Science Robotics
AAAS

Supplementary Materials for

Representation granularity enables time-efficient autonomous exploration in large, complex worlds

C. Cao *et al.*

Corresponding author: C. Cao, ccao1@andrew.cmu.edu

Sci. Robot. **8**, eadf0970 (2023)
DOI: 10.1126/scirobotics.adf0970

The PDF file includes:

Other Supplementary Material for this manuscript includes the following:

Supplementary Materials

Additional Comparison Results in Simulation

In addition to NBVP, GBP and MBP, we further compared our method against two additional baseline methods including a topology-based strategy, specifically GVGExp (*15*), and an information gain-based strategy, denoted as Information-Theoretical Exploration (ITE) (*51*). We evaluated ITE in all five simulation environments, and evaluated GVGExp only in the tunnel, indoor, and forest environments, since its implementation utilized a 2D occupancy grid and thus could only handle flat environments.

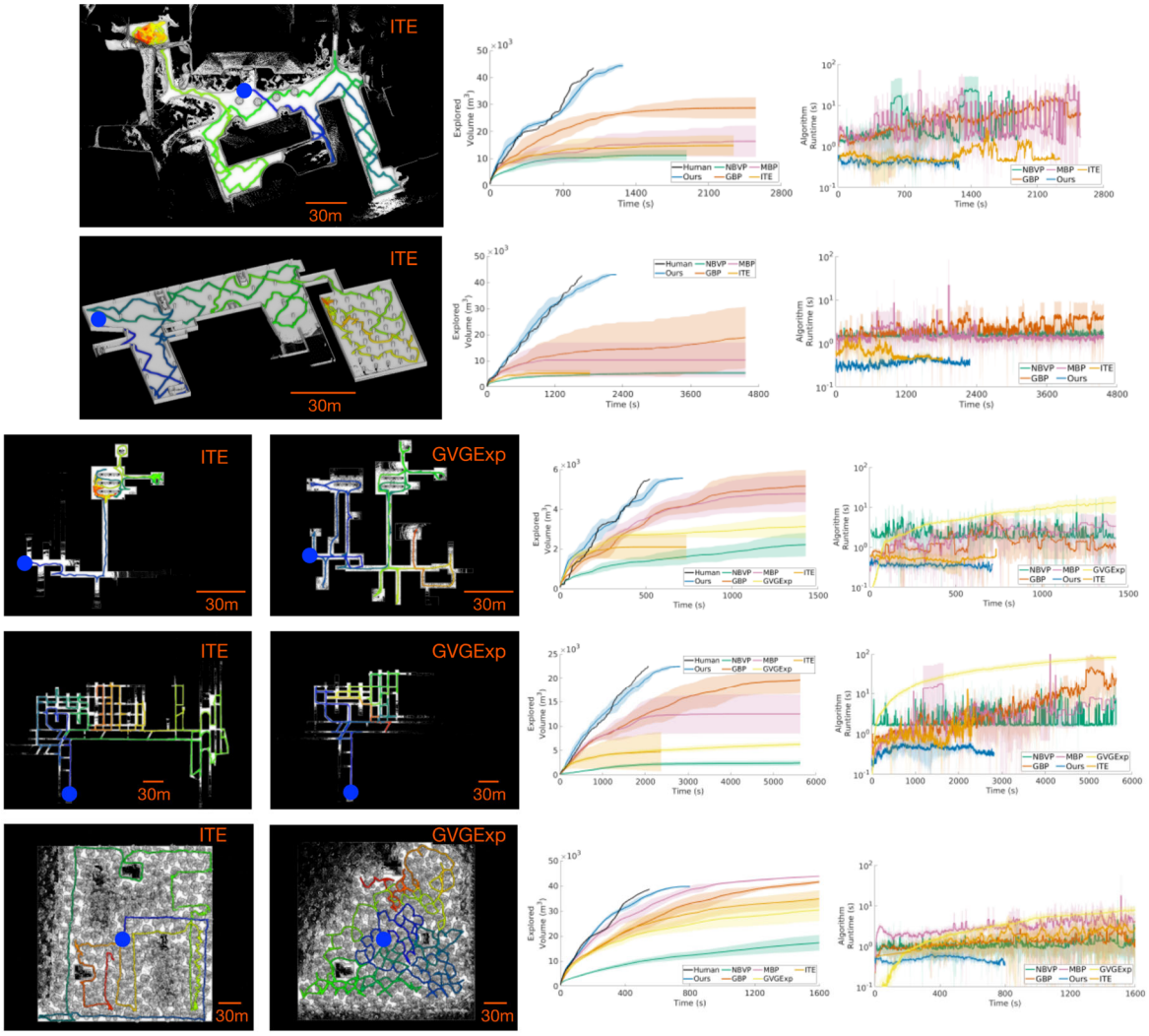

Figure S1 ***Additional ground-based autonomous exploration in simulation****. The layout is similar to Figure 3. Each row shows the resulting point cloud maps and vehicle trajectories in an environment, followed by the plots on exploration efficiency and algorithm runtime over time. The starting points of the exploration are indicated by the blue dots. The*

robot trajectories are color-coded to indicate temporal progression, starting with blue and ending in red. The curves in the plots show mean values and the shaded areas represent the standard deviation over 10 trials.

Figure S1 presents the results. In the experiments, both GVGExp and ITE demonstrated superior performance compared to NBVP. However, when considering exploration efficiency, our proposed methods, GBP, and MBP outperformed both GVGExp and ITE. Notably, GVGExp exhibited competitive exploration efficiency in the initial stages of the run, but its runtime rapidly increased as explored areas became larger, leading to slower exploration progress later in the run, as shown in the algorithm runtime plots in Figure S1. This issue arised because the Generalized Voronoi Graph (GVG), which was closely related to the structural layout of the environment, could experience substantial changes when the map was updated. Instead of maintaining the same graph over time through extensive data association between changes, GVGExp reconstructed the GVG periodically. As a result, the runtime of GVGExp increased considerably as the size of the explored area grew. Additionally, the GVG constructed in cluttered environments could be overly complex, causing the exploration to become easily diverted towards nooks and crannies, as shown by the vehicle trajectory in the forest environment.

ITE, on the other hand, had a relatively low runtime. However, ITE adopted a greedy approach to selecting the direction with maximum information gain. Consequently, it tended to persistently explore a direction until it reached an obstacle, at which point it turned 90 degrees to explore sideways and repeated the process. This behavior was motivated by the fact that the information gain was highest on the sideways, compared to the path that the vehicle had traversed and the obstacles ahead. Further, ITE frequently encountered challenges in detecting unexplored regions, resulting in premature termination of the exploration. This was due to its limited local map, and its robot-centric motion primitives that lacked the ability to plan long-term routes in complex environments. From the experiments, it is evident that this approach is better suited for mostly open and easily accessible environments that do not require extensive route planning through numerous obstacles, such as the forest environment shown above, and the cave environments presented in (*51*).

It is worth noting that the NBVP, GBP, and MBP all belong to the family of RRT/RRG-based approaches and represent a series of continuous improvements over time. Furthermore, GBP and MBP had been empirically verified in real-world complex scenarios, such as the SubT challenge, and are therefore considered state-of-the-art. We therefore extensively evaluated our proposed method against them.

Overall time efficiency and runtime comparison

Table S1. ***Comparison of averaged exploration time efficiency****. ϵ is defined as the average volume (m^3) explored per second throughout the entire run, and relative efficiency r_ϵ is defined as the ratio compared to the best human practice, or our method when human practice is not available.*

Test	Vehicle Type	Environment	Human		NBVP		GBP		MBP		Ours	
			ϵ	r_ϵ	ϵ	r_ϵ	ϵ	r_ϵ	ϵ	r_ϵ	ϵ	r_ϵ
1	Ground	Real-world Garage	-	-	3.0	0.24	6.8	0.55	5.3	0.43	**12.3**	**1.0**

2	Ground	Real-world Indoor	-	-	0.56	0.33	0.8	0.47	0.72	0.42	**1.72**	**1.0**
4	Ground	Simulation Campus	**43.7**	**1.0**	11.1	0.25	12.1	0.28	11.7	0.27	**36.7**	**0.84**
5	Ground	Simulation Garage	**25.3**	**1.0**	0.90	0.04	6.0	0.24	2.8	0.11	**19.6**	**0.78**
6	Ground	Simulation Indoor	**10.6**	**1.0**	1.2	0.11	3.6	0.34	3.3	0.31	**8.6**	**0.81**
7	Ground	Simulation Tunnel	**10.9**	**1.0**	0.50	0.05	3.1	0.28	4.0	0.37	**8.4**	**0.78**
8	Ground	Simulation Forest	**70.3**	**1.0**	9.3	0.13	15.4	0.22	21.8	0.31	**54.8**	**0.78**
9	Aerial	Simulation Campus	-	-	15.7	0.079	23.9	0.12	26.9	0.13	**199.7**	**1.0**

Test	Vehicle type	Environment	ITE		GVGExp	
			ϵ	r_ϵ	ϵ	r_ϵ
4	Ground	Simulation Campus	11.8	0.27	-	-
5	Ground	Simulation Garage	4.3	0.17	-	-
6	Ground	Simulation Indoor	3.6	0.34	0.60	0.06
7	Ground	Simulation Tunnel	3.3	0.30	1.1	0.10
8	Ground	Simulation Forest	20.8	0.30	9.9	0.14

Table S1 compares the exploration efficiency of all methods. On average, our method was 80 percent more efficient than the other methods in all tests.

*Table S2. **Comparison of algorithm runtime**. Each entry shows the mean value and standard deviation over the entire run in all trials. All methods were evaluated based on a 4.1GHz i7 CPU.*

Test	Vehicle Type	Environment	NBVP (s)	GBP (s)	MBP (s)	ITE (s)	GVGExp (s)	Ours (s)
1	Ground	Real-world Garage	0.88 ± 1.5	2.6 ± 2.5	12.8 ± 35.6	-	-	**0.42** ± 0.16
2	Ground	Real-world Indoor	2.6 ± 7.6	1.8 ± 1.7	2.4 ± 23.9	-	-	**0.13** ± 0.05
4	Ground	Simulation Campus	4.6 ± 12.6	5.0 ± 6.5	5.1 ± 12.1	0.54 ± 0.26	-	**0.21** ± 0.06
5	Ground	Simulation Garage	1.6 ± 0.60	1.5 ± 3.1	2.7 ± 2.9	0.62 ± 0.40	-	**0.22** ± 0.05
6	Ground	Simulation Indoor	2.3 ± 2.7	1.5 ± 1.8	2.2 ± 2.0	0.51 ± 0.19	17.8 ± 12.3	**0.23** ± 0.06
7	Ground	Simulation Tunnel	2.3 ± 3.9	16.1 ± 32.4	4.2 ± 15.0	0.86 ± 0.77	43.3 ± 25.3	**0.27** ± 0.06
8	Ground	Simulation Forest	1.3 ± 1.3	3.9 ± 11.7	4.0 ± 8.7	1.8 ± 1.6	8.2 ± 5.4	**0.27** ± 0.07
9	Aerial	Simulation Campus	4.7 ± 0.90	11.3 ± 10.9	6.5 ± 2.1	-	-	**0.20** ± 0.06

Table S2 gives the comparison of algorithm runtime for all methods. The runtime of our method was usually an order of magnitude lower and was 50 percent less than that of others in all tests.

Table S3. ***Runtime breakdown of our method.*** *The runtime for local planning includes the time for updating world representation at the local level, sampling viewpoints for observing the robot's surrounding environment, and computing and optimizing the local detailed path. The runtime for global planning included updating the world representation at the global level and computing the global coarse path, which were reported together. Each entry shows the mean value and standard deviation over the entire run in all trials.*

Test	Vehicle Type	Environment	Local Planning			Global Planning (ms)	Overall (ms)
			(1). Update Representation (ms)	(2). Sample Viewpoints (ms)	(3). Find/Opt. Path (ms)		
1	Ground	Real-world Garage	388.6 ± 152.2	4.3 ± 7.2	5.3 ± 14.8	24.8 ± 4.5	423.0 ± 161.6
2	Ground	Real-world Indoor	121.6 ± 44.3	0.02 ± 0.17	2.5 ± 11.9	8.6 ± 1.0	132.7 ± 47.0
3	Ground	Real-world Campus	106.9 ± 49.6	1.9 ± 6.4	45.6 ± 97.7	9.6 ± 1.6	164.0 ± 139.5
4	Ground	Simulation Campus	165.6 ± 57.7	0.06 ± 0.3	2.5 ± 4.5	49.1 ± 3.6	217.3 ± 59.7
5	Ground	Simulation Garage	146.2 ± 38.0	0.12 ± 0.44	3.9 ± 7.9	71.4 ± 27.5	221.6 ± 54.6
6	Ground	Simulation Indoor	183.0 ± 55.0	0.11 ± 0.59	3.0 ± 6.3	47.9 ± 4.2	234.0 ± 58.6
7	Ground	Simulation Tunnel	188.5 ± 53.2	0.05 ± 0.3	2.9 ± 5.7	80.8 ± 31.8	272.3 ± 63.5
8	Ground	Simulation Forest	180.0 ± 57.9	0.23 ± 0.61	4.8 ± 7.4	80.3 ± 23.3	265.3 ± 68.2
9	Aerial	Simulation Campus	129.7 ± 43.8	0.66 ± 1.3	44.9 ± 27.0	28.6 ± 8.9	203.9 ± 63.8
10	Aerial	Real-world Campus	140.6 ± 36.6	0.30 ± 0.9	36.7 ± 47.3	24.5 ± 5.1	202.1 ± 75.4

Table S4. ***Runtime breakdown of our method with the "Pursuit" strategy****. The tests were conducted in the simulated tunnel environment with a 30-meter communication range, where our method was run with the "Pursuit" strategy for inter-robot communication. The runtime components are the same as those in Table S3.*

Number of Vehicles	Local Planning			Global Planning (ms)	Overall (ms)
	(1). Update Representation (ms)	(2). Sample Viewpoints (ms)	(3). Find/Opt. Path (ms)		
2	240.0 ± 62.5	0.01 ± 0.3	1.5 ± 4.8	390.5 ± 115.3	632.0 ± 267.5
4	218.3 ± 63.2	0.03 ± 0.4	3.3 ± 3.9	416.8 ± 261.0	638.4 ± 270.7
6	215.9 ± 65.1	0.02 ± 0.3	3.4 ± 5.9	428.9 ± 239.2	648.2 ± 238.8
8	222.6 ± 63.3	0.04 ± 0.3	3.9 ± 4.1	467.5 ± 188.9	694.0 ± 272.4
10	297.9 ± 55.2	0.03 ± 0.4	4.8 ± 6.8	448.9 ± 220.7	751.6 ± 309.1
12	222.8 ± 67.8	0.01 ± 0.1	1.3 ± 3.1	487.1 ± 250.1	711.2 ± 263.3
14	257.8 ± 66.6	0.02 ± 0.4	1.9 ± 3.7	552.0 ± 224.1	811.7 ± 240.7
16	263.2 ± 64.2	0.01 ± 0.1	1.6 ± 3.3	563.0 ± 236.8	827.8 ± 252.9
18	261.0 ± 80.0	0.04 ± 0.2	5.5 ± 8.8	636.9 ± 227.7	903.4 ± 296.0
20	272.4 ± 71.9	0.02 ± 0.2	2.9 ± 5.5	628.4 ± 234.3	903.7 ± 257.8

Table S3 gives the runtime breakdown for our method. The numbers indicate that there were no substantial differences in overall runtime between environments with different scales and complexity levels. One can see that the time on updating representation in local planning dominated the overall runtime. This confirmed our insight that keeping the local planning horizon relatively small reduced the computational complexity and enables fast processing. The computation at the global scale was lightweight due to the usage of low-resolution data. Table S4 gives the runtime breakdown for our method in the multi-robot exploration scenario, where the "Pursuit" strategy was used for inter-robot communication. The test was conducted in the tunnel environment, which had the largest footprint among all simulation environments and results in the longest global planning time in the single-robot case. The communication range was set to 30 meters. Compared to the single-robot scenario, the global planning in the multi-robot scenario took substantially longer, mainly because of the requirement to solve the VRP multiple times, accounting for approximately two-thirds of the total runtime. Moreover, the global planning time slowly increased with the number of robots. Nevertheless, our approach maintained an overall runtime of approximately 1 second or less even with 20 robots.

Furthermore, we repeated Test 9 with different sizes of local planning horizons $\mathcal{H}$ to inspect the corresponding algorithm runtime. Table S5 shows the result. As expected, the computation in local planning increased dramatically as the size of $\mathcal{H}$ increased from the default value. The largest $\mathcal{H}$ on the right covered the entire environment, where the runtime for local planning was equivalent to that of computing a detailed path using a uniformly high-resolution map for the entire environment. Note that the number of global subspaces outside the local planning horizon decreased as the local planning horizon expanded. The reduction in global subspaces resulted in a decrease in the runtime required for updating global map representations and computing the TSP path among subspaces. Nevertheless, the decrease in global-level runtime was negligible compared to the increase in local-level runtime. The result confirmed the strength of our dual-resolution framework in producing high-efficiency processing.

Table S5. ***Runtime with different local planning horizon $\mathcal{H}$ size for Test 9****. The dimensions were measured for width × length × height in meters. The runtime for local and global levels were measured in the same way as in Table S3. Each entry shows the mean value and standard deviation over the entire run in all trials.*

Dimensions of $\mathcal{H}$	80m × 80m × 30m (default)		160m × 160m × 60m		320m × 320m × 120m	
Average Runtime (ms)	Local	Global	Local	Global	Local	Global
	175.3 ± 64.8	**28.6 ± 8.9**	2197.6 ± 1624.6	27.9 ± 7.6	5336.0 ± 3355.9	23.25 ± 2.6

Local Planning – Viewpoint Sampling Algorithm

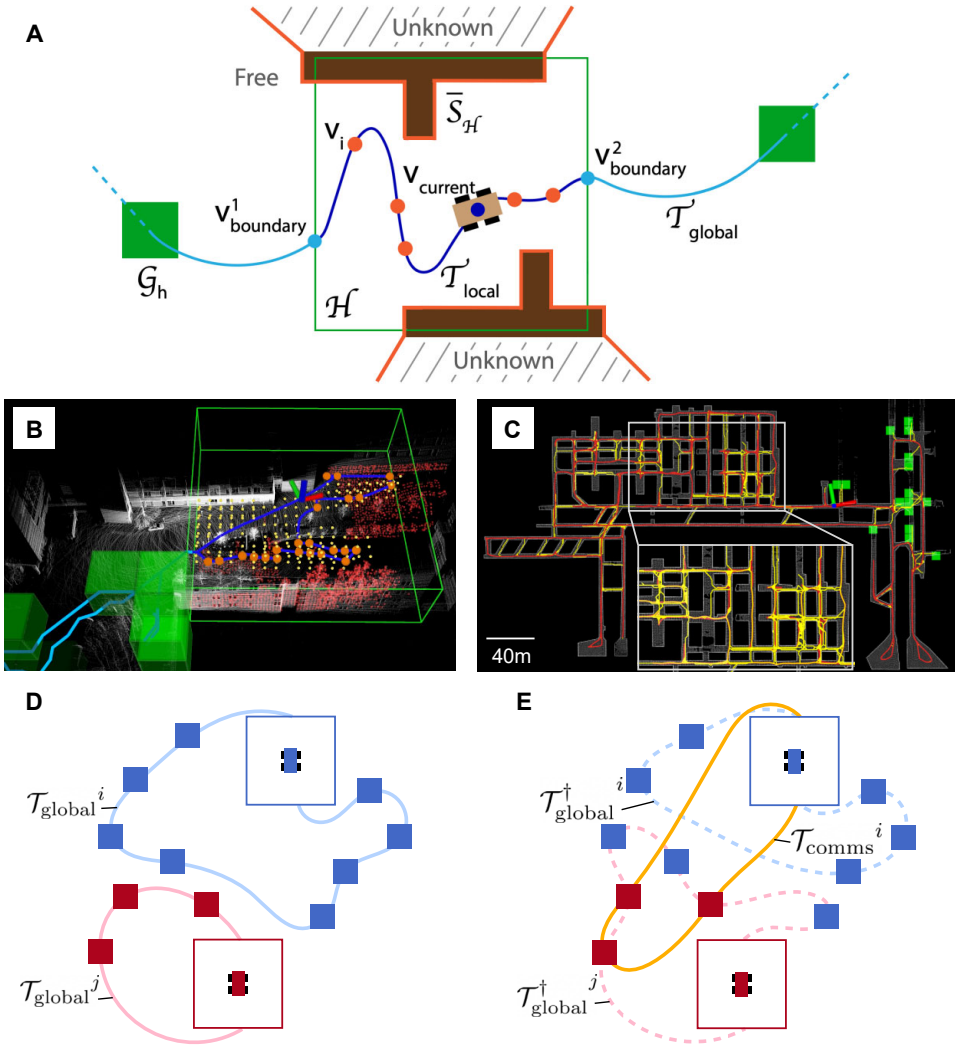

Figure S2. ***Illustrations of the methodology****.* ***(A)*** *The open green box represents the local planning horizon $\mathcal{H}$, The solid green squares represent the exploring subspaces $\mathcal{G}_h, h \in \mathbb{Z}^+$. The dark-blue curve is the local path $\mathcal{T}_{local}$. The light-blue curve is the global path $\mathcal{T}_{global}$. The dark-blue dot on $\mathcal{T}_{local}$ is the current viewpoint $v_{current}$. The light-blue dots are viewpoints $v^1_{boundary}$ and $v^2_{boundary}$ on the boundary of $\mathcal{H}$ where $\mathcal{T}_{local}$ and $\mathcal{T}_{global}$ are connected. The orange dots are sampled viewpoints v_i, $i \in \mathbb{Z}^+$. The orange lines are uncovered surfaces $\overline{S}_{\mathcal{H}}$ to be covered by the viewpoints.* ***(B)*** *An example exploration process with real data, which shows elements from both the local and global planning processes: LiDAR scan data (white), uncovered surfaces (red), local planning horizon (green box frame), viewpoint candidates (small yellow dots), sampled viewpoints (orange dots), local path $\mathcal{T}_{local}$ (light blue), global path $\mathcal{T}_{global}$ (dark blue), exploring global subspaces (solid green boxes).* ***(C)*** *The global roadmap (yellow) constructed while the vehicle is exploring a tunnel environment in simulation. The coordinate frame represents the vehicle. The past trajectory of the vehicle is in red. Exploring global subspaces are represented by green boxes and the explored areas are in white.* ***(D)****. A scenario where two robots are out-of-comms with each other. Robot i has a longer route than robot j.* ***(E)****. A scenario where robot i pursues robot j for delivering information. By traveling along the communication route (yellow), robot i has a chance to meet robot j for exchanging information, which can lead to new distribution of global subspaces and shortening the longest route of the two robots (dotted curves).*

Algorithm 1: Compute Local Path

Input: traversable C-space $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$, uncovered surfaces $\overline{\mathcal{S}}_{\mathcal{H}}$, current viewpoint $\mathbf{v}_{\text{current}}$, boundary viewpoints $\mathbf{v}_{\text{boundary}}^{1}$ and $\mathbf{v}_{\text{boundary}}^{2}$

Output: local path $\mathcal{T}_{\text{local}}$

1 Generate a set of viewpoint candidates $\mathcal{V}$ in $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$

2 Initialize priority queue Q

3 For every $\mathbf{v} \in \mathcal{V}$, estimate coverage $\overline{\mathcal{S}}_{\mathbf{v}}$, then push $\mathbf{v}$ into Q with the priority set to its reward $A_{\mathbf{v}}$

4 $\mathcal{T}_{\text{local}} \leftarrow \emptyset$, $c_{best} \leftarrow +\infty$

5 **For** $i := 1$ to K **do**

6 $\quad Q' \leftarrow Q$, $\mathcal{V}' \leftarrow \{\mathbf{v}_{\text{current}}, \mathbf{v}_{\text{boundary}}^{1}, \mathbf{v}_{\text{boundary}}^{2}\}$

7 $\quad$ **While** $Q' \neq \emptyset$ and Q' contains at least one non-zero priority

8 $\quad\quad$ Probabilistically pick viewpoint $\mathbf{v}'$ in Q', then remove $\mathbf{v}'$ from Q'

9 $\quad\quad \mathcal{V}' \leftarrow \mathcal{V}' \cup \mathbf{v}'$

10 $\quad\quad$ Update priorities for all viewpoints in Q'

11 $\quad$ **end**

12 $\quad$ Compute smooth path $\mathcal{T}'_{\text{smooth}}$ and cost c'_{smooth} using Algorithm 2

13 $\quad$ **If** $c'_{\text{smooth}} < c_{\text{best}}$ **then**

14 $\quad\quad \mathcal{T}_{\text{local}} \leftarrow \mathcal{T}'_{\text{smooth}}$, $c_{\text{best}} \leftarrow c'_{\text{smooth}}$

15 $\quad$ **end**

16 **end**

17 **Return** $\mathcal{T}_{\text{local}}$

Algorithm 1 presents the process of viewpoint sampling. The algorithm first generates a set of viewpoint candidates $\mathcal{V}$ uniformly from a lattice pattern in $\mathcal{H}_{\text{trav}}$ (line 1). Second, it computes the rewards $A_{\mathbf{v}}$ for all viewpoint candidates $\mathbf{v} \in \mathcal{V}$ by estimating their coverages $\overline{\mathcal{S}}_{\mathbf{v}}$ with the updated environment representation (line 3). Then, a process is iterated for K times to determine the viewpoints. At each iteration, the algorithm randomly samples a subset of viewpoints from $\mathcal{V}$ that covers $\overline{\mathcal{S}}_{\mathcal{H}}$ (line 6-11). Three viewpoints are pre-selected (line 6), one as the current viewpoint $\mathbf{v}_{\text{current}}$, and the other two as the viewpoints on the boundary of $\mathcal{H}$, $\mathbf{v}_{\text{boundary}}^{1}$ and $\mathbf{v}_{\text{boundary}}^{2}$, connecting the local path and global path. The process of determining $\mathbf{v}_{\text{boundary}}^{1}$ and $\mathbf{v}_{\text{boundary}}^{2}$ is given in Section IV-C. A priority queue Q' is used to manage the viewpoint candidates. The priority of a viewpoint $\mathbf{v}$ is set to its reward $A_{\mathbf{v}}$. Viewpoints are selected from the priority queue with probabilities proportional to their rewards (line 8). Due to the submodularity, the rewards of the remaining viewpoints in the priority queue are reduced accordingly after a viewpoint is selected, accounting for the overlapping field-of-views (line 10). The viewpoint sampling process finishes when the priority queue is empty or the marginal reward of adding a new viewpoint is negligible. The algorithm calls a subroutine (Algorithm 2 in Supplementary Materials) to generate a path through the sampled viewpoints (line 12). Via iterations, paths with lower costs are found and the path with the lowest cost is returned as the local path, denoted as $\mathcal{T}_{\text{local}}$.

Local Planning – Path Generation and Smoothing

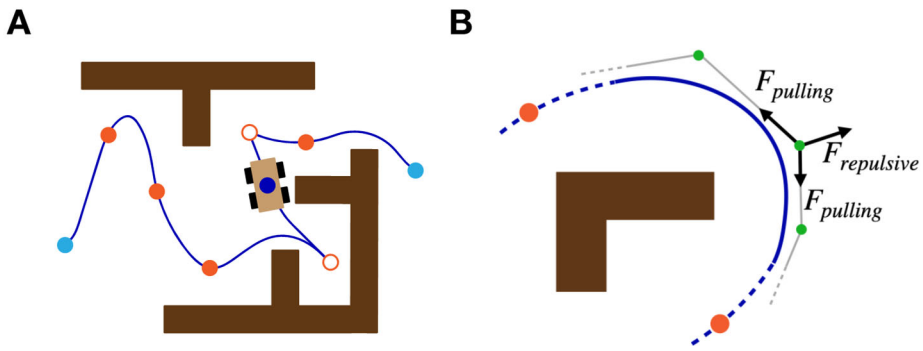

Figure S3. ***Illustration of the path smoothing procedure.*** *The dark-blue curves represent path segments and orange dots are sampled viewpoints.* ***(A)****. The dark-blue dot is* $\boldsymbol{v}_{current}$*. The light-blue dots are* $\boldsymbol{v}^1_{boundary}$ *and* $\boldsymbol{v}^2_{boundary}$ *connecting to* $\mathcal{T}_{global}$*. Hollow-centered orange dots are break-points between the path segments.* ***(B)****. The small green dots represent control points of the B-spline. Each control point is applied with pulling forces from neighboring control points and repulsive forces from the nearest obstacles.*

Given a set of sampled viewpoints $\mathcal{V}'$, we want to generate a path through each of the viewpoints. Ideally, we would like the path to be kinodynamically feasible, which can be followed by the vehicle at a high speed. Here, we focus on the curvature constraint such that the maximum curvature of the path is determined by the vehicle's minimum turning radius when moving at the desired speed.

Due to the distribution of the viewpoints in $\mathcal{V}'$ and structures in the environment, finding a continuous path that meets the curvature constraint can be impossible. The method computes the path in smooth segments as shown in Figure S3A. At the end of each smooth segment, the vehicle comes to a complete stop and then turns towards the next segment. These transition points, where the vehicle must stop and turn between adjacent smooth segments, are denoted as break-points hereafter. Notably, this procedure requires that the vehicle can turn in place. In contrast, the points within the smooth segments are referred to as inner-points. Considering the focus of this paper is not on path planning using sophisticated motion models, we adopt the generic differential motion model, and for aerial vehicles, with independent altitude control. Let us denote the path as $\mathcal{T}'_{\text{smooth}} = [\mathbf{v}_{(1,1)}, \mathbf{v}_{(2,1)}, \dots][\mathbf{v}_{(1,2)}, \mathbf{v}_{(2,2)}, \dots]\dots$ where $[\cdot]$ represents a segment and $\mathbf{v}_{(j,k)}$, is the j-th viewpoint on the k-th segment, $j, k \in \mathbb{Z}^+$. Note that the last viewpoint on a segment and the first viewpoint on the following segment is the same. Let $\tilde{n} \in \mathbb{Z}^+$ be the number of segments, $\tilde{n} \geq k$. Define l_k as the length of the k-th segment. We discourage stopping too frequently by applying a penalty p at each stop. The cost of $\mathcal{T}'_{\text{smooth}}$ is defined as

$$c'_{\text{smooth}} = \sum_{k=1}^{\tilde{n}} l_k + p(\tilde{n} - 1) \tag{1}$$

The problem of computing the path can be stated as follows.

Problem 3: Given viewpoints $\mathcal{V}'$, find a path $\mathcal{T}^*_{\text{smooth}} = [\mathbf{v}_{(1,1)}, \mathbf{v}_{(2,1)}, \dots][\mathbf{v}_{(1,2)}, \mathbf{v}_{(2,2)}, \dots]\dots$ with the lowest c'_{smooth} such that $\mathcal{T}^*_{\text{smooth}}$ visits each viewpoint in $\mathcal{V}'$ once and each segment on $\mathcal{T}^*_{\text{smooth}}$ satisfies the curvature constraint.

Problem 3 is NP-hard given that it is an extended version of TSP (*44*). Instead of attempting to find an exact solution, we solve the problem using approximation algorithms in two steps. First, we solve for an order of the viewpoints with a standard TSP. Second, we separate the viewpoints into segments following the order. This is to determine if a viewpoint, except the first and the last ones, is an inner-point within a segment or a break-point between two segments.

Let $n' \in \mathbb{Z}^+$ be the number of viewpoints in $\mathcal{V}'$. Define $\mathbf{x} = [x_1, x_2, \ldots, x_{n'-2}]$ as a sequence of boolean variables describing the status (being inner-point or break-point) of the $n' - 2$ viewpoints (excluding the first and last) and a function $f(\mathbf{x}) = c'_{\text{smooth}}$. The problem of determining the viewpoint status can be formulated as,

Problem 4: Given a sequence of boolean variables $\mathbf{x}$ and a function $f(\mathbf{x}) = c'_{\text{smooth}}$, find $\mathbf{x}^*$ such that

$$\mathbf{x}^* = \underset{\mathbf{x}}{argmin}\ c'_{\text{smooth}} \tag{2}$$

Problem 4 is a nonlinear integer programming problem and known to be NP-hard (*58*). Especially, finding smooth paths involves time-consuming trajectory optimization. Instead of using an existing heuristic method such as (*59*) with higher computational complexity, our method applies a greedy strategy to check each viewpoint only once in order to maximally reduce the runtime.

Algorithm 2: Compute Local Path from Viewpoints

Input: traversable C-space $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$, viewpoints $\mathcal{V}'$
Output: smooth path $\mathcal{T}'_{\text{smooth}}$, cost c'_{smooth}

Compute shortest paths between viewpoint pairs in $\mathcal{V}'$, then create distance matrix $\mathbf{D}'$
Compute path $\mathcal{T}'$ by solving TSP using $\mathbf{D}'$
Initialize $\mathcal{T}'_{\text{smooth}}$ using $\mathcal{T}'$, then set viewpoints $\mathbf{v}_2$, $\mathbf{v}_3$, ..., $\mathbf{v}_{n-1}$ on $\mathcal{T}_{\text{smooth}}$ as break-points
Smooth path segments between consecutive viewpoints on $\mathcal{T}'_{\text{smooth}}$ and compute cost c'_{smooth}
For $i := 2$ to $n' - 1$ **do**
    Temporarily set viewpoint $\mathbf{v}_i$ as an inner-point by connecting the two segments on both sides of $\mathbf{v}_i$
    Smooth the path segment through $\mathbf{v}_i$ and compute cost c'_{temp}
    If $c'_{\text{temp}} < c'_{\text{smooth}}$ and $\mathcal{T}'_{\text{smooth}}$ meets curvature constraint **then**
        Finalize $\mathbf{v}_i$ on $\mathcal{T}'_{\text{smooth}}$ as an inner-point
        $c'_{\text{smooth}} \leftarrow c'_{\text{temp}}$
    end
end
Return $\mathcal{T}'_{\text{smooth}}$, c'_{smooth}

Algorithm 2 gives the procedure of computing $\mathcal{T}'_{\text{smooth}}$ from a set of viewpoints $\mathcal{V}'$. The algorithm finds the shortest collision-free path between every viewpoint pair in $\mathcal{V}'$ using the A* algorithm (*60*) and constructs a distance matrix $\mathbf{D}'$ containing the length of the

paths (line 1). Next, the algorithm solves a TSP for a traversal order of the viewpoints (line 2). Upon determining the segments on $\mathcal{T}'_{\text{smooth}}$, the algorithm initializes all $n'-2$ viewpoints as break-points (line 3) and smooths the segments between consecutive viewpoints (line 4). Then, the algorithm attempts to reduce the cost c'_{smooth} by sequentially setting each viewpoint $\mathbf{v}_i$ as an inner-point and re-smoothing the segment through $\mathbf{v}_i$ (lines 5-11). Each path segment is smoothed with a trajectory optimization method similar to (*61*), where the segment is modelled as a set of cubic B-splines connected at the viewpoints. Boundary conditions are applied at the viewpoints to ensure connectivity and smoothness. Control points for the splines are placed on the initial paths given by the A* search and adjusted by a nonlinear optimization solver (*62*) to account for collision clearance and path smoothness. To obtain the distance between a control point to nearest obstacles, we maintain a Euclidean distance field (EDF) within the local planning horizon that is updated incrementally (*63*). The control points are adjusted in an elastic band optimization manner (*64*), where each control point is applied with the pulling forces from neighboring control points and repulsive forces from the nearest obstacle, as shown in Figure S3B. To accelerate processing, the nonlinear optimization is marginalized where only the splines between the two adjacent viewpoints of $\mathbf{v}_i$ are optimized. The algorithm returns $\mathcal{T}'_{\text{smooth}}$ with c'_{smooth}.

Global Planning – Single Robot Exploration

Algorithm 3: Compute Exploration Path

Input: local planning horizon $\mathcal{H}$, traversable C-space $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$, uncovered surfaces $\overline{\mathcal{S}}_{\mathcal{H}}$, exploring subspaces $\hat{\mathcal{G}}$, current viewpoint $\mathbf{v}_{\text{current}}$
Output: exploration path $\mathcal{T}$

1 Compute shortest paths between centroids in $\hat{\mathcal{G}}$ and $\mathbf{v}_{\text{current}}$, then create distance matrix **D**
2 Compute global path $\mathcal{T}_{\text{global}}$ by solving TSP using **D**
3 Extract $\mathbf{v}_{\text{boundary}}^1$ and $\mathbf{v}_{\text{boundary}}^2$as the intersections between $\mathcal{T}_{\text{global}}$ and the boundary of $\mathcal{H}$
4 Compute local path $\mathcal{T}_{\text{local}}$ using Algorithm 1
5 Concatenate $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$ to generate $\mathcal{T}$
6 **Return** $\mathcal{T}$

Algorithm 3 gives the overall procedure for computing the exploration path for a single robot. The algorithm begins by constructing a distance matrix (line 1), denoted as **D**, that contains the pairwise shortest distances between the centroids of every pair of global subspaces in $\hat{\mathcal{G}}$. The shortest distance between any two global subspaces is determined using the A* algorithm to find the shortest path between them on the global roadmap. Using the distance matrix, the algorithm solves a TSP (line 2) to obtain $\mathcal{T}_{\text{global}}$. The two points on $\mathcal{T}_{\text{global}}$ intersecting with the boundary of $\mathcal{H}$ are extracted as $\mathbf{v}_{\text{boundary}}^1$ and $\mathbf{v}_{\text{boundary}}^2$ (line 3). Then, Algorithm 1 is used to compute $\mathcal{T}_{\text{local}}$ (line 4). Finally, $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$ are concatenated by substituting the part of $\mathcal{T}_{\text{global}}$ inside $\mathcal{H}$ with $\mathcal{T}_{\text{local}}$ (line 5). Figure S2C shows an example of the exploration where the dark-blue and light-blue paths represent $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$.

When the exploration completes in $\mathcal{H}$ ($\overline{\mathcal{S}}_{\mathcal{H}} = \emptyset$), $\mathcal{T}_{\text{local}}$ reduces to the shortest paths connect from $\mathbf{v}_{\text{current}}$ to $\mathbf{v}^1_{\text{boundary}}$ and $\mathbf{v}^2_{\text{boundary}}$, which are further connected to the adjacent exploring subspaces on $\mathcal{T}_{\text{global}}$. The vehicle follows the path to transit to an exploring subspace to resume exploration. In other words, the algorithm implicitly transitions between exploration and relocation to another area to explore further. If $\overline{\mathcal{S}}_{\mathcal{H}} = \emptyset$ and $\hat{\mathcal{G}} = \emptyset$, the exploration terminates.

Global Planning – Multi-robot Exploration with Limited Communication

Algorithm 4 presents a unified procedure for computing the exploration path for robot *i* under limited communication. Different from the Algorithm 3, Algorithm 4 takes extra steps to update and share knowledge among the robots (line 1, 8) and compute paths to regain communication with out-of-comms robots (line 5), depending on if the communication condition satisfies (line 4). For the rendezvous-based strategies, the condition is satisfied if the traveling time to the rendezvous location is no less than the remaining exploration time. For our "Pursuit" strategy, the condition is satisfied if the benefit of communication outweighs its cost, as discussed above. In addition, Algorithm 4 computes global paths for all robots to share with others but only execute the one corresponding to robot *i* itself (line 7).

Algorithm 4: Compute Multi-robot Exploration Path for Robot *i*

Input: local planning horizon $\mathcal{H}$, traversable C-space $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, uncovered surfaces $\overline{\mathcal{S}}_{\mathcal{H}}$, current viewpoints of all robots $\{\mathbf{v}_{\text{current}}{}^j\}$, knowledge of all robots $\{\tilde{\mathcal{G}}_i{}^j\}$, global paths from other robots with the minimum cost $\{\mathcal{T}_{global}{}^j\}^*$
Output: exploration path $\mathcal{T}$

1 Update $\tilde{\mathcal{G}}_i{}^i$ with $\{\tilde{\mathcal{G}}_i{}^j\}, j \neq i$
2 Compute shortest paths between centroids of exploring subspaces in $\tilde{\mathcal{G}}_i{}^i$ and $\{\mathbf{v}_{\text{current}}{}^j\}$, then create distance matrix **D**
3 Compute global paths $\{\mathcal{T}_{global}{}^j\}$ by solving VRP using **D** with an initial guess $\{\mathcal{T}_{global}{}^j\}^*$
4 **If** the communication condition satisfies **then**
5 Use $\{\tilde{\mathcal{G}}_i{}^j\}$ and $\{\mathcal{T}_{global}{}^j\}$ to compute global paths for communication $\{\mathcal{T}_{comms}{}^j\}$, replace $\{\mathcal{T}_{global}{}^j\}$ with $\{\mathcal{T}_{comms}{}^j\}$
6 **end**
7 Extract $\mathbf{v}^1_{\text{boundary}}$ and $\mathbf{v}^2_{\text{boundary}}$ as the intersections between $\mathcal{T}_{global}{}^i$ and the boundary of $\mathcal{H}$
8 Compute local path $\mathcal{T}_{\text{local}}$ using Algorithm 1
9 Concatenate $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{global}{}^i$ to generate $\mathcal{T}$
10 Communicate $\{\tilde{\mathcal{G}}_i{}^j\}$ and $\{\mathcal{T}_{global}{}^j\}$ to other robots
11 **Return** $\mathcal{T}$

Theoretical Analysis - Computational Complexity

Let $n \in \mathbb{Z}^+$ be the number of viewpoint candidates generated in Algorithm 1. For a fixed number of iterations, the sampled viewpoints are no more than n. To maintain the priority queue of viewpoints in Algorithm 1, at most $O(n\log\ n)$ time is needed, which includes adding or removing viewpoints and adjusting rewards of remaining viewpoints following the selection and removal of a viewpoint. In Algorithm 3, we model the time of finding the shortest path between two viewpoints and the time of smoothing a path segment as bounded by two constants (the trajectory optimization is marginalized where only a bounded number of control points are optimized). The time complexity of constructing the distance matrix is $O(n^2)$. The TSP is solved using the Lin–Kernighan heuristic which takes $O(n^{2.2})$ time (*3*). The path smoothing processes each viewpoint once and takes $O(n)$ time. In Algorithm 3, computing the distance matrix takes $O(m^2)$time and solving the TSP runs in $O(m^{2.2})$ time, where $m \in \mathbb{Z}^+$ is the number of exploring subspaces. Concatenating $\mathcal{T}_{\text{local}}$ and $\mathcal{T}_{\text{global}}$ takes constant time. Overall, Algorithm 3 has the time complexity of $O(n^{2.2} + m^{2.2})$.

For multi-robot scenarios, we can apply the same analysis on the local-level planning that uses Algorithm 1, which has a complexity of $O(n^{2.2})$. For global-level planning, the VRP can be solved using transformation-based methods that incorporate the Lin-Kernighan heuristic (*65*), maintaining a complexity of $O(m^{2.2})$. In the "Pursuit" strategy, the VRP must be solved 3 times for each randomly selected set of target robots: once for non-pursuit, once for pursuit, and once for computing the pursuit path. As discussed in the methodology section, the target robot selection process is distributed over planning cycles, where only a small number k (usually 1 or 2) of target robot combinations are randomly selected without replacement at each planning cycle. As a result, the time complexity is $O(n^{2.2} + 3km^{2.2})$ for each planning cycle.
Therefore, we have

Theorem 1: Algorithm 3 runs in $O(n^{2.2} + m^{2.2})$ time and Algorithm 3 runs in $(n^{2.2} + 3km^{2.2})$ time.

Theoretical Analysis - Probabilistic Completeness

Recall that $\overline{\mathcal{S}}_{\mathcal{H}} \subset \mathcal{S}_{\mathcal{H}}$ is the uncovered surfaces that can be perceived from viewpoints in $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$ and the problem of viewpoint sampling is to select a minimum set of viewpoints in $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$ to cover $\overline{\mathcal{S}}_{\mathcal{H}}$. In Algorithm 1, viewpoints are sampled uniformly at random from $\mathcal{C}_{\text{trav}}^{\mathcal{H}}$. Define $\mathcal{V}_i \subseteq \mathcal{C}_{\text{trav}}^{\mathcal{H}}$ as the set of viewpoints that can perceive a surface patch $s_i \in \overline{\mathcal{S}}_{\mathcal{H}}$. By definition, $\mu(\mathcal{V}_i)/\mu(\mathcal{C}_{\text{trav}}^{\mathcal{H}}) \geq \epsilon > 0, \forall i \in \{1, \ldots |\overline{\mathcal{S}}_{\mathcal{H}}|\}$, where μ gives the volume of the specified region of configuration space and $|\overline{\mathcal{S}}_{\mathcal{H}}|$ is the number of surface patches comprising $\overline{\mathcal{S}}_{\mathcal{H}}$. From (*1*), we have the following lemma.

Lemma 1: Algorithm 1 is probabilistically complete. The probability that $\overline{\mathcal{S}}_{\mathcal{H}}$ cannot be covered after sampling n viewpoints is bounded by

$$Pr[FAILURE] < |\overline{\mathcal{S}}_{\mathcal{H}}| \cdot \frac{e}{e^{n\epsilon/2}} \tag{3}$$

Note that in (*4*), the original analysis is given for the Coverage Sampling Problem, where a robot is set to cover structural surfaces known a priori. In our formulation, we consider the generalized surfaces for exploration, yet the same analysis applies.

Lemma 1 states that our method is probabilistically complete in completing coverage inside the local planning horizon. As the exploration proceeds, the local planning horizon moves along with the vehicle and covers all subspaces in the configuration space. The exploration terminates when the local planning horizon and all subspaces are covered, $\overline{\mathcal{S}}_{\mathcal{H}} = \emptyset$ and $\hat{\mathcal{G}} = \emptyset$. Therefore, we have

Theorem 2: Algorithm 3 is probabilistically complete.

Theoretical Analysis – Approximation Ratio

Let us analyze the approximation ratio introduced by the dual-resolution representation. To simplify the problem, we evaluate a path with its length and ignore its kinodynamic feasibility first. The effect of considering kinodynamic feasibility will be discussed later in Theorem 4 of this section. Define $\mathcal{T}^*$ as the shortest possible path to complete the coverage. $\mathcal{T}^*$ is computed without using the dual-resolution representation and considered the theoretically optimal solution. Denote l^* as the length of $\mathcal{T}^*$. Let $D_{\mathcal{H}}$ be the longest distance needed to travel between any two viewpoints in $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, in other words, for all the shortest paths between any two viewpoints in $\mathcal{C}^{\mathcal{H}}_{\text{trav}}$, $D_{\mathcal{H}}$ is the length of the longest path. Recall that D is the distance limit for covering surfaces and a subspace in $\hat{\mathcal{G}}$ is a place that the robot has passed by previously but not yet fully covered the surfaces around. It defines a neighboring area around the centroid of the subspace, where there are uncovered surfaces within a distance range of D. To fully cover the area, the robot must return to the area and be no more than $2D$ away from the subspace's centroid. Let us approximate the area using a sphere with the radius $r = 2D$ centered at the centroid of a subspace in $\hat{\mathcal{G}}$. Let $D_{\mathcal{G}}$ be the longest distance needed to travel between any two viewpoints in such a sphere. Let $\mathcal{V}_{\mathcal{G}}$ be the volume of a subspace. The approximation ratio introduced by the dual-resolution representation is analysed as follows.

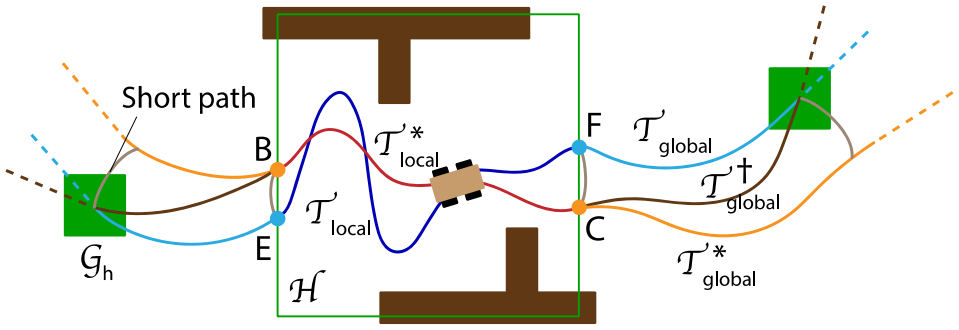

Figure S4. Illustration of mathematical definitions in the proof of Theorem 3

As shown in Figure S4, let $\mathcal{T}^*_{\text{local}}$ be the part of $\mathcal{T}^*$ inside $\mathcal{H}$ and let $\mathcal{T}^*_{\text{global}}$ be the part outside $\mathcal{H}$. The length of $\mathcal{T}^*_{\text{local}}$ and $\mathcal{T}^*_{\text{global}}$ are denoted as l^*_{local} and l^*_{global}, respectively. Define $\mathcal{T}^\dagger_{\text{global}}$ as the shortest global path connecting to $\mathcal{T}^*_{\text{local}}$ at B, C and going through the centroid of each subspace in $\hat{\mathcal{G}}$. The length of $\mathcal{T}^\dagger_{\text{global}}$ is denoted as $l^\dagger_{\text{global}}$. Note that $\mathcal{T}^*_{\text{global}}$ must go into all spheres of $\hat{\mathcal{G}}$ to completely cover all surfaces, the length of $\mathcal{T}^*_{\text{global}}$ is lower bounded by the solution to a corresponding TSP with neighborhoods (TSPN) (*66*), which seeks the shortest tour visiting a set of neighborhoods. We here consider the

neighborhoods to be the spheres around $\hat{\mathcal{G}}$. Denote the TSPN solution $\tilde{\mathcal{T}}_{\text{global}}$ and its length $\tilde{l}_{\text{global}}$, we have $\tilde{l}_{\text{global}} \leq l^*_{\text{global}}$. This can be easily proved by contradiction. If $\tilde{l}_{\text{global}} > l^*_{\text{global}}$, $\mathcal{T}^*_{\text{global}}$ is a feasible and shorter path visiting all neighborhoods, thus a contradiction occurs. Note that $\mathcal{T}^\dagger_{\text{global}}$ is the shortest path visiting the centroids of $\hat{\mathcal{G}}$, thus a TSP tour with respect to $\hat{\mathcal{G}}$. We now establish the relationship between $l^\dagger_{\text{global}}$ and $\tilde{l}_{\text{global}}$.

First, we consider the case of disjoint spherical neighborhoods. We have:

Proposition 1. Given a set of m disjoint equal-size spheres with radius r, we have

$$l^\dagger_{\text{global}} \leq (1 + \frac{6D_{\mathcal{G}}}{r})\tilde{l}_{\text{global}} + 16D_{\mathcal{G}}. \tag{4}$$

Proof. For brevity, we drop the "global" subscript from notations in the following. We extend the proof of (*14*) from the planar case to 3D. Since $\tilde{\mathcal{T}}$ visits all spheres, the volume $V_{\tilde{\mathcal{T}}}$ swept by a sphere of radius $2r$, whose center moves along $\tilde{\mathcal{T}}$, covers all m spheres. This volume is bounded as:

$$m\frac{4}{3}\pi r^3 \leq V_{\tilde{\mathcal{T}}} \leq \pi 4r^2\tilde{l} + \frac{4}{3}\pi(2r)^3 \tag{5}$$

thus,

$$m \leq (3/r)\tilde{l} + 8. \tag{6}$$

By constructing a tour going along $\tilde{\mathcal{T}}$ and making a detour of length at most $2D_{\mathcal{G}}$ at the point where $\tilde{\mathcal{T}}$ first visits the centre of each sphere, we have

$$l^\dagger \leq \tilde{l} + 2mD_{\mathcal{G}}, \tag{7}$$

which gives

$$l^\dagger \leq (1 + \frac{6D_{\mathcal{G}}}{r})\tilde{l} + 16D_{\mathcal{G}} \tag{8}$$

■

Next, we consider the case where the spheres can overlap. We can compute a maximal independent (pairwise disjoint) set I of spheres. Let $n = |I|$. We have the following inequalities:

$$n \leq (3/r)\tilde{l}_n + 8 \tag{9}$$

$$l^\dagger_n \leq (1 + \frac{6D_{\mathcal{G}}}{r})\tilde{l}_n + 16D_{\mathcal{G}} \tag{10}$$

$$\tilde{l}_n \leq \tilde{l} \tag{11}$$

$$l^\dagger \leq l^\dagger_n + (m-n)2D_{\mathcal{G}} \tag{12}$$

$$m \leq \frac{4\pi r^3}{3\mathcal{V}_{\mathcal{G}}}n \tag{13}$$

Eq. (9)-(10) directly follows from the case of disjoint spheres discussed above. Eq. (11) follows from the fact that I is a subset of all spheres. Eq. (12) can be verified by constructing a tour along $l^\dagger_n$ and making detours of length at most $2D_{\mathcal{G}}$ from the centers of the n spheres to visit the centers of the remaining $m - n$ spheres. Eq. (13) is given by our

scheme's discretization of the space, where a sphere can overlap with at most $\frac{4\pi r^3}{3\mathcal{V}_\mathcal{G}}$ other spheres. Putting Eq. (9)-(13) together, we have

$$l^\dagger \leq (1+\frac{8\pi r^2 D_\mathcal{G}}{\mathcal{V}_\mathcal{G}})\tilde{l}+\frac{64\pi r^3 D_\mathcal{G}}{3\mathcal{V}_\mathcal{G}} \tag{14}$$

Therefore, we have

$$l^\dagger_{\text{global}} \leq (1+\frac{8\pi r^2 D_\mathcal{G}}{\mathcal{V}_\mathcal{G}})l^*_{\text{global}}+\frac{64\pi r^3 D_\mathcal{G}}{3\mathcal{V}_\mathcal{G}} \tag{15}$$

Let l_{global} be the length of the global path $\mathcal{T}_{\text{global}}$ found by our method. Here, we do not consider the effect of the approximation algorithm used to solve the TSP. In other words, we assume an exact solution is obtained from solving the TSP and $\mathcal{T}_{\text{global}}$ is the shortest possible path connecting to E, F and going through the centroid of each subspace in $\hat{\mathcal{G}}$. We can show that

$$l_{\text{global}} \leq l^\dagger_{\text{global}}+2D_\mathcal{H}. \tag{16}$$

Again, we use proof by contradiction. We find two short paths connecting between B, E and C, F, respectively. Each short path can be no longer than $D_\mathcal{H}$. This way, we form a new path with $\mathcal{T}^\dagger_{\text{global}}$ and the two short paths. The new path connects to E, F and visits the centroid of each subspace in $\hat{\mathcal{G}}$. If Eq. (16) does not hold, the new path becomes shorter than $\mathcal{T}_{\text{global}}$. Therefore, a contradiction occurs. Further, let l_{local} be the length of the local path $\mathcal{T}_{\text{local}}$ found by our method. We do not consider the effect of the approximation algorithm solving the TSP or require the path being "kinodynamically" feasible. $\mathcal{T}_{\text{local}}$ is assumed to be the shortest possible path connecting to E, F and fulfilling a coverage within $\mathcal{H}$. we have

$$l_{\text{local}} \leq l^*_{\text{local}}+2D_\mathcal{H}. \tag{17}$$

Eq. (17) can be proved by forming a new path with $\mathcal{T}^*_{\text{local}}$ and the two short paths connecting between B, E and C, F. If Eq. (17) does not hold, the new path becomes shorter than $\mathcal{T}_{\text{local}}$ and a contradiction occurs. Combining Eq. (15)-(17) gives

$$l_{\text{local}}+l_{\text{global}} \leq (1+\frac{8\pi r^2 D_\mathcal{G}}{\mathcal{V}_\mathcal{G}})l^*+\frac{64\pi r^3 D_\mathcal{G}}{3\mathcal{V}_\mathcal{G}}+4D_\mathcal{H}. \tag{18}$$

Therefore, we have the following theorem

Theorem 3: The approximation ratio of the path length resulting from the dual-resolution representation is

$$\sigma_{\text{dual}} \leq \left(\left(1+\frac{8\pi r^2 D_\mathcal{G}}{\mathcal{V}_\mathcal{G}}\right)l^*+\frac{64\pi r^3 D_\mathcal{G}}{3\mathcal{V}_\mathcal{G}}+4D_\mathcal{H}\right)/l^*. \tag{19}$$

It is noteworthy that the constants $1+\frac{8\pi r^2 D_\mathcal{G}}{\mathcal{V}_\mathcal{G}}$ and $\frac{64\pi r^3 D_\mathcal{G}}{3\mathcal{V}_\mathcal{G}}+4D_\mathcal{H}$ depend on the parameters chosen for our dual-resolution representation and do not depend on the size of the environment. In addition, the bound gives the worst-case scenario where the maximum number of spheres of $\hat{\mathcal{G}}$ overlaps together. In practice, the environment is large-scale, the exploring subspaces are usually sparsely distributed through the environment with little overlapping. With a large l^* and a 2D environment, where the neighbourhoods can be

approximated by non-overlapping circles with radius r, σ_{dual} is given by $\frac{8D_{\mathcal{G}}}{\pi r}+1$ following a similar proof in (*67*).

Finally, we analyze the approximation ratio resulting from the segmented path smoothing, which is not considered in Theorem 3. Recall that p is the penalty for stopping the vehicle at a breakpoint and n' is the number of viewpoints on $\mathcal{T}'_{\text{smooth}}$. We reuse l_{local} to denote the length of the local path where all $n'-2$ viewpoints are treated as break-points. This path has the shortest length since no boundary conditions are enforced at the viewpoints, in contrast to the case where any of the viewpoints is designated as an inner-point. We have

Theorem 4: The approximation ratio of cost c'_{smooth} by the path smoothing has

$$\sigma_{\text{smooth}} \leq (l_{\text{local}} + p(n'-2))/l_{\text{local}}. \tag{20}$$

Proof: Since l_{local} is the length of the shortest local path, the cost of the theoretically optimal local path has

$$c^*_{\text{smooth}} \geq l_{\text{local}}. \tag{21}$$

In Algorithm 2, the algorithm starts with initializing all $n'-2$ viewpoints as break-points and reduces the cost by setting each viewpoint as an inner-point and re-smoothing the segment through the viewpoint. The resulting path has a cost

$$c'_{\text{smooth}} \leq l_{\text{local}} + p(n'-2). \tag{22}$$

Therefore, the approximation ratio is bounded by

$$\sigma_{\text{smooth}} \leq (l_{\text{local}} + p(\tilde{n}-2))/l_{\text{local}}. \tag{23}$$

■

Theorem 4 implies that our path smoothing is most suitable when p is relatively small as compared to l_{local}.

Implementation Details

Choosing parameters: In our implementation, the workspace $\mathcal{W}$ is evenly divided into square blocks. Each subspace $\mathcal{G}_i$ consists of a block. The local planning horizon $\mathcal{H}$ consists of 5×5×3 blocks with the vehicle in the center block. For aerial vehicle experiments, the block size is set to 16 meters × 16 meters × 10 meters, and for ground vehicle experiments, the block size is set to 8 meters × 8 meters × 5 meters. During exploration, if the vehicle crosses the boundary of a block, a rollover is introduced where the blocks behind the vehicle roll out of $\mathcal{H}$ and the blocks in front of the vehicle become inside $\mathcal{H}$. Accordingly, the memory that maintains data in these blocks for local planning is reallocated. It is noteworthy that the local level planning accounts for the bulk of the computation, as illustrated in Table S3. Consequently, the above parameter values are chosen to enable real-time execution of this stage with a minimum frequency of 1 Hz. Typically, the size of the exploration area and the level of clutteredness of the environment are estimated to determine appropriate parameter values.

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

Following the TSP tour: Note that the local and global paths form a TSP tour over the environment visiting all unexplored areas. The current position of the vehicle is a node on the TSP tour. At any given moment, the vehicle can follow the path in two directions. In our implementation, when there are still viewpoints in $\mathcal{H}$, the vehicle follows the path in the direction that visits more viewpoints before exiting $\mathcal{H}$. When the exploration completes in $\mathcal{H}(\overline{\mathcal{S}}_{\mathcal{H}} = \emptyset)$, the vehicle follows the direction that leads to the nearest global subspace.

Handling perception noise: In general, perception noise resulting from reflective surfaces such as windows and glass walls, coupled with state estimation drift and mapping misalignment, can lead to spurious obstacles that prevent the robot from finding feasible paths. Our approach tackles the issues through two techniques that include Ray casting to the latest LiDAR points to eliminate spurious obstacles (*68*) and adjusting the map representations in accordance with the loop-closure events from the SLAM module to eliminate mapping misalignments (*69*). It is noteworthy that both techniques have been employed in the field of autonomous navigation.

Handling cumulative localization errors: Cumulative localization errors can cause map misalignments that may hinder the robot from finding feasible paths or lead to redundant visiting or no visiting of some areas. In the case of a single robot, as explained above on handling perception noise, our approach relies on two techniques, which include ray casting to eliminate "ghost" structures caused by map alignment and adjusting the environment representation based on loop closure events to reduce topological errors.

In the context of multi-robot exploration, our approach only shares among robots the coarse global representations, specifically, global subspaces indicating whether a region has been explored or not. This approach is less sensitive to map misalignment compared to directly sharing the detailed point cloud maps. Our method can effectively manage typical localization errors present in modern LiDAR-based SLAM algorithms, which are typically lower than the size of a global subspace (for example, 8 meters) used in our method while the robots travel in the range of 3-4 kilometers. During the SubT final competition, our SLAM system demonstrated an average drift of 1-2 meters per kilometer, which does not substantially affect the shared global subspaces among robots. However, when the drift exceeds the size of a global subspace, our method performs data association between two robots' global subspaces to address misalignment. This involves utilizing overlapping global subspaces that share the same state of explored/unexplored to form a pattern, and

identifying any shift in the pattern to recover the local offset between the two frames using data association techniques.

Handling dynamic obstacles: Dynamic obstacles can result in obsolete point cloud “trails” on the map that could potentially mislead the exploration and hinder the robot from reaching a desired navigation goal. Our approach addresses the issue at both the perception and navigation levels. At the perception level, we use a ray-casting technique (*68*) that is similar to the approach used for handling perception noise, as discussed above. This technique effectively removes outdated "trails" left by dynamic obstacles, such as moving pedestrians and cars, from the map. When dynamic obstacles are other robots, we assume that they can share their real-time locations in close proximity, thus further preventing the registration of laser points corresponding to other robots onto the map. Additionally, our method can identify alterations in the environment, such as a previously closed door being opened, by comparing the current sensor readings with the map. This enables it to detect new surfaces that require coverage, which subsequently leads the robot into previously unexplored regions. At the navigation level, our approach relies on the navigation modules at layer 2 (shown in Figure 1), including collision avoidance and terrain analysis that are below the exploration planner layer, to navigate the robot amidst dynamic obstacles. The terrain analysis module utilizes a refined version of the ray-casting technique, discussed previously, to distinguish between mobile objects and stationary backgrounds. The resulting output is then utilized by the collision avoidance module to find feasible trajectories and control commands toward the goal point generated by the exploration planner.

Frontiers versus Object Surfaces as Exploration Objective

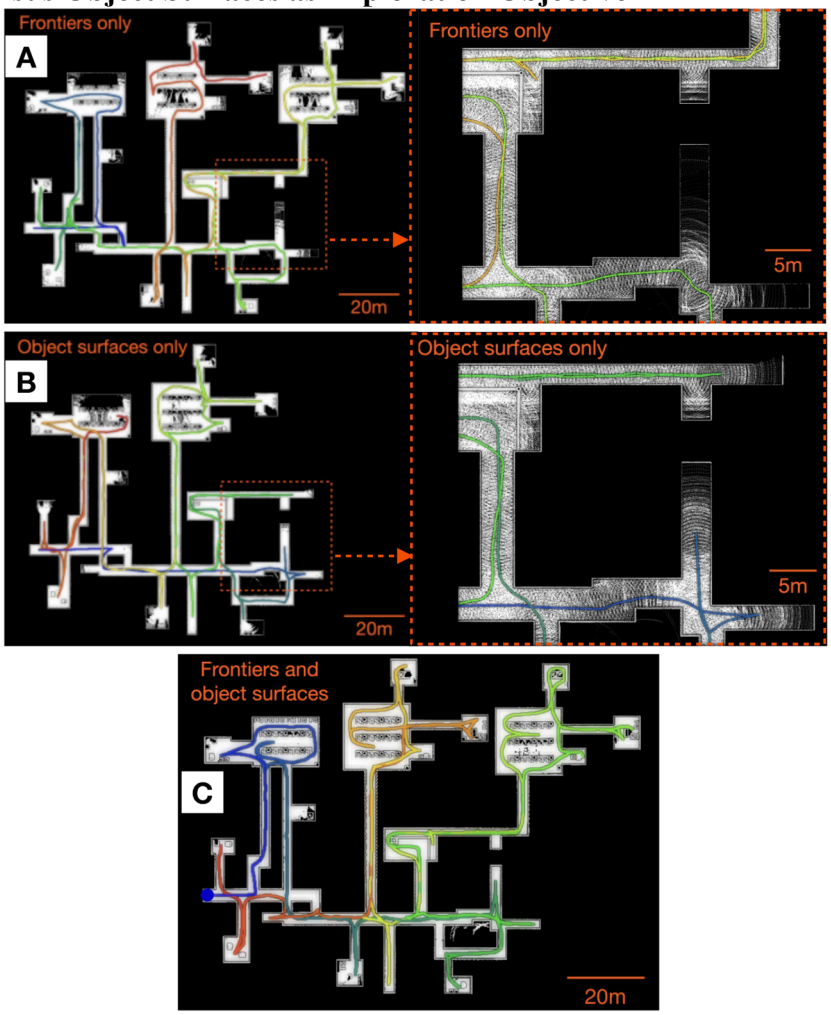

Figure S5. ***Comparison of using frontiers versus object surfaces as objective for exploration.*** ***(A)*** *Our method configured to use frontiers only for exploration.* ***(B)*** *Our method configured to use object surface only for exploration. Neither (A) nor (B) is able to produce complete and good coverage of the environment.* ***(C)*** *Our method configured to use the default generalized surfaces, which achieves complete exploration.*

We here compare using the generalized surfaces to using conventional frontiers as the coverage objective for our exploration algorithm. Recall that the generalized surface is defined as the boundary between free and non-free space. In the classic occupancy mapping scheme, the non-free space consists of unknown and occupied space. Therefore the generalized surfaces consist of frontiers - the boundaries between free and unknown space, and object surfaces - the boundaries between free and occupied space.
We conducted a test in simulation to compare the effect of using the generalized surfaces versus using frontiers in the indoor environment, as shown in Figure S5. With only frontiers used as the coverage objective, there were areas in the map that are not well covered with lower density of laser points, as shown in Figure S5A and the highlighted areas. The vehicle lacked motivation to move closer to object surfaces when there were no frontiers left in the area. This was because the ability to detect frontier cells was dependent on the resolution of the underlying occupancy grid map. For instance, when using an occupancy grid map with a 0.2-meter resolution, the presence of a laser point every 0.2 meter on the wall implies no unknown cells exist on the wall, resulting in no frontiers. However, the resulting point cloud map with a resolution of 0.2-meter is generally inadequate for use in other applications. Therefore, relying solely on frontiers for exploration may failed to produce sufficiently dense point cloud maps.

On the other hand, with only object surfaces as the coverage objective, the algorithm could not fully explore the environment, as shown in Figure S5B. This is because that object surfaces behind corners cannot be perceived by the sensor, thus is not able to drive the vehicle to go around corners for covering more surfaces. The test showed that the combination of frontiers and object surfaces, namely, the generalized surfaces, for guiding the exploration achieved better performance in terms of map completeness and quality, as shown in Figure S5C.

Development Environment and Benchmarking System

In this section we present the development environment and systems for benchmarking ours against other state-of-the-art exploration algorithms. The development environment functions as a platform for developing and benchmarking high-level planning algorithms for ground vehicle navigation. In the development environment, we provide five environment models, fundamental navigation modules, and visualization and debugging tools. In addition, our development environment supports custom-built or third-party environment models, such as the photorealistic ones provided by Matterport3D (*70*).

Environment Models

The environment models resemble real-world settings where robotic systems are commonly deployed. Each of the environment models is distinctive with unique features and challenges. Figure 3 gives an overview of the environment models and Table S6 summarizes their characteristics. In particular, Campus (340 meters × 340 meters) is large-scale environment as part of the Carnegie Mellon University campus, containing undulating terrains and convoluted layout. Indoor (130 meters × 100 meters) consists of long and narrow corridors connected with lobby areas. Obstacles such as tables and columns are present. Garage (140 meters × 130 meters, 5 floors) is an environment with multiple floors and sloping terrains to test autonomous navigation in a 3D environment. Tunnel (330 meters × 250 meters) is a large-scale environment containing tunnels that form a network, provided by the Autonomous Robots Lab at University of Nevada, Reno. Forest (150 meters × 150 meters) contains mostly trees and a couple of houses in a cluttered setting.

Table S6. Environment model characteristics

Environment	Large Scale	Convoluted Topology	Multi-level	Undulating Terrains	Cluttered Obstacles	Thin Structures
Campus	Y	Y		Y		
Indoor		Y			Y	Y
Garage			Y	Y		
Tunnel	Y	Y				
Forest					Y	

Collision Avoidance Planner

The collision avoidance planner (*71*) warrants safety in reaching waypoints that are sent by high-level planners. It computes and follows collision-free paths that lead to the waypoint. The module pre-computes a motion primitive library and associates the motion primitives to 3D locations in the vicinity of the vehicle. The motion primitives are modelled as Monte Carlo samples and organized in groups. In real-time, when a location is occupied by obstacles, the module can determine motion primitives that are in collision with the obstacle within milliseconds. The module then selects the group of motion

primitives with the maximum likelihood toward the waypoint. In Figure 1 (red box) the red paths represent the collision-free motion primitives. For ground vehicles, the traversability of the vehicle is determined by the terrain characteristics. The collision avoidance planner takes in the terrain map from the terrain analysis module, detailed in the next section. The module also has interface to take in additional range data for collision avoidance as an extension option.

Terrain Traversability Analysis

The terrain analysis module examines the traversability of the local terrain surrounding the vehicle. The module builds a cost map where each point on the map is associated with a traversal cost. The cost is determined by the local smoothness of the terrain. We use a voxel grid to represent the environment and analyze the distributions of data points in adjacent voxels to estimate the ground height. The points are associated with higher traversal costs if they are further apart from the ground. Figure 1 (red box) gives an example terrain map covering a 40m x 40m area with the vehicle in the center. The green points are traversable and the red points are non-traversable. In addition, the terrain analysis module can handle negative obstacles that often result in empty areas with no data points on the terrain map. When negative obstacle handling is turned on, the module treats those areas as non-traversable.

Visualization and Debugging Tools

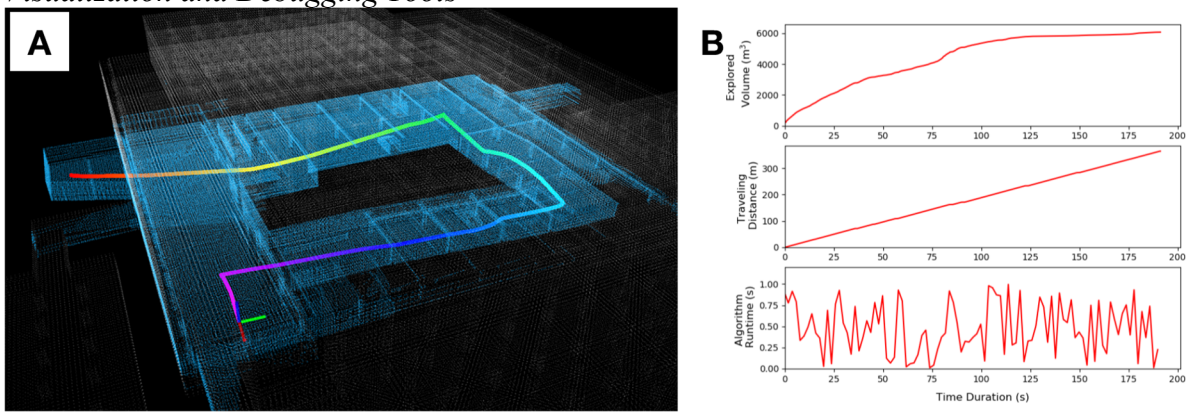

Figure S6. ***Visualization tools****.* ***(A)*** *Vehicle trajectory (rainbow color), explored areas (blue) and overall areas (white).* ***(B)*** *Plots of metrics.*

To facilitate algorithm development, we provide a set of tools to visualize the algorithm performance. The visualization tools display the overall map, explored areas, and vehicle trajectory as shown in Figure S6A. Metrics such as explored volume, traveling distance, and algorithm runtime are plotted and logged to files, shown in Figure S6B. Further, the system supports using a joystick controller to interfere with the navigation, switching among multiple operation modes to ease the process of system debugging. Detailed information is available on the project website (www.cmu-exploration.com).

System Integration

Figure 1 shows the interfacing between our method and the development environment. The development environment functions as the mid-layer between upper-level exploration planner and lower-level motion controllers. It takes in the state estimation output and navigation-related requests from the exploration planner including waypoints, navigation boundaries and desired vehicle speed. It produces the terrain map for the exploration planner and sends command velocity to the motion controllers that drive the vehicle. The

stack remains the same when run in simulation, except that the state estimation is provided by the simulator and the motion controller drives a simulated vehicle.

System Integration for the DARPA Subterranean Challenge

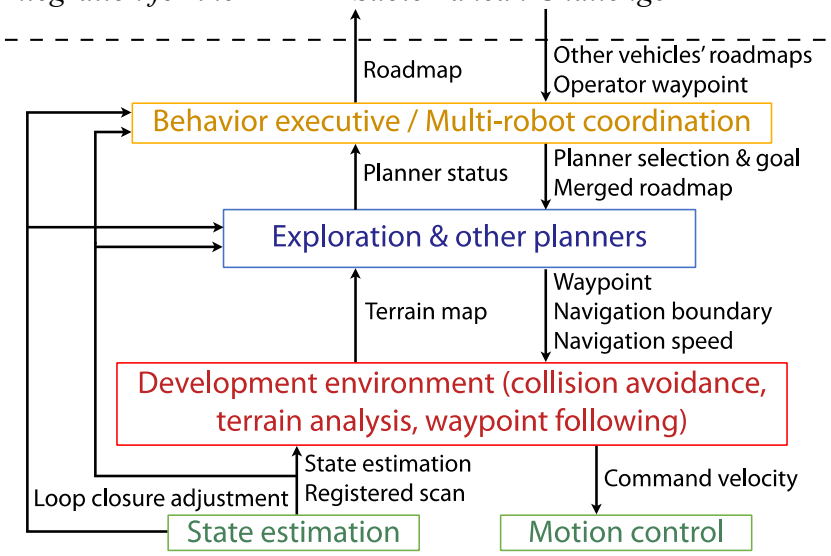

Figure S7. System diagram for the ground vehicles used by the CMU-OSU Team (Team Explorer) in DARPA Subterranean Challenge. The system is extended from our basic system in Figure 1.

The ground vehicles used by the CMU-OSU Team (Team Explorer) are 4-wheel-drive and skid-steer platforms as shown in Figure 6B. The navigation system is an extended version of our fundamental system in Figure 1. The system diagram is shown in Figure S7. The development environment is used as the mid-layer in the navigation system, and our method is used as the main exploration method. The state estimation module can detect and introduce loop closures. The module outputs state estimation in the odometry frame generated by 3D LiDAR-based odometry containing accumulated drift. When loop closure is triggered, it outputs loop closure adjustments to globally relax the vehicle trajectory and the corresponding maps. Our method uses the loop closure adjustments to update the positions of global subspaces and nodes on the global roadmap. Our method, together with other planners (route planner, stuck recovery, and so on) are run in parallel for tasks such as exploration, go-to waypoint, and return home. On top of these planners, behavior executive and multi-robot coordination modules are built specifically for the challenge.

Table S7 Frequently Used Notations

Notation	Meaning
$\mathcal{W}$	The workspace to be explored.
$\mathcal{W}_{\text{trav}}$	The traversable subspace of the workspace.
$\mathcal{S}$	Union of surfaces perceived by sensors along a path.
$\mathcal{S}_{\text{cov}}$	A subset of the perceived surfaces that is considered "covered".
$\overline{\mathcal{S}}$	A subset of the perceived surfaces that is considered not "covered".
$\mathcal{H}$	The local planning horizon.
$\mathcal{H}_{\text{trav}}$	The traversable subspace of the local planning horizon.
$\mathcal{C}_{\text{trav}}^{\mathcal{H}}$	The corresponding configuration space of $\mathcal{H}_{\text{trav}}$.
$\mathcal{T}$	The overall exploration path for a single robot.
$\mathcal{T}_{\text{local}}$	The local portion of the exploration path that is within $\mathcal{H}$.

$\mathcal{T}_{\text{global}}$	The global portion of the exploration path.
$\{{\mathcal{T}_{\text{global}}}^{i}\}, i \in \{1, \ldots N\}$	A set of global paths for all robots, which is the output of each robot's global planning process.
$\hat{\mathcal{G}}$	A set of exploring subspaces, which need to be visited by robots for completing the exploration.
$\tilde{\mathcal{G}}$	A set of explored and exploring subspaces.
$\tilde{\mathcal{G}}_i^j$	What robot i believes about robot j 's knowledge of the environment, which is a set of explored and exploring subspaces.
$\tilde{\mathcal{G}}_i^{i\dagger}$	Robot i's updated belief about robot j 's knowledge of the environment, after communicating with robot j.
$\hat{\mathcal{G}}_{\text{new}}$	The newly discovered exploring subspaces, which are only known to robot i before it communicates to other robots.
${\mathcal{T}_{global}^{\dagger}}^{j}$	A robot's updated global path for robot j, which hypothetically assume that robot j knows about $\hat{\mathcal{G}}_{\text{new}}$.
${\mathcal{T}_{\text{comms}}}^{i}$	Robot i's global path to pursue other robots, which serves to deliver the information about $\hat{\mathcal{G}}_{\text{new}}$ to other robots.