2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Exploring Large and Complex Environments Fast and Efficiently

Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang

Abstract— **This paper describes a novel framework for autonomous exploration in large and complex environments. We show that the framework is efficient as a result of its hierarchical structure, where at one level it maintains a sparse representation of the environment and at another level, a dense representation is used within a local planning horizon around the robot. The exploration path is computed at the two levels, coarsely at the global scale and finely around the robot. Such a framework produces detailed paths in the vicinity of the robot, while trades off data resolution far away from the robot for computational efficiency. In experiments, we evaluate our method with a real robot exploring large and complex indoor and outdoor environments. Results show that our method is twice as efficient in covering spaces while using less than one-fifth of processing in comparison to state-of-the-art methods.**

I. Introduction

We consider the problem of covering a three-dimensional space unknown *a priori* with an autonomous robot. The term coverage has been used in a variety of contexts, but in this paper, we define it to be passing the "footprint" of a sensor or detector over all points in a target space. This footprint is sometimes called "field of view". Coverage requires constantly maintaining a representation of the environment to keep track of areas covered by the robot. As the robot explores, it searches for feasible paths to traverse the environment and complete the coverage. In cases where the environment is large-scale, structurally complex, and three-dimensional, the problem becomes computationally complex, and ensuring complete coverage can become a challenge.

Existing work [1]–[4] greedily maximizes the marginal reward, i.e., the added benefit of subsequent movement by optimizing the next one or a few instant steps. The resulting path is often inefficient, producing back-and-forth motion of the robot and visiting the same areas repetitively. Further, excessive computation is another issue limiting the exploration efficiency. The robot spends a considerable amount of time waiting for onboard processing to finish, instead of exploring continuously, resulting in "stop-and-go" motions.

The method described in this paper aims at highly efficient exploration for large, complex environments. The method utilizes a hierarchical data structure to achieve fast computation – one level in the data structure maintains data sparsely at the global scale, while another level maintains data densely within a local planning horizon around the robot. The exploration path is computed at the two levels, coarsely at the global scale and finely around the robot, and then joined together. We leverage the key insight that only detailed paths are necessary in the vicinity of the robot, while coarse paths provide sufficient utility far away from the robot. The method maintains low-resolution data at the global scale for computational efficiency. Further, the method seeks the shortest exploration path overall instead of maximizing the instant reward, which results in more efficient paths without repetitive visiting to the same areas.

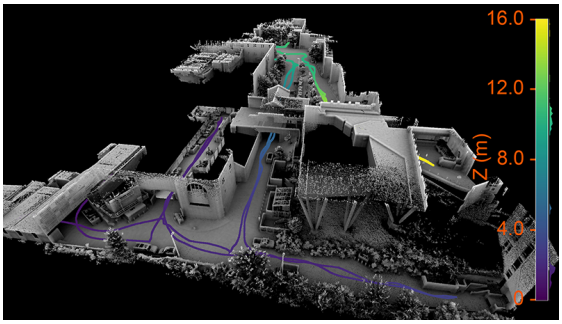


Fig. 1. A representative result produced by our method in a complex, 3D environment including indoor and outdoor scenes. The vehicle autonomously explores over 1.4km in 20 minutes. The trajectory spans 16.0m in elevation, which is color-coded by height in the figure. More experiment details are in Section V.

In experiments, we evaluate the method on a real robot in various indoor and outdoor environments. We compare to state-of-the-art methods in a cluttered indoor environment, a large indoor environment where the robot travels close to a kilometer. Comparison shows that our method covers spaces twice as fast as the start-of-the-art and at the same time using runtime less than one-fifth of the start-of-the-art. Our simulation environment and source-code are publicly available[1] and experiment results are in a video[2].

II. Related Work

Many methods to autonomous exploration formulate the problem using frontiers, i.e., the boundary between mapped and unmapped areas [5]–[9]. The vehicle moves toward the boundary and therefore extends the mapped areas. Other methods model the environment based on information theory [10]–[13] and topological representations [14]–[16]. Machine learning has also been used to determine the exploration strategy [17]–[19]. The aforementioned work can be extended to multi-robot scenarios [20]–[24]. Most existing

All authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh PA. Emails: {ccao1, hongbiaz, choset, zhangji}@cmu.edu

[1]Sim. environment and source-code: www.cmu-exploration.com
[2]Experiment video: https://youtu.be/pIo64S-uOoI

978-1-7281-9077-8/21/$31.00 ©2021 IEEE

methods rely on greedy strategies for exploration, where the efficiency suffers from being myopic.

Recent work in exploration develops a framework based on the Rapidly-exploring Random Tree (RRT) [25] or Rapidly-exploring Random Graph (RRG) [26]. The framework spans RRTs or RRGs through the environment taking into account the sensor model and vehicle traversability. Specifically, a method named Next-Best-View Planner (NBVP) [1] is considered the state-of-the-art which models the nodes on the RRT as viewpoints. The method greedily selects the branch with the maximum collective reward from the associated viewpoints. Witting et al. [2] extend the method by seeding the RRT with the vehicle's past trajectory, which allows for further exploration in the areas previously passed by the vehicle. However, using solely the past trajectory can result in suboptimality of the exploration. A Graph-Based Path Planning (GBP) method [3] improves the scheme by constructing a global roadmap along the past trajectory. The method searches the roadmap for routes to the areas that need to be explored further. Heuristics are used to explicitly switch between local exploration and global relocation. Recently, a variant of [3] integrates motion primitives with the plan for local smoothness of the exploration path [4]. In essence, due to the randomness of RRT and RRG, these methods are prone to overlooking areas that are not completely explored.

The main contribution of our work is an efficient exploration framework that optimizes the full exploration path. In the framework, planning at the global level guides the planning at the local level. When the exploration completes inside the local planning horizon, the method implicitly relocates to a different area for further exploration. In comparison to [2]–[4], our method does not involve heuristics for mode switching. We compare our method to NBVP [1] and GBP [3] and show that our method significantly outperforms both methods in exploring large, complex environments.

III. Problem Definition

Define $\mathcal{Q} \subset \mathbb{R}^3$ as the workspace to be explored. Let $\mathcal{Q}_{\text{trav}} \subset \mathcal{Q}$ be the traversable subspace. We define a viewpoint $\mathbf{v} = [\mathbf{p_v}, \mathbf{q_v}] \in \mathbf{SE(3)}$ to describe the placement of the onboard sensor, where $\mathbf{p_v} \in \mathcal{Q}_{\text{trav}}$ denotes the sensor position and $\mathbf{q_v} \in \mathbf{SO(3)}$ denotes the orientation. We define "surface" to be the generalized boundary between free space and non-free space, the latter includes occupied and unknown spaces. Let $\mathcal{S_v} \subset \mathcal{Q}$ be the surfaces that are perceived by the sensor located at $\mathbf{v}$. Let $\mathcal{L} \subset \mathbf{SE(3)}$ be a set of viewpoints along the vehicle's past trajectory. The perceived surfaces so far are

$$\mathcal{S} = \bigcup_{\mathbf{v} \in \mathcal{L}} \mathcal{S_v}. \tag{1}$$

Let $\mathcal{S}_{\text{cov}} \subset \mathcal{S}$ be the covered surfaces so far, and let $\bar{\mathcal{S}} = \mathcal{S} \setminus \mathcal{S}_{\text{cov}}$ be the yet uncovered surfaces. We would like to find the shortest path $\mathcal{T}^*$ that passes through a sequence of viewpoints, which completely cover $\bar{\mathcal{S}}$ when followed by a robot with a sensor. Let $\mathbf{v}_{\text{current}}$ be the viewpoint located at the vehicle's current sensor pose. Our exploration problem can be defined as follows,

Problem 1: Given $\bar{\mathcal{S}}$ and $\mathbf{v}_{\text{current}}$, find the shortest path $\mathcal{T}^*$ that passes through $\mathbf{v}_1, \mathbf{v}_2, ...$, which when followed by the robot covers $\bar{\mathcal{S}}$, and $\mathbf{v}_{\text{current}} \in \mathcal{T}^*$.

Problem 1 is solved at each step as the vehicle explores, i.e. the vehicle finds the shortest path at every instant time of replanning to incorporate the latest sensor readings.

IV. Methodology

A. Local Planning

We begin with defining the criteria for a surface point to be covered. Given a surface patch centered at $\mathbf{p}_s \in \mathbb{R}^3$ with normal $\mathbf{n}_s \in \mathbb{R}^3$, the center point is covered by a viewpoint $\mathbf{v}$ if the following two conditions are met,

$$|\mathbf{p}_s - \mathbf{p_v}| \leq D, \tag{2}$$

$$\mathbf{n}_s \cdot (\mathbf{p_v} - \mathbf{p}_s)/|\mathbf{n}_s||\mathbf{p_v} - \mathbf{p}_s| \geq T, \tag{3}$$

where D and T are two constants. The criteria limits the relative distance and orientation of the surface patch. Ideally, we would like the surfaces to be perceived by the sensor perpendicularly from a relatively short distance. Meeting the criteria makes sure the surfaces are perceived well.

The local planning problem solves for a path $\mathcal{T}_{\text{local}}$ within the local planning horizon around the vehicle. Define $\mathcal{H} \subset \mathcal{Q}$ as the local planning horizon as shown in Fig. 2. Let $\mathcal{H}_{\text{trav}} \subset \mathcal{H}$ be the traversable subspace and $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$ be the corresponding C-space considering rotation and translation. Recall that $\bar{\mathcal{S}}$ denotes the uncovered surfaces. We define $\bar{\mathcal{S}}_{\mathcal{H}} \subset \bar{\mathcal{S}}$ as the uncovered surfaces to be perceived from viewpoints in $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$. The problem of computing $\mathcal{T}_{\text{local}}$ can be stated as follows,

Problem 2: Given $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, find the shortest path $\mathcal{T}^*_{\text{local}} = \mathbf{v}_1, \mathbf{v}_2, ...$ where $\mathbf{v}_i \in \mathcal{C}^{\mathcal{H}}_{\text{trav}},\ i = 1, 2, ...$ to cover $\bar{\mathcal{S}}_{\mathcal{H}}$.

Problem 2 is solved in an iterative random-sampling process. At each iteration, we randomly sample a set of viewpoints that completely covers $\bar{\mathcal{S}}_{\mathcal{H}}$, then we form a path using the viewpoints. Through the iterations, paths with shorter lengths are found and the path that has the shortest length is kept. We use $\bar{\mathcal{S}}_{\mathbf{v}} \subset \bar{\mathcal{S}}_{\mathcal{H}}$ to denote the uncovered

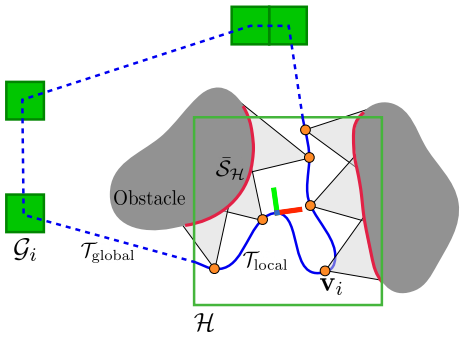


Fig. 2. Illustration of our method. The coordinate frame represents the vehicle. The green box represents the local planning horizon $\mathcal{H}$. The solid green squares represent the exploring subspaces $\mathcal{G}_i$, $i \in \mathbb{Z}^+$. The solid blue curve is the local path $\mathcal{T}_{\text{local}}$. The dashed blue line is the global path $\mathcal{T}_{\text{global}}$. The orange dotes on $\mathcal{T}_{\text{local}}$ are the viewpoints $\mathbf{v}_i$, $i \in \mathbb{Z}^+$. The red curves are the uncovered surfaces $\bar{\mathcal{S}}_{\mathcal{H}}$ to be perceived by the viewpoints. The method uses an iterative random-sampling process in determining the viewpoints to cover $\bar{\mathcal{S}}_{\mathcal{H}}$.

surfaces to be perceived from $\mathbf{v} \in \mathcal{C}^{\mathcal{H}}_{\rm trav}$. The reward of $\mathbf{v}$ is defined as the area of $\bar{\mathcal{S}}_{\mathbf{v}}$, denoted as $A_{\mathbf{v}}$. It is worth mentioning that the problem exhibits submodularity [27], i.e. the more viewpoints selected, the less reward from an additional viewpoint being selected. This is because of the field-of-view overlaps among the viewpoints where the same surface can be perceived from multiple viewpoints. As a result, the reward of a viewpoint is dependent on the viewpoints visited earlier on the path. Let $\mathbf{v}_i$, $i \in \mathbb{Z}^+$, be the i-th viewpoint on $\mathcal{T}_{\rm local}$. The uncovered surfaces to be perceived from $\mathbf{v}_i$, $\bar{\mathcal{S}}_{\mathbf{v}_i}$, needs to be adjusted to $\bar{\mathcal{S}}_{\mathbf{v}_i} - \bigcup_{j=1}^{i-1}(\bar{\mathcal{S}}_{\mathbf{v}_i} \cap \bar{\mathcal{S}}_{\mathbf{v}_j})$, and $A_{\mathbf{v}_i}$ is adjusted accordingly.

Algorithm 1 solves Problem 2. The algorithm begins with uniformly generating a set of viewpoint candidates $\mathcal{V}$ in $\mathcal{C}^{\mathcal{H}}_{\rm trav}$ (line 1). Then, it computes the rewards $A_{\mathbf{v}}$ for all viewpoint candidates $\mathbf{v} \in \mathcal{V}$ by estimating their coverages $\bar{\mathcal{S}}_{\mathbf{v}}$ (line 3). To compute $\mathcal{T}_{\rm local}$, the following two steps are repeated for a number of $K \in \mathbb{Z}^+$ iterations. In the first step, the algorithm randomly samples a min-cardinality subset of viewpoints from $\mathcal{V}$ that covers $\bar{\mathcal{S}}_{\mathcal{H}}$ (line 6-14). This uses a priority queue $\mathcal{Q}$ to manage the candidate viewpoints. The priority of a viewpoint $\mathbf{v}$ is set to its reward $A_{\mathbf{v}}$. Viewpoints are selected from the priority queue with probabilities proportional to their rewards (line 8). Due to the submodularity as we have discussed, the rewards of the remaining viewpoints in the priority queue are reduced properly after a viewpoint is selected to account for the effect of field-of-view overlaps between the viewpoints (line 11-13). The viewpoint sampling process finishes when the priority queue is empty or the marginal reward of adding a new viewpoint is smaller than a threshold $A_{\rm min}$.

In the second step, the algorithm computes a path using the sampled viewpoints. Here, two viewpoints $\mathbf{v}_{\rm start}$ and $\mathbf{v}_{\rm end}$ are set as the start and end viewpoints of $\mathcal{T}_{\rm local}$ (line 15). This is by choosing the closest viewpoints to the two adjacent nodes on the global path $\mathcal{T}_{\rm global}$ (more details in section IV-B). Then, the algorithm finds the shortest collision-free path between every viewpoint pair in the set of sampled viewpoints $\mathcal{V}_i$ and construct a distance matrix $\mathbf{D}_i$ containing the length of the paths (line 16). This uses the A* algorithm [28] to search for the paths in $\mathcal{H}_{\rm trav}$. Next, the algorithm solves a Traveling Salesman Problem (TSP) [29] to produce a path that starts at $\mathbf{v}_{\rm start}$ and ends at $\mathbf{v}_{\rm end}$ (line 17). In the end, upon completion of the iterations, the algorithm returns the shortest path found as $\mathcal{T}_{\rm local}$ (line 22).

Let n be the number of viewpoint candidates generated on line 1 in Algorithm 1. At each iteration, the number of sampled viewpoints is no more than n. After sampling each viewpoint, adjusting the rewards of the viewpoints in the priority queue takes $O(n)$ time (lines 11-13). We model the time of finding the shortest path between two viewpoints as bounded by a constant. For all viewpoint pairs, the time complexity is $O(n^2)$. The TSP is solved using the Lin–Kernighan heuristic which consumes $O(n^{2.2})$ time [29]. The process is over a constant K iterations. Therefore, the time complexity of Algorithm 1 is $O(n^{2.2})$.

Algorithm 1: ComputeLocalPath

input : traversable C-space $\mathcal{C}^{\mathcal{H}}_{\rm trav}$, uncovered surfaces $\bar{\mathcal{S}}_{\mathcal{H}}$, global path $\mathcal{T}_{\rm global}$, current viewpoint $\mathbf{v}_{\rm c}$
output: local path $\mathcal{T}_{\rm local}$

1 $\mathcal{V} \leftarrow$ GenerateViewpointCandidates($\mathcal{C}^{\mathcal{H}}_{\rm trav}$);
2 $Q \leftarrow$ new PriorityQueue();
3 For every $\mathbf{v} \in \mathcal{V}$, estimate its coverage $\bar{\mathcal{S}}_{\mathbf{v}}$, push $\mathbf{v}$ into Q with the priority equal to its reward $A_{\mathbf{v}}$;
4 $\mathcal{T}_{\rm local} \leftarrow \emptyset$, $l_{best} \leftarrow +\infty$;
5 **for** $i := 1$ to K **do**
6 $\mathcal{V}_i \leftarrow \{\mathbf{v}_{\rm c}\}$, $Q_i \leftarrow Q$;
7 **while** $Q_i \neq \emptyset$ **and** Q_i.front().priority $\geq A_{\rm min}$ **do**
8 $\mathbf{v} \leftarrow$ ProbabilisticPick(Q_i);
9 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathbf{v}$;
10 Q_i.erase($\mathbf{v}$);
11 **for** $\mathbf{v}' \in Q_i$ **do**
12 Q_i.UpdatePriority($\mathbf{v}'$);
13 **end**
14 **end**
15 $\mathbf{v}_{\rm start}, \mathbf{v}_{\rm end} \leftarrow$ GetStartAndEndViewpoints($\mathcal{T}_{\rm global}$, $\mathcal{V}_i$);
16 $\mathbf{D}_i \leftarrow$ ComputeShortestPaths($\mathcal{C}^{\mathcal{H}}_{\rm trav}$, $\mathcal{V}_i$);
17 $\mathcal{T}_i \leftarrow$ SolveTSP($\mathbf{D}_i$);
18 **if** Length($\mathcal{T}_i$) $< l_{\rm best}$ **then**
19 $\mathcal{T}_{\rm local} \leftarrow \mathcal{T}_i$, $l_{\rm best} \leftarrow$ Length($\mathcal{T}_i$);
20 **end**
21 **end**
22 **return** $\mathcal{T}_{\rm local}$;

B. Global Planning

The space outside $\mathcal{H}$ is divided into subspaces. Each subspace stores the covered and uncovered surfaces developed during the exploration. In addition, each subspace holds a status from one of three categories: "unexplored", "exploring", and "explored". If a subspace does not contain any covered or uncovered surfaces, the status is "unexplored". If a subspace contains only covered surfaces, the status is "explored". If a subspace contains uncovered surfaces, the status is "exploring". Here, we only consider the exploring subspaces as these are the subspaces used in global planning. Let $m \in \mathbb{Z}^+$ be the number of exploring subspaces. A subspace is denoted as $\mathcal{G}_i \subset \mathcal{Q}$, $i = 1, 2, ..., m$. The set of exploring subspaces is denoted as $\hat{\mathcal{G}} = \{\mathcal{G}_i \mid i = 1, 2, ..., m\}$. We state the global planning problem as follows.

Problem 3: Given $\mathcal{H} \subset Q$ and $\hat{\mathcal{G}} = \{\mathcal{G}_i \mid i = 1, 2, ..., m\}$, find the shortest path $\mathcal{T}^*_{\rm global}$ that goes through the centroids of $\mathcal{H}$ and $\mathcal{G}_i$, $i = 1, 2, ..., m$.

Problem 3 is solved as a TSP. As shown in Algorithm 2, we compute the distances between each two exploring subspaces (line 1). Here, the Euclidean distance between the centroids of the two subspaces is used. However, in complex environments, Euclidean distances can be insufficient to accurately measure the traveling distance. Alternatively, the algorithm can use a roadmap constructed in the traversable

Algorithm 2: ComputeExplorationPath

input : local planning horizon $\mathcal{H}$, traversable C-space $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, uncovered surfaces $\bar{\mathcal{S}}_{\mathcal{H}}$, exploring subspaces $\hat{\mathcal{G}}$, current viewpoint $\mathbf{v}_{\text{c}}$

output: exploration path $\mathcal{T}$

1 $\mathbf{D} \leftarrow$ ComputeDistanceMatrix($\mathcal{H}$, $\hat{\mathcal{G}}$);
2 $\mathcal{T}_{\text{global}} \leftarrow$ SolveTSP($\mathbf{D}$);
3 $\mathcal{T}_{\text{local}} \leftarrow$ ComputeLocalPath($\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, $\bar{\mathcal{S}}_{\mathcal{H}}$, $\mathcal{T}_{\text{global}}$, $\mathbf{v}_{\text{c}}$);
4 $\mathcal{T}$ = ConcatenatePaths($\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, $\mathcal{T}_{\text{local}}$, $\mathcal{T}_{\text{global}}$);
5 **return** $\mathcal{T}$;

space along the past trajectory and search the roadmap for the shortest path between the two exploring subspaces.

Algorithm 2 gives the procedure of computing the exploration path $\mathcal{T}$. The TSP is solved using the Lin–Kernighan heuristic [29] (line 2). The resulting global path $\mathcal{T}_{\text{global}}$ consists of the centroids of $\mathcal{H}$ and all subspaces in $\hat{\mathcal{G}}$. Then, the algorithm calls Algorithm 1 to compute $\mathcal{T}_{\text{local}}$ (line 3). Finally, $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$ are concatenated. This is by finding the shortest collision-free paths from the start point $\mathbf{v}_{\text{start}}$ and end point $\mathbf{v}_{\text{end}}$ on $\mathcal{T}_{\text{local}}$ to the boundary of $\mathcal{H}$ and replacing the centroid of $\mathcal{H}$ on $\mathcal{T}_{\text{global}}$ with the boundary points. Fig. 3 shows an example of the exploration. The solid and dashed blue paths represent $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$, respectively.

If the exploration completes in $\mathcal{H}$ ($\bar{\mathcal{S}}_{\mathcal{H}} = \emptyset$), the processing reduces to the case that the shortest paths connect the vehicle to the boundary of $\mathcal{H}$, which are further connected to the adjacent subspaces along $\mathcal{T}_{\text{global}}$. The vehicle follows the path to transit to an exploring subspace to resume exploration. In other words, the algorithm implicitly transitions between exploration and relocation to another area to explore further. If $\bar{\mathcal{S}}_{\mathcal{H}} = \emptyset$ and $\hat{\mathcal{G}} = \emptyset$, the exploration terminates.

In Algorithm 2, computing the distance matrix takes $O(m^2)$ time and solving the TSP runs in $O(m^{2.2})$ time, where m is the number of exploring subspaces. Recall that Algorithm 1 takes $O(n^{2.2})$ time where n is the maximum number of viewpoints. We model the time of concatenating $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$ as bounded by a constant. Theorem 1 states the time complexity of our algorithm.

Theorem 1: Algorithm 2 runs in $O(n^{2.2} + m^{2.2})$ time.

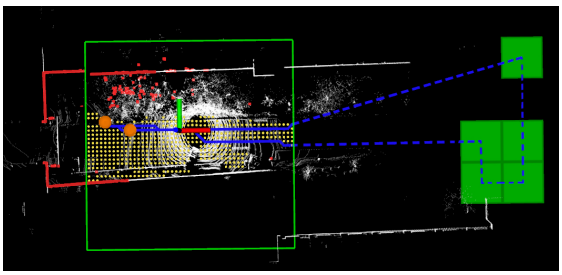


Fig. 3. An example exploration process with real data. The figure uses the same color code as Fig. 2. The white points show lidar scan data, with which the method extracts the uncovered surfaces (red points). The yellow dots are the viewpoint candidates, from which, the method samples the viewpoints (orange dots) to cover the red points.

C. Implementation Details

In our implementation, the configuration space $\mathcal{Q}$ is evenly divided into square blocks. Each subspace $\mathcal{G}_i$ consists of a block. The local planning horizon $\mathcal{H}$ consists of 5×5 blocks with the vehicle in the center block. During exploration, if the vehicle crosses the boundary of a block, a rollover is introduced where a number of 5 blocks behind the vehicle roll out of $\mathcal{H}$ and another 5 blocks in front of the vehicle become inside $\mathcal{H}$. Accordingly, the memory that maintains data in these blocks for local planning is reallocated.

To accelerate processing, we use a piece of dedicated memory for estimating the coverage of each viewpoint. The memory keeps track of the surrounding geometry of the viewpoint candidates. Specifically, the index of an element in the memory denotes a direction seen from the viewpoint candidate, and the value denotes the distance from the viewpoint candidate to the closest object in that direction. At a planning cycle, we take one frame of registered sensor data and compare it to the previous frames to extract the "changes". This is implemented using a voxel grid – if both new and old data points locate in the same voxel, we consider it as no change, otherwise, we consider it as a change. The "changed" data points are used to update the memory. The method then estimates the surfaces perceived from the viewpoint candidates. Using dedicated memory allows us to incrementally update a viewpoint's coverage, which avoids re-computation between planning cycles thus considerably accelerates the process.

V. Experiments

We conduct experiments using the vehicle platform in Fig. 4. 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 [30]. Collision avoidance uses a trajectory library-based method [31]. Our exploration algorithm runs on a laptop computer with a 4.1GHz i7 CPU and uses a single CPU thread for processing.

In our method, the local planning horizon is set as a 40m×40m area around the vehicle. Each subspace is an 8m×8m area. The covered and uncovered surfaces are represented as 3D points at the resolution of 0.2m. The viewpoint candidates are generated at the resolution of 0.5m. The method replans at the frequency of 1Hz and processes lidar scan data stacked in the last second. We compare with two existing methods that are considered state-of-the-art.

Fig. 4. Experiment vehicle platform.

- *NBVP* [1]: A method using a Rapidly-exploring Random Tree (RRT) [25] to span the space. It finds the most informative branch in the RRT as the path to the next viewpoint. We evaluate using open-source code.
- *GBP* [3]: An extension of NBVP where the method builds a roadmap through the traversable space and searches the roadmap for a route to relocate the vehicle. The method explicitly switches between exploration mode and relocation mode. We use open-source code tuned and adapted to the testing environments.

The methods are evaluated in three distinctive environments. Test 1 is conducted in an indoor environment as shown in Fig. 5. The environment consists of cluttered scenes and narrow passways. Fig. 5(a) gives the resulting trajectories of all three methods. Carefully examining the trajectories reveals that NBVP misses some local areas while GBP and our method complete the coverage. Based on our understanding of the method, NBVP is theoretically limited in transiting between separate areas to explore. The issue is mitigated in GBP by using a graph-based search to help the vehicle relocate. Fig. 5(d) and Fig. 5(e) compare the explored volumes and traveling distances. The explored volumes are calculated using the lidar scan data. It takes 119s for our method to explore the environment while NBVP takes 415s and GBP takes 274s, both consume more than twice of our method. Fig. 5(f) shows the runtime. The average runtime for NBVP is 2.6s, for GBP is 1.5s, and for our method is 0.17s. Ours is less than one-fifth of NBVP and GBP.

Test 2 is in a large indoor environment consisting of lobbies and dining areas connected by long corridors as shown in Fig. 6. Note that the map is cleaned up for visualization purpose. The space expands on both sides of the corridors and windows and glass walls add additional difficulty to the environment. NBVP and GBP are not able to cover the space completely leaving a large portion uncovered. NBVP is set to explore the maximum area (80m×80m) without leading to a performance drop. GBP produces endless back-and-forth motion to the end of the run and is stopped. Our method completes the exploration after traveling over 988m in 1167s. Especially, we can see in the close-up view in Fig. 6(a) that the green trajectory (NBVP) is repetitive and inefficient. Fig. 6(e) presents the explored volumes. As the method terminates, NBVP takes 1228s to cover 687m^3 while the same amount of space is covered by our method in 206s. GBP takes 1801s to cover 1388m^3 and that amount of space is covered by our method in 550s. In both cases, our method is twice more efficient. In Fig. 6(f), NBVP uses 1228s to travel over 231m and the same distance is completed by our method in 265s. GBP travels over 958m overall, almost the same as our method. Since the vehicle often needs to navigate through explored areas for relocating to a different area to explore further, we consider the explored volume as a better measure of the efficiency. Fig. 6(g) presents the runtime. The average runtime for NBVP is 2.5s, for GBP is 1.8s, and for our method is 0.13s. Again, our method is more than five times faster than NBVP and GBP in terms of processing.

Test 3 is in a complex environment including both indoor and outdoor scenes over a 3D terrain. The results are shown in Fig. 1 and Fig. 7. Our method completes the exploration after traveling over 1403m in 1217s. The resulting trajectory spans over 16.0m in elevation. In particular, the green box in Fig. 7(a) indicates the area where Test 1 is conducted.

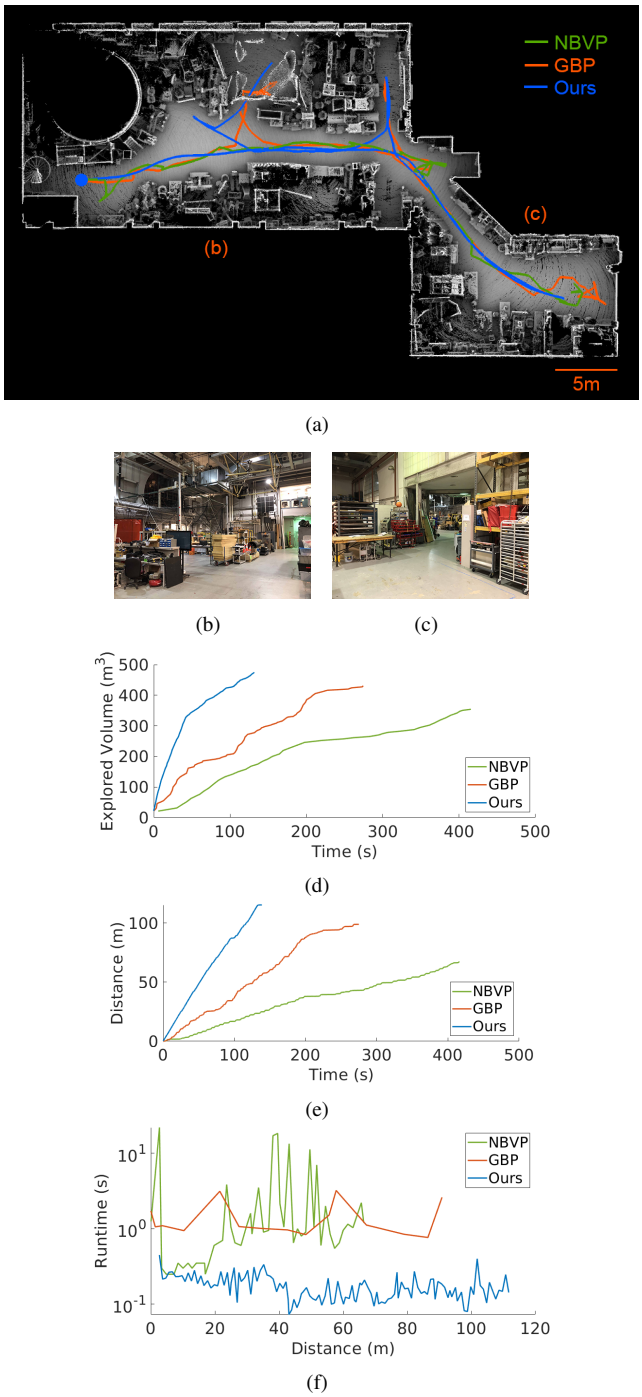
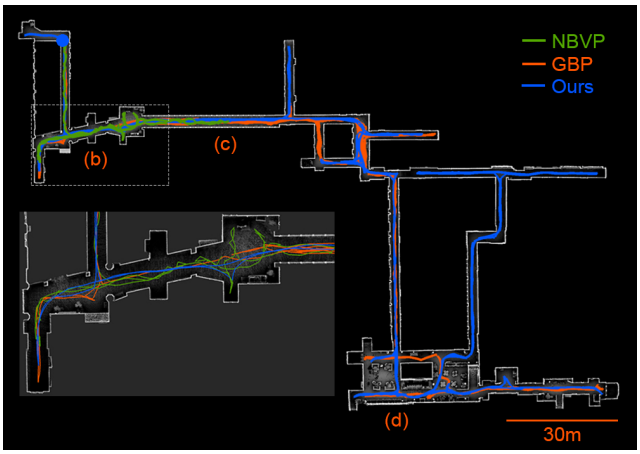
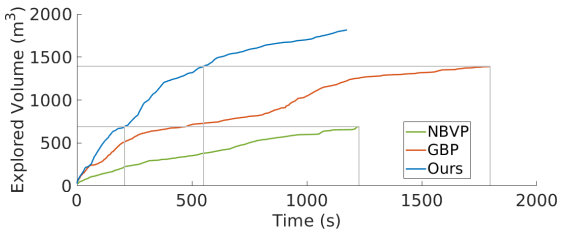


Fig. 5. Results of Test 1 in a cluttered indoor environment. (a) shows the resulting map of our method and the trajectories of NBVP, GBP, and our method overlaid on the map. The blue dot indicates the start point of all three trajectories. (b)-(c) are two photos taken from the locations as labeled in (a). (d) is the explored volumes vs. time. (e) is the traveling distances vs. time. (f) is the runtime vs. traveling distance. In comparison, NBVP and GBP respectively take 415s and 274s to complete the exploration while our method takes 119s. The average runtime for NBVP is 2.6s, for GBP is 1.5s, and for our method is 0.17s.

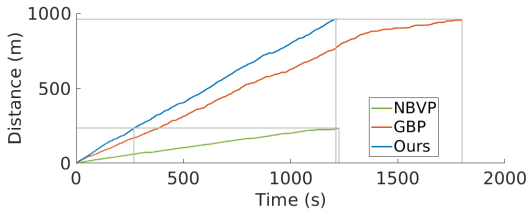


(a)

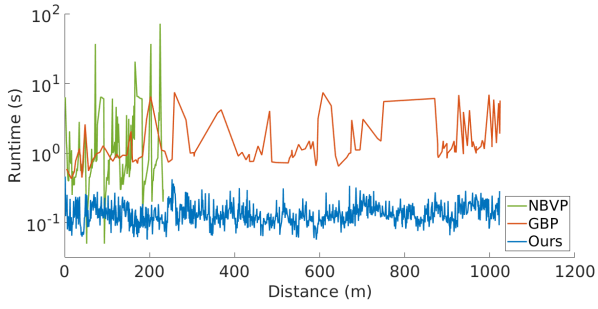
(b) (c) (d)



(e)



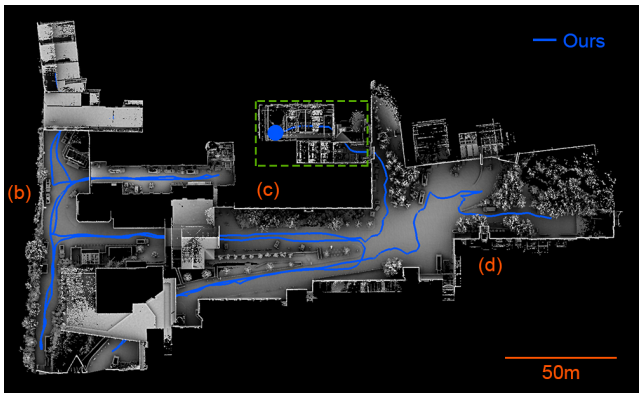
(f)



(g)

Fig. 6. Results of Test 2 in a large indoor environment. The figure shares the same layout as Fig. 5. Neither NBVP nor GBP is able to explore the space completely. Both leave a large portion of the space unexplored. Our method completes after traveling over 988m in 1167s.

Here, we present the results of our method only as the difficulty of this environment is beyond what the other two methods are able to handle. Finally, Table I gives the



(a)

(b) (c) (d)

Fig. 7. Results of Test 3 in a complex environment including indoor and outdoor scenes over a 3D terrain. Our method finishes exploring the environment after traveling over 1403m in 1217s. Fig. 1 shows another view of the resulting map and trajectory. The trajectory spans 16.0m in elevation. The green box in (a) indicates the area of Test 1.

runtime breakdown for our method. The numbers indicate that there is not a significant difference in the overall runtime between indoor and outdoor environments. We see the time on updating representation in local planning dominates the overall runtime. This confirms our insight that using the hierarchical framework to keep the local planning horizon relatively small reduces the computational complexity and keeps the processing fast. The computation at the global scale is lightweight due to usage of sparse data and coarse paths.

VI. Conclusion

We propose a method for autonomous exploration of large, complex environments. The method uses a hierarchical framework to plan a detailed path within the local planning horizon and trade off data density at the global scale for computational efficiency. Further, the method plans a full exploration path rather than finding instant viewpoints to maximize the marginal coverage. The resulting path is efficient without redundant revisiting. We evaluate the method in physical experiments in various indoor and outdoor environments. We compare the results to two state-of-the-art methods and conclude that our method covers spaces more than twice as fast as the state-of-the-art while the average runtime is less than one-fifth of the start-of-the-art.

TABLE I

Runtime Breakdown

Test	Local Planning			Global Planning	Overall
	Update Representation	Sample Viewpoints	Find Path		
Test 1	147.4ms	1.2ms	12.5ms	8.3ms	169.4ms
Test 2	121.6ms	1.3ms	2.5ms	8.6ms	134.0ms
Test 3	106.9ms	5.7ms	45.6ms	9.6ms	167.8ms

REFERENCES

[1] 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.

[2] C. Witting, M. Fehr, R. Bähnemann, H. Oleynikova, and R. Siegwart, "History-aware autonomous exploration in confined environments using mavs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, Oct. 2018.

[3] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "Graph-based path planning for autonomous robotic exploration in subterranean environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, Nov. 2019.

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

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

[6] 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)*, Munich, Germany, 2010.

[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] F. Bourgault, A. A. Makarenko, S. B. Williams, B. Grocholsky, and H. F. Durrant-Whyte, "Information based adaptive robotic exploration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, Oct. 2002.

[11] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters." in *Robotics: Science and Systems*, Cambridge, MA, June 2005.

[12] W. Tabib, M. Corah, N. Michael, and R. Whittaker, "Computationally efficient information-theoretic exploration of pits and caves," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, Oct. 2016.

[13] S. Bai, J. Wang, F. Chen, and B. Englot, "Information-theoretic exploration with bayesian optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, Oct. 2016.

[14] H. Choset, S. Walker, K. Eiamsa-Ard, and J. Burdick, "Sensor-based exploration: Incremental construction of the hierarchical generalized voronoi graph," *The International Journal of Robotics Research*, vol. 19, no. 2, pp. 126–148, 2000.

[15] E. U. Acar and H. Choset, "Sensor-based coverage of unknown environments: Incremental construction of morse decompositions," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002.

[16] S. Kim, S. Bhattacharya, R. Ghrist, and V. Kumar, "Topological exploration of unknown and partially known environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3851–3858.

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

[18] 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.

[19] 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.

[20] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[21] M. S. Couceiro, R. P. Rocha, and N. M. Ferreira, "A novel multi-robot exploration approach based on particle swarm optimization algorithms," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, Kyoto, Japan, Oct. 2011.

[22] A. Bautin, O. Simonin, and F. Charpillet, "Minpos: A novel frontier allocation algorithm for multi-robot exploration," in *International conference on intelligent robotics and applications*, Montréal, Canada, Oct. 2012.

[23] C. Nieto-Granda, J. G. Rogers III, and H. I. Christensen, "Coordination strategies for multi-robot exploration and mapping," *The International Journal of Robotics Research*, vol. 33, no. 4, pp. 519–533, 2014.

[24] M. Corah and N. Michael, "Efficient online multi-robot exploration via distributed sequential greedy assignment." in *Robotics: Science and Systems*, Cambridge, MA, July 2017.

[25] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[26] 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.

[27] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3D scanning," in *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy, Oct. 2017.

[28] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[29] C. H. Papadimitriou, "The complexity of the lin–kernighan heuristic for the traveling salesman problem," *SIAM Journal on Computing*, vol. 21, no. 3, pp. 450–465, 1992.

[30] 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.

[31] 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*, 2020.