Informed Autonomous Exploration of Subterranean Environments

Aliakbar Akbari[1] and Sara Bernardini[1]

***Abstract*—Autonomous exploration is highly challenging in subterranean applications due to the constraints imposed by the nature of the environments (e.g., dead-end branches, unstructured regions, narrow passages and bifurcations). Robots need to constantly balance their exploration objectives with measures to ensure safety. We present an informed exploration approach to address these challenges, which exploits a reachability graph to represent the environment's structure and adaptive navigation to find collision-free motions. Our system makes the inspection task tractable and maximizes the information acquired about the environment while preserving safety. We evaluate our navigation and exploration techniques against several challenging cave scenarios reconstructed using real data. Our experimental results demonstrate that our method enables the robot to make informed decisions and perform exploration more efficiently than existing techniques in confined spaces.**

***Index Terms*—Robotics in Hazardous Fields, Mining Robotics, AI-Based Methods, AI-Enabled Robotics, Aerial Systems: Perception and Autonomy.**

I. Introduction

ROBOTIC inspection of subterranean spaces is a challenging problem because of the presence of hazards such as vapors, fumes, and gases and due to the configuration of such spaces, which often present voids as well as unstructured and possibly dead-end branches (see Figure 1, measures refer to the bounding box around the mine). There are also challenges associated with the robot's health system; for instance, battery life imposes limitations on the available exploration time. To deal with all those factors, the robots should be fully autonomous and capable of robustly mapping and exploring the environment beyond visual line of sight.

Aerial robots are considered promising to carry out inspection of subterranean environments [1], [2]. To be effective in sensing and mapping the environment, the robot needs to measure the information it can gain by visiting areas not yet seen. We call this ability *informed exploration*. Since exploration consists in continuously moving the robot towards unknown spaces, the robot also needs to be equipped with *adaptive navigation*, which allows it to move between chosen points in the environment. This paper presents an integrated, informed exploration and adaptive navigation system that allows an aerial robot to safely and autonomously inspect confined spaces. Our notion of *safety* encompasses two aspects, which reflect two crucial requirements of underground operations: i) the robot must avoid collision with the environment at all times to preserve its own and the environment's safety; and ii) the aerial robot needs to exit the confined space at the end of the mission. Since confined spaces such as abandoned mines often present only one entry point (*home*), the second constraint requires that the robot navigates back to such point and exit the space before running out of battery [2].

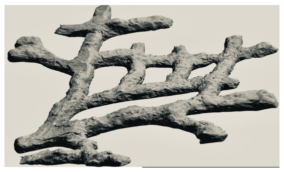
Fig. 1: A big mine in the UK ($96 \times 95 \times 15$ m, Network Rail).

To achieve safe and autonomous inspection, we put forward the following contributions. As for navigation, we modify the algorithm *RRT-Connect* [3], which is one of the most efficient approaches to single-query path planning to account for sensing actions and look for *safe* paths based on 3D map information. The algorithm ensures that the robot avoids collisions and is able to go back to the entry point when it is about to run out of battery. Furthermore, we combine the planner with a *re-planning* mechanism that regenerates real-time motions once a collision is detected. Regarding exploration, we introduce a *double weighted reachability graph* that represents information about the structure of the environment and the unfolding of the exploration process. The graph is constructed automatically and dynamically. It ensures efficient exploration by enabling the robot to detect dead-ends, spaces already visited and areas that need further exploration. We also formulate a *utility function* that incorporates not only information gain and path length as state-of-the-art techniques but also safety constraints (e.g. distance to obstacles). Our utility function drives the robot towards regions where it can maximize its knowledge of the environment while preserving safety. Finally, we devise an *exploration manager* that combines all the above components in a hierarchical structure. The manager exploits a frontier sampling mechanism and a set of lower-level reasoners, which are used by the robot to choose the best course of action when considering the environmental challenges. After describing our contributions, we present extensive experiments on several challenging real-world scenarios, which demonstrate the power of our approach.

II. Related Works

The frontier-based approach to exploration was pioneered by Yamauchi [4]. Following that, various advancements have

Manuscript received: February, 23, 2021; Revised May, 27, 2021; Accepted July, 13, 2021.

This paper was recommended for publication by Editor Pauline Pounds upon evaluation of the Associate Editor and Reviewers' comments. This work is partially supported by the Innovate UK.

[1]Aliakbar Akbari and Sara Bernardini are with Department of Computer Science, Royal Holloway University of London, Egham, UK. ali.akbari@rhul.ac.uk, sara.bernardini@rhul.ac.uk

Digital Object Identifier (DOI): see top of this page.

been proposed to effectively drive the robot towards unknown regions [5], [6], with the use of utility metrics being particularly successful. The approach by Almadhoun *et al.* [7] present a function that factors in information theory, model density, traveled distance, and predictive measures. Stachniss *et al.* [8] offer information gain-based utility functions for exploration, mapping, and localization, while Palazzolo *et al.* [9] present an approach to cope with unknown 3D environments. Frontier-based approaches have also been proposed for underwater exploration [10], [11]. Recent work by Dai *et al.* [12] reports a utility function which is calculated based on the robot's candidate pose and the total path travel time. All these approaches are not designed to tackle the challenges of subterranean settings. They do not deal with large voids, narrow passages, and dead-end branches, which are instead common in underground scenarios.

Differently from related techniques, our utility function measures the distance of the chosen frontier node from the nearest obstacle and tries to maintain the robot at the centre of narrow passages, keeping it safe while it explores the environment. Furthermore, our function computes the information gain not only for the selected frontier node, as the other techniques, but also for the full motion from the robot's current state to the frontier node, which reflects the total information acquired by the robot via its motion.

Other approaches to deal with the exploration of subterranean environments exist ([13], [14]). In this direction, several authors ([15], [16]) offer different versions of multi–modal mapping and sensing techniques combined with exploration planning. Dang *et al.* [17] present graph-based exploration path planning in subterranean environments. Akbari *et al.* [1]'s approach presents an intelligent exploration technique with semantic knowledge to tackle confined spaces. Despite these approaches address the subterranean challenges, they do not implement an informed strategy to efficiently guide the search, which is particularly detrimental in large environments. Compared to these approaches, we present an original informed exploration technique, which employs a reachability graph and a sampling-based mechanism that lead the robot to find unexplored regions and sample promising frontier nodes. Moreover, we offer a navigation module that is able to find safe plans (i.e. plans unfolding in known areas) whenever possible.

III. Problem Formulation and Solution Overview

We tackle the challenge of an aerial robot inspecting and mapping an unknown subterranean environment with no prior information and no human intervention. We identify two problems that need to be solved to address this challenge, *exploration* and *navigation*, which are defined below.

We denote the robot's configuration by its 3D position and yaw angle relative to the world frame, $q = [x, y, z, \psi]$. We assume that the robot's 3D sensing system Ω has a 360° range r_Ω varying from d_{min} to d_{max} and $\Delta = d_{max} - d_{min} \geq 1\ m$. We also assume that the robot is powered by a battery with limited energy capacity and can always access information regarding its remaining flight time. It can also access its accurate location, acquired by using a localization system (see Section VIII for details).

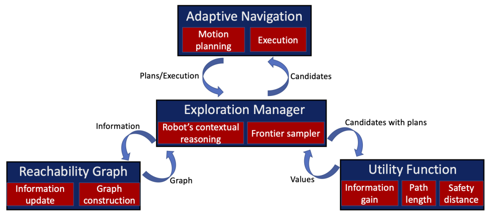

Fig. 2: Main components of our informed exploration approach.

Based on sensing data, the robot incrementally constructs a *3D occupancy map* [18] $\mathcal{M}$, which represents the real-world environment as divided into equally sized units of 3D space called *voxels*. Each point in $\mathbb{R}^3$ lies in exactly one voxel, and we identify each voxel with the point at the center of it. Voxels that correspond to known points in space are classified into *free* and *occupied*, depending on whether they are free or occupied by obstacles. They are indicated as $\mathcal{M}_{free}$ and $\mathcal{M}_{occupied}$. We put $\mathcal{M}_{known} = \mathcal{M}_{free} \cup \mathcal{M}_{occupied}$. For any point that is still unknown to the robot, we say that the corresponding voxel is also *unknown* (unexplored space is not voxelized yet). Unknown voxels are indicated as $\mathcal{M}_{unknown}$. We call *frontier* the set of free voxels that are adjacent to the unknown space and *frontier candidate* a voxel sampled from the frontier.

Exploration Problem *Given an unknown 3D space $\mathcal{U}$, the exploration problem $\mathcal{P}_e$ consists in constructing a 3D occupancy map of known voxels, $\mathcal{M}_{known}$, of the largest 3D space $\mathcal{E} \subseteq \mathcal{U}$, given the robot's flying time.* Our exploration algorithm works by continuously looking for the best frontier candidate to visit and then moving the robot from its current position to the candidate's position until the robot has run out of resources. Hence, to solve the exploration problem, the robot also needs to solve the *navigation problem*, which involves being able to move from an initial to a goal configuration. The robot must be capable of computing a collision-free trajectory in the presence of obstacles and, then, executing it.

Navigation Problem *A navigation problem is a tuple $\mathcal{P}_n = \langle \mathcal{C}, q_{init}, q_{goal} \rangle$, where $\mathcal{C}$ is the configuration space built based on a 3D map $\mathcal{M}$, $q_{init} \in \mathcal{M}_{free}$ is the initial configuration and $q_{goal} \in \mathcal{M}_{free} \cup \mathcal{M}_{unknown}$ is the goal configuration of the robot. A solution for a navigation problem $\mathcal{P}_n$ is a motion plan $Q = \langle q_0, ..., q_k \rangle$, which is a sequence of robot's configurations where $q_0 = q_{init}$ and $q_k = q_{goal}$ and each $q_i \in \mathcal{M}_{free} \cup \mathcal{M}_{unknown}$.*

We tackle problems $\mathcal{P}_n$ and $\mathcal{P}_e$ via a system of interacting components (Figure 2), which, taken together, form our *informed exploration* approach. The *exploration manager* (Section VII) generates frontier candidates based on the information contained in the *double weighted reachability graph* (Section V), which is a dynamically constructed data structure that represents the structure of the environment and the progress of the exploration process. The candidates sampled by the manager are passed to the *adaptive navigation* module (Section IV), which looks for motion solutions from the robot's current position to the candidates' positions and send the solutions back to the manager. The manager calls the *utility function* module (Section VI), which computes the utility value of each frontier candidate based on information gain, path length, and safety distance to the obstacles. The manager then picks the candidate with the highest utility and pass it to navigation for execution. As part of execution, the navigation module constructs the 3D map $\mathcal{M}_{known}$ and verifies the motion plan against safety constraints. Once the robot reaches the chosen frontier voxel, the entire process starts again until the space is fully explored or the robot goes back to the initial position having exhausted its resources.

IV. Adaptive Navigation

In a known environment, navigation deals with finding a collision-free motion plan. In our scenario, we have the additional challenge that the frontier between known and unknown space changes dynamically while the mission unfolds. We also need to ensure that, when the robot is running out of battery, it can safely get back to the home position.

We build on the *RRT-Connect* sampling-based motion planner [3] to implement our navigation strategy, which has two sides: motion planning and motion execution with re-planning. We choose *RRT-Connect* because it is one of the most efficient single-query algorithms, which usually offer better performance in the context of exploration. They remain focused on a given query in visiting the configuration space, even when the environment changes as a result of the exploration process. The original version of the planner grows two trees ($\mathcal{T}_a$ and $\mathcal{T}_b$) in $\mathcal{M}_{free}$, rooted at the *initial* and *goal* configurations. The two trees try to meet while growing. When they become connected, a path is found between the initial and goal configurations.

We modify the *RRT-Connect* planner to reason about the known and unknown space. In particular, if the goal configuration lies in the unknown area, the planner follows the original procedure for the tree extension by exploiting both $\mathcal{M}_{free}$ and $\mathcal{M}_{unknown}$. More specifically, the planner selects $q_i \in \mathcal{M}_{free} \cup \mathcal{M}_{unknown}$ to obtain the plan: $Q = \langle q_0, ..., q_n\rangle \in \mathcal{M}_{free} \cup \mathcal{M}_{unknown}$. However, if the goal configuration lies in the known space, the motion planner tries to find a path within the known areas only, i.e., $Q \in \mathcal{M}_{free}$. This is done by sampling the robot's positions inside the known regions and by growing the two trees within these regions rather than the whole space. Our technique prioritizes finding paths in the known space because they allow the robot to make accurate calculations of the battery needed to follow them. Based on such calculations, the robot can return home when it is running out of battery and be retrieved. Avoiding the unknown space might lead to suboptimal paths but satisfies the robot's constraint of being able to navigate back home by the end of the mission.

We also augment the *RRT-Connect* planner with an executive and replanning mechanism. In executing a plan Q, the system keeps monitoring the configurations found and, for plans that unfold in $\mathcal{M}_{free} \cup \mathcal{M}_{unknown}$, if a collision with a newly observed obstacle is detected ($q_i \in Q$ and $q_i \in \mathcal{M}_{occupied}$), the system replans from scratch, looking for a new path from the current position to the goal. To ensure safety in the intermediate parts of the path, our planner uses a cost function, which tracks if a minimum distance between the robot's position and the closest obstacles is kept.

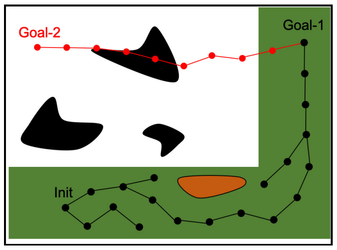

Fig. 3: Two navigation problems. The robot first plans to move from *Init* to *Goal-1* within the known area (green) and, then, from *Goal-1* to *Goal-2* within the unknown area (white). Obstacles in brown color are known to the robot, those in black are not.

This cost function is applied when the planner evaluates the configurations and adds them to the trees. In this way, we can always keep the robot at a safe distance from obstacles.

Figure 3 shows an example of our technique. Suppose that a robot is located at *Init* and is required to move to *Goal-1*. The proposed planner identifies that the goal is located within the known area, $\mathcal{M}_{free}$ (region in green color). Therefore, the trees are grown inside this safe region, $\mathcal{T}_a, \mathcal{T}_b \in \mathcal{M}_{free}$. The visible obstacle space (shown in brown color) is avoided. The second motion planning problem is to find a path from *Goal-1* to *Goal-2*. Since the second goal configuration is placed in the unknown area, the planner considers the whole space to extend the two trees. The planner does not detect the invisible obstacle space (the black regions) from the initial state due to limitations in sensor coverage and creates a plan that goes through one of the obstacles. When the robot starts executing the planned path and gets closer to the obstacle, it detects the collision, so it stops and triggers re-planning, which looks for a new path from the current configuration to *Goal-2*.

V. Double Weighted Reachability Graph

We propose a data structure that we call *Double Weighted Reachability Graphs* because labels are assigned to both vertices and edges. The graph is used by the robot to keep track of seen and unseen areas of the environment and reason about what frontier voxel to explore next.

The graph $\mathcal{G}$ is a tuple $(V, E, \mathcal{F}, \mathcal{C})$. The vertices V and edges E are added incrementally during navigation as follows. Whenever the robot executes a motion plan, the algorithm considers voxels over the path equally spaced by the sensor's range d_{max}. We use bounding boxes to collect the frontier around a path. Starting from the first voxel c in the path, the algorithm creates a bounding box around it, with fixed side corresponding to $2d_{max}$, and considers the voxels which lie in the box. If there are frontier voxels within the bounding box, a vertex v corresponding to c is added to the graph. The vertex indicates that the area around c needs further exploration. A vertex for the goal is always added to the graph. When a vertex v is created, it is labelled by the function $\mathcal{F}(v) = \langle \mathcal{W}, exp\rangle$, where $\mathcal{W}$ represents the collected frontier voxels in the bounding box associated with c (with their distances to the nearest obstacles) and exp is a binary flag, initially set to false, that is used to indicate whether the area around c has been fully explored (i.e. all frontier voxels in the box for c have been explored) or not.

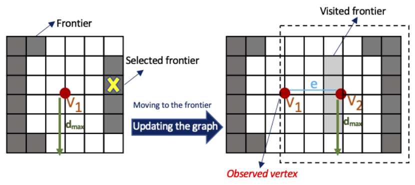

Fig. 4: Insertion of a vertex and an edge in the graph (2D view).

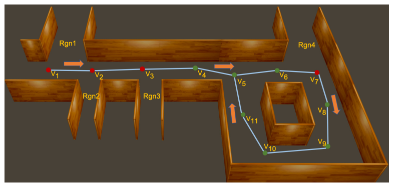

Fig. 5: Double Weighted Reachability Graph construction.

While collecting the frontier voxels associated with the next voxel c' and generating the vertex v', the algorithm also identifies whether some frontier voxels associated with an existing vertex v do not belong to the frontier any longer (because the robot has now observed some of the unknown voxels beyond it). Vertices with this characteristic are called *observed vertices*. The algorithm adds an edge (e, v, v') between the new vertex v' and each observed vertex v, which indicates that going from v to v' has allowed the robot to explore previously unseen space. The edge $e \in E$ is labelled with a cost $\mathcal{C}(e)$, which is the path length between the corresponding vertices. The example in Figure 4 shows how this process is performed. The robot is located at v_1 and the frontier voxels around this vertex are highlighted in grey. When the robot navigates to the selected frontier voxel (in yellow), the algorithm creates v_2, which corresponds to the new voxel, and identifies that some of the frontier voxels associated with v_1 do not belong to the frontier any longer. v_1 becomes an *observed* vertex, and an edge is created between the two vertices. The dashed line includes the visited and new frontier voxels identified when the robot is at v_2. The graph supports the following operations:

- *Insert*(e, v, v'): inserts the edge (e, v, v'), along with the related information, to $\mathcal{G}$;
- *Query*(e, v, v'): retrieves the latest information about (e, v, v') by applying the functions $\mathcal{F}$ and $\mathcal{C}$;
- *Update*(v): updates the information about vertex v.

The graph is equipped with an implementation of the A* algorithm, which is a heuristic method to find the cheapest path between two nodes [19]. The Euclidean distance between the voxels corresponding to the vertices is used as the heuristic function. We implement a query $\mathcal{P}(v, v')$ to compute the cheapest path between a start vertex v and the goal vertex v' based on the graph information.

The example in Figure 5 illustrates how some components of the reachability graph are dynamically constructed. The robot starts navigating the environment from v_1. The other vertices are added to the graph incrementally during the robot's motion towards v_{11}. A vertex v is red when the flag exp is false (i.e. there are unexplored frontier voxels associated with v), while it is green when exp is true (i.e. the frontier voxels associated with v have been completely explored). The arrows represent the direction of the robot's motion. When the robot reaches v_2, v_1 becomes an *observed* vertex because the robot has now visited some of the frontier voxels previously associated with v_1. An edge is added between v_1 and v_2. Both vertices are red because there are still unexplored frontier voxels associated with them (corresponding to regions *Rgn1* and *Rgn2*). When adding v_{11}, the robot identifies that the unexplored frontier voxels associated with v_5 are now all visited and there is also no frontier left corresponding to vertex v_{11}. So, an edge is added between v_5 and v_{11}, they become green and the robot detects that it has fully visited this part of the graph. However, vertices v_1, v_2, v_3 and v_7 are still red (i.e. flagged as not fully explored). At this point, the robot looks for the next best region to explore among those around v_1, v_2, v_3 and v_7 (i.e. *Rgn1, Rgn2, Rgn3*, and *Rgn4*) by using the approach that we will describe in Section VII. The example shows that a key feature of our graph-based reasoning is that it detects regions that do not need further exploration. When the robot has fully explored the current vertex and there is no frontier left around it, it concludes that it has reached a dead end or a visited branch and moves on.

VI. Utility function

We use a utility function that measures the amount of information corresponding to frontier voxels and the paths to them to choose the best one to visit next. The function takes into consideration *information gain*, *safety constraints*, and *path length*. The information gain is computed using Shannon's entropy [20], which is a popular approach to measure uncertainty. Given a discrete random variable X with possible outcomes $x_1, ..., x_n$ having probability $p(x_1), ..., p(x_n)$, the entropy of X is given by:

$$\mathcal{H}(X) = -\sum_{i=1}^{n} p(x_i)\, log\, p(x_i) \tag{1}$$

with logarithm base 2. If X is a binary random variable, the function becomes 1 with probability p and 0 with probability $1 - p$. In the 3D occupancy map, a voxel c is represented by a random variable, which assumes probability $p(c) < 0.5$ when it is free ($c \in \mathcal{M}_{free}$), and $p(c) > 0.5$ when occupied ($c \in \mathcal{M}_{occupied}$). The entropy of a voxel c is minimized when $p(c) = 0$ or $p(c) = 1$ (no more information can be gained) and has its greatest value when $p(c) = 0.5$ (the voxel is unknown). For a configuration q, the uncertainty of the map, using the binary entropy, is expressed as:

$$\mathcal{H}(q) = -\sum_{c \in \mathcal{M}_{rc}} \{p(c)\, log\, p(c) + (1 - p(c))\, log\, (1 - p(c))\} \tag{2}$$

where $\mathcal{M}_{rc}$ are the voxels along the rays casting outward from the position of the robot, 360° around it, with the length of the ray corresponding to d_{max}. For the voxels in each direction, the function keeps considering them until the first voxel assumes a probability greater than 0.5, which indicates that the voxel is occupied and the robot cannot see beyond it in that direction. Figure 6 displays a frontier candidate and the ray-casting mechanism when $d_{max} = 4m$. The figure shows that, when a ray hits an occupied voxel c where $p(c) > 0.5$, it stops there. The approach computes the information gain of the frontier candidates given the probability values of the voxels hit by the rays.

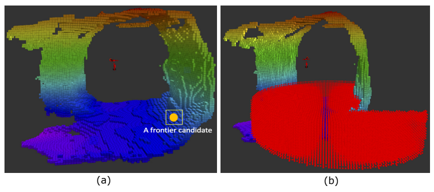

Fig. 6: (a) Frontier candidate; (b) Corresponding ray-casting procedure (a red dot is shown in every voxel that a ray passes through).

We compute a total entropy function $\mathcal{H}_{total}$ based on the following information: i) a frontier voxel c; ii) its corresponding position, indicated as $c.q$; and iii) a subset Q' of the path Q from the initial position to $c.q$, with Q' being of length $L(Q')$ and containing only those positions that maintain a distance of d_{max} between each other. We use d_{max} because, in Eq. (2), the entropy is computed on voxels at a maximum distance d_{max} from the frontier candidate. The total entropy function $\mathcal{H}_{total}(c)$, which returns a numeric value indicating how much new relevant information about the space can be acquired by visiting c, is defined as:

$$\mathcal{H}_{total}(c) = \alpha\mathcal{H}(c.q) + \beta \sum_{i=0}^{L(Q')-2} \mathcal{H}(Q'.q_i) \tag{3}$$

where $Q'.q_i$ are the configurations along the path Q' and $\alpha, \beta \in \mathbb{R}$ are customizable factors that can be use to give more weight to the frontier voxel at the end of the path (first term) or the entire motion (second term).

Lastly, we define the *utility function* of a frontier voxel c:

$$U(c) = \delta\mathcal{H}_{total}(c)\ exp(-\gamma L(Q))\ (1 + \zeta\mathcal{D}) \tag{4}$$

where $L(Q)$ is the length of the path Q and D is the distance between c and the nearest occupied voxel. The parameters $\delta, \gamma, \zeta \in \mathbb{R}$ can be tuned according to the user's preferences to give more or less importance to the various parts of the utility function, which combines the total information gain with the length of the motion and the safety distance from the obstacles. We can set ζ to a high value when the robot navigates within narrow areas to maintain it in the centre of those areas. The function $U(c)$ is called by the exploration manager to measure the utility value of frontier candidates.

VII. Exploration Manager

We use a high-level exploration manager to integrate all the system's components together and choose the best frontier candidate to visit next. The manager exploits a set of reasoners and a sampling-based mechanism as follows.

The *region-based reasoner* $\mathcal{R}_{rgn}$ aims to identify the type of region in which the robot is currently located using a ray-casting technique. It casts rays along the coordinate frame of the robot. If the rays hit occupied voxels along the $x-axis$ and $z-axis$ or the $y-axis$ and $z-axis$ up to a threshold distance, the robot detects that it is surrounded by obstacles and, in consequence, it is exploring a narrow region. This region is labelled as a *branch*. If close obstacles are not detected, the region is labelled as a *void*.

The *contextual reasoner* $\mathcal{R}_{cr}$ computes whether the robot has enough time to execute the motion plan and go back home. Given the information in the reachability graph $\mathcal{G}$, $\mathcal{R}_{cr}$ estimates the flight time to home by considering the cost of the current path and the velocity of the robot. If there seems to be no sufficient time to go back home, $\mathcal{R}_{cr}$ calls the motion planner to see if there is a shorter path to go back home that satisfies the time. If not, the robot goes back home immediately by following a motion plan that satisfies the energy constraints.

The *frontier sampler* $\mathcal{S}_{\mathcal{F}}$ samples the current frontier voxels to generate a subset of them. This is done to reduce complexity as the system performs several calculations based on the frontier. A minimum distance d_s between two samples is chosen and a candidate sample is discarded if its distance from one of the previously selected samples is less than d_s. $\mathcal{S}_{\mathcal{F}}$ employs a collision detection module to confirm that the candidates are geometrically feasible. The sampler mechanism is based on the environmental features. If the robot is in a void, the sampler considers the whole frontier set. If the robot is exploring a branch, the sampler is biased to consider the frontier voxels located in the centre of the area for safety and stability reasons. The bias is achieved as follows. While collecting the frontier voxels, the algorithm computes their distance from the nearest occupied voxel. It then calculates the average and maximum distances of the frontier voxels from the obstacles and, based on them, it sets a higher rate of sampling for the frontier voxels that lie between the average and the maximum distance.

The sampling density depends on the reachability graph's information. Given a vertex v, the sampler finds the neighboring vertices of v, $\mathcal{N}(v) = \{v_0, ..., v_i\}$, and samples the frontier by choosing the same number of samples in the frontiers associated with each of the vertices $v_0, ..., v_i$. If the number of voxels in the frontiers of all vertices in $\mathcal{N}(v)$ is below a given threshold or there are no unexplored vertices, the sampler concludes that the robot is back to a known area or has arrived to a dead-end area. In this case, the algorithm uses a ranking function *Rank*$(\mathcal{G}, v)$ to sort the graph's vertices that are not fully explored based on the shortest distance between them and the current vertex v by using the A* query $\mathcal{P}$ (see Section V). The sampler selects the i nearest candidates and, for each candidate, calculates the samples in its frontier. The sampling rate f_s is higher for the vertices that are closer to v:

$$Rank(\mathcal{G}, v) = \langle v'_0, ..., v'_i \rangle \implies f_s(v'_0) > ... > f_s(v'_i) \tag{5}$$

This mechanism provides an efficient way to sample the frontier where it is useful instead of in the whole environment. For instance, when the robot is at v_{11} in Figure 5, all the neighboring vertices are fully explored. Vertices v_1, v_2, v_3, and v_7, which are not fully explored yet, are the vertices where sampling occurs. Vertices v_1 and v_7 lead the robot towards big regions similar in size, and vertices v_2 and v_3 guide the robot towards small regions similar in size. The sampling rate is higher for v_7 and v_3 than for v_2 and v_1 ($f_s(v_7) > f_s(v_3) > f_s(v_2) > f_s(v_1)$), because they are closer to the current position of the robot.

Algorithm 1 describes our *informed exploration planner*, which integrates all the modules together. The algorithm gets the sensing data as a point cloud $\mathcal{PC}$ and outputs the occupancy map $\mathcal{M}$. The functions GetInitMap [line 2] and InitGraph [line 3] construct the initial map and the reachability graph, respectively. An iterative process is implemented to explore the environment [lines 4-17]. The mission is completed if there are no frontier voxels to explore, which is detected by EndMission [line 5], or if ContReas, which implements $\mathcal{R}_{cr}$, returns false [line 10] (i.e. the robot has no sufficient resources to explore further). In both cases, GoHome [line 16] brings the robot back to the home position.

At each iteration, the robot reasons about the current environmental features by calling RgnReasoning [line 6], which uses $\mathcal{R}_{rgn}$, and samples the frontier by calling the sampler $\mathcal{S}_{\mathcal{F}}$ (Sample [line 7]). Then, the motion planner is triggered (MotionPlanning [line 8]) to find if collision-free motions exist from the current position to the frontier samples. The planner discards samples with non collision-free motions. The function BestAct chooses the best frontier candidate [line 9]. This procedure performs two tasks: i) it calculates the utility function in Eq. (4) for each sample; and ii) it selects the one that maximizes the value $s^* = \arg\max_{s \in S} U(s)$. Then, if there is enough battery, the robot traverses the path via the function Navigate, which implements the navigation module described in Section IV [line 11]. The reachability graph is expanded using ExpGraph [line 14], which uses the functions *Insert* to add a new edge, *Query* to retrieve the latest graph information, and *Update* to update it.

Algorithm 1: Informed exploration

```
   inputs  : Point cloud $\mathcal{PC}$
   outputs: Occupancy map $\mathcal{M}$
1  Mission ← False
2  $\mathcal{M}$ = GetInitMap($\mathcal{PC}$)
3  $\mathcal{G}$ = InitGraph($\mathcal{M}$)
4  while !Mission do
5      if !EndMission($\mathcal{G}$) then
6          $\kappa$ = RgnReasoning($\mathcal{M}$)
7          $\mathcal{S}$ = Sample($\mathcal{M}$, $\mathcal{G}$, $\kappa$)
8          $\mathcal{S}$ = MotionPlanning($\mathcal{M}$, $\mathcal{S}$)
9          $s$ = BestAct($\mathcal{M}$, $\mathcal{S}$, $\kappa$)
10         if ContReas($\mathcal{M}$, $\mathcal{G}$, $s$) then
11             $\mathcal{M}$ = Navigate($\mathcal{M}$, $\mathcal{P}$, $s$)
12         else
13             break
14         $\mathcal{G}$ = ExpGraph($\mathcal{M}$, $s$)
15     else
16         GoHome($\mathcal{M}$, $\mathcal{G}$)
17         Mission ← True
18 return {$\mathcal{M}$, $\mathcal{G}$}
```

VIII. Experimental Results

We evaluate our approach against several navigation and exploration problems. We developed our motion planning algorithms using Open Motion Planning Library (*OMPL*) [21]. We use the simplification method offered by *OMPL* to shorten and smooth paths. We integrated *Octomap* [18] into our system for the 3D mapping. All modules are implemented in Robot Operating System (*ROS*). All experiments were run on an Intel Core i9-9980HK 2.40 GHz CPU machine with 32 GB memory. We use 360° LiDAR sensors to capture

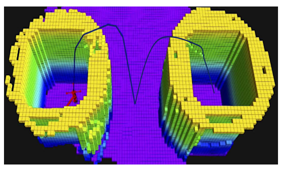
Fig. 7: Final map and solution of the two-shaft example. The dimensions of each shaft are $2.5 \times 2.5 \times 4\ m$.

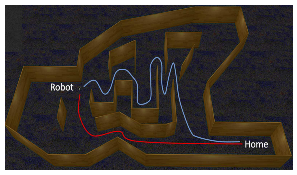

Fig. 8: Room scenario ($40 \times 30 \times 4\ m$). Path from current pose to home (our approach in red; *RRT-Connect* in blue);

the environment. We use the Prometheus drone [2] to run our experiments, which is a reconfigurable platform that can reduce its diameter to fit through boreholes. Our system is however independent from the platform used as long as the assumptions laid out in Section III are verified.

We use the following values for the parameters, which are based on our practical experience with running exploration missions in mines: a) maximum robot's velocity of $0.25\ m/s$; b) $d_{min} = 0.1\ m$ and $d_{max} = 4.0\ m$; c) maximum motion planning time $10.0\ s$; d) $\alpha = 1.0$, $\beta = 0.2$, $\delta = 1$, $\gamma = 0.5$, $\zeta = 0.2$ in wide areas and $\zeta = 0.6$ in narrow passages.

A. Results on Navigation

We consider several challenging navigation scenarios to study the benefits of our navigation components. Figure 7 shows the final map and solutions of a two-shaft scenario. The robot is required to navigate from *shaft-1*, on the right, to *shaft-2*, on the left. The initial robot's knowledge is limited to the parts of *shaft-1* surrounding its initial position. Given the unknown goal region, the navigation planner looks for a path $Q \in \mathcal{M}_{free} \cup \mathcal{M}_{unknown}$. The planner re-plans as soon as a collision is detected. Figure 8 illustrates a complex room with 35 walls. From the current position, the robot needs to go back home to be able to exit the space by the end of the mission. As the home position is known, our adaptive navigation planner tries to find $Q \in \mathcal{M}_{free}$. The motion solution is shown in red color. The path in blue color indicates the motion's solution found by the original *RRT-Connect* planner augmented with re-planning; in the latter case, the robot maps a bigger area to reach home.

Considering the scenarios above, we perform experiments on the following classes of problems: a) *2-SHAFT*: the two-shaft example solved by our planner; b) *X-OBS*: the room examples with an increasing number X of obstacles (walls with various dimensions that make the problems challenging by blocking straight-line paths between the initial and goal locations), with $X \in \{15, 25, 35\}$; c) *MINE*: the massive mine represented in Figure 1, which has been reconstructed by using real data collected by Network Rail in the UK. Table I reports the average navigation performance for all the problems for five trials. We compare our adaptive navigation, *A-NAV*, with the original *RRT-Connect*, *R-NAV*. For the *2-SHAFT* scenario, our planner re-plans several times because the goal is located in an unknown area. For the same reason, in this case, our technique and the original *RRT-Connect* with re-planning have the same performance, which however indicates that our modifications of the *RRT-Connect* do not introduce inefficiencies. On the other hand, for the room examples (*X-OBS*), our approach reduces motion planning time, execution time and number of replanning occurrences. It also provides shorter paths than the original *RRT-Connect* in all cases. This is due to the fact that our approach exploits the known space whenever possible and so avoids triggering replanning due to unobserved obstacles. Figure 9 compares the total navigation time (sum of motion planning and execution times) of our planner against the original one for the room trials. In all cases, our planner is more time-efficient than the original one. For the *MINE* problem, we decrease the sensing range d_{max} from to $4m$ to $2m$ to make the problem more challenging as the robot cannot easily identify walls. Although both planners need to re-plan, our planner is more efficient than *R-NAV*.

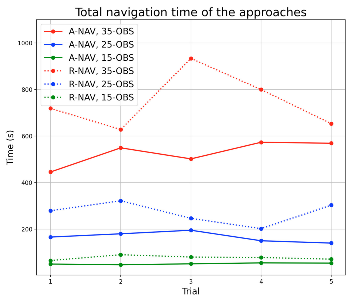

Fig. 9: Comparison on navigation time between our adaptive navigation (*A-NAV*) and *RRT-Connect* (*R-NAV*) for the *X-OBS* problems.

To deal with unreliable state estimation, in our experiments, while the robot is executing a path, the re-planning module keeps comparing the robot's pose, computed by the localization system, with the pose of the waypoint where the robot is navigating to, which is calculated by the planner. If the deviation between the two poses becomes higher than a given threshold, the re-planning conclude that the robot is drifting and is not following the path properly. Accordingly, the plan fails and the planner re-plans from the current pose to the waypoint of interest to make the robot converge to the path.

Problem	MP (s)	Execution (s)	Re-plan	Path L (m)
2-SHAFT	0.62	125.30	17.80	19.66
A-NAV,15-OBS	0.04	51.40	0.0	6.61
R-NAV,15-OBS	0.11	76.80	3.80	8.74
A-NAV,25-OBS	0.10	166.20	0.0	21.63
R-NAV,25-OBS	1.50	269.00	14.40	27.97
A-NAV,35-OBS	0.19	527.60	0.0	33.19
R-NAV,35-OBS	3.07	743.49	20.60	74.00
A-NAV, MINE	0.08	98.95	2.20	17.50.
R-NAV, MINE	0.63	156.66	14.6	25.98

TABLE I: The average performance of navigation in terms of motion planning (*MP*) time, execution time, number of re-planning calls, and path length for the navigation problems.

B. Results on Exploration

We tested our approach against several cave and mine scenarios whose CAD models have been generated using real data. Figures 10 and 11 represent the map of two caves, *cave-1* and *cave-2*, whose models have been provided to us by Network Rail (UK). These are challenging environments due to the presence of uneven structures, long paths, narrow passages, multiple loops, and dead-end branches. To explore them, the robot needs to cope with these challenges and accounts for battery constraints.

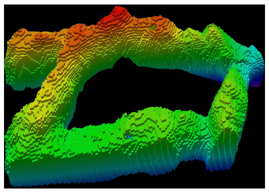
Fig. 10: The *cave-1* map.

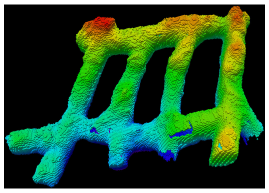
Fig. 11: The *cave-2* map.

We define several problems for both caves with different exploration time and map resolutions (r). The problems that correspond to Figure 10 are: a) *Problem-1* with 600 s and $r = 0.1\ m$; and b) *Problem-2* with 1500 s and $r = 0.2\ m$. The problems that correspond to Figure 11 are: c) *Problem-3* with 600 s and $r = 0.1\ m$; and d) *Problem-4* with 1900 s and $r = 0.2\ m$. Table II reports the performance for all these problems. For each scenario, more exploration time allows the robot to map larger unknown spaces, which also results in increasing the size of the reachability graphs. When a maximum exploration time is set, the robot needs not only to explore but also to verify that there is enough time to get back home while mapping new areas.

Problem	V	E	Nav. (s)	Exp. (s)	Vol. (m^3)
Problem-1	32	52	508	595	858
Problem-2	90	161	1306	1434	2604
Problem-3	55	109	452	591	859
Problem-4	207	553	1603	1853	3062

TABLE II: The performance of the problems in terms of number of vertices and edges in the reachability graph, navigation time, exploration time, and explored volume.

We compared our informed exploration (*IE*) approach against several state-of-the-art techniques. The first is an implementation of the frontier-based exploration (*FE*) by Yamauchi et al. [4]. The *FE* technique goes through the entire 3D map to collect frontier voxels and then selects the nearest frontier candidate to explore unknown areas. To evaluate our utility function, we developed an approach (indicated as *UE*) in which we switch off our reachability graph, use the complete map to reason about the frontier and implement the utility function proposed by Dai et al. [12]. Figure 12 shows explored volume versus time for all the approaches and different map resolutions. For $r = 0.2\ m$, the voxels are bigger and our approach maps larger unknown areas. The problems become more challenging when $r = 0.1\ m$. All approaches need more time to collect frontier voxels and find collision-free paths as the voxels' size gets smaller. Our approach explores the space much more effectively than the other ones in all cases.

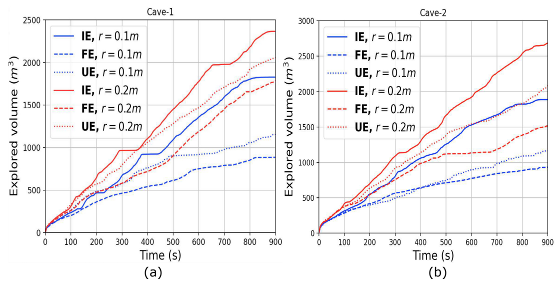

Fig. 12: Comparison on total explored volume between our approach (*IE*), the frontier-based exploration (*FE*) and the utility-based approach (*UE*) by Dai et al. [12].

We also tested *IE* against *UE* for a large mine (bounding box of $225 \times 117 \times 62\ m$) located in Snailbeach (UK), which has been reconstructed using real data collected by Geoterra (UK). Figure 13 depicts the 3D map ($800s$ and $r = 0.1\ m$) of the mine after exploration by our approach. Figure 14 shows the performance of the techniques in terms of time, explored volume, and distance to the nearest obstacle. Our approach explores larger areas in the mine than *UE*, while maintaining a higher distance from the obstacles, which reflects its safety.

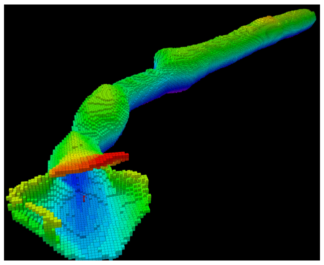
Fig. 13: 3D map of Snailbeach mine (UK) after exploration.

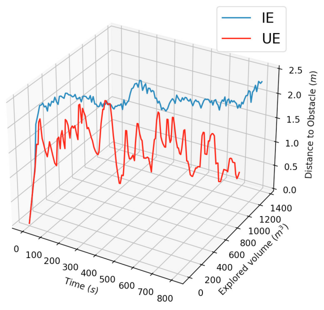

Fig. 14: A comparison between *IE* and *UE* on Snailbeach mine.

Finally, we evaluated our exploration approach on the big mine represented in Figure 1 and compared our solution with another state-of-the-art technique, indicated as (*GE*) [17]. Figure 15 plots the results of the exploration tasks with different sensing ranges and when $r = 0.2\ m$. In all cases, our planner efficiently explores bigger areas than *GE*.

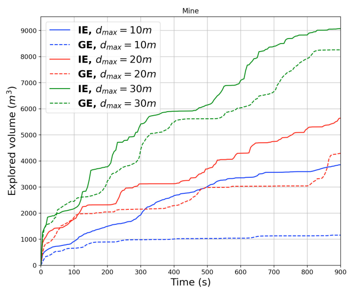

Fig. 15: A comparison between *IE* and *GE* on the mine in Figure 1.

IX. Conclusion

We present an autonomous exploration and navigation approach for subterranean environments. Our approach enables the robot to maximize information gain while preserving safety. Our experimental results show the power of our techniques and their superiority over related methods.

In future work, we will investigate the use of asymptotically optimal planners (e.g. RRT* [22]) for navigation as they have been successfully employed in underwater exploration [23]. We also intend to work on efficient frontier detection for 3D SLAM based on occupancy grid sub-maps [24]. Finally, we plan to conduct extensive real-world experiments using the Prometheus drone [2] in real mines in the UK as soon as COVID-19 restrictions for confined spaces are lifted.

References

[1] A. Akbari, P. Chhabra, U. Bhandari, and S. Bernardini, "Intelligent exploration and autonomous navigation in confined spaces," in *International Conference on Intelligent Robots and Systems*. IEEE, 2020.
[2] L. Brown, R. Clarke, A. Akbari, U. Bhandari, S. Bernardini, P. Chhabra, O. Marjanovic, T. Richardson, and S. Watson, "The design of prometheus: A reconfigurable uav for subterranean mine inspection," *Robotics*, vol. 9, 2020.
[3] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE international conference on robotics and automation (ICRA)*, vol. 2. IEEE, 2000.
[4] B. Yamauchi, "A frontier-based approach for autonomous exploration." in *cira*, vol. 97, 1997.
[5] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *International Conference on Intelligent Robots and Systems*, 2017.
[6] M. Faria, I. Maza, and A. Viguria, "Applying frontier cells based exploration and Lazy Theta* path planning over single grid-based world representation for autonomous inspection of large 3D structures with an UAS," *J. of Intelligent & Robotic Systems*, vol. 93, 2019.
[7] R. Almadhoun, A. Abduldayem, T. Taha, L. Seneviratne, and Y. Zweiri, "Guided next best view for 3d reconstruction of large complex structures," *Remote Sensing*, vol. 11, 2019.
[8] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters." in *Robotics: Science and Systems*, vol. 2, 2005.
[9] E. Palazzolo and C. Stachniss, "Information-driven autonomous exploration for a vision-based mav," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, 2017.
[10] E. Vidal, N. Palomeras, K. Istenič, J. D. Hernández, and M. Carreras, "Two-dimensional frontier-based viewpoint generation for exploring and mapping underwater environments," *Sensors*, vol. 19, 2019.
[11] E. Vidal, N. Palomeras, K. Istenič, N. Gracias, and M. Carreras, "Multi-sensor online 3d view planning for autonomous underwater exploration," *Journal of Field Robotics*, vol. 37, 2020.
[12] A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas, and S. Leutenegger, "Fast frontier-based information-driven autonomous exploration with an mav," *arXiv preprint*, 2020.
[13] D. Silver, D. Ferguson, A. Morris, and S. Thayer, "Topological exploration of subterranean environments," *Journal of Field Robotics*.
[14] A. Morris, D. Ferguson, Z. Omohundro, D. Bradley, D. Silver, C. Baker, S. Thayer, C. Whittaker, and W. Whittaker, "Recent developments in subterranean robotics," *Journal of Field Robotics*.
[15] F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "A multi-modal mapping unit for autonomous exploration and mapping of underground tunnels," in *IEEE aerospace conference*, 2018.
[16] C. Papachristos, S. Khattak, F. Mascarich, T. Dang, and K. Alexis, "Autonomous aerial robotic exploration of subterranean environments relying on morphology–aware path planning," in *International Conference on Unmanned Aircraft Systems*. IEEE, 2019.
[17] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "Graph-based path planning for autonomous robotic exploration in subterranean environments," in *International Conference on Intelligent Robots and Systems*. IEEE, 2019.
[18] 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, 2013.
[19] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2002.
[20] R. M. Gray, *Entropy and information theory*. Springer Science & Business Media, 2011.
[21] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, 2012.
[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, 2011.
[23] J. D. Hernandez, E. Vidal, M. Moll, N. Palomeras, M. Carreras, and L. E. Kavraki, "Online motion planning for unexplored underwater environments using autonomous underwater vehicles," *J. Field Rob.*, vol. 36, no. 2, 2019.
[24] J. Orsulic, D. Miklic, and Z. Kovacic, "Efficient dense frontier detection for 2-d graph slam based on occupancy grid submaps," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, 2019.