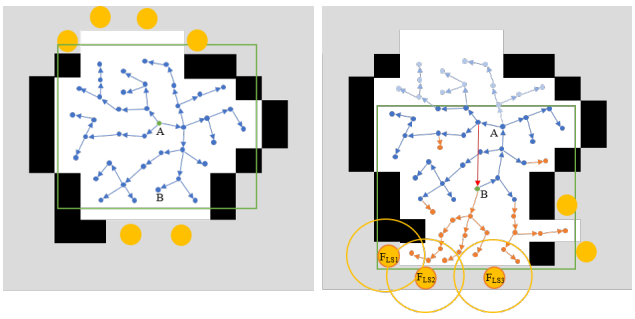
$$\mathcal{F}_L \in \mathcal{B} \tag{1}$$

$$\exists V \; s.t. \; \mathcal{F}_L \; is \; in \; FOV(V) \; or \tag{2}$$

$$\mathcal{F}_L \; is \; in \; FOV(\mathcal{P}_R) \tag{3}$$

In Eq.(1), $\mathcal{B}$ represents a boundary within which the local frontiers are extracted. This region is slightly bigger than the planning horizon. Condition (2) and (3) require the frontier to be in the field of view (FOV) of at least one viewpoint or in the FOV of the robot. Note that line-of-sight check is performed to ensure that the frontier can be observed by the viewpoint or the robot without occlusion. Under these two conditions, the hybrid frontiers that are around the viewpoints and around the robots can be extracted. Note that when extracting frontiers, the sensor range of the viewpoints is set to smaller than that of the robot. This is because that viewpoints are expanded closer to frontiers, where a shorter range is sufficient for observing the fronteirs and can reduce overall computation. Condition (2) and (3) also serve as the noise-filtering process, where in complex environments occulusion and sensor noise can result in noisy frontiers that are not worth exploring. Given that frontiers are only used as guidance, they can be grouped into sparse clusters.

Among all $\mathcal{F}_L$, we select $\mathcal{F}_{LS}$ to be the closest ones to the current exploration direction. The selected frontiers bias the tree expansion as shown in Fig. 2b, where $\mathcal{F}_{LS1}$, $\mathcal{F}_{LS2}$, and $\mathcal{F}_{LS3}$ are the frontiers selected. The biased sampling scheme is described as follows. We first uniformly sample a number between 0 and 1. If the number is larger than θ, a threshold for regulating the sampling area, then we randomly sample points in the selected frontiers' sensor range. Otherwise, we sample points in other regions. The probability of sampling points falling around the selected frontiers is much higher than in other regions. Thus, the tree tends to expand towards the frontier, resulting in more viewpoints close to the current exploration direction. With



(a) (b)

Fig. 2: Exploration stage. (a) shows the tree and local frontiers obtained in the previous iteration. Grey area is unknown space and black area is occupied space. Yellow solid circles are local frontiers $\mathcal{F}_L$. The green square denotes the planning horizon $\mathcal{H}$. (b) is the new tree generated in the current iteration. The light blue dots are pruned viewpoints that are out of the current planning horizon. Blue dots stand for useful viewpoints from the old tree and orange dots are new sampled viewpoints. $\mathcal{F}_{LS1}$, $\mathcal{F}_{LS1}$ and $\mathcal{F}_{LS1}$ are three selected local frontiers used to guide the extension of local tree. Yellow hollow circles are the sensor ranges of the selected frontiers.

Algorithm 1: Exploration

```
Set S_lb, root position P_rob and F_local
Update F_LS
V ← DynamicRRT()
BestGain ← 0
for i from 1 to N do
    Compute Gain(B_i)
    if Gain(B_i) > BestGain then
        BestGain ← Gain(B_i)
        BestBranch ← B_i
    end
end
Previous Tree ← Current Tree
```

the guidance of local frontiers, the viewpoints distributed more densely near unknown areas in $\mathcal{H}$. This can help make the sampling process more effective.

Define $\mathcal{V} = \{V_1, V_2, ..., V_n\}$ as the set of viewpoints, where the subscript indicates the order in which the corresponding viewpoint is generated. Eqs. (4) and 5 show the utility function used to compute the gain of each branch in the tree. It is similar to the method used in [13].

$$Gain(B_i) = \sum_{V_i^j \in B_i} Gain(V_i^j) \cdot e^{-DTW(B_i)\cdot\lambda_1} \tag{4}$$

$$Gain(V) = VoxelGain(V) \cdot e^{-dist(V)\cdot\lambda_2} \tag{5}$$

where B_i represents the branch from root viewpoint V_0 to V_i and V_i^j represents the jth viewpoint on B_i. $VoxelGain(V)$ is the number of unknown voxels in the FOV of viewpoint V. $dist(V)$ denotes the distance of the tree branch from V_0 to V, and λ_1 is a parameter that penalizes traveling distance. Function $DTW(B_i)$ is based on Dynamic Time Warping method [22] that computes the similarity between branch B_i and the branch selected in the last iteration, which also reflects the exploring direction. The more similar these two branches are, the lower the value of $DTW(B_i)$. λ_2 is a parameter that penalizes the difference between B_i and the last trajectory. The branch with the greatest gain will be picked as the next trajectory.

Algorithm 1 and 2 illustrate the process of the exploration stage. The local frontiers are updated at a constant frequency. $\mathcal{F}_{LS}$ are selected from all local frontiers at the beginning of each iteration. Then, with the guidance of $\mathcal{F}_{LS}$, new viewpoints are sampled after pruning and reconstruction of the previous tree. Eventually, Eq. (4) is used to compute the gain of each branch and determine the final trajectory.

B. Relocation Stage

When there is no local frontiers within the planning horizon, the planner switches from the exploration stage to the relocation stage. The relocation stage involves the global graph and global frontiers. The main utility of the global graph $\mathcal{G}$ is to record all the valuable viewpoints sampled at the exploration stage and search for the shortest path between

Algorithm 2: Dynamic Expansion

Set $\mathcal{S}_{lb}$, root position $\mathcal{P}_{rob}$ and $\mathcal{F}_S$
Prune(Previous Tree)
Rebuild(Previous Tree)
while $N_{new} < \mathcal{N}$ **do**
  Sample $u \sim U[0,1]$
  if $u <= \theta$ **then**
    Random Sample viewpoints in $\mathcal{S}_{lb}$
  else
    Random Sample viewpoints around $\mathcal{F}_S$
  end
end

two viewpoints. In each iteration of the exploration stage, viewpoints in branches with positive $Gain()$ are added as vertices to the global graph. When adding a new vertex v_{new} to $\mathcal{G}$, an edge between v_{new} and the closest existing vertex is added as well. In addition, if an existing vertex v meets the following two conditions, an edge between it and the new vertex will also be added.

$$\begin{cases} D_E(v, v_{new}) < \delta \\ D_{\mathcal{G}}(v, v_{new}) / D_E(v, v_{new}) > \gamma \end{cases} \tag{6}$$

where D_E is the euclidean distance and $D_{\mathcal{G}}$ is the closest distance along the graph. δ and γ are two parameters to restrict the euclidean distance and the ratio between the two distances. Eq. (6) ensures that the graph is not too dense while providing short paths between vertices. Further, to ensure that all edges in the graph are collision-free, we trim off edges that are in collision due to dynamic or previously overlooked obstacles. The graph is then adjusted after the pruning to ensure connectivity. Note that vertices in the graph only includes position information, without considering the $VoxelGain$.

Global frontiers $\mathcal{F}_G$ are composed of local frontiers that are left out previously. Note that they can be observed by at least one viewpoint in the global graph. Every time the local frontiers are updated, they are added to $\mathcal{F}_G$. Meanwhile, all frontiers in $\mathcal{F}_G$ are rechecked and removed if they are cleared.

The detailed process of the relocation stage is presented in Algorithm 3. Define $\mathcal{F}_i$ as the ith frontier in $\mathcal{F}_G$, $\mathcal{F}_{GS}$ as the selected global frontier to be observed and v_S as the vertex that is selected as the goal. Taking Fig. 3 as an example. First, the planner searches for a vertex that is able to observe any global frontier in $\mathcal{G}$ determined by ray tracing. For a vertex v_i in $\mathcal{G}$, the larger the value of i is, the later it appears and the closer it is to the robot position. The same rule also applies to the global frontiers. Thus, furthest global frontiers are selected first, making the final selection close to the current position. As in Fig. 3, point B is the vertex that can observe the selected frontier $\mathcal{F}_{GS}$. This process corresponds to lines 4 to 14 in Algorithm 3. Then the method searches the graph from the end again to find the closest vertex that can observe $\mathcal{F}_{GS}$, such as point A, which corresponds to

Algorithm 3: Relocation

1 Update $\mathcal{F}_{global}$ and $\mathcal{G}$
2 $Flag \leftarrow False$ and $Dist \leftarrow 0$
3 **for** *i from N to 1* **do**
4 **for** *j from M to 1* **do**
5 **if** $\mathcal{F}_i$ *in FOV*(v_i) **then**
6 $v_S \leftarrow v_i$, $\mathcal{F}_{GS} \leftarrow \mathcal{F}_i$
7 $Dist \leftarrow$ Dist($\mathcal{F}_i$, v_i), $Flag \leftarrow True$
8 break;
9 **end**
10 **if** *Flag is True* **then**
11 break;
12 **end**
13 **end**
14 **end**
15 **if** *Flag is True* **then**
16 **for** *i from N to 1* **do**
17 **if** $\mathcal{F}_{GS}$ *in FOV*(v_i) **then**
18 **if** *Dist*($\mathcal{F}_{GS}$, v_i) < $Dist$ **then**
19 $v_S \leftarrow v_i$
20 **end**
21 **end**
22 **end**
23 **else**
24 Exploration Complete.
25 **end**

lines 16 to 23. After this process, the final target viewpoint v_S is obtained. The robot then moves to v_S along the graph and enters exploration mode again. If no vertex is found in the first step, the exploration is completed, line 24. With global frontiers, this method guarantees all traversable areas are covered. In addition, there is no need to compute and update $VoxelGain$ of each viewpoint in the global graph, which saves much computation.

IV. Benchmark Environment

The environment serves as a platform for benchmarking exploration algorithms. It is also meant for leveraging system development and robot deployment for ground-based

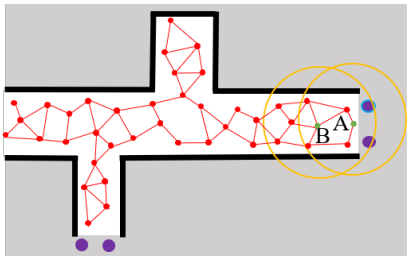
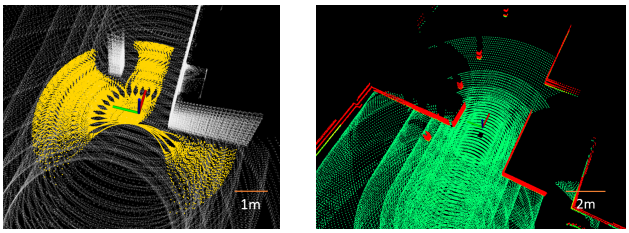


Fig. 3: Relocation stage. The purple dots are global frontiers and the purple dot with blue edge is the selected global frontier $\mathcal{F}_{GS}$.The red dots and red lines have the same definitions as Fig. 1. The green viewpoint B is the first one that could observe $\mathcal{F}_{GS}$ if searching from the end of the global graph. The green viewpoint A is the best one to observe $\mathcal{F}_{GS}$.The yellow circles denote the sensor ranges of viewpoint A and B.



(a) (b)

Fig. 4: (a) Collision avoidance. The yellow dots indicate collision-free paths. (b) Terrain map (40m x 40m). The green points are traversable and the red points are non-traversable.

autonomous navigation. The environment contains a variety of simulation environments, fundamental navigation modules such as collision avoidance, terrain traversability analysis, waypoint following, and a set of visualization tools. Table I lists the characteristics of the simulation environments.

- *Campus* (340m × 340m): A large-scale environment as part of the Carnegie Mellon University campus, containing undulating terrains and convoluted layout.
- *Indoor* (130m × 100m): Consists of long and narrow corridors connected with lobby areas. Obstacles such as tables and columns are present.
- *Garage* (140m × 130m, 5 floors): An environment with multiple floors and sloped terrains to test autonomous navigation in a 3D environment.
- *Tunnel* (330m × 250m): A large-scale environment containing tunnels that form a network, provided by Tung Dang at University of Nevada, Reno.
- *Forest* (150m × 150m): Containing mostly trees and a couple of houses in a cluttered setting.

The collision avoidance module [23] makes sure the vehicle navigates safely. It computes collision-free paths in the vicinity of the vehicle and guides the vehicle to navigate between obstacles. The collision avoidance module utilizes pre-generated motion primitives. When the vehicle navigates, it in real-time determines the motion primitives occluded by obstacles. The resulting collision-free paths, as shown in Fig. 4a, are for vehicle navigation. The collision avoidance module takes as input the terrain maps from the terrain analysis module to determine terrain traversability.

The terrain analysis module analyzes the local smoothness of the terrain and associates a cost to each point on the terrain map. This uses a voxel representation and checks the distribution of the points in adjacent voxels. Advanced functionalities such as handling negative obstacles are optional. Fig. 4b gives an example terrain map covering a 40m x 40m area with the vehicle in the center. The green points are traversable and the red points are non-traversable.

TABLE I: Simulation environment characteristics

	Large Scale	Convoluted	Multi Storage	Undulating Terrain	Cluttered Obstacles	Thin Structure
Campus	X	X		X		
Indoor		X			X	X
Garage			X	X		
Tunnel	X	X				
Forest					X	

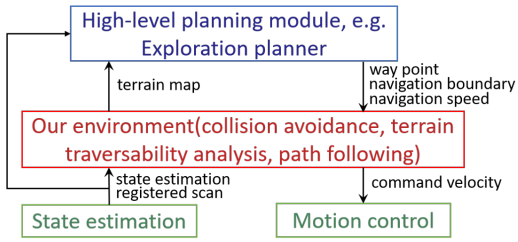


Fig. 5: System integration diagram

The visualization tools display the overall map and explored areas during the course of the exploration. Exploration metrics such as explored volume, traveling distance, and algorithm runtime are plotted and logged to inspect the performance. The environment is constructed with facilitating development of autonomous navigation systems in mind. When integrated on a real robot, it takes the role as the middle layer in the autonomous navigation systems, as illustrated in Fig. 5. Further, the environment supports using a joystick controller to interfere with the navigation, easing the process of system debugging. Detailed information is available on the project website (link provided on first page).

V. Experiments

A. Evaluation in Benchmark Environment

We conduct simulation experiments with the benchmark environment in three environments, i.e. indoor, campus and garage. The vehicle navigates at 2 m/s. Our method sets the exploration planning horizon $\mathcal{H}$ to a 30m×30m area and the frontier boundary $\mathcal{B}$ to a 40m×40m area. The resolution of the octomap is set to 0.35m. We compare with three state-of-art methods in the experiments, all using open-source code adapted to the evaluation environments.

- *NBVP* [2]: A method using RRT to span the space. It finds the most informative branch in the RRT as the path to the next viewpoint.
- *GBP* [13]: An extension of NBVP where the method builds a global RRG through the traversable space and searches the RRG for routes to relocate the vehicle.
- *MBP* [12]: An extension of GBP where the method builds the local RRT using motion primitives. The resulting paths are smoother and span in constrained directions.

Each method is run 10 times. A run is ended if the exploration algorithm reports completion, the vehicle almost stops moving (less than 10m of movement within 300s), or a time limit is reached. Here, the time limit is set to twice of the longest run of our method. Among the evaluated methods, only our method reports completion. In the following results, the trajectories are the best of the 10 runs and the evaluation metrics (explored volume, traveling distance, and algorithm runtime) are the average of the 10 runs. The algorithm runtime is evaluated based on a 4.1 GHz i7 CPU. All algorithms use a single CPU thread.

Fig. 6 shows the results for the indoor environment. in which Fig. 6a includes the best trajectory of each method.

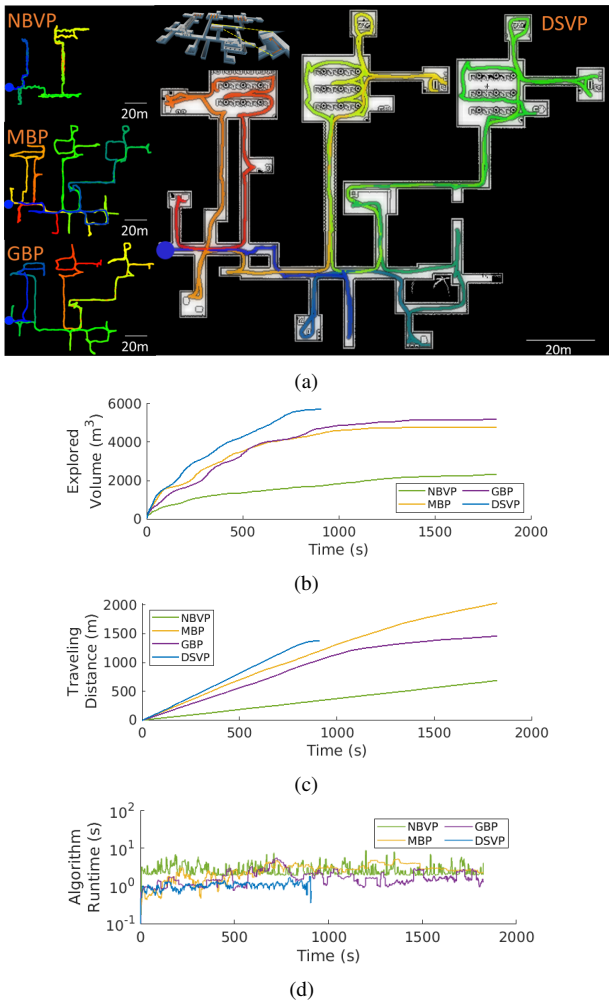


Fig. 6: Simulation results of the indoor environment.(a) shows the resulting map of our method and trajectories of all methods. The blue dot indicates the start point of all trajectories. (b) is the average explored volumes vs. time.(c) is the average traveling distances vs. time. (d) is the average runtime vs. time.

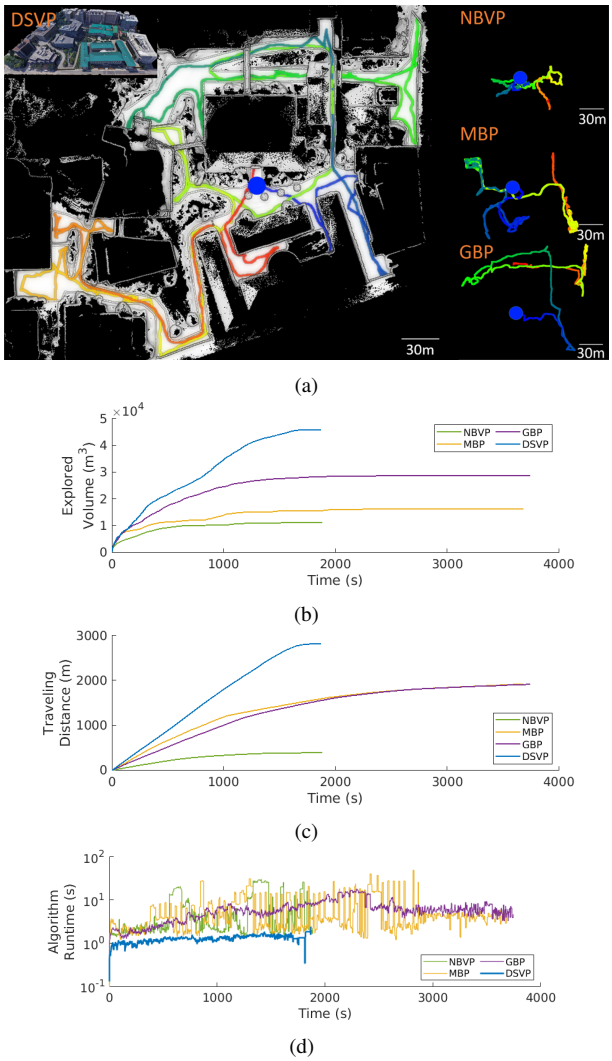


Fig. 7: Simulation results of the campus environment. The figure shares the same layout as Fig.6.

As can be seen, GBP and MBP are both capable of exploring the entire area while NBVP can only cover a limited area because it has limitations in relocating the vehicle. Our method covers the complete space. Fig. 6b and 6c compare the average explored volume and traveling distance of all methods. Our method completes the exploration after traveling 1384m over 912s. It can be seen that GBP can not cover the whole space every time, causing the average volume much less than our methods. GBP and MBP are often trapped in dead end areas. Our method can cover the space fully in all evaluated runs.

Fig. 7 and 8 demonstrate the results for campus and garage environments. From the best trajectories of NBVP, GBP and MBP in Fig. 7a and Fig. 8a, we can see that these methods are unable to cover the entire environment. For GBP and MBP, we observe that they have difficulty in triggering the relocation mode when the environment is open. After traveling 2828m in 1872s in the campus environment and 5347m in 3652s in the garage environment, our method finishes the exploration.

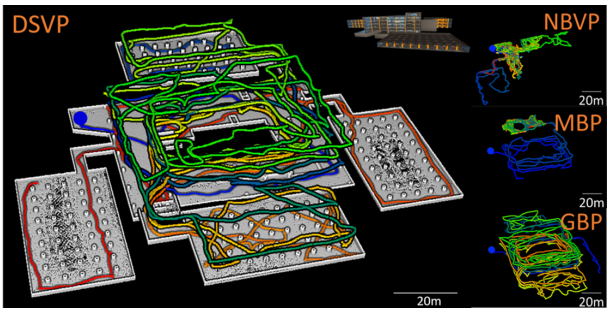
Tables II and III compares the exploration efficiency and algorithm runtime between our method and the other methods. In Table II, $\epsilon(m^3/s)$ is the average value of the efficiency of all runs. The efficiency of one run is defined as the average explored volume per second in that run. r_ϵ is the relative efficiency compared to our method. The average runtime of our method is the smallest in all evaluated environments. With dynamically-expanded RRT, our method does not need to regenerate a new dense local tree every

TABLE II: Comparison of exploration efficiency

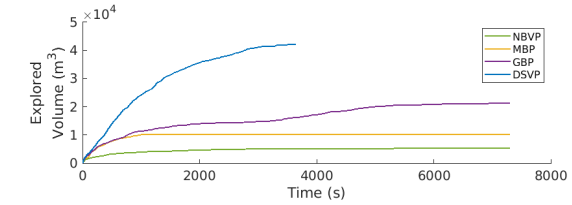
	NBVP		MBP		GBP		DSVP	
	ϵ	r_ϵ	ϵ	r_ϵ	ϵ	r_ϵ	ϵ	r_ϵ
Indoor	1.2	0.18	3.3	0.49	3.6	0.53	**6.8**	**1.0**
Campus	11.1	0.39	11.7	0.41	12.1	0.42	**28.8**	**1.0**
Garage	0.9	0.06	2.8	0.2	6.0	0.43	**14.0**	**1.0**

TABLE III: Comparison of planning time

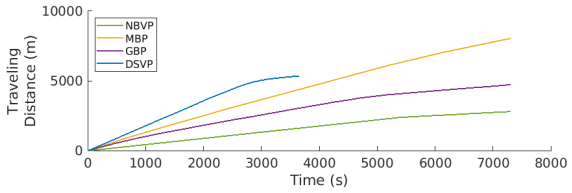
	Average Runtime (s)				CPU Load (%)			
	NBVP	MBP	GBP	DSVP	NBVP	MBP	GBP	DSVP
Indoor	2.48	1.97	1.46	**0.81**	47	27	26	**19**
Campus	2.41	3.31	3.08	**1.03**	37	47	33	**19**
Garage	1.84	1.71	2.33	**0.72**	40	22	44	**17**



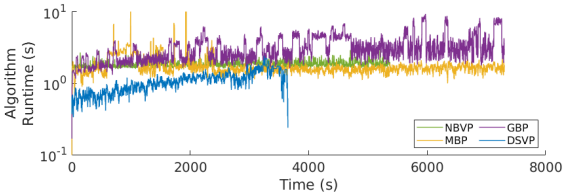
(a)



(b)



(c)



(d)

Fig. 8: Simulation results of the garage environment. The figure shares the same layout as Fig.6.

time, which saves much time especially when the space is open. Further, our method leverages the global frontiers with the global graph to eliminate the need of a dense global graph - the method uses a relatively sparse global graph to navigate to the vicinity of the global frontiers and then uses the global frontiers to guide the vehicle further. While for GBP and MBP, they incrementally add more nodes to the global RRG by randomly sampling viewpoints in the relocation mode, which leads to a dense global RRG. In addition, GBP and MBP need to compute the reward of each viewpoint in the global RRG continuously to decide which one has the highest reward, which takes considerable computation time. Our method, however, neglects the reward of the viewpoints in the global graph and only checks if a viewpoint can observe a given frontier, as described in Algorithm 3, which takes considerably less time.

B. Physical Experiment

We conduct experiments using the vehicle platform in Fig. 9. The vehicle is equipped with a Velodyne Puck lidar, a camera at 640×360 resolution, and a MEMS-based IMU. The system uses our prior method for state estimation as well as mapping explored areas [24]. The system also incorporates navigation modules from our benchmark environment, e.g. collision avoidance, terrain traversability analysis, way-point following, as the mid-layer. During exploration, the collision avoidance module [23] further prevents collisions and warrants safety. Our exploration algorithm is at the top layer in the system, running on a computer with a 4.1GHz i7 CPU.

Fig. 9: Experiment vehicle platform

The experiment is conducted in an outdoor environment at the university campus as shown in Fig. 10. The environment includes several intersections, dead ends and trees. Fig. 10a gives the final trajectories of all methods. The trajectory of NBVP reveals considerable back-and-forth behaviors through the whole process. One issue of MBP and GBP is that they have trouble handling thin structures such as tree branches. The reason is relevant to what is mentioned above that they extend the global RRG randomly in the relocation mode. Due to the fact that the sampled viewpoints in relocation mode are distant from the robot, lidar scan data can miss the thin structures causing the places to be considered traversable. As the vehicle navigates closer, the sampled viewpoints are not effectively eliminated from the global graph, causing the vehicle to be trapped around trees. In contrast, our method actively eliminates edges on the local RRT and global graph that interfere with obstacles. This gives our method the advantage of dealing with thin structures in the environment from which laser returns are inconsistent. Fig. 10b and Fig. 10c compare the explored volume and traveling distance of four methods. NBVP spends 2160s covering 8677m^3 while our method spends 322s covering the same space. GBP spends 1984s covering 15192m^3 which only takes 843s for our method. MBP explores almost the same space as our method while the time is almost double. Fig. 10d shows the runtime of each method. The average runtime for NBVP is 2.02s, for GBP is 3.9s and for MBP is 1.05s. The average runtime for our method is 1.4s. Even though our runtime is slightly longer than MBP, the variation is much smaller as one can see a large spike in the runtime of MBP.

VI. Conclusion

We propose a method for efficiently exploring environments at a high-degree of convolution. By switching between exploration stage and relocation state, our method is able to cover the entire environment. The method dynamically expand the local RRT and global graph, and use hybrid frontiers to guide the expansion. We evaluate the method in a real

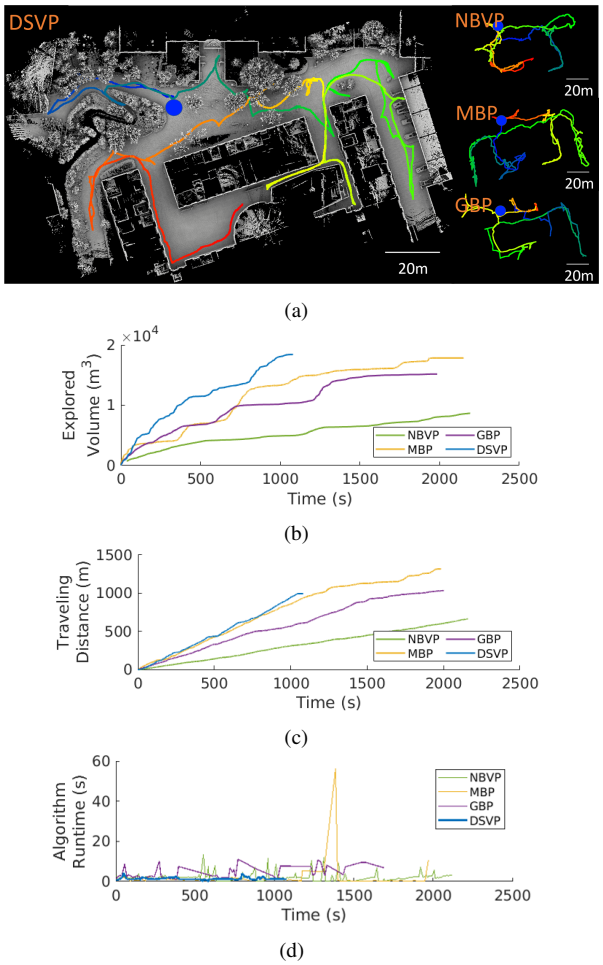


Fig. 10: Results of the experiment in an outdoor environment. (a) is the resulting map of our method and trajectories of all methods. The blue dot indicates the start point of all trajectories. (b) is the explored volumes vs. time.(e) is the traveling distances vs. time. (f) is the runtime vs. time.

outdoor environment and three simulation environments, i.e. indoor, campus and garage environments in the benchmark environment that we develop to facilitate development of autonomous navigation systems. Our method is compared to three state-of-art methods. The results show that our method covers the space twice as fast as the other methods while taking less computation.

References

[1] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[2] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon" next-best-view" planner for 3D exploration," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016.

[3] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997, pp. 146–151.

[4] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the efficiency of frontier-based exploration strategies," in *41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK)*, 2010, pp. 1–8.

[5] Z. Meng, H. Qin, Z. Chen, X. Chen, H. Sun, F. Lin, and M. H. Ang, "A two-stage optimized next-view planning framework for 3-d unknown environment exploration, and structural reconstruction," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1680–1687, 2017.

[6] B. Fang, J. Ding, and Z. Wang, "Autonomous robotic exploration based on frontier point optimization and multistep path planning," *IEEE Access*, vol. 7, pp. 46 104–46 113, 2019.

[7] C. Dornhege and A. Kleiner, "A frontier-void-based approach for autonomous exploration in 3D," *Advanced Robotics*, vol. 27, no. 6, pp. 459–468, 2013.

[8] L. Heng, A. Gotovos, A. Krause, and M. Pollefeys, "Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015.

[9] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canda, Sept. 2017.

[10] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-d environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[11] C. Wang, H. Ma, W. Chen, L. Liu, and M. Q.-H. Meng, "Efficient autonomous exploration with incrementally built topological map in 3-d environments," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 12, pp. 9853–9865, 2020.

[12] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based agile exploration path planning for aerial robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020.

[13] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[14] R. Reinhart, T. Dang, E. Hand, C. Papachristos, and K. Alexis, "Learning-based path planning for autonomous exploration of subterranean environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020.

[15] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," in *Proceedings of Seventh International Conference on Learning Representations (ICLR)*, New Orleans, LA, May 2019.

[16] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[17] T. Kollar and N. Roy, "Trajectory optimization using reinforcement learning for map exploration," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 175–196, 2008.

[18] M. Kulich, J. Faigl, and L. Přeučil, "On distance utility in the exploration task," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[19] M. Kulich, J. Kubalík, and L. Přeučil, "An integrated approach to goal selection in mobile robot exploration," *Sensors*, vol. 19, no. 6, 2019.

[20] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[21] 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.

[22] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proceedings of the 2001 SIAM international conference on data mining*. SIAM, 2001, pp. 1–11.

[23] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1300–1313, 2020.

[24] J. Zhang and S. Singh, "Laser-visual-inertial odometry and mapping with high robustness and low drift," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1242–1264, 2018.