UFOExplorer: Fast and Scalable Sampling-Based Exploration With a Graph-Based Planning Structure

Daniel Duberg, *Graduate Student Member, IEEE*, and Patric Jensfelt, *Member, IEEE*

***Abstract*—We propose UFOExplorer, a fast and efficient exploration method that scales well with the environment size. An exploration paradigm driven by map updates is proposed to enable the robot to react quicker and to always move towards the optimal exploration goal. For each map update, a dense graph-based planning structure is updated and extended. The planning structure is then used to generate a path using a simple exploration heuristic, which guides the robot towards the closest exploration goal. The proposed method scales well with the environment size, as the planning cost is amortized when updating and extending the planning structure. The simple exploration heuristic performs on par with the most recent state-of-the-art methods in smaller environments and outperforms them in larger environments, both in terms of exploration speed and computational efficiency. The implementation of the method is made available for future research.**

***Index Terms*—Motion and path planning, mapping.**

I. Introduction

MOBILE robots are becoming increasingly more capable and autonomous. An ambition is for robots to function in any environment. Most environments are, however, partially or fully unknown beforehand. Autonomous exploration allows robots to explore and map an environment without any a priori information, making it an essential capability for achieving fully autonomous robots.

Sampling-based exploration is one of the most common approaches for solving the autonomous exploration problem. These approaches sample poses in the currently known space and move to one of the poses based on some heuristics. By continually repeating the process, the environment is explored.

The heuristics are generally defined based on how much new information the robot will gain about the environment by moving to a pose and is called *information gain*. Different information gain formulations have been proposed, focusing on fast exploration [1] or reconstruction [2].

Manuscript received September 9, 2021; accepted December 22, 2021. Date of publication January 13, 2022; date of current version January 28, 2022. This letter was recommended for publication by Associate Editor G. Grisetti and Editor S. Behnke upon evaluation of the reviewers' comments. This work was supported in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, in part by the SSF Project FACT, and in part by the VR Grant XPLORE3D. *(Corresponding author: Daniel Duberg.)*

The authors are with the Division of Robotics, Perception and Learning (RPL), KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden (e-mail: dduberg@kth.se; patric@kth.se).

Digital Object Identifier 10.1109/LRA.2022.3142923

State-of-the-art sampling-based exploration methods utilize a receding horizon approach [1]–[4]. A *best* path consisting of multiple sampled poses is selected. The first step of the path is then executed and when reached, a new path is calculated.

The best path is typically a path that maximizes a score function taking into account the distance travelled and the information gain along the path. Multiple score functions have been proposed [1], [2], [4]; however, the effect of these functions has not been thoroughly studied.

Only executing the first step along the path allows the robot to quickly react to what is being discovered. As more of the map is uncovered continuously, it is often possible to find a better path after only having executed the first step. In this work, we show that it is beneficial to calculate a new path even before the first step has been fully executed; and that updating the path should coincide with the map being updated.

A tree structure is most commonly used to generate the paths. In [1], [3], [4], the tree is rebuilt, with the previous best path as a seed, for every new path. This means a large part of the tree is discarded and has to be recomputed, wasting computational resources. To address the computational issue, [2] proposed to maintain a single tree throughout the whole exploration process. This single tree is rewired to conform to the current state of the map and the robot's pose.

From a planning perspective, the tree structure is limited as it only guarantees finding the optimal path from the root node to any other node in the tree, but not between any two nodes. This restricts the methods to single goal paths. In this work, we propose to use an undirected graph structure [5] instead. This enables finding optimal paths from any two nodes. Being undirected means that no rewiring based on the robot's pose is required, making for a mostly static structure and any-time planning possible. The static structure removes the stochastic nature of sampling in the path planning, ensuring a path will always be found if one exists, and that a certain query results in the same path every time. We build the planning structure gradually as the map is explored, thereby amortizing the cost over time and reusing it.

The contributions of this paper are as follows:

1) A planning structure, which continuously expands a dense undirected graph, maintaining information about how all uncovered parts of the map are connected.
2) An exploration paradigm driven by map updates instead of time or distance, minimizing the time spent on following paths which are no longer optimal.

3) Showing that a simple exploration heuristic of moving to the closest part of space with information gain left, ignoring how much, outperforms more complex heuristics as well as global tours.

The remainder of this paper is organized as follows. Related work is presented in Section II. The proposed exploration planner is presented in Section III with implementation details in Section IV. The experimental setup and evaluation is presented in Section V and Section VI, respectively, followed by an ablation study in Section VII. A real world experiment is presented in Section VIII. Finally, Section IX concludes the paper.

II. Related Work

Autonomous exploration is the task where a previously fully or partially unknown environment is mapped. The task is generally considered complete when all accessible space has been classified as either occupied or free. It can also be when a person or object has been localized, for example, in a search-and-rescue scenario. Depending on the use case, the focus can be on speed [6]–[8], reconstruction accuracy [2], [9], object search [10], or other factors such as multi-robot collaboration and battery limitations [11]. In this paper, the use case is for a single robot to quickly explore the environment.

The topic has been studied for decades. Frontier exploration [12] introduced frontier regions, regions where free and unknown space meet. By continually creating and visiting new frontier regions, the environment is explored. The exploration process is complete when no frontier regions remain.

In [12], the closest frontier region is selected as the next target to visit. By prioritising frontier regions in the field of view (FoV), a higher velocity can be maintained [6]. This results in faster exploration, outperforming existing methods at the time in terms of speed.

Sampling-based exploration is another approach to solve the exploration task. As mentioned in Section I, poses are sampled in the known free space and for each pose, the information gain is calculated. Sampling-based exploration originates from next-best-view (NBV) [13]. NBV was initially developed as a method to reconstruct 3D objects from range images and was later adopted for exploration [14]. Sampling-based approaches have the advantage that they allow any kind of information gain formulation, which enables the use of them for a wide range of exploration applications [2], [10], [14]. This flexibility is why sampling-based methods are the basis of this work.

Receding horizon next-best-view planner (RH-NBVP) [1] expands a rapidly-exploring random tree (RRT) [15] with viewpoints. For each node n in the tree a utility value is calculated

$$v_{\text{exp}}(n) = v_{\text{exp}}(\text{parent}(n)) + g(n) \cdot \exp\left(-\lambda \cdot c(n)\right) \qquad (1)$$

where $\text{parent}(n)$ is the parent of n, $g(n)$ is the information gain at n, $c(n)$ is the cost of moving to n from the root node, and λ is a tuning factor. The path to the node with the highest utility value is executed in a receding horizon (RH) fashion, meaning only the first node along the path is executed. When the first node is reached, a new tree is expanded with the best branch from the previous iteration as a seed.

RH-NBVP has been extended to avoid local minima, by keeping track of the planner's history [16], or by combining RH-NBVP for local planning with frontier-based global planning, as in AEP [3]. Another hybrid approach [17], proposes to sample poses close to frontiers, reducing the number of samples needed.

Another sampling-based approach expands and maintains a single tree [2], using a rewiring technique similar to RRT* [18] and RT-RRT* [19]. By maintaining the same tree throughout the whole exploration process, they reach global coverage. They also introduce a *global normalization* utility function

$$v_{\text{GN}}(n) = \frac{\sum_{n_i \in P(n)} g(n_i)}{\sum_{n_i \in P(n)} c(n_i)} \qquad (2)$$

where n is a node in the tree, $g(n)$ is the node gain, $c(n)$ is the cost of moving to the node, and $P(n)$ is the sequence of nodes connecting n to the root of the tree. As in RH-NBVP, the node with the highest utility is moved to in a receding horizon fashion.

The aforementioned sampling-based methods all use a fixed number of samples for the tree to bound the computational cost. However, this means that only part of space, the limited part that has been sampled, may be reached. This makes it hard, for example, to determine when to terminate the exploration process.

A two-stage planning approach, where frontier viewpoints are first sampled followed by finding a global shortest tour through all viewpoints is presented in [20]. Similarly, Fast UAV Exploration (FUEL) [8] introduced an incrementally updated frontier information structure (FIS), which stores information about the explored space to facilitate high frequency exploration planning. The FIS is utilized in their proposed hierarchical planner. The hierarchical planner first generates a global tour, similar to [20], that covers all frontier regions. The costs of moving between every pair of frontier regions are cached inside the FIS to make the global tour computationally feasible. Next, a local segment of the tour is refined and a safe minimum-time trajectory, accounting for the dynamics of the robot, is generated for the first part of the refined tour. Using this proposed hierarchical planner, they claim to be able to perform safe, efficient, and fast exploration; although, from the evaluation of our work, these claims can be questioned.

III. Proposed Approach

The fundamental idea of our approach is to build a single undirected graph that is maintained and expanded every time the map is updated, in the region where the map has been updated. By doing this, for every map update, we achieve scalability and ensure that there is always a path from the robot's current position to every accessible part of the map.

Furthermore, one of the key insights that underpin our design is that the amount of information gain is not important beyond that there is more information. This means that all such nodes have to be visited and that it is therefore best to visit the closest one first, as in [12]. This also helps improve scalability as we do not need to search the entire map to find the best node to move to, but can terminate the search at the closest node.

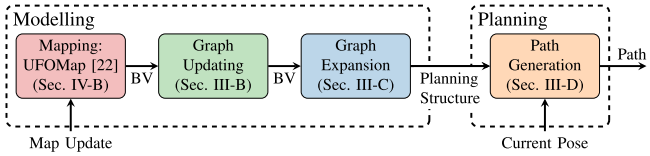


Fig. 1. Planner overview. The map is built concurrently from sensor data in a separate process. *Map updates* are transmitted to the exploration planner when this map is updated. For each map update the planner's internal map and the graph-based planning structure are updated, followed by generating a new *path* to the closest exploration goal using the *planning structure*. The bounding volume (*BV*) of the map update is extracted and propagated, to efficiently update and expand only the corresponding part of the planning structure. The robot's *current pose* comes from the positioning system.

An overview of how the graph is maintained and expanded can be seen in Fig. 1. The details of these steps, along with details on the graph structure, will be presented in the following sections.

A. Graph Structure

The undirected graph $G = (N, E)$ is represented as a set of nodes N and a set of edges E connecting the nodes. A pair of nodes has an edge between them if they are within a certain distance l_{max} from each other and it is possible to move in a straight line between them in free space, i.e., not occupied nor unknown. To ensure that planning in the graph is efficient, the density is limited by allowing only M nodes within a radius R from each node.

Each node is associated with an information gain from the node's position and yaw. The yaw is set to maximize the information gain. We distinguish between two types of nodes. Nodes with enough information gain $N_{\text{IG}} \subseteq N$ to be of interest and nodes with less $N_{\text{O}} \subseteq N$, such that $N_{\text{IG}} \cap N_{\text{O}} = \emptyset$. If the information gain for a node $n \in N_{\text{IG}}$ falls below or equals a threshold t_g, it is demoted to belong to N_{O}, which only stores the position. In a static environment, a node $n \in N_{\text{O}}$ is never promoted to belong to N_{IG}. When $N_{\text{IG}} = \emptyset$, the exploration process is finished.

The information gain is traditionally the most expensive part of a sampling-based exploration planner, consuming up to 95% of the planner's run-time [2]. By differentiating between nodes with and without information gain, we limit the number of times the information gain has to be calculated.

B. Graph Updating

As the map changes, nodes and edges can become invalid, i.e., in collision. Therefore, when the map is updated, all nodes and edges within the updated region of the map are reassessed. First, nodes and edges in collision are removed. Second, new edges between the remaining nodes are added where possible. For the remaining nodes N_{IG} within the sensor range of the updated region, the information gain and yaw are updated to reflect the current state of the map.

It is worth noting, depending on the assumptions on the environment and the sensor used, it is possible to skip removing nodes and edges. For example, when exploring a static environment with a noise-free sensor. However, as those conditions rarely hold in the real world, we have chosen to always reassess all nodes in this work, even when performing experiments in static environments in simulation.

We do not create edges in unknown space. Hence, it is always necessary to perform the step of adding new edges where possible, as additional unknown space may have been uncovered. Whenever the graph is updated, it has to be extended to ensure that the graph density is consistent, even if no new space has been uncovered.

C. Graph Expansion

During the exploration process, more of the map is uncovered, making it necessary to continuously expand the graph to cover all free space. New nodes have to be added to the uncovered region. The number of new nodes generated is the volume V of the uncovered region times the expected node density M/R,

$$V \cdot \frac{M}{R}. \tag{3}$$

Each generated node n_{g} is assigned a sampled position within the updated part of the map. If a node n_{g} is further than l_{max} from the closest node n_{c} already in the graph, n_{g} is moved towards n_{c} such that they are l_{max} apart.

For a generated node to be added to the graph, it has to fulfill the following criteria:

1) Not be in collision with any occupied or unknown space.
2) Have less than M nodes within a radius R of its position.
3) Have at least one edge to an existing node.

The information gain and yaw are calculated for every newly added node.

D. Path Generation

The last step is to generate a new exploration path based on the robot's current pose and the graph. In this work, we use the insight that all N_{IG} should be visited. Therefore, we argue that the amount of information gain should not have an influence on the path generation. We have identified in other works that the exploration method gets greedy since it priorities regions with a lot of information gain and therefore leaves smaller regions of information gain behind which it has to come back to later [22]. These path generators combine the path cost and the amount of information gained along the path [1]–[4] to determine the best path. We instead pick the shortest path that reaches a node $n \in N_{\text{IG}}$. In other words, we pick the shortest path to a node with enough information gain, ignoring the amount of gain as long as it is over the threshold.

IV. Implementation

This section describes the key implementation details of the proposed method. UFOExplorer is made available at: https://github.com/UnknownFreeOccupied/UFOExplorer.

A. Graph

Updating (Section III-B) and extending (Section III-C) the graph requires a couple of spatial queries on the nodes N, such

as, retrieving all nodes within a region or retrieving the closest node to a position. To perform these queries quickly and to ensure scalability with the number of nodes in the graph, the nodes are stored in an R*-tree [23].

B. Mapping

To ensure safety while exploring, it is important that the collision checking is performed with respect to both occupied and unknown space. In other exploration methods, such as [3], unknown space is treated as free space when performing collision checking, meaning collision checking is only performed against occupied space. This is because of the high cost of accessing the unknown space in the used map representation. The lack of collision checking against unknown space means some paths are close to unknown space, which can be occupied, and if followed results in a collision. Therefore, UFOMap [21] is used as the map representation in this work. The explicit representation of unknown space together with the fast collision checking capabilities, makes it a suitable choice.

UFOMap also provides builtin support for retrieving the updated region of the map, which simplifies the implementation. However, any volumetric mapping framework can be used.

C. Path Execution and Obstacle Avoidance

The path generated from the graph only contains straight line segments, following these would result in longer and non-smooth motion. To allow for shorter paths and smoother motion, a pure pursuit [24] path tracking controller is used. As the controller will cut corners, an obstacle avoidance method [25] is also added. This decouples the path execution and obstacle avoidance from the exploration planning, as opposed to, for example, [8] that incorporates local trajectory optimization.

D. Method Scalability

A key aspect of an exploration method is how well it scales with the environment size. As the environment grows larger, sampling-based methods, such as [1], [3], [4], perform increasingly worse. This is because the closest unknown region can be far away, meaning more nodes have to be sampled. These approaches usually sample a fixed number of nodes and need to identify and solve such scenarios by either backtracking or switching to a different mode. Other approaches, such as [8], instead have the problem that the path that has to be generated to a frontier region is far away.

Our method scales better with the environment size, as the cost of updating and extending the graph is approximately the same at any point during the exploration. The graph is then used for path planning, reducing the cost of finding a path significantly compared to [8].

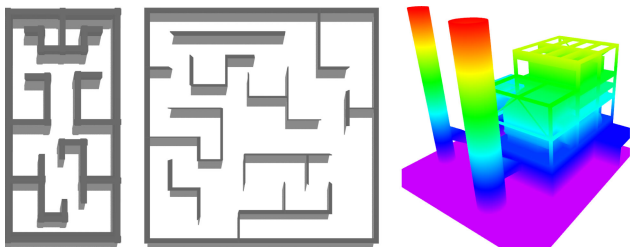
We use multiple-goal A* [26] for the path generation with all N_{IG} nodes as goals, stored in an R*-tree [23] for scalability, and the robot's pose being the start. The first goal to be reached is picked as the target with the corresponding path. The cost for this depends on the number N_{IG} nodes and the distance to the closest N_{IG} node measured by path length. Path generation is most expensive when the closest N_{IG} node is far away. However, in these cases the robot will, by definition, mainly travel through explored space and hence the graph will not be changed until the end of the path. This means that the path does not have to be updated for a long time. It would therefore be possible to only perform path generation when the graph has changed. However, in our experiments, even in the largest environments, the cost for the path generation was always negligible.

TABLE I
PARAMETERS USED IN THE EXPERIMENTAL EVALUATION

Parameter	Value	Parameter	Value
Voxel size	10 cm	Sensor Range	4.5 m
Linear Velocity	2.0 m/s	Sensor FoV	80° × 60°
Angular Velocity	0.9 rad/s	Sensor Rate	10 Hz

TABLE II
UFOEXPLORER PARAMETERS

Parameter		Value	Parameter		Value
Gain threshold	t_g	30 %	Max local nodes	M	1
Max edge length	$l_{\max}$	2.0 m	Local density radius	R	1.2 m



(a) Simple Maze [8]. (b) Complex Maze [2]. (c) Power Plant [3], [6].

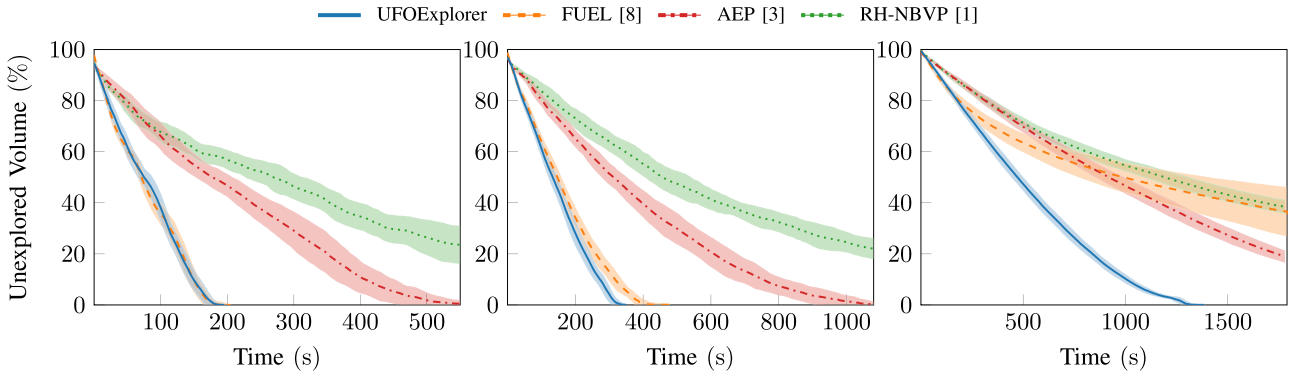
Fig. 2. The three environments used in the experimental evaluation.

V. EXPERIMENTAL SETUP

We compare UFOExplorer with the receding horizon next-best-view planner (RH-NBVP) [1], autonomous exploration planner (AEP) [3], and the more recent fast UAV exploration (FUEL) [8]. RH-NBVP and AEP are both sampling-based methods, while FUEL is a frontier-based method. All three methods have shown to perform well in indoor scenarios.

Identical system constraints were used (Table I) for all experiments. We used the parameter settings proposed in the respective paper for the methods we compare to and the parameters listed in Table II for UFOExplorer. All methods were evaluated using their respective open-source implementations, with minimum changes. RH-NBVP, AEP, and our method were run in the Gazebo-based RotorS simulator [27], while FUEL was run in their simulator for the fairest evaluation.

We decided to not create our own environments, instead chose to use environments presented in the literature. This is to ensure that the environments have not been tailored to our method, and to show the generality of UFOExplorer. The simple maze (Fig. 2(a)) and complex maze (Fig. 2(b)) environments were



(a) Simple Maze (Fig. 2a). (b) Complex Maze (Fig. 2b). (c) Power Plant (Fig. 2c).

Fig. 3. Comparison of exploration progress of the four methods in the three environments depicted in Fig. 2. Mean and standard deviation are shown.

adapted from [8] and [2], respectively. It is not a one-to-one copy as they used a different format, therefore there are some subtle differences. The power plant environment (Fig. 2(c)) is the same as in [3], [6].

The simple maze (Fig. 2(a)) is $20 \times 40 \times 3\text{m}^3$ large, containing only dead-ends and no loops; therefore, it is rather simple to explore as any depth first or wall following approach is optimal. The complex maze (Fig. 2(b)), is more complex as it contains both loops and dead-ends, and is also larger at $40 \times 40 \times 3^3$. The power plant (Fig. 2(c)), is a large environment in all three dimensions, $31 \times 31 \times 26^3$. It contains a lot of free space and differs from the others in that it is the outside of a building that is explored. The robot starts in the center of the simple and complex maze scenarios, and in the bottom right corner in the power plant scenario.

We evaluate the methods both in terms of exploration performance and computational time. As in the work we compare with, we define performance as how fast an environment is explored; in order words, the amount of space uncovered over time. Each method is run 100 times per environment. Each run is allotted a maximum of 30 min and is stopped if it exceeds this. We only take into account the successful runs without collisions. In the case of FUEL, this is important as it was noted that the method could travel through walls, since their simulator does not simulate collisions. We therefore check if the center-point of the robot is in collision and discard the run if it happens.

The experiments were performed on an Intel Core^{TM} i7-8700 K with 64GB of RAM.

VI. Experimental Evaluation

In this section, simulation results obtained by the proposed sampling-based exploration method UFOExplorer described in the previous sections are presented and analysed. Additionally, videos showcasing the proposed method can be found at: https://youtube.com/playlist?list=PLrgh6NykJdiO9NsEwJ12CSK1a-_fq6SUU.

A. Exploration Performance

One of the goals of the proposed method was to be able to explore environments quickly. Therefore, in this first evaluation, a comparison of the exploration performance between the four methods is conducted.

In Fig. 3, the exploration performance of the four methods in the three environments is shown. In the two maze environments (Fig. 3(a) and Fig. 3(b)) the relative performance of the methods is similar, with UFOExplorer and FUEL outperforming AEP and RH-NBVP by a factor of 3 to 5. In the simple maze environment, FUEL finished on average a second ahead of UFOExplorer, 169s compared to 170s. However, in the complex maze environment UFOExplorer has the advantage and takes on average 312s compared to FUEL's 381s, over a full minute faster.

At first glance, these results seem surprising. As our proposed planner is greedy in the sense that it always moves to the closest node with enough information gain, i.e., N_{IG}, compared to FUEL which plans a longer global tour. However, it is important to remember that exploration is an iterative process and that it is impossible to calculate an optimal route through the environment before it has been fully explored. The global tour generated by FUEL is therefore only optimal for a specific state of the map. As soon as more of the environment has been explored, the plan is most likely no longer optimal.

The behavior of FUEL's global tour is often similar to depth-first search, meaning the robot will continue to explore in the same direction until it has reached a dead-end or a previously visited state. This is because the global tour only accounts for what is currently known, thus assuming that the robot will have to backtrack if it does not continue in the same direction. Therefore, usually the first subgoal on the global tour is similar to moving to the closest node, as in UFOExplorer. The depth-first search behavior of FUEL explains the results in the simple maze environment, as it is the optimal strategy for that environment.

In the larger power plant scenario (Fig. 3(c)), UFOExplorer was the only method able to fully explore the environment within the allotted 30 min, finishing on average after 21.3 min. Notably, AEP performed better than FUEL in this environment, managing to explore around 80% of the environment. FUEL

TABLE III
COMPUTATIONAL TIME PER ITERATION (MS)

Method	Environment	Min	Max	Mean & SD
UFOExplorer	Simple Maze	0.1	28.9	3.9 ± 2.3
FUEL [8]	(Fig. 2a)	4.0	3118.2	41.3 ± 111.3
UFOExplorer	Complex Maze	0.1	30.2	6.0 ± 2.9
FUEL [8]	(Fig. 2b)	5.5	3929.3	98.3 ± 86.6
UFOExplorer	Power Plant	0.2	69.3	12.3 ± 6.4
FUEL [8]	(Fig. 2c)	5.9	8185.1	1204.8 ± 958.2

started off exploring at the same rate as UFOExplorer; however, at around 250s it started to stagnate (see Section VI-B), resulting in RH-NBVP catching up and both ending at 60% explored.

In the power plant environment (Fig. 2(c)) we performed an additional experiment with Rapid [6] as it reported competitive results in this environment. We used the same parameter settings as in [6], i.e., linear velocity 2.5m/s, angular velocity 1.5rad/s, sensor range 7m, and sensor FoV $115^\circ \times 60^\circ$. The UFOExplorer specific parameters were kept the same as in all other experiments (Table II). Over 10 runs it took on average 273(16)s for UFOExplorer, compared to Rapid's 582(26)s, as stated in their paper. Resulting in 113% faster exploration rate for UFOExplorer.

B. Computational Time

Another goal of UFOExplorer was scalability. Thus, this section investigates the computational time in the three environments. Given that FUEL and UFOExplorer performed so much better than the other methods at exploration, in the first two environments, we limit the evaluation of computational complexity to these methods. The computational time per iteration for each environment is summarised in Table III.

For UFOExplorer, the computational time are similar in the two maze environments. In the power plant scenario, an increase by a factor of two is observed. The reason for this is the increased size in the z-direction, as well as there being a lot more open space. This results in larger map updates, which in turn means more nodes being added to the graph in a single iteration. It is mainly the maintaining of the graph that affects the computational time, as the path generation has an average and maximum computational time of 0.3 ms and 12.0 ms, respectively, in all three environments.

Comparing UFOExplorer and FUEL, the mean computational time in the simple and complex maze scenarios, where the two methods exploration performance were similar, is an order of magnitude lower for UFOExplorer. The advantage is further increased in the power plant scenario, where UFOExplorer is two orders of magnitude quicker.

The mean and standard deviation together with the maximum computational time can explain the poor performance of FUEL in the power plant scenario. The global tour that FUEL computes every iteration is inherently expensive as it is a form of the travelling salesman problem (TSP), which is not solvable in polynomial time. In environments with multiple possible paths, or large open spaces, the number of frontier regions grows large, increasing the cost of computing the global tour. Out of the 1.2s average computational time, 0.5s is spent on computing the distance matrix between the frontier clusters and 0.5s on solving the TSP. The maximum time to compute the distance matrix was 5.4s and the TSP 3.1s.

The minimum computational times were similar in each environment. This is expected as the minimum is often found at the beginning, when very little about the environment is known; hence, there is not much to calculate.

These results demonstrate the scalability of UFOExplorer. The main contributing factor is the undirected graph planning structure. As the time taken to maintain the graph is approximately the same at any point.

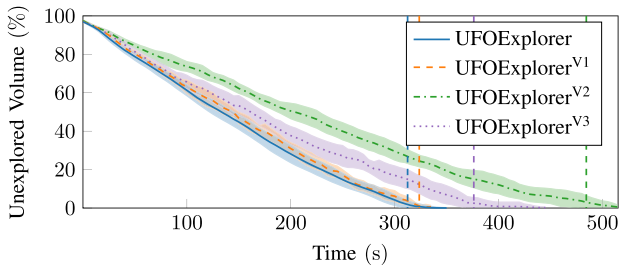


Fig. 4. Comparison of exploration performance of four exploration paradigms in the complex maze environments depicted in Fig. 2(b). Mean and standard deviation are shown, with the vertical lines representing the mean end time.

VII. ABLATION STUDY

In this section UFOExplorer is dissected to study where the gains in exploration performance originates. Each experiment in this section was run 10 times.

A. Exploration Paradigm

First, we look at the impact of the map update driven exploration paradigm. In Fig. 4 the exploration performance in the complex maze of four different configurations is shown. UFOExplorer updates the path for every map update, while UFOExplorer^{V1} updates it when there is no more information gain along the path or after a certain time, whichever comes first, as in FUEL [8]. UFOExplorer^{V2} uses the receding horizon approach of RH-NBVP [1] and AEP [3].

UFOExplorer and UFOExplorer^{V1} sends the whole path to the closest N_{IG} to the controller, while UFOExplorer^{V2} only sends the first subgoal/node along the same path. To account for this difference and to be fairer towards the receding horizon approach, UFOExplorer^{V3} is included. It is the same as UFOExplorer^{V2} except that the whole path is given to the controller. Both of them update the path once they reached the first subgoal. As seen in Fig. 4, this often overlooked difference has a substantial impact on the exploration performance.

Comparing UFOExplorer and UFOExplorer^{V3} shows the importance of updating the path each time the map is updated, as it prunes non-informative paths as soon as they are not consistent with the current map, thus improving the exploration

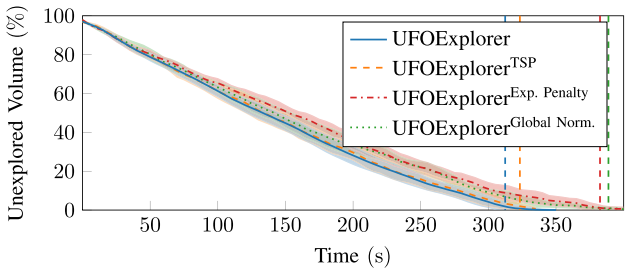


Fig. 5. Comparison of exploration performance of four exploration heuristics in the complex maze environments depicted in Fig. 2(b). Mean and standard deviation are shown, with the vertical lines representing the mean end time.

TABLE IV
PATH GENERATION COMPUTATIONAL TIME PER ITERATION (MS)

Method	Max	Mean & SD
UFOExplorer	6.4	0.2 ± 0.4
UFOExplorer$^{\text{TSP}}$	68.0	6.9 ± 13.2
UFOExplorer$^{\text{Exp. Penalty}}$	16.3	3.3 ± 1.7
UFOExplorer$^{\text{Global Norm.}}$	17.0	3.8 ± 2.3

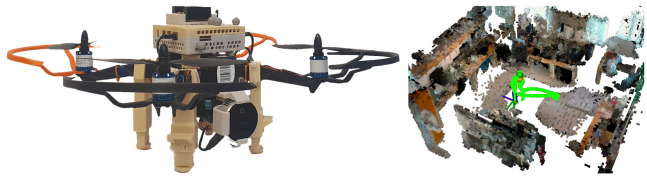


Fig. 6. Experiment in an indoor environment. The micro air vehicle (left). The exploration result, green line showing the path travelled (right).

performance. By reducing the update rate of UFOExplorer the performance would approach that of UFOExplorer^{V3}. An important note is that there is no reason to update the path more often than UFOExplorer, as it would not be possible to find a better path without the map changing.

By following the whole path and stopping prematurely, when the path contains no new information (UFOExplorer$^{\text{V1}}$), as in FUEL, the exploration performance is on par with UFOExplorer, where the path is updated as soon as the map is updated, 323s compared to 312s.

These results are from a static environment. In dynamic environments UFOExplorer is arguably preferred, as it can react the quickest to environmental changes.

Noteworthy is that these are implemented in UFOExplorer. The results may be different in another planner. The any-time planning requirement, enabled by the graph planning structure, makes it difficult to implement in RH-NBVP and AEP.

B. Exploration Heuristic

Another aspect investigated is the exploration heuristic, which decides where the robot should move next. A comparison of four exploration heuristics can be seen in Fig. 5.

UFOExplorer simply moves to the nearest N_{IG} node. UFOExplorer^{TSP} computes a global tour, similar to FUEL, by clustering the N_{IG} nodes using DBSCAN [28]. However, the distance between the clusters are not cached in our case, as planning is accelerated using the undirected graph planning structure. UFOExplorer$^{Exp.Penalty}$ uses the exponential penalty (1) to select the N_{IG} node with highest utility value, as in RH-NBVP and AEP. UFOExplorer$^{GlobalNorm.}$ instead uses the global normalization (2), proposed in [2].

UFOExplorer and UFOExplorer^{TSP} perform similarly, as expected from the discussion in Section VI-A. It may be surprising that UFOExplorer outperforms UFOExplorer$^{Exp.Penalty}$ and UFOExplorer$^{GlobalNorm.}$, 312s compared to 383s and 389s, respectively. The reason is that both the exponential penalty and global normalization fragment the unexplored space, leaving nodes here and there that have to be visited later. RH-NBVP and AEP could gain an additional 25% exploration performance by switching exploration heuristics.

Lastly, there is an additional cost of computing the paths using UFOExplorer$^{Exp.Penalty}$ and UFOExplorer$^{GlobalNorm.}$, as a path to every N_{IG} node has to be computed. This results in an order magnitude higher path generation time compared to UFOExplorer (Table IV). The computational cost of UFOExplorer^{TSP} is even more expensive; however, not as expensive as in FUEL, because of the planning structure.

VIII. Real World Experiment

To further validate the proposed method, we conducted a small real world experiment in the same indoor environment as in [3] on a micro air vehicle (MAV) (left Fig. 6). The reconstructed environment is shown to the right in Fig. 6, with the green line showing the path the MAV travelled.

The MAV was equipped with an Intel NUC NUC8i7HNK with 32 GB of RAM and an Intel RealSense$^{\text{TM}}$ L515 sensor. The parameters were kept the same (Tables I and II). State information was provided by a motion capture system to remove the associated errors induced by state estimation drift.

IX. Conclusion

In this work, a new sampling-based exploration method was proposed. The proposed method employs a novel exploration paradigm driven by map updates, enabling the robot to always move towards the optimal exploration goal and react quickly. A dense graph-based planning structure is updated and maintained for each map update, amortizing the cost of planning. The graph structure enables efficient planning between any two locations in the uncovered space, ensuring scalability of the method with respect to the environment size. Lastly, a simple exploration heuristic is used, which moves the robot to the closest unknown space. This heuristic is less long-term greedy and simpler to compute compared to the heuristics used in recent state-of-the-art methods.

The method was evaluated against three state-of-the-art methods. In smaller environments, the proposed method was on par with, or even slightly faster than, the state-of-the-art methods in terms of exploration speed, and outperformed them by more than an order of magnitude in terms of computational performance. In

larger environments, the advantage increases, showing that the proposed method scales well with the environment size. Only the proposed method was able to explore the largest environment in the allotted time, and the computational cost was similar to the smaller environments. This resulted in the computational cost being two orders of magnitude smaller compared to the other methods.

REFERENCES

[1] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3D exploration," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 1462–1468.

[2] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto, "An efficient sampling-based method for online informative path planning in unknown environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1500–1507, Apr. 2020.

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

[4] F. S. Barbosa, D. Duberg, P. Jensfelt, and J. Tumova, "Guiding autonomous exploration with signal temporal logic," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3332–3339, Oct. 2019.

[5] L. E. Kavralu, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[6] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2135–2142.

[7] M. Dharmadhikari *et al.*, "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 179–185.

[8] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "FUEL: Fast uav exploration using incremental frontier structure and hierarchical planning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 779–786, 2021.

[9] S. Song and S. Jo, "Online inspection path planning for autonomous 3D modeling using a micro-aerial vehicle," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 6217–6224.

[10] T. Dang, C. Papachristos, and K. Alexis, "Autonomous exploration and simultaneous object search using aerial robots," in *Proc. IEEE Aerosp. Conf.*, 2018, pp. 1–7.

[11] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-UAV exploration with limited communication and battery," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 2230–2235.

[12] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Automat.*, 1997, pp. 146–151.

[13] C. I. Connolly, "The determination of next best views," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1985, vol. 2, pp. 432–435.

[14] C. Papachristos, S. Khattak, and K. Alexis, "Uncertainty-aware receding horizon exploration and mapping using aerial robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 4568–4575.

[15] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Technical Rep. no. 98-11, Oct. 1998.

[16] C. Witting, M. Fehr, R. Bähnemann, H. Oleynikova, and R. Siegwart, "History-aware autonomous exploration in confined environments using MAVs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[17] A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas, and S. Leutenegger, "Fast frontier-based information-driven autonomous exploration with an MAV," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 9570–9576.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[19] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT*: A real-time path planning algorithm based on RRT*," in *Proc. 8th ACM SIGGRAPH Conf. Motion Games*, 2015, pp. 113–118.

[20] Z. Meng *et al.*, "A two-stage optimized next-view planning framework for 3-D unknown environment exploration, and structural reconstruction," *IEEE Robot. Automat. Lett.*, vol. 2, no. 3, pp. 1680–1687, Jul. 2017.

[21] D. Duberg and P. Jensfelt, "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 6411–6418, Oct. 2020.

[22] L. Ericson, D. Duberg, and P. Jensfelt, "Understanding greediness in map-predictive exploration planning," in *Proc. Eur. Conf. Mobile Robot.*, 2021, pp. 1–7.

[23] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1990, pp. 322–331.

[24] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," *Robot. Inst.*, Carnegie Mellon Univ., Pittsburgh, PA, USA, Rep. no. CMU-RI-TR-92-01, 1992.

[25] D. Duberg and P. Jensfelt, "The obstacle-restriction method for teleoperation of unmanned aerial vehicles with restricted motion," in *Proc. 15th Int. Conf. Control, Automation, Robot. Vis.*, 2018, pp. 266–273.

[26] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[27] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS-A Modular Gazebo MAV Simulator Framework*. Cham, Switzerland: Springer International Publishing, 2016, pp. 595–625.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *KDD*, vol. 96, no. 34, 1996, pp. 226–231.