SphereMap: Dynamic Multi-Layer Graph Structure for Rapid Safety-Aware UAV Planning

Tomáš Musil [ID], Matěj Petrlík [ID], Martin Saska [ID]

Abstract—A flexible topological representation consisting of a two-layer graph structure built on-board an Unmanned Aerial Vehicle (UAV) by continuously filling the free space of an occupancy map with intersecting spheres is proposed in this letter. Most state-of-the-art planning methods find the shortest paths while keeping the UAV at a pre-defined distance from obstacles. Planning over the proposed structure reaches this pre-defined distance only when necessary, maintaining a safer distance otherwise, while also being orders of magnitude faster than other state-of-the-art methods. Furthermore, we demonstrate how this graph representation can be converted into a lightweight shareable topological-volumetric map of the environment, which enables decentralized multi-robot cooperation. The proposed approach was successfully validated in several kilometers of real subterranean environments, such as caves, devastated industrial buildings, and in the harsh and complex setting of the final event of the DARPA SubT Challenge, which aims to mimic the conditions of real search and rescue missions as closely as possible, and where our approach achieved the 2nd place in the virtual track.

Index Terms—Autonomous Vehicle Navigation, Planning under Uncertainty, Mapping, Aerial Systems: Perception and Autonomy, Robotics in Hazardous Fields

MULTIMEDIA MATERIALS

The paper is supported by the multimedia materials available at mrs.felk.cvut.cz/papers/ral2022spheremap.

I. INTRODUCTION

THANKS to the rising affordability and rapid development of onboard software, autonomous UAVs are increasingly being deployed in Search and Rescue (SAR) missions [1] and inspection of industrial [2] or historical [3] structures. The high mobility of UAVs makes them an ideal tool for first-responders and rescue teams. A quick flight through an unknown and potentially dangerous environment provides the operator with quick situational awareness that allows better decision making and avoids risking human lives when exploring uncharted environment with unknown safety conditions.

Systems for accomplishing autonomous tasks by UAVs are available in literature [4], [5] and can be deployed in various

Authors are with the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, 166 36 Prague 6, {tomas.musil|matej.petrlik|martin.saska}@fel.cvut.cz
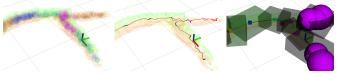
Fig. 1: The SphereMap created during the final event of DARPA SubT Challenge in the tunnel section of the course. The free space is filled with spheres (left), forming an undirected graph with spherical centers as vertices. Middle image shows portals (black) between pairs of roughly convex regions with cached intra-region paths (red). The bounding boxes of the LTVMap is shown on the right with darker colors representing less explored/inspected regions, and purple spheres marking exploration goals.

scenarios, such as inspection [6], firefighting [7], [8] and radiation localization [9]. These applications often rely on GNSS for UAV localization and additional sensors for mostly simple reactive behavior. On the other hand, the navigation of UAVs in large unstructured environments without reliable GNSS data is still an open topic, one in which advancements have been recently accelerated thanks to the DARPA SubT tunnel circuit [10], urban circuit [11], and cave circuit [12].

Finding a collision-free path through any environment requires a representation of the free space and obstacles. The commonly employed volumetric occupancy grids based on octree data structures (e.g. the OctoMap [13] or UFOMap [14] implementations) encode this information as the states of cells. Even though octree-based occupancy maps are memory efficient and allow fast access times, naive planning on an occupancy grid may be unfeasible in real time for long distance planning (global navigation), especially with complex topology typical for subterranean environments. Timeout-based workarounds exist [11], which output the path with an endpoint of the lowest Euclidean distance from the initial goal. This approach avoids long idle times while waiting for the plan, but might prevent reaching the goal altogether in cases of non-trivial topology. Many approaches to the global navigation problem utilize building a map exploiting the topology of the environment for rapid path planning, such as [15]–[17]. How-

ever, most of these methods build the topological map after an initial run-through of the environment and not incrementally throughout the mission. Thus they cannot be deployed on a first responder UAV into a previously unmapped environment, which is a typical requirement for SAR scenarios. Those that are built incrementally often do not incorporate obstacle clearance information into the map for safety-aware planning. As such, even those methods cannot be used when the UAV needs to quickly decide whether to explore a nearby narrow passage with a high risk of collision, or to explore a farther goal for which a safer path exists.

This letter presents an incrementally built graph structure (called SphereMap), consisting of a sparse topological graph of roughly convex regions of free space, and an underlying dense graph of intersecting spheres. This structure can be built continually on-board a UAV during the mission, and allow it to make decisions about mission goals, weighing the safety of paths, their length, and remaining flight time in real time. Furthermore, the structure can generate a lightweight representation (called LTVMap) of the environment consisting of a graph of convex shapes representing the environment's volume to share with other robots or human operators at any point.

A. Related works

A lightweight topological map representation designed for quick path planning was presented in [16]. In this work, the authors grow convex clusters of free space from a Euclidean Signed Distance Field (ESDF) created offline after one run of the robot through the environment. The obtained maps are then used for rapid path planning in the environment. Since the clusters are convex, the path is planned on a navigation graph where the vertices are points (called portals) in which the clusters coincide. Edges are straight-line paths between two portals through a cluster. The approach requires a fixed voxel size for the clusters, which can pose problems when the environment has both very narrow, and large open areas. Moreover, the offline creation of an ESDF prevents exploratory deployment in an unknown environment and cannot handle dynamically changing environments.

Another approach of creating a sparse graph representation of the environment is found in [15]. A 3D Voronoi diagram is created from an ESDF map and converted to a sparse graph, which enables planning that is orders of magnitude faster than A* or RRT* [18] on the ESDF. The sparse graph is, however, not built dynamically, but only after an initial exploration mission. Clearance information is also not incorporated into the graph, though the authors state this is to be done in their future work.

A popular method of constructing a topological graph for the purpose of autonomous exploration, called GBPlanner, was presented in [19]. In this approach, a global graph is incrementally built during the mission and is used for homing and repositioning the UAV to new frontiers when a dead end is reached. The downside of this approach is that it does not enable easy querying of a path between any two points in the occupancy map, and also that the graph is built to provide the shortest path home. This does not allow for weighing path length vs. path safety when planning towards new frontiers.

A similar approach was presented in [17], where the authors incrementally build a sparse topological graph of coverage points from which they cast rays through an occupancy map and create a convex polyhedron at each coverage point. These coverage points are also merged if a centroid of a polyhedron is contained inside another polyhedron. This is an incremental and very lightweight topological representation, but it does not contain implicit clearance information, and the authors present path planning based only on distance.

A technique for safety-aware path planning was presented in [20], where the authors construct the path optimization criterion to contain information about the distance from obstacles up to a maximum distance d_{max} at a given point on the path. In this approach, the authors solve the optimization problem by constructing a graph at a distance of d_{max} from obstacles, or a lower distance between two obstacles. The path is found across this graph, so that the UAV travels either at d_{max} from obstacles, through a ridge between two obstacles, or in a straight line between two obstacles. The method does not scale well with the environment size as only a local map with a size of 3–4 times the sensor range is maintained and thus it does not allow global navigation in a larger environment. In contrast to our method, which was verified in both outdoor and indoor/subterranean environment, the authors also only assume an outdoor environment.

For planning outside the local map boundaries, [21] builds a sparse graph containing two types of vertices: the history of the robot's past positions to quickly navigate through an already explored environment, and the visibility nodes that represent potential pathways to new space. The safety of local trajectories is ensured by including a collision probability penalization. This penalization is based on the pose estimate covariance in the cost function of the motion primitive selection. The presented method lacks a technique for rerouting the sparse graph in a dynamic environment, where a previously free path can suddenly become obstructed. The evaluation was also realized only in a simplified simulation and the method does not consider common real world issues such as imperfect sensor data, map drift, control error, and external disturbances.

B. Contributions

In context of the above discussed literature, the contributions of this work can be summarized as follows:

- a novel dynamically built graph structure (SphereMap) with clearance information that allows for rapid safety-aware path planning between any two points in known space and can handle arbitrarily structured and changing environments,
- a method for quickly generating a lightweight volumetric graph representation (LTVMap) of the environment, that can be shared with other robots using low-bandwidth communication and carry mission-critical information,
- experimental evaluation showing that planning on the SphereMap can be orders of magnitude faster than state-of-the-art approaches for long-distance planning,
- validation of the proposed methods in real world scenarios, most importantly in the DARPA SubT competition, where robots were deployed into both real and simulated

previously unvisited environments [22].

II. PROBLEM DESCRIPTION

The planning problem tackled in this paper is the computation of long-distance paths through known space represented by an occupancy map in real time, while also weighing the path length and path safety. The UAV is assumed to be modeled as a sphere with a radius r_{min}. The dynamic constraints of the UAV are not considered, as the obtained path is used as an input into a local planner. Formally, we define the planning problem as finding a solution path $\mathbf{x}(k)$ parametrized by $k \in [0,1]$ which minimizes the criterion

$$J(\mathbf{x}(k)) = L(\mathbf{x}(k)) + Z(\mathbf{x}(k)), \qquad (1)$$

divided into the path length $L(\mathbf{x}(k))$ and path risk $Z(\mathbf{x}(k))$ defined as

$$L(\mathbf{x}(k)) = \int_0^1 \|\mathbf{x}'(k)\| \, dk \qquad (2)$$

$$Z(\mathbf{x}(k)) = \xi \int_0^1 \|\mathbf{x}'(k)\| \left[\max(0, d_{max} - d(\mathbf{x}(k)))\right]^2 dk \quad (3)$$

while satisfying

$$\min_{k \in (0,1)} d(\mathbf{x}(k)) > r_{min} \qquad (4)$$

where $d(\mathbf{x})$ is the nearest obstacle distance at point $\mathbf{x}$ and ξ is a parameter specifying the weight of the path safety on the overall cost of the path. The cost cutoff distance d_{max} is determined based on the UAV's size and the maximum possible deviation caused by disturbances and control errors.

The second problem to be solved is the ability to efficiently generate a lightweight topological-volumetric representation of the visited part of the environment at any time, which can carry mission-relevant information about sections of the environment to share with the rest of the robot and human team. We further specify the following requirements and assumptions for the problem:

1) Mission: to explore the given environment and share exploration and mission-relevant information to other UAVs and human operators when communication is available.
2) Environment: unstructured and unknown dynamic subterranean or outdoor environment with narrow passages, sections with unreliable low bandwidth communication, and without access to GNSS.
3) UAV Platform: assumed to carry depth sensors (LiDAR or depth cameras) and build an occupancy map of the environment onboard while localizing itself using a SLAM method. The UAV is modeled as a sphere with no physical collision tolerance of the platform, such as in [23].
4) Planning time: the method must be able to find paths to goals which are hundreds of metres away. This has to be done in tens of milliseconds, so that the UAV can evaluate paths to all potential goals at any time, as sometimes a safe, but distant goal might be preferable to a nearer goal that is reachable only by a risky passage.

III. THE SPHEREMAP

In this section, the proposed multi-layer graph structure, the method of incrementally building the structure, and the method for quickly extracting a lightweight topological-volumetric map from the structure are described.

A. Topological map representation

We propose representing the topology of the environment as an undirected graph of spheres $\mathcal{G}$ divided into roughly convex segments $\mathcal{S}$. Each node ν on the sphere graph is assigned a position $\mathbf{p}_\nu$, a radius r_ν equal to the nearest obstacle distance from $\mathbf{p}_\nu$, and a label σ_ν signifying which segment the node belongs to. We connect every two spheres for which the radius of their intersection is bigger than r_{min}. For planning, the graph of spheres, illustrated in Fig. 3, is similar to an ESDF used in [24] as it inherently encodes the distance from obstacles. However, in contrast to [24], the proposed map representation is not bound to a fixed resolution grid, where narrow passages can appear blocked due to discretization errors when the voxel size is set too high, and where the planning can be greatly slowed down if the voxels are too small. The graph of spheres is also stored in an octree structure for quick nearest-neighbor querying and short insertion and deletion time.

The graph of spheres is divided into roughly convex segments of nodes $\mathcal{S}$, which form a topological graph. The convexity condition is not strict, since the paths between portals are collision-free intra-segment paths introduced in Sec. III-B3, and not straight lines as in [16]. We add a connection to the topological graph for every two regions which have some connected nodes, and store the pair of nodes $\nu_1 \in \sigma_1, \nu_2 \in \sigma_2$ with the largest sphere-sphere intersection radius. We call these pairs *portals* between segments, as in [16]. To allow for efficient long-distance planning, we also iteratively compute and store optimal (according to (1)) paths between every two portals of a given segment, as illustrated in Fig. 3.

B. Growing the SphereMap

The proposed structure is built incrementally during a mission at a low rate of 2 Hz. The graph of spheres is grown so that the spheres cover the entire reachable known space of the environment, and redundant nodes are pruned from the sphere graph to speed up computations. During a single update iteration, only the part of the map inside a cube with a pre-defined size centered on the UAV is modified. As input, the update iteration only needs the current position of the UAV and the most recent occupancy information about the space in the cube (meaning the information whether a given point in the cube is occupied, free, or unknown). Any onboard-running occupancy mapping method can be used to provide this information, such as the efficient occupancy map implementations OctoMap [13] or UFOMap [14].

To enable fast querying of the nearest obstacle distance at any given point, we compute a k-D tree from occupied points (e.g. centroids of occupied voxels of an occupancy grid) in the cube of nearby space. To prevent the spheres from expanding into unknown space, we also add points that lie at the boundary of free and unknown space into the k-D tree. Finding these boundary points (referred to as frontiers) is non-trivial in most
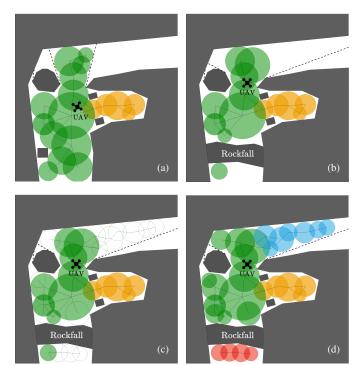
Fig. 2: Outline of one iteration of the SphereMap update algorithm. The dashed lines represent the boundary of known free space and unknown space. (a): Map at the end of the previous iteration. (b): Sphere radius update and pruning of unsafe and redundant nodes. (c): Sampling, creating and connecting new nodes into the graph. (d): Dividing disconnected (●) segment into two segments (●,●), expanding existing segments and growing a new segment (●).

occupancy maps and usually slower than constructing the k-D tree, but it is necessary for safe path planning and also useful for exploration.

The SphereMap is built in approximately constant-time iterations, which converge to an optimal distribution of spheres with increasing number of iterations in a given area. As such, it can occasionally happen that the spheres do not cover the entire reachable space, which can cause some points to be unreachable through the SphereMap or cause the found paths to be far from optimal. However, this can be handled by mission control. A single SphereMap update iteration, further outlined in Fig. 2, consists of three steps in this order:

1) Recomputing and pruning step

Because the safe planning requires the most recent occupancy information, we first recompute the radius of every nearby node $\nu \in \mathcal{N}_{near}$ and update the connections of the nodes for which the radius has changed, as shown in Alg. 1. The radius computation is done by querying the nearest point distance from the current obstacle k-D tree K for each node in the updated part of the sphere graph. Since we assume a dynamic environment, some nodes can become untraversible. This can occur due to obstacles moving in the environment or noisy sensor data. These nodes are pruned from the graph.

Furthermore, some nodes can grow in size due to more free space being uncovered, making other nodes obsolete. We mark a node as redundant if there exists a larger node, which covers at least a certain percentage of the node's volume. All redundant nodes are then pruned.

Algorithm 1: SphereMap radius update and pruning step. Only nodes $\mathcal{N}_{near}$ in a bounding box around the UAV are updated.

```
1: procedure RECOMPUTESPHEREMAP
2:     Input:
3:         G                                          ▷ SphereMap graph
4:         K                                          ▷ obstacle k-D tree
5:         r_min                        ▷ minimal distance from obstacles
6:     for ν ∈ N_near do      ▷ nodes to be updated in a cube around UAV
7:         r_ν ← nearestObstacleDistance(p_ν, K)
8:         if r_ν < r_min then
9:             G ← G \ ν
10:        else
11:            updateNodeConnections(ν, G)
12:    for ν ∈ N_near do
13:        if isRedundant(ν, G) then
14:            G ← G \ ν
```

2) Expansion step

In the next step, shown in Alg. 2, new non-redundant spheres are added to the graph by sampling points in the vicinity of the UAV. The number of sampled points S heavily influences the runtime of this step. Sampling a large amount of points will ensure the reachable space is filled with spheres, but it demands more computational resources. A sparser sampling method will speed up the expansion step, but in the newly seen large areas, it might cause some reachable space to not be covered by the sphere graph. In our implementation, we chose to sample one point per each voxel of the used occupancy octree, and also cast rays in random directions from the UAV and sample points along the rays.

Algorithm 2: SphereMap expansion step. New nodes that are not inside already existing nodes are added to the graph at sampled positions.

```
1: procedure EXPANDSPHEREMAP
2:     Input:
3:         G                                          ▷ SphereMap graph
4:         K                                          ▷ obstacle k-D tree
5:     P_s ← samplePointsInNearFreeSpace(p_uav)
6:     for p_s ∈ P_s do
7:         ν_new ← initNode()
8:         p_ν_new ← p_s
9:         r_ν_new ← nearestObstacleDistance(p_s, K)
10:        σ_ν_new ← ∅
11:        if r_ν_new ≥ r_min and not isRedundant(ν_new, G) then
12:            G ← G ∪ ν_new
13:            updateNodeConnections(ν_new, G)
14:            for ν' ∈ connectedNodes(ν_new, G) do
15:                if isRedundant(ν', G) then
16:                    G ← G \ ν'
```

3) Segmentation step

In the last step, shown in Alg. 3, we update the segmentation of the map. Each node ν in the graph, except for the nodes that were created in the current expansion step, belongs to some region σ_ν. In this step, we first assign each new node to a segment. This is done by first expanding the existing regions using a flood fill over the sphere graph, limiting the maximum bounding sphere of the region to a radius r_{exp}. Then, we iterate over the remaining newly added nodes without an assigned segment, and initialize and grow new segments (also bounded by r_{exp}) from them, until all nodes belong to some segment.

Since this approach can create a very granular segmentation of the graph, we attempt to merge all the regions that have been modified in this step with their neighboring regions. A merge is allowed only if the resulting region does not exceed the bounding sphere radius r_{merge}, and also if the centers of the segments are mutually visible. By using these conditions

and setting r_{exp} very low compared to r_{merge}, we can achieve roughly convex segmentation with low computational effort.

Additionally, we recompute and store the optimal paths $\mathcal{C}_{(p_1,p_2)}$ with respect to the criterion (1) between each pair of portals (p_1, p_2) inside every segment that has been modified in the iteration. A segment can become modified either by the segmentation step or by the update and pruning step, meaning that some of its nodes were altered or some edges between nodes were deleted. These paths, henceforth called intra-segment paths, are computed using the A* algorithm with the path cost and heuristic functions described in Sec. IV, and then cached for rapid on-demand long-distance path planning.

Algorithm 3: SphereMap segmentation step. Intra-segment paths between portals are stored in cache $\mathcal{C}$ for faster planning.

```
 1: procedure SEGMENTSPHEREMAP
 2:    Input:
 3:       S_G                                    ▷ SphereMap graph segments
 4:       r_exp           ▷ maximum radius of segment expansion bounding sphere
 5:       r_merge         ▷ maximum radius of segment merging bounding sphere
 6:    S_near ← getNearSegments(S_G)
 7:    S_near ← splitDisconnectedSegments(S_near)
 8:    for σ ∈ S_near do                        ▷ try expanding existing segments
 9:       σ ← expandSegment(σ, r_exp)
10:       recomputeSegmentPortals(σ)
11:    for ν ∈ N_near, σ_ν = ∅ do                ▷ grow new segments
12:       σ_new ← initSegment(ν)
13:       σ_new ← expandSegment(σ_new)
14:       S_near ← S_near ∪ σ_new
15:       recomputeSegmentPortals(σ_new)
16:    for (σ_1, σ_2) ∈ getAdjacentSegmentPairs(S_near) do    ▷ merge segments
17:       if canMergeSegments(σ_1, σ_2, r_merge) then
18:          σ_1 ← mergeSegments(σ_1, σ_2)
19:          S_near ← S_near \ σ_2
20:    for σ ∈ S_near do                         ▷ recompute intra-segment paths
21:       if segmentWasAltered(σ) then
22:          for (p_1, p_2) ∈ getSegmentPortals(σ) do
23:             C_(p_1,p_2) ← computePathInSegment(σ, p_1, p_2)
```

C. Lightweight map creation and sharing

Another key feature of segmentation of the sphere graph is that the roughly convex regions of space can be represented by convex 3D shapes (in our implementation, we used 4DOF bounding boxes determined by width, height, depth, and z axis rotation, as shown in Fig. 7). After each update iteration of the SphereMap, we compute and store the best-fitting convex shape envelope of a segment's spheres for any nearby segments that were modified. Therefore a topological-volumetric graph representation of the environment (LTVMap) can be extracted at any time. Each vertex of the LTVMap graph represents the pose and dimensions of a segment's envelope, with edges added to the graph for every two connected segments.

This representation is more lightweight than both the SphereMap itself and the underlying occupancy map, and can therefore be used in low-bandwidth situations, in which sharing the full information is not possible. For SAR missions, additional information (such as the percentage of surfaces covered by the UAV's cameras or information about frontiers in a given region) can easily be appended to each region. This information can then be shared with other robots during the mission for cooperation purposes, and also with human operators to whom this map representation can provide a good overview of the environment and the mission progress [22].

D. Computational complexity

The leading factors in every update iteration's complexity include the determination of the sphere's radius around a given point as $\mathcal{O}(\log|K|)$, and checking the nearest spheres for possible intersections, which is $\mathcal{O}(n_c \log|\mathcal{N}_{near}|)$. Herein, n_c is the average number of nearest spheres that need to be checked, depending on the density of spheres and the largest sphere in the update bounding box. Hence, the entire update is $\mathcal{O}((|\mathcal{N}_{near}|+|S|)(n_c \log|\mathcal{N}_{near}|+\log|K|))$. By only updating in a small area around the UAV, the update time does not grow with the total number of spheres in the SphereMap, rather it depends on the sphere density in the local area. By pruning the sphere graph, the method keeps n_c and $|\mathcal{N}_{near}|$ reasonably low, thus also keeping the runtime low.

IV. PLANNING IN THE SPHEREMAP

For planning paths on a graph or grid with the optimization criterion (1), the criterion first needs to be discretized. Let us assume a segment of a path between two nearby points $\mathbf{p}_1, \mathbf{p}_2$ that have the distance from the nearest obstacle equal to $r_{\mathbf{p}_1}, r_{\mathbf{p}_2}$ respectively. Naturally, the length increment can be written as

$$\Delta L_{\mathbf{p}_1\mathbf{p}_2} = \|\mathbf{p}_1 - \mathbf{p}_2\|, \tag{5}$$

but the computation of the safety increment ΔZ is trickier if we only access obstacle distances from a k-D tree. The integral is approximated by defining the increment as

$$\Delta Z_{\mathbf{p}_1\mathbf{p}_2} = \xi\left[\max\left(0, d_{max} - \frac{r_{\mathbf{p}_1} + r_{\mathbf{p}_2}}{2}\right)\right]^2 \|\mathbf{p}_1 - \mathbf{p}_2\|. \tag{6}$$

During planning, we also ensure that there are no points along the line between the points $\mathbf{p}_1$ and $\mathbf{p}_2$ (using the size of the intersection of spheres around $\mathbf{p}_1$ and $\mathbf{p}_2$ with radii $r_{\mathbf{p}_1}$ and $r_{\mathbf{p}_2}$) that would be closer to obstacles than the minimal allowed distance r_{min}. We use A* with a transition cost between two points given by

$$g(\mathbf{p}_1, \mathbf{p}_2) = \Delta L_{\mathbf{p}_1\mathbf{p}_2} + \Delta Z_{\mathbf{p}_1\mathbf{p}_2} \tag{7}$$

and a heuristic

$$h(\mathbf{p}_2) = \|\mathbf{p}_2 - \mathbf{p}_{goal}\| \tag{8}$$

for planning in the SphereMap.

The main advantage of the segmentation and caching of the optimal paths between portals is that whenever a path between points A and B is queried, we only need to compute paths across the sphere graph from the start and end points to the portals of the segments the points are in. The rest of the path can be planned using the cached intra-segment paths, as shown in Fig. 3.

This technique provides a major speedup of 2 orders of magnitude over computing the path on the entire sphere graph, as shown in Tab. I, but the resulting paths are forced to pass through the portals, through which the optimal path might not pass.
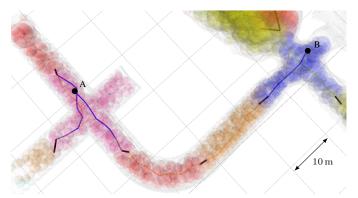
Fig. 3: Illustration of the proposed planning approach. When a path from A to B is queried, only a few paths (——) need to be computed across the sphere graph, namely the paths leading from A to the portals (——) of the segment that A is in, and then paths from portals of B's segment to B, when the portals are reached. The cached intra-segment paths (thin colored lines) between portals are used for the rest of the computation.

V. EXPERIMENTS

This section aims to validate the proposed method on multiple simulated environments and compare SphereMap with state-of-the-art methods. Furthermore, we analyze the runtime of the SphereMap growing steps and show that the lightweight map generated from the SphereMap is substantially smaller than a binary occupancy octree even with a lowered resolution, while keeping details about the environment connectivity and the size and geometric shape of each region. We also demonstrate deployment of the proposed method onboard a real UAV platform in both subterranean and outdoor environment.

A. Planning analysis

To demonstrate the ability of the SphereMap to quickly generate paths optimizing the criterion (1), we conducted two experiments in simulated environments where we compare the proposed planning approach with the following state-of-the-art planning methods:

- *A* through an occupancy grid:* A simple solution for path planning is to run A* in an occupancy grid with a fixed voxel size. Using the transition cost and heuristic defined in Sec. IV and a granular voxel size, this approach can find paths that are closer to the optimum defined in (1) than the SphereMap approach, but as we show in Tab. I, such a method is unusable on large distances due to the long computation times.
- *RRT* through an occupancy grid:* Additionally, we compare the proposed planning approach with a popular planning algorithm, the asymptotically optimal RRT* [18]. The path used for the comparison is the initial solution, i.e., the path is not further optimized by continuing the tree growth after finding the solution, which would generate solutions closer to the optimum, at the cost of increased computation time.

Firstly, we compared the planning methods in a situation where the UAV needs to find paths to a large amount of goals that can be up to $300\,\mathrm{m}$ away from the UAV. We used a standard laptop computer, a $0.2\,\mathrm{m}$ resolution occupancy octree for the SphereMap building and a $0.8\,\mathrm{m}$ resolution for the grid-based A* computation. The safety-based planning parameters

were set to $\xi = 7, d_{max} = 2\,\mathrm{m}, r_{min} = 0.8\,\mathrm{m}$. Results of this comparison are presented in Tab. I, and found paths are shown in Fig. 4 (a). It is evident from the results, that the state-of-the-art methods are too slow for such situations and do not meet the requirements set by this letter.

In the second planning experiment, we analyzed the path length and risk as defined in (1) for paths found by the compared methods to a single goal. To better outline the differences between the approaches, a $0.4\,\mathrm{m}$ resolution was used for the grid-based A*. The same safety-based planning parameters as in the first experiment were used. The results of this analysis can be seen in Fig. 4 (b) and Tab. II. As can be seen, our approach outperforms the state-of-the art methods by orders of magnitude, while still being close to the optimal solutions found by grid-based A*.
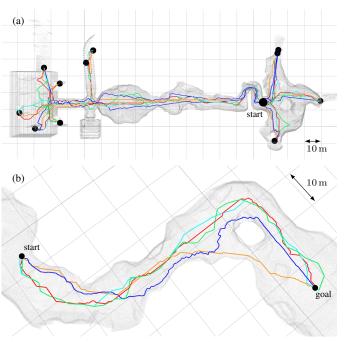

Fig. 4: Visualization of path planning to multiple potential exploration goals (a) and to a single goal (b) using the following methods: path generated by A* on the SphereMap using cached intra-segment paths (——); the path planned through the entire sphere graph (——); grid-based A* using cost function (1) (——); grid-based A* with path cost equal to path length on the occupancy grid with voxel resolution $r = 0.8\,\mathrm{m}$ (a) and $r = 0.4\,\mathrm{m}$ (b) (——); RRT* path (——).

Planner	Time [ms]	Found paths	Mean Cost [-]
Grid A*	59524	11/11	270.36
Grid A* length-only	58795	11/11	477.39
RRT*	96830	6/11	385.05
SphereMap	1868	11/11	**257.77**
SphereMap caching	**22**	11/11	322.73

TABLE I: Quantitative results for the multiple-goal path planning experiment in a simulated environment, shown in Fig. 4 (a). The cost of each path was computed by summing the discretized transition cost (7). The computation time is summed for all found paths. Even with a $10\,\mathrm{s}$ planning timeout RRT* did not manage to find 5 of the paths. Compared to the state-of-the-art methods which find the paths in tens of seconds, planning over the entire sphere graph of the SphereMap runs for only $1.8\,\mathrm{s}$, and if the intra-segment path caching is used, the paths are found in only $22\,\mathrm{ms}$.

Planner	Time [ms]	Length [m]	Risk [-]	Cost [-]
Grid A*	28064	101.11	33.88	134.99
Grid A* length-only	15376	80.33	376.21	456.54
RRT*	544	96.79	413.61	510.40
SphereMap	56	**104.14**	**45.98**	**150.12**
SphereMap caching	**0.496**	110.46	49.37	159.83

TABLE II: Quantitative results for the single-goal planning experiment in a simulated cave environment, shown in Fig. 4 (b). The cost of each path was computed by summing the discretized transition cost (7). Our method is 3 orders of magnitude faster than the initial solution found by RRT* and 5 orders of magnitude faster than grid-based A* with a 0.4 m resolution. The A* optimization of the path length only naturally finds the shortest path, but as shown in Fig. 4 (b), the path leads through the riskier narrow passage.

B. Iteration runtime analysis

To demonstrate the light computational load of building the SphereMap, we conducted an experiment in a simulated exploration mission through a vast cave environment identical in scale to the ones shown in the other experiments, and measured the runtime of individual parts of the SphereMap growing process.

The results of this experiment can be seen in Fig. 5. The experiment demonstrates that the update iteration of the SphereMap reaches a mean computation time of approx. 150 ms. It is important to note that the overall growth runtime depends heavily on the expansion step and can be sped up by using a sparser sampling method.
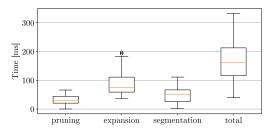

Fig. 5: Boxplot illustration of runtimes of individual parts of one SphereMap growing iteration during an exploration mission in a large cave environment. The outlier computation times usually occur when the UAV peeks around a corner and discovers a vast free-space area.

C. Map compression analysis

To show the usefulness and light weight of the LTVMap generated from the SphereMap, we conducted an experiment in a simulated cavernous environment spanning hundreds of meters and compared the LTVMap with the original binary occupancy octree (0.2 m resolution), from which the SphereMap is built. We also compare the size of the LTVMap and an occupancy octree that is generated by lowering the original octree to a resolution of 1 m. The development of the size of the three representations throughout the mission is shown in Fig. 6. The LTVMap and the original map at the end of the experiment are visualized in Fig. 7.

According to Fig. 6, the lowered-resolution occupancy octree could seem like a good choice for volumetric information sharing. However, the resolution lowering can cause narrow passages to become obstructed in the resulting maps, which could give other robots false information about the environment's connectivity. The LTVMap on the other hand keeps this connectivity information while also being smaller than the lowered-resolution map.

One disadvantage of the shown LTVMap is that since we use 4DOF bounding boxes for the region shape approximation, there is a large percentage of space in the bounding boxes, that is not free in the original occupancy octree. The amount of misclassified space could be reduced by choosing different bounding shapes, such as polygons or 6DOF boxes, which would better approximate the segments' shapes at the expense of higher computational complexity and higher communication bandwidth. Nevertheless, for the purposes of decentralized cooperative planning, we found the 4DOF boxes to be sufficient in the DARPA SubT Challenge [22].
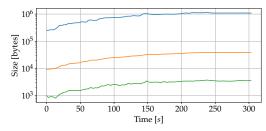

Fig. 6: Development of the size of the LTVMap (——), the binary occupancy octree (——) used by the SphereMap (implemented by the OctoMap library [13]), and an octree generated by lowering the original octree to a 1 m resolution (——) in the experiment described in Sec. V-C.
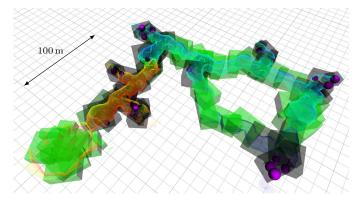

Fig. 7: Visualization of the original occupancy octree (pointcloud colored by a heatmap encoding the point height) and the LTVMap generated from the SphereMap built from it in the experiment described in Sec. V-C. The purple spheres signify potential exploration goals appended to the LTVMap and the color of the blocks signifies how close they are to unexplored space.

D. Real world deployment

The SphereMap described in this letter was successfully used in the final round of the DARPA SubT Challenge (Fig. 1) for the purpose of navigation, cooperative exploration, goal pathfinding, and map sharing among robots and with an operator. The control and state estimation part of the deployed system is described further in [4]. The SphereMap in a more crude form (without pruning of nodes) was used also in the virtual circuit of the DARPA SubT Challenge Finals, where our team CTU-CRAS-NORLAB won the 2nd place [22].

Lastly, the SphereMap was also tested on a large outdoor environment where a UAV autonomously explored the space around a house surrounded by heavy shrubbery, crossed a bridge into a vast open field, and safely returned home. With a minimal amount of fine-tuning (only lowering the number

of sampled points to compensate the large free space volume), the total SphereMap growth runtime was on average around $200\,\mathrm{ms}$ on a real-world UAV platform equipped with an Intel Core i7 processor running SLAM, state estimation, visual detection and control in addition to SphereMap, which demonstrates that the proposed method is usable on common hardware even in large outdoor areas. Video outputs from the outdoor exploration experiment and from both the virtual and physical DARPA SubT Final events are available in the supplementary materials.

VI. CONCLUSIONS

A method for flexible in-flight building of a segmented graph of spheres with cached intra-segment paths designed for rapid safety-aware path planning and generating a lightweight topological-volumetric graph representation of the explored environment on-demand was proposed in this letter. The proposed method does not require any fine-tuning of the map resolution based on apriori knowledge of the environment. Thus, for achieving safe and flexible multi-robot exploration of unknown unstructured environment, the proposed approach requires only a map of occupied, free and unknown space in the vicinity of the UAV that can be efficiently provided by the onboard sensors at every iteration of the map update. Planning in the proposed SphereMap is designed to find paths that optimize both safety and length. The topological-volumetric LTVMap generated from the proposed structure takes up significantly less bandwidth than the occupancy map even with a lowered resolution, while still keeping information about the connectivity of the environment and its approximate shape, which is crucial for real-time multi-robot exploration of subterranean environment, where reliable communication among robots is a bottleneck.

Among numerous simulations and statistical analyses using world models from scanned caves, mines and industrial complexes, the proposed method was also verified onboard real UAV platforms during multiple exploratory missions and was able to find paths 3 orders of magnitude faster than state-of-the-art planning algorithms such as grid-based A* or RRT*, navigate robots in the environment safely, and enable cooperation among the UAVs sharing the lightweight maps. The performance of the entire UAV system using the proposed method was confronted with competition consisting of leading robotic teams selected for finals of DARPA SubT Challenge. The challenge provided trustworthy conditions for comparison of different solutions in real-world conditions of demanding cave, mine and subway domains. The proposed method successfully navigated a team of UAVs through dust-filled narrow tunnels with dynamically blocked paths in the systems track and was a crucial part of the system deployed in the virtual track, gaining the 2nd place.

REFERENCES

[1] J. Delmerico et al., "The current state and future outlook of rescue robotics," *Journal of Field Robotics*, vol. 36, no. 7, pp. 1171–1191, 2019.

[2] R. Carvalho et al., "A UAV-based framework for semi-automated thermographic inspection of belt conveyors in the mining industry," *Sensors*, vol. 20, no. 8, p. 2243, 2020.

[3] P. Petráček et al., "Dronument: System for Reliable Deployment of Micro Aerial Vehicles in Dark Areas of Large Historical Monuments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2078–2085, April 2020.

[4] T. Baca et al., "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 26, pp. 1–28, May 2021.

[5] J. L. Sanchez-Lopez et al., "Aerostack: An architecture and open-source software framework for aerial robotics," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 332–341.

[6] V. Krátký et al., "Autonomous Aerial Filming with Distributed Lighting by a Team of Unmanned Aerial Vehicles," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7580–7587, October 2021.

[7] V. Spurny et al., "Autonomous Firefighting Inside Buildings by an Unmanned Aerial Vehicle," *IEEE Access*, vol. 9, pp. 15872–15890, January 2021.

[8] J. Quenzel et al., "Autonomous Fire Fighting with a UAV-UGV Team at MBZIRC 2020," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 934–941.

[9] T. Baca et al., "Gamma Radiation Source Localization for Micro Aerial Vehicles with a Miniature Single-Detector Compton Event Camera," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2021, pp. 338–346.

[10] M. Petrlík et al., "A Robust UAV System for Operations in a Constrained Environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2169–2176, April 2020.

[11] V. Krátký et al., "An autonomous unmanned aerial vehicle system for fast exploration of large complex indoor environments," *Journal of Field Robotics*, vol. 38, no. 8, pp. 1036–1058, May 2021.

[12] P. Petracek et al., "Large-Scale Exploration of Cave Environments by Unmanned Aerial Vehicles," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7596–7603, October 2021.

[13] A. Hornung et al., "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[14] D. Duberg et al., "UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6411–6418, 2020.

[15] H. Oleynikova et al., "Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, 2018.

[16] F. Blöchliger et al., "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.

[17] T. Dang et al., "Graph-based Path Planning for Autonomous Robotic Exploration in Subterranean Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3105–3112, 2019.

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

[19] T. Dang et al., "Graph-based Path Planning for Autonomous Robotic Exploration in Subterranean Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3105–3112, 2019.

[20] H. Cover et al., "Sparse Tangential Network (SPARTAN): Motion planning for micro aerial vehicles," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2820–2825, 2013.

[21] M. Collins et al., "Efficient Planning for High-Speed MAV Flight in Unknown Environments Using Online Sparse Topological Graphs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, inproceedings, pp. 11450–11456.

[22] M. Petrlik et al., "UAVs Beneath the Surface: Cooperative Autonomy for Subterranean Search and Rescue in DARPA SubT," *preprint arXiv:2206.08185*, 2022, submitted to Field Robotics Special Issue: DARPA Subterranean Challenge.

[23] P. De Petris et al., "Resilient collision-tolerant navigation in confined environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 2286–2292.

[24] H. Oleynikova et al., "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1366–1373, 2017.