Air-Ground Collaboration with SPOMP: Semantic Panoramic Online Mapping and Planning

Ian D. Miller, Fernando Cladera, Trey Smith, Camillo Jose Taylor, Vijay Kumar*

Abstract

Mapping and navigation have gone hand-in-hand since long before robots existed. Maps are a key form of communication, allowing someone who has never been somewhere to nonetheless navigate that area successfully. In the context of multi-robot systems, the maps and information that flow between robots are necessary for effective collaboration, whether those robots are operating concurrently, sequentially, or completely asynchronously. In this paper, we argue that maps must go beyond encoding purely geometric or visual information to enable increasingly complex autonomy, particularly between robots. We propose a framework for multi-robot autonomy, focusing in particular on air and ground robots operating in outdoor 2.5D environments. We show that semantic maps can enable the specification, planning, and execution of complex collaborative missions, including localization in GPS-denied settings. A distinguishing characteristic of this work is that we strongly emphasize field experiments and testing, and by doing so demonstrate that these ideas can work at scale in the real world. We also perform extensive simulation experiments to validate our ideas at even larger scales. We believe these experiments and the experimental results constitute a significant step forward toward advancing the state-of-the-art of large-scale, collaborative multi-robot systems operating with real communication, navigation, and perception constraints.

1 Introduction

Multi-robot heterogeneous systems provide benefits in a wide range of tasks (Rizk et al., 2019). Consider one such task: exploratory mapping, where a robot team is tasked with mapping objects of interest within some user-defined region. Such a use case arises in domains ranging from agriculture (Ribeiro and Conesa-Muñoz, 2021; Chen et al., 2020) to search-and-rescue (Lindqvist et al., 2022), to radiation mapping (Gabrlik et al., 2021). Employing robots with diverse sensing and mobility characteristics often leads to a team that is more capable, flexible, and efficient. However, to efficiently utilize these robots, we must find a suitable representation for data communicated between robots.

It is first worth distinguishing between representations *internal* to a robot and those *shared* between robots. A robot may have some set of environment representations that it uses internally, but we do not require that robots exchange that information verbatim. Robots only share some subset of their state, not the entirety. We call the representation used by the robot that built it *internal*, and the information communicated to other robots the *shared* representation.

This leads us to the following list of properties for our shared representation:

*Corresponding author: Ian D. Miller

- **Sufficiency:** Given a suitable task-performance function, we want to minimize the penalty for communicating the *shared* representation as opposed to the full *internal* one.
- **Efficiency:** We want to minimize the resources needed for communication, such as time, power, and bandwidth.
- **Commonality:** The shared representation should be usable by all robots, even those with diverse sensing and mobility. That is, it should be invariant to variations in each robot platform.
- **Interpretability:** The representation should be understandable not only by robots, but also by human operators.

To help build intuition, and in light of our desired property of interpretability, it is helpful to consider how humans approach this problem. We typically communicate not in terms of geometry, but rather in terms of objects and their meaning for a particular task. For instance, we might communicate a room layout by describing what wall or corner a piece of furniture should be against, not in terms of cartesian coordinates or a full 3D model of the space. Because this information is encoded in terms of human language, we refer to it as *semantics*.

We argue that human language is not simply convenient, but actually reflects more fundamental properties of the environment. Semantics naturally represent properties that are invariant under a wide variety of transformations. A chair remains a chair under translation, rotation, scaling, color shifting, sensing, *et cetera*. Its "chair-ness" is an intrinsic property not dependent on any external factor. This invariance is captured by our properties of commonality and interpretability, and even sufficiency and efficiency since encoding only invariant properties throws out extraneous information that may not be useful to other agents and expensive to share.

Thus far, we have taken a very general definition of semantics. However, in practice, most works have encoded semantics into an environment by labeling geometry or objects with some set of semantic labels, or possibly a probability distribution over some set (Garg et al., 2020). The choice of this set of labels is highly dependent on the tasks a robot is intended to complete, and has the benefit of being human-interpretable by definition. For instance, classes such as `graspable` or `slippery` might be relevant to a robotic manipulator, classes such as `coffee mug` or `towel` relevant to a home assistant robot, and classes such as `road` or `pedestrian` relevant to an autonomous vehicle. In recent years, deep learning methods for rapidly and accurately labeling images and even point clouds based on training data have become widely adopted. We therefore leverage these methods throughout this work and adopt a set of classes for this work that are distinctive, detectable from the air and ground through cameras or LiDAR, and relevant for traversability, thereby ensuring that we meet all the requirements for our representation.

In this paper, we present an integrated system for air-ground robotic collaboration built on top of semantic maps. Like our prior work (Miller et al., 2022), we adopt the architecture of an aerial robot with multiple ground robots consuming the aerial data. We select this architecture because it allows us to leverage the unique capabilities of aerial and ground robots. Aerial robots can fly at high altitude to view large portions of the environment, as well as explore more rapidly and with less concern for obstacles. In addition, due to their altitude, aerial vehicles can often have line-of-sight to multiple ground robots, thereby naturally being an effective communication relay. Because of these properties, our architecture is designed to primarily exploit an aerial robot as an initial explorer and communication relay, and ground robots for more close-up investigation.

A distinguishing characteristic of our work is our emphasis on semantics at every level of autonomy. We adopt a cross-view localization system built on semantics from our prior work (Miller et al., 2021), but also employ semantics for ground robot global planning. Secondly, we heavily employ integrated depth panoramas constructed by our LiDAR odometry algorithm, LLOL (Qu et al., 2022). These panoramas are semantically segmented and used for localization, and are also used directly for local terrain assessment and

local planning. By building our approach around semantics, depth panoramas, and aerial maps, we simplify our overall autonomy stack around these core representations, all of which are compact and efficient to use.

We devote a large portion of this paper to the experimental verification and validation of our approach. While our primary contribution of this work is an autonomy framework which can be the foundation for a variety of tasks, we here focus on a particular scenario to experimentally study and characterize the system. In this scenario, the overall objective of the team is to visit all clusters of cars in a predefined region with at least one ground robot. We adopt this particular mission because it is motivated by real-world search and rescue applications and it leverages ground and aerial robots. It is also straightforward to compute informative performance metrics for this task, such as the number of clusters successfully visited and the time taken to do so. However, we note that this scenario is not our goal *in itself*, rather, we seek to construct a more general autonomy framework with it as a motivating example.

Our contributions are as follows:

- An integrated system for multi-robot collaboration, the code of which we open source (`https://github.com/KumarRobotics/spomp-system`). A video demonstrating SPOMP in operation is available there as well.
- An efficient depth panorama-based local planner for ground robots, along with a mechanism for learning and sharing compact traversability information between ground robots while incorporating aerial context.
- Experiments in simulation and the real world demonstrating active autonomous collaboration between aerial and ground robots based on a single high-level mission specification.

2 Background

2.1 Ground Robot Terrain Analysis and Planning

Motion planning approaches can be broadly categorized into classical approaches and learning-based approaches (Dong et al., 2023). Most classical approaches begin with some form of geometric map representation. This representation can take the form of a digital elevation map (DEM), voxel grid, or pointcloud, and approaches can be purely geometric or appearance-based (Borges et al., 2022).

Early works typically used a LiDAR for sensing and constructed an elevation map from the sensor pointcloud. They then estimated traversability using a metric computed on that terrain (Howard and Seraji, 2000; Larson et al., 2011). Some more recent methods, recognizing the inherent polar nature of 3D LiDAR, instead directly compute traversability on the points of an organized sweep (Reddy and Pal, 2016). This approach takes advantage of the adjacency of points in an organized point cloud. However, approaches that operate on these point-based representations ultimately bin these points into a grid, either cartesian (Neuhaus et al., 2009) or polar (Martínez et al., 2020), for planning purposes.

While geometric algorithms are straightforward and work well in many environments, not all obstacles are characterized by geometry alone. For instance, tall grass may look like an obstacle but be easily driven over, and muddy or sandy regions may appear smooth but be treacherous nonetheless. 3D LiDARs are also expensive and complex compared to cameras. To address these challenges, many have proposed vision-based classification systems (Sevastopoulos and Konstantopoulos, 2022). One early use of vision leveraged LiDAR to compute a traversable region, and then used vision to extrapolate that region to parts of an image with a similar appearance (Thrun et al., 2006). More recent approaches use supervised learning in conjunction with convolutional neural networks to directly classify images (Hosseinpoor et al., 2021). Some authors have recognized the complementary benefits of geometric LiDAR and learning-based image classification and

proposed methods for fusing the two in a hybrid system (Zhou et al., 2022) or directly learning traversability from LiDAR (Shaban et al., 2022) on the basis of human labels. Schilling et al. (2017) follow the idea of fusing LiDAR and vision, but also employ a semantic segmentation network as a pre-processing step for the image data. Semantic class, therefore, can be a useful input to the traversability model, an idea which we employ in this work.

Supervised learning-based methods typically rely on human labels of traversability. This human labelling is time consuming as well as non-transferable between different robots which may have very different mobility constraints. As a result, many works have sought to develop methods for self-supervision of traversability prediction models, where the labels are provided by a robot's own experience. In an early work (Stavens and Thrun, 2006), a ground robot statistically learns the roughness of terrains based on its interoceptive sensing. Another method (Ho et al., 2013) learns a kernel function to predict what a robot's configuration would be on a given region of terrain. Prágr et al. (2019) learn a heatmap of traversability on top of a satellite image on the basis of experienced data. More recently, methods have been proposed to learn traversability based on data from teleoperating the robot using a deep neural network (Wellhausen et al., 2019). End-to-end approaches have also been proposed, such as ViKiNG (Shah and Levine, 2022), which does not explicitly model traversability on a per-pixel or per-voxel basis, but rather directly determines the best plan based on imagery. ViKiNG additionally constructs a topological graph as it explores to aid in global planning. This approach builds on the work of other authors that have sought to build roadmaps on top of occupancy grid maps (Park et al., 2012). These traversability graphs are much sparser than a dense grid map, enabling more rapid planning, and our approach takes advantage of this.

A logical next step for self-supervised learning-based methods is to not only build a training dataset from a robot's experience, but to train the robot online *during* its experiences. These methods can use Bayesian clustering (Lee et al., 2017) or a Multi-layer Perceptron (MLP) (Frey et al., 2023) to rapidly train online on the basis of experienced data. Both approaches perform image segmentation into superpixels (Lee et al., 2017) or learned segments (Frey et al., 2023) as a preprocessing step. Our approach follows this idea on online learning with an MLP, but for the purpose of extrapolating classical geometric terrain analysis to areas that only have aerial UAV imagery.

In our work, we draw inspiration from approaches that operate directly on organized LiDAR sweeps. However, we avoid binning into grid cells and instead plan directly in this space. In addition, we incorporate ideas from learning and traversability graphs in order to use semantically labeled aerial maps to extrapolate local terrain experience across the global map. We perform this fitting and extrapolation process in real-time, feeding it back into our planner for constant refinement. Unlike these works, we also incorporate information from other robots as part of the online estimation and develop a compact representation which can be efficiently shared between robots.

2.2 Air-Ground Teaming

It has long been recognized that heterogeneous teams of air and ground robots can achieve better performance than either robot individually. Several authors have developed a taxonomy for different roles that Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) can assume in a team, including acting as a sensor, actuator, or playing an auxiliary role (Ding et al., 2021; Liu et al., 2022a).

Some of the most recent large-scale work on air/ground collaboration has come from the DARPA SubT Challenge (Chung et al., 2023; Tranzatto et al., 2022; Morrell et al., 2022; Hudson et al., 2022). These teams all utilized heterogeneous robot teams to locate various objects in diverse underground environments in a task similar to ours. Nonetheless, the challenges and constraints of underground operation are very different from those on the surface; the domain we focus on here. In particular, the advantage of a quadrotor as an eye-in-the-sky is reduced in favor of its utility in visiting difficult to reach areas. There is no way to obtain a high-altitude view of a tunnel system, but we argue that if possible for a given environment, doing so is highly valuable. In our work, we employ the UAV as both a sensor for gathering an aerial map as well as an

active communication relay. We here discuss prior work addressing both of these roles.

2.2.1 UAV for Mapping

Some authors have used UAVs and UGVs in cooperative mapping in relatively homogenous ways (Yue et al., 2020; Scherer et al., 2021). In these works, the UAVs fly at a relatively low altitude and use similar sensing to UGVs. Map representations, sensing, and viewpoints are all similar, making map fusion primarily between homogeneous maps. However, the heterogeneity of mobility of both types of robots can be used to reach different parts of the environment.

By contrast, many authors have recognized the particular strengths of UAVs in high-altitude flight, gaining a perspective with a speed that a UGV cannot compete with. One approach (Vandapel et al., 2006) takes LiDAR maps built from a crewed helicopter and uses them for localizing a UGV equipped with LiDAR, as well as for planning its global trajectories. Other approaches have replaced the expensive and heavy LiDAR sensor on the aerial platform with a vision-equipped UAV. Some of these approaches operate at relatively small scales. In one work (Su et al., 2021), a UAV identifies targets for a UGV to track. However, the UGV planning and localization operates completely independently of the UAV, rather than using the UAV only to identify goals. Fankhauser et al. (2016) similarly operate at a small scale, but perform localization and mapping cooperatively on both robots. However, the quadrotor flies low enough that image features can be directly registered between robots, and the resulting elevation map is high enough resolution for planning on it directly.

Other works perform experiments at larger scales. A major early work (Grocholsky et al., 2006) employ a team of fixed-wing UAVs and UGVs to localize targets, but use GPS for localization. Obstacles are not considered. More recently, Lazna et al. (2018) use UAV data for UGV planning, but construction of the aerial map is a computationally demanding procedure that must be run offline. Peterson et al. (2018) build an orthomap online which they use to localize obstacles for the UGV to avoid, but communication is always assumed to be available.

Our work operates completely online and does not require constant communication. In addition, we employ the aerial map for planning and localization, but do not specify traversability *a priori*, and rather allow UGVs to learn to generalize their experience using this map. We also focus on large-scale experiments, both in terms of area covered as well as team size.

2.2.2 UAV for Communication

UAVs also have properties highly conducive to facilitating communication (Li et al., 2021). They can operate at high altitude where line-of-sight to multiple UGVs is feasible, and can also travel more quickly without needing to worry about obstacle avoidance. Nonetheless, algorithms are needed to determine where UAVs should go to maximize team communication and performance.

Some authors formulate this as the relay placement problem. In a simple form, this problem can be expressed as follows: given a number of agents in the environment, select where to place relay nodes to keep the network fully connected (Mox et al., 2022). Other authors have generalized the problem further, such as having a number of set task points and robot points that a single UAV is tasked with visiting (Ding et al., 2019). This then turns into a variant of the Travelling Salesman Problem (TSP).

However, both of these authors assume that the positions of the task and/or robot points are always known. If communication is not constant, this is not a realistic assumption. Some methods have sought to address this problem. In one approach (Chamseddine et al., 2016), the UAV uses the bearing and signal strength from different UGVs to try to optimize its position. In another work (Wu et al., 2020), the UAV explicitly tries to estimate the UGV positions using a filter-based approach. Nonetheless, both of these methods rely on some form of detection of ground robots, and if communication is lost completely may struggle to determine

where to go to regain a connection.

In our work, we do not assume or try to always maintain connectivity. Rather, we seek to balance the UAV's time between exploring new regions and acting as a relay for UGVs. To keep track of UGV locations, the UGVs communicate their current position and current goal and the UAV uses this last-known information to try to revisit robots. Our approach therefore makes no assumptions about the communication model, connectivity graph, or current knowledge of robot positions, and we argue that this makes it particularly robust in real-world deployments.

2.3 Authors' Prior Work

In this work, we employ a similar high-level design to our prior work (Miller et al., 2022). This includes a modified version of the localizer from our works (Miller et al., 2021, 2022), the communication system from (Miller et al., 2022, 2020; Cladera et al., 2023), the aerial mapper from (Miller et al., 2021), and the aerial planner from (Cladera et al., 2023). Here, we focus on our new approaches for UGV terrain mapping, navigation, and learning from the aerial map, though we provide summaries of modules previously described in other works. Nevertheless, our emphasis in this paper is on the *integrated air/ground system*, and not purely on the terrain analysis component. Our experiments are much more extensive than our prior work, incorporating 3 robots across two real-world diverse and challenging environments, totalling 17.3 km autonomously travelled by the UGVs. We additionally perform many ablation and scaling experiments in simulation to further validate our approach.

3 Method

A simplified overall block diagram of our system, SPOMP (Semantic Panoramic Online Mapping and Planning), is shown in Fig. 1. The aerial robot is responsible for building a semantically segmented orthomap of the target region. As this map is being constructed, updated maps are transmitted to ground robots when they happen to be in communication range using our communication framework and distributed database MOCHA (Cladera et al., 2023). Ground robots use this semantic aerial map to localize using the cross-view localization algorithm described in our prior works (Miller et al., 2021, 2022). Robots opportunistically share data with each other using MOCHA when they are able to. All robots know their own poses in a common coordinate frame, and ground robots are in possession of the aerial orthomap. However, for the system to be autonomous, we additionally need algorithms for the ground and aerial robots to make decisions about where they can and should travel, both locally and globally. In Section 3.1 we first discuss the aerial autonomy stack, beginning with the aerial mapper ASOOM in Sec. 3.1.1 and moving to the aerial planner in Sec. 3.1.2. We then move to the ground autonomy stack in Section 3.2, beginning by describing the high level mission manager and goal selector (Section 3.2.1) and moving to the LiDAR odometry, segmentation (Section 3.2.2), and cross-view localization algorithms (Section 3.2.3) that provide a common reference frame to the robots. We conclude with the UGV local and global planners responsible for planning to the selected goals while remaining safe (Sections 3.2.4 and 3.2.5, respectively).

3.1 Aerial Autonomy

3.1.1 Aerial Mapper

We employ the same Aerial Semantic Online Ortho-Mapping (ASOOM) algorithm first presented in our prior work (Miller et al., 2022). ASOOM uses odometry from ORBSLAM3 (Campos et al., 2021) and GPS position in order to construct a semantic orthomap. This is the only use of GPS in our system. The orthomap includes several layers, including RGB color, semantic class from ErfNet (Romera et al., 2017), and elevation. Each layer is `png` compressed before being transmitted to other robots, resulting in the map

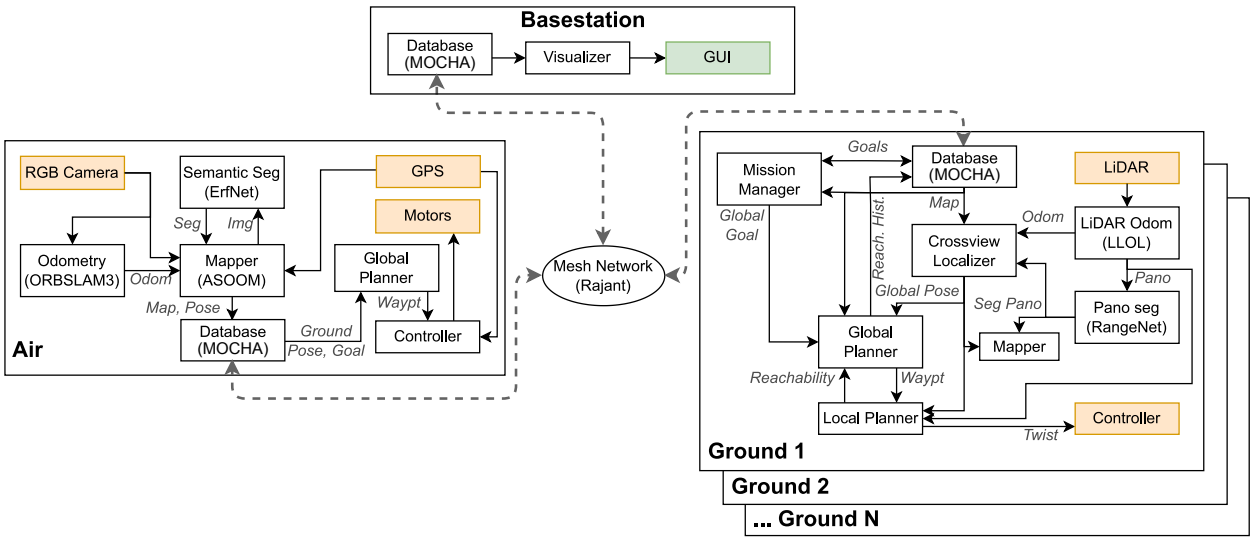

Figure 1: The SPOMP architecture described in terms of ROS nodes. Sensors and actuators are shown in orange and visualization outputs in green.

size being reduced by about a factor of 10.

3.1.2 Aerial Planner

Our aerial planner is addressed in more detail in our prior work (Cladera et al., 2023), but we provide a brief overview here. To exploit the UAV strengths of viewpoint and speed, we assign it two goals:

1. Explore the environment to build a semantic orthomap for UGV navigation.

2. Facilitate communication both to transmit the orthomap to UGVs as well as relay data between ground agents.

These goals may be contradictory: if the UAV is only exploring, it will be in new regions that the UGVs have never been, and therefore never have the opportunity to send them the aerial orthomap. By contrast, if the UAV is only facilitating communication, it will stay with the UGVs and never explore new regions. In either scenario, the team is ineffective.

The simplified state machine for the aerial planner is shown in Fig. 2. We employ a time-sharing system, where the UAV alternates between tasks, assigning a fixed amount of time to each. In exploration mode, the robot follows a preset collection of waypoints. When the timer expires, the UAV seeks out a UGV on the basis of its knowledge of the UGV's last known location and goal. The target UGV is selected in round-robin fashion. If more recent information is received en route, the UAV adjusts accordingly. Once either all data in the distributed communication system has been synchronized with the target UGV, or the UAV times out, it returns back to its last exploration waypoint, restarts the timer, and resumes exploration. Once exploration has completed, the UAV simply spends all its time flying between the UGVs sequentially.

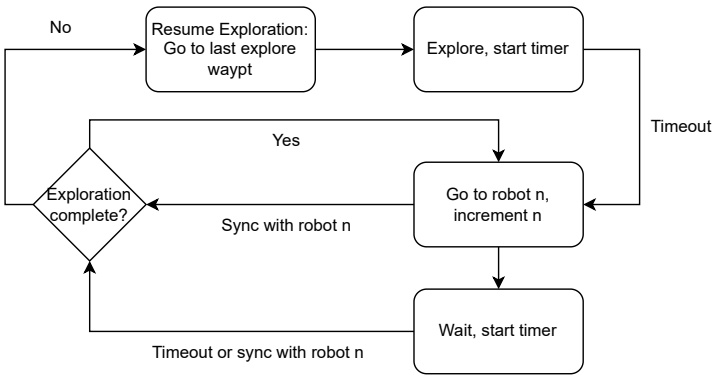

Figure 2: Simplified state machine determining the UAV high level goal.

3.2 Ground Autonomy

3.2.1 High-Level Coordination

The UGVs coordinate at the level of goal locations $g \in \mathbb{R}^2$. These goal locations can either be manually specified by an operator or selected automatically from the aerial map. If manually specified, the goals are added to MOCHA (Cladera et al., 2023) and then distributed by MOCHA through the mesh network to all of the robots. If automatically detecting goals, each robot identifies clusters of the target class (vehicles in our experiments) in the aerial semantic map, and shares the goals that it has *identified* via MOCHA. Clusters are *identified* by performing a morphological close followed by a morphological open operation on the binary class mask in order to remove noise and combine groups of vehicles. We then select the goal to be the point furthest from an obstacle within a threshold distance of a cluster in order to avoid placing goals inside buildings or other unreachable areas.

Robots also share each goal's status: whether a goal is *claimed*, *visited*, or only *identified*. A *claimed* goal is a goal that a robot is currently *en route* to, but has not yet *visited*. Goals from other robots that are within a threshold distance (10 meters in our case) of a robot's goal are considered to be the same goal for the purposes of deconfliction. The goal detection procedure is intentionally done on each robot in order to make the system entirely distributed, though goals can be provided from a central source if desired.

We consider a mission to be completed when all goal locations have been visited by at least one UGV. A given UGV will always select the closest goal in cartesian distance from itself that is not already claimed by another robot. We implement a priority mechanism to disambiguate goals, with priorities pre-assigned at mission start: if a robot learns that another robot is going to the same goal, it will switch to a new goal if the other robot has a higher priority. If a UGV is unable to find a path to a goal, or exceeds a time limit while trying to get there, it will mark the goal as unreachable and release it for other agents to attempt to reach. This procedure is described as a state machine in Fig. 3.

3.2.2 Ground Odometry and Semantic Segmentation

The basis for the ground odometry stack is a modified[1] version of LLOL (Low-Latency Odometry for spinning LiDARs) (Qu et al., 2022). LLOL takes as input the raw LiDAR sweeps and IMU and outputs per-sweep odometry as well as integrated depth and intensity panoramas approximately every meter that the robot moves. These panoramas can be thought of as virtual super-resolution LiDAR sweeps.

We then segment these sweeps using RangeNet++ (Milioto et al., 2019), also incorporating intensity as an

[1]The source code for this modified version can be found here: `https://github.com/versatran01/rofl-beta`

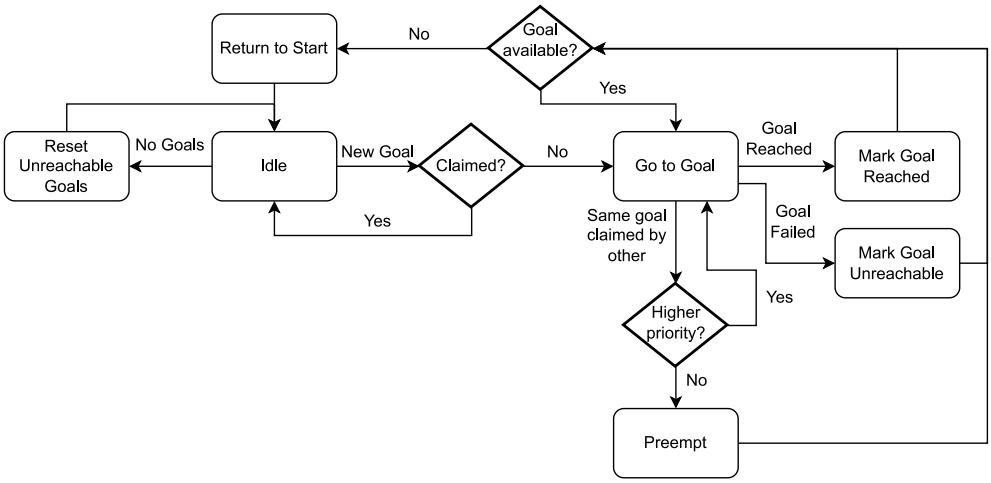

Figure 3: State machine describing the algorithm for assigning goals to ground vehicles.

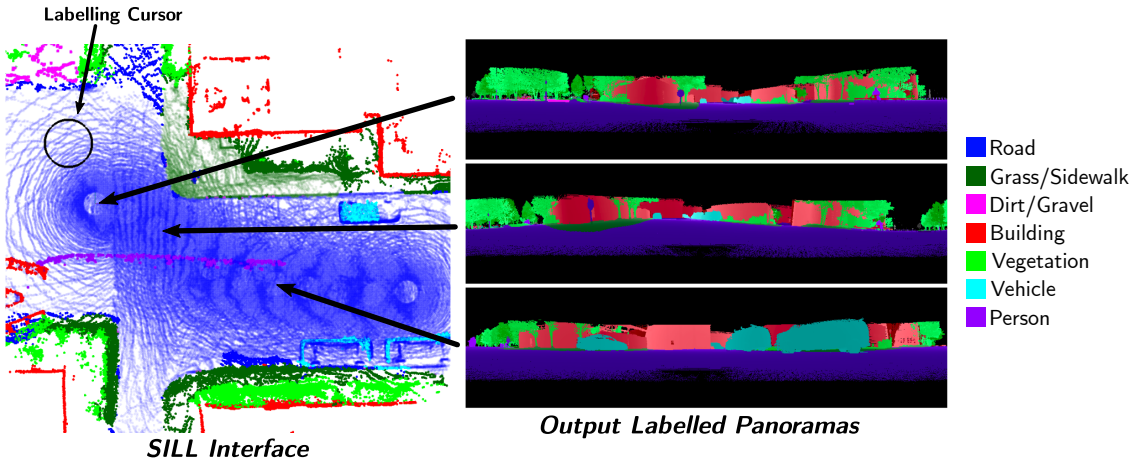

Figure 4: The Semantic Integrated LiDAR Labelling (SILL) tool and output labelled depth panoramas.

input. The classes we use are shown in Fig. 4. The `Person` class is ignored by all downstream processing. We make several modifications to the network. We first shrink the network size using the same modified structure as Liu et al. (2023). Secondly, we replace the normalized x, y, and z coordinates with x, y, and z components of the surface normal at that point. We also upweight the loss with range to encourage the network to more accurately classify more distant and sparse points.

In order to generate training data, we developed a custom labeling tool, SILL (Semantic Integrated LiDAR Labelling)[2]. The SILL interface and output labeled panoramas are shown in Fig. 4. SILL operates by first using LLOL to integrate panoramas into a large pointcloud. This pointcloud is then downsampled by voxel filtering. The key idea of SILL is to label from a top-down perspective, slicing at varying points on the z axis. Consider the example of labeling an outdoor scene. The user first selects a z value just above ground level, causing SILL to only display points at or below the ground. The user can therefore focus on labeling just the various types of ground before increasing the z threshold, at which point they can easily label above-ground objects. This is because points which have already been labeled at lower z are frozen and will not be overwritten by the new classes, making it easy to label overhanging vegetation without changing the class of the road beneath. Using SILL, we were able to rapidly label training data from our test environments.

[2]Source: `https://github.com/iandouglas96/sill`

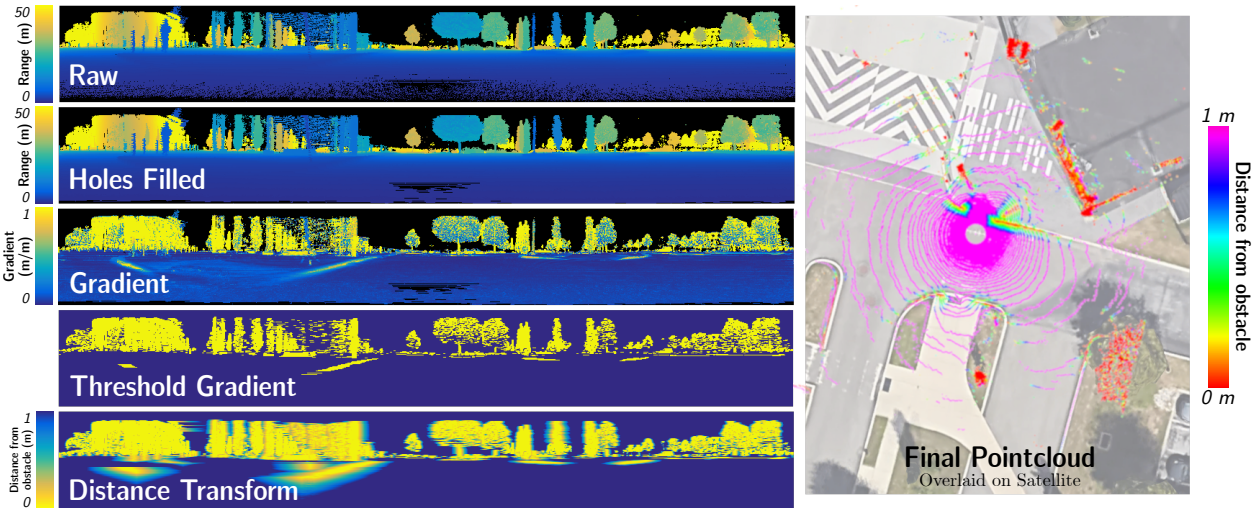

Figure 5: Visualization of the panorama after each processing step, and the final projected obstacle point cloud.

3.2.3 Cross-View Localizer

The primary modification made to the localizer (Miller et al., 2022) was to operate on integrated semantic panoramas instead of each LiDAR sweep. This significantly reduces the computational load, because even though the higher resolution panoramas take longer to segment, they are generated far less frequently than the 10 Hz LiDAR sweeps. Additionally, downstream processes such as localization run less often. Because new panoramas are generated less frequently and higher frequency pose estimates are needed for planning and control, we rely on the LLOL odometry in between panoramas.

3.2.4 Local Ground Planner

A key design decision that we made was to use depth panoramas as much as possible for planning. While it is traditional to construct an occupancy grid or elevation map and plan on that representation (Papadakis, 2013), by directly planning in panorama space we are able to optimize the local planner by parallelizing operations on a per-row basis, increasing computation speed. When parallelized, the entire terrain analysis pipeline takes approximately 3.5 ms per-panorama on an AMD Ryzen 3900X. The nature of 3D spinning LiDARs inherently results in higher resolution data close to the robot and coarser data further away, which is convenient due to regions near the robot being more important for planning. We therefore avoid the loss of data, particularly near the robot, that would result from binning points into a cartesian grid.

Let $p = (p_e, p_a)$ be a pixel coordinate on the panorama, where p_a and p_e are the coordinates in the azimuthal and elevation directions respectively. Then $\rho(p) = \rho(p_e, p_a)$, $\theta(p)$, and $\phi(p)$ are the range, elevation, and azimuth at that point respectively, and $x(p)$, $y(p)$, and $z(p)$ the cartesian coordinates. We also let $r(p) = \sqrt{x^2(p) + y^2(p)}$, and let the panorama have dimension $E \times A$. Finally, we assume that there is noise in the depths of the panorama of order η. We decompose this into $\eta_z(p) = \eta \sin(\theta(p))$ and $\eta_{xy}(p) = \eta \cos(\theta(p))$, the components perpendicular to and parallel to the xy plane.

Terrain analysis consists of a number of steps operating on each panorama. We consider each step in more detail here, and the process is illustrated in Fig. 5.

Fill holes and project: The depth panorama is typically of higher resolution than the raw LiDAR data, and so therefore there are often holes in the panorama with unknown depth. We therefore loop through each row in the panorama and fill holes below a certain threshold D_{px} in size with linearly interpolated depths. This threshold is scaled with the range of the points in question such that it can be approximately specified in meters D_m. More formally, at a point p we have the arc length between adjacent points in the panorama

$$s(p) = (2\pi/A) * \rho(p). \tag{1}$$

Then we have

$$D_{px}(p) = D_m/s(p). \tag{2}$$

Once this has been completed, we compute the per-pixel cartesian projection of the depths, needed for the later gradient computation. Both operations can be done in parallel.

Compute and threshold gradients: We now compute the gradients on a per-pixel basis. To make this operation more efficient, we take advantage of the known adjacency in the panorama structure and compute the gradient along the azimuthal and elevation directions. This is an approximation, because there is no guarantee that the vectors from the current point to the neighboring points in the panorama are in fact perpendicular, but it is a reasonable assumption on an approximately planar surface. To minimize noise, we want to compute the gradient using a window of approximately W meters in size. Due to the polar nature of the coordinate system, the corresponding window size in pixels varies depending on its position in the panorama. We therefore first compute the window sizes w_a and w_e for the azimuthal and elevation directions

$$\begin{aligned} w_a(p) &= \min\{W/s_{xy}(p), W_{max}\}/2 + 1 \\ w_e(p) &= \min\left\{\delta \in \mathbb{N} \,\middle|\, h_s * \left|\frac{1}{\tan\theta(p)} - \frac{1}{\tan\theta(p_e - \delta, p_a)}\right| \geq W\right\} \end{aligned} \tag{3}$$

where s_{xy} is the arc length on the $x-y$ plane, in other words $s_{xy}(p) = (2\pi/A)r(p)$ and h_s is the approximate height of the LiDAR above the ground when the robot is on a flat surface. w_a is computed using the known range at p to estimate the arc length, and w_e is computed by assuming that the robot is on a flat surface and determining the closest distance at which successive rows of the panorama are separated by at least W.

With these in hand, we are now ready to compute the gradients:

$$\begin{aligned} \nabla_a(p) &= \frac{\max\{|z(p_e, p_a - w_a(p)) - z(p_e, p_a + w_a(p))| - \eta_z(p), 0\}}{s(p) * (w_a(p) * 2 + 1)} \\ \nabla_e(p) &= \frac{\max\{|z(p) - z(p_e - w_e(p), p_a)| - \eta_z(p), 0\}}{|r(p) - r(p_e - w_e(p), p_a)| + \eta_{xy}(p)} \end{aligned} \tag{4}$$

Note that we add and subtract the components of η in order to yield the minimum gradient that could be present if the noise is of magnitude η. Finally, the total gradient magnitude is given by $|\nabla(p)| = \sqrt{\nabla_a^2(p) + \nabla_e^2(p)}$. We threshold the panorama at a particular gradient magnitude ∇_{thresh} to form a binary image.

Compute per-point closest distance to obstacle: At this point, we have a per-pixel panorama and point cloud labeled as obstacle or not. We then need to inflate the obstacles for planning purposes. Since our design goal is to avoid any sort of grid projection, we now want to efficiently compute the distance for each point to the nearest obstacle point. We do this with successive sweeps along the altitude and then the azimuth of the panorama. The algorithm is formally described in Alg. 1.

We first iterate down each column, keeping track of the last time an obstacle was encountered and the distance to it. Note that we only care about the distance to an obstacle coming from the robot's direction, so therefore we only need to make this one column pass. We then make two passes along each row in both directions, again adding up the distances from the last closest obstacle. Note that this addition overestimates

Algorithm 1 Distance Transform Approximation

Input: Depth panorama $\rho(p)$, Obstacle panorama $O(p) \in \{\texttt{true}, \texttt{false}\}$
Output: Distance transformed obstacle panorama $D(p) \in [0, D_{max}]$

$D \leftarrow D_{max} \forall p$
$D_a \leftarrow D_{max} \forall p$
for $p_a \in \{1, \ldots, A\}$ **do**
    $D_{obs} \leftarrow \infty$
    for $p_e \in \{1, \ldots, E\}$ **do**
        if $O(p_e, p_a)$ **then**
            $D_a(p_e, p_a) = 0$
            $D_{obs} \leftarrow \min\{\rho(p_e, p_a), D_{obs}\}$
        else
            $D_a(p_e, p_a) \leftarrow \min\{|\rho(p_e, p_a) - D_{obs}|, D_a(p_e, p_a)\}$
        end if
    end for
end for

function UpdateAz(p_e, p_a, p_a^{last})
    $D_{obs} \leftarrow D_a(p_e, p_a^{last}) + |p_a - p_a^{last}| * s(p_e, p_a^{last})$
    if $D_{obs} > D_a(p_e, p_a)$ **then**
        $p_a^{last} \leftarrow p_a$
        $D(p_e, p_a) \leftarrow \min\{D_a(p_e, p_a), D(p_e, p_a)\}$
    else
        $D(p_e, p_a) \leftarrow \min\{D_{obs}, D(p_e, p_a)\}$
    end if
end function

for $p_e \in \{1, \ldots, E\}$ **do**
    $p_a^{last} \leftarrow 1$
    for $p_a \in \{1, \ldots, A\}$ **do**
        UpdateAz(p_e, p_a, p_a^{last})
    end for
    $p_a^{last} \leftarrow A$
    for $p_a \in \{A, \ldots, 1\}$ **do**
        UpdateAz(p_e, p_a, p_a^{last})
    end for
end for

the actual distance to the obstacle, but in practice, this algorithm yields a good enough approximation and is very fast.

As a final step, we compute what we call the *reachability scan* and denote $S(\phi) \in \mathbb{R}^+ \times \{\texttt{true}, \texttt{false}\}$. For each ϕ, S gives us the known safe distance that the robot can travel in that direction, and a boolean value for whether there is a known obstacle at the end of that distance or not. S is computed by scanning up the specified column of the panorama until either the distance between successive scan rows exceeds a threshold (no obstacle known) or the distance to an obstacle drops below a threshold (known obstacle). As a result, we have a known safe region, known unsafe regions, and regions that could be safe or unsafe but we are too far away to determine.

Controller: Given a global goal $\mathbf{G}$, we must first compute a local goal for the controller. To do this, we randomly sample uniform points $\mathbf{x}$ within the known safe region specified by S. We compute the cost of a choice of local goal as

$$C(\mathbf{x}) = \|\mathbf{x} - \mathbf{G}\|_2 + \alpha \frac{\|\mathbf{x} \times \mathbf{g}_{old}\|_2}{\|\mathbf{g}_{old}\|_2} \tag{5}$$

where $\mathbf{g}_{old}$ is the last local goal, and α is some constant. We then select the new local goal $\mathbf{g} = \text{argmin}_{\mathbf{x}} C(\mathbf{x})$. A goal that is in the known safe region, close to the global goal, and close to the old last goal will have the lowest cost. The final term incentivizes the robot to be consistent in its choice of local goal instead of, for instance, switching back and forth constantly between two possible paths around an obstacle.

For our low-level controller, we take the Dynamic Window Approach (DWA) (Fox et al., 1997). We uniformly sample the control space of the robot, which for our skid-steer platform is $(v, \omega) \in \mathbb{R}^2$, where v and ω are the linear and angular velocities, respectively. We perform this sampling around the current control inputs which naturally imposes a smoothness constraint on the controller. We then roll out that control for a time window T_w into a trajectory $\tau(t)$, $t \in [0, T_w]$ and compute a cost C for each trajectory τ:

$$\begin{aligned} C(\tau) &= C_g(\tau) + C_o(\tau) \\ C_g(\tau) &= \|\mathbf{g} - \tau(T_w)\|_2 + \frac{\|\mathbf{g} \times \tau(T_w)\|_2}{\|\mathbf{g}\|_2} \\ C_o(\tau) &= \frac{1}{T_w} \sum_{0 \le t \le T_w} -D(\Phi(\tau(t))) \end{aligned} \tag{6}$$

where Φ is a function that maps from a cartesian coordinate to the indices of a nearby point in the panorama. When sampling a point on the trajectory, we choose a point that is slightly in front of the actual robot position. This implicitly incentivizes the robot to turn to face in the direction of the goal. We then act on this computed control input, and replan when we receive new odometry.

3.2.5 Global Ground Planner

The core of our global planner is a traversability graph. Each edge e_i possesses a cost $C(e_i) = -\log P(e_i)$, where $P(e_i)$ is the probability of a robot successfully traversing e_i. Therefore, when we solve for the lowest cost path, we are finding the path with the overall highest probability of successful traversal, assuming that the probabilities for each edge are independent. We employ a sparse traversability graph for several reasons:

- Sparsity makes graph search very fast, especially for large environments.
- Maintaining a traversability graph allows us to keep track of traversed edges, thereby building a roadmap of known routes.
- Traversability, we argue, is fundamentally a property of an edge. It encodes not whether a robot can exist at a point, but rather whether a robot can get from point A to point B.

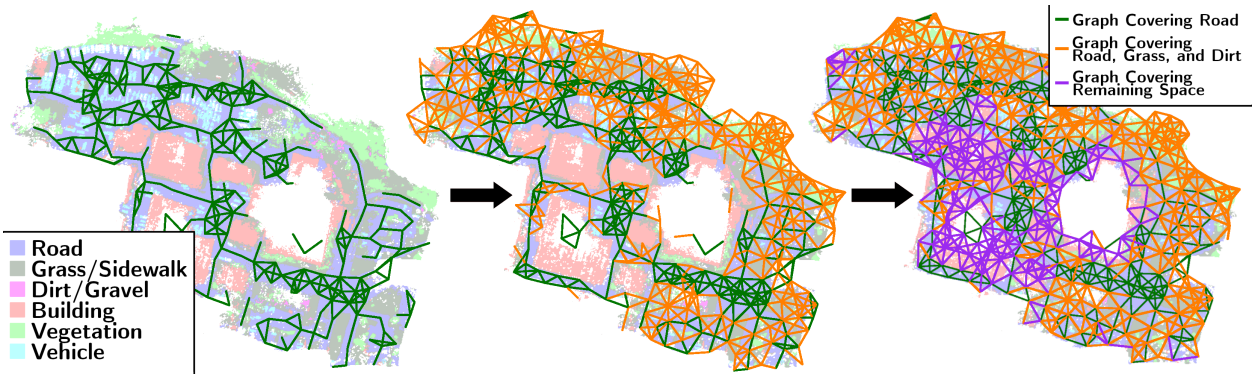

Figure 6: Visualization of the sequence of graph construction. We begin by building a graph that well-covers the easiest to traverse class (road), and then extend the graph to additionally cover the next easiest to traverse, each class at a time.

This notion of constructing a traversability graph on top of a grid map is not new (Park et al., 2012). However, we combine this graph with a learned traversability model that exists on top of the aerial orthomap, especially the semantic map. In addition, we allow a robot to update this graph with its own and other robots' experienced data. Here we first discuss the construction of the graph, then how it is updated from the robots' own experience, and finally our method of learning to extrapolate those experiences using the aerial map.

Graph Construction: We take a similar approach to graph construction as in our previous work (Miller et al., 2022). For a given region that we wish to build a graph over, we sample the point that is the farthest away from the edges of the region. We then raytrace a region around that point, which is considered covered by the graph, and repeat the sampling process ignoring points within the covered region. This process also allows us to preserve the already-built graph as the aerial map is expanded, which is important to retain traversability data encoded in the edges. The choice of region to cover is configurable. We can either cover the entire known region, or we can consecutively cover regions for each class in a specified order, typically approximately from most to least traversable, as shown in Fig. 6. The progressive method ensures that we do have edges in the regions that are traversable while keeping possible path options in case all the roads are blocked. At the end of this process, we have both a graph and a map of the regions raytraced around each node. We use this map to rapidly lookup accessible nodes on the graph nearby to any arbitrary point.

Learning Experience Extrapolation: It is broadly experientially true that regions that appear similar from the air are often similar on the ground. More concretely, a region of the map that is traversable and looks like a road suggests that other road-like patches of the map are also readily traversable. Some authors have noticed this and used this observation to fit a traversability model to local observations by a ground robot (Prágr et al., 2019), in this case using Locally Weighted Projection Regression (LWPR). In a follow-up work, the authors use a haptic sensor to probe traversability, thereby learning a traversability model online (Prágr et al., 2023). Taking note of these works, and observing that we already have a large aerial map available to us, we postulate that the ground robots should be able to extrapolate their local experience to the broader map, without any *a priori* information about class traversability. We additionally observe that there is a strong predictive correlation between semantic class and traversability, as noted in Section 3.2.5. For this reason, and because we already have access to the aerial map, we employ the semantic classes as one input into our learned terrain model. We do use an *a priori* ranking of terrain classes for graph construction, but we wish to construct a more sophisticated model based on the robots' own experience.

We model the probability of traversability as $T(p) = f(\mathbf{C}(p), \mathbf{S}(p), h(p))$, where p is the cell of the orthomap, $\mathbf{C}(p)$ is the RGB color vector at that point, $\mathbf{S}(p)$ is a vector of the distance to the nearest pixel of each

semantic class at that point, and $h(p)$ is the elevation at that point. Note that using the distance to all classes provides richer information to the model than the class at the point, and also allows the model to incorporate information in the local region of the point in question without needing to use convolutions. Therefore, $f : \mathbb{R}^{3+N_C+1} \mapsto [0, 1]$, where N_C is the number of semantic classes. We approximate f with a Multi-Layer Perceptron (MLP). We make this choice because we want to approximate an arbitrary and nonlinear function, and we need a very small and simple model that can be trained quickly onboard the robots on the CPU. Our fully-connected MLP has a single hidden layer of size 10 with a sigmoid activation function and L2 regularization of 0.01, and is trained online using RMSProp (Hinton et al., 2012) as implemented in mlpack[3] with default parameters. We normalize each input channel and perform a softmax on the output to obtain the final traversability probability.

The primary output of the local terrain analyser is the reachability scan $S(\phi)$, which is also associated with a pose. $S(\phi)$ defines a star-shaped region that is known to be safe as well as the edges of obstacles. Using the associated pose of S, we can label the traversability map pixels within the safe region as `trav` and the pixels just beyond the untraversable boundary `not_trav`. We assume that obstacles have a width of at least the size of a pixel of the obstacle map. This assumption does not hold for thin obstacles like fences, but these obstacles are often not detected from the air, and in practice, we have nonetheless obtained good results. We therefore now have a set of pixels with labels and feature vectors, and can train our MLP on this data, predicting traversability probabilities for all pixels on the map. Training is done online on each robot while it is exploring, and is repeated every 10 seconds. We perform no prior training and retain no data or weights between experiments, rather, all training data is gathered by the robot during the course of each experiment.

Given a graph edge $e = (e_1, e_2)$, we now want to compute its cost $C(e)$ for the purposes of graph search. Recall that $C(e) = -\log P(e)$, or the negative log of the probability of traversing the edge. In order to approximate this, we discretize e by sampling uniformly along the edge at points $s_{e,i}$ for $i \in [0, N_d]$. Then

$$C(e) = \sum_i -\log T(s_{e,i}) \tag{7}$$

Note that C is the sum of negative log-likelihoods, meaning that when we solve for the lowest-cost path, we are maximizing the probability of traversal under the assumption that the traversabilities of all cells are independent. This assumption is of course not true, but it does provide some physical intuition for the planner. In practice, in order to handle pixels with unknown descriptors, we take the sum over the known points and then divide it by the number of known points, finally multiplying by N_d.

Updating Graph with Experience: As we have observed, $S(\phi)$ defines a known safe region as well as known unsafe edges. Given this information and an edge in the traversability graph, we would like to determine if the edge is traversable (`trav`), not traversable (`not_trav`), or we are currently unable to determine the edge's traversability (`unk_trav`). These cases are illustrated in Fig. 7.

Because the known safe region is defined as a star, we can compute a set of line segments between all the neighboring points. We then check for intersections between this set of line segments and the query traversability edge. If there are none and the endpoints lie within the safe region, then the edge must lie entirely in the safe region and therefore be `trav`. Alternatively, if both edges lie outside the safe region and there are no intersections, the edge is `unk_trav`. If the edge has more than one intersection, we also assign it `unk_trav`, adopting a wait-and-see approach. If an edge has a single intersection, there are again two cases. If there is an unknown point in S within $\Delta\phi$ of the crossing, we assign `unk_trav`, and if not we finally determine the edge to be `not_trav`.

If an edge is observed to be traversable, then we can assign it a very high traversability probability, and if it is not a very low one. We override the estimated traversability from the map to do this. In practice, observations are noisy, and it is important to not mark an edge as traversable if it is in fact not. In this case, the robot may get stuck trying to traverse the edge and not realize that it is not possible to do so.

[3] https://mlpack.org/

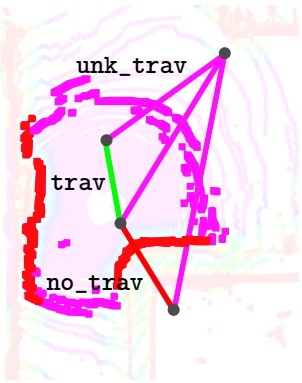

Figure 7: Reachability scan with several overlaid edges in the traversability graph labeled with their traversability determination. The `trav` edge is considered traversable, `no_trav` edge considered not traversable, and the `unk_trav` edges considered unknown.

To address this, we require repeated, consistent observations to mark an edge. Once an edge is marked `not_trav` and `locked`, it can never be changed. In addition, it cannot be traversed at all and is therefore effectively removed from the graph for the purposes of graph search. An edge marked `trav` can be changed to `not_trav`, but it requires even more consistent observations. This procedure is described in more detail in Fig. 8.

Sharing Experience: Using this framework, we now let robots share the accumulated sets of reachability scans that each have made. All the robots are in the same coordinate frame due to the cross-view localizer, so robots can treat scans received by other robots in precisely the same way that they treat their own. Scans are both incorporated into the map for the traversability model input as well as to directly update the graph. We do prevent edges from being marked `not_trav` on the basis of data from other robots. In other words, the only way an edge can be completely removed from the graph is from a robots' own observations. This prevents noisy or incorrect data from other robots from irrevocably affecting other robots' graphs.

4 Simulation Experiments

We perform a number of experiments in our Unity-based simulator environment shown in Fig. 9. There are 15 clusters of cars in the environment, though in practice on occasion the goal detector will select multiple goals near one cluster.

For these experiments, we run the entire autonomy stack for all robots, with two main exceptions. Firstly, semantic segmentation is simulated. ASOOM is running, but just uses the ground truth image segmentation instead of ERFNet. Secondly, instead of running LLOL, we use ground truth odometry with a small amount of Gaussian noise added. We additionally simulate the LLOL depth panoramas and their semantically segmented counterparts every second, which are fed into the downstream autonomy stack. We make these choices for several reasons:

- Segmentation and odometry are the most computationally expensive portions of the system, so simulating them allows us to test larger teams.
- Segmentation and LiDAR odometry are very well-studied algorithms, and not the focus of this work.
- High-fidelity sensor simulation for reasonable segmentation and odometry testing is difficult to obtain, and without it, running these algorithms in the loop is not representative of real-world performance.

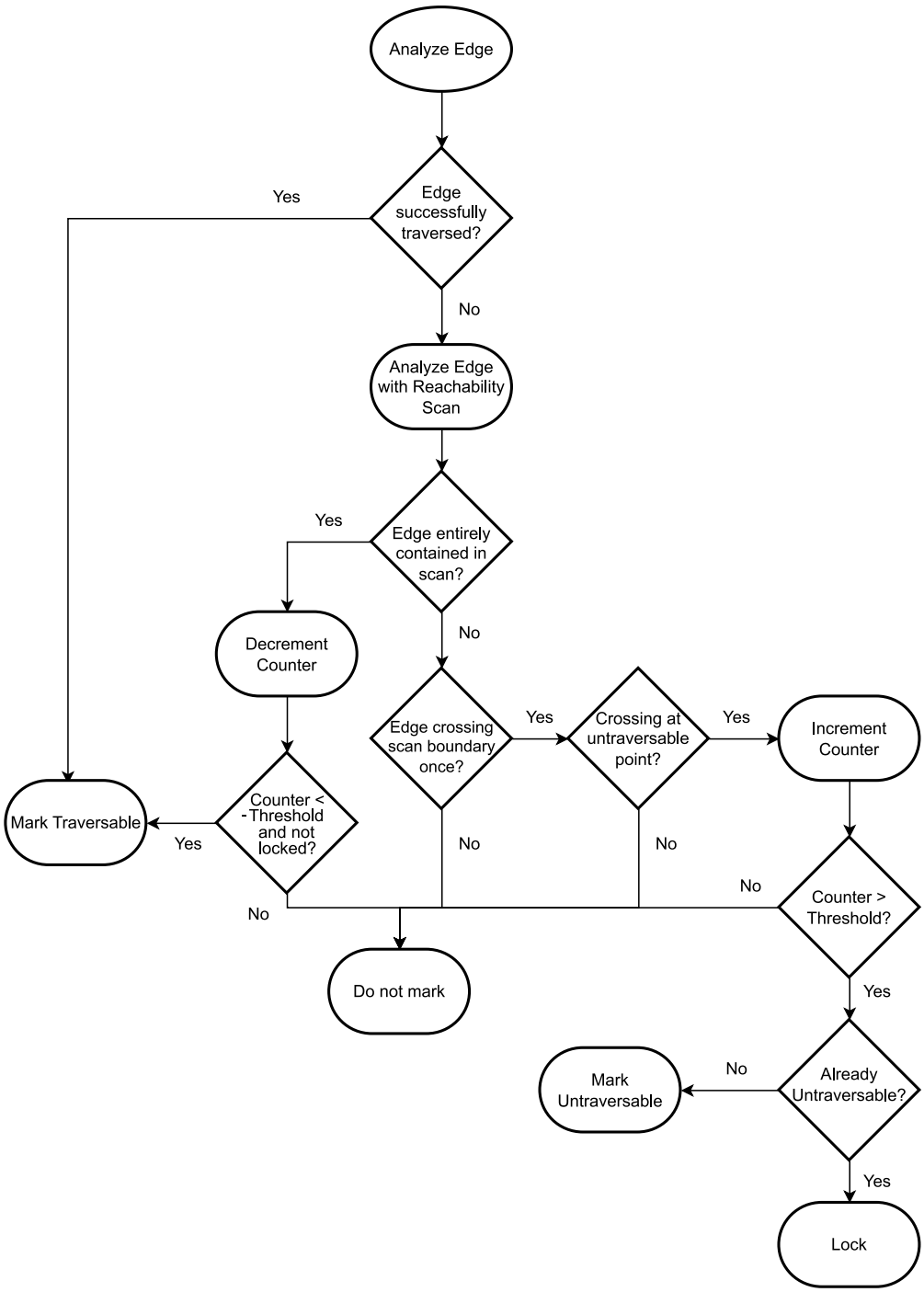

Figure 8: Flowchart showing the assignment of different edge states.

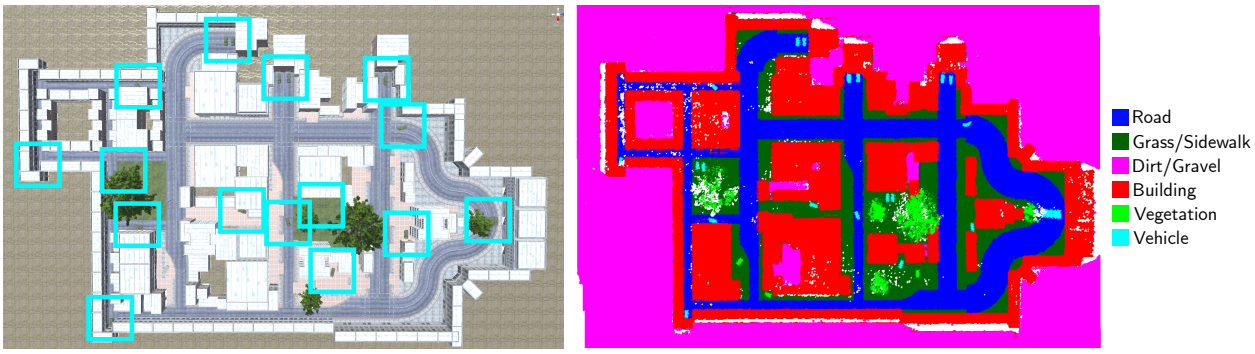

Figure 9: Simulation environment, with a color aerial image on the left where clusters of vehicles are highlighted with blue squares. The semantic map constructed by the UAV is on the right.

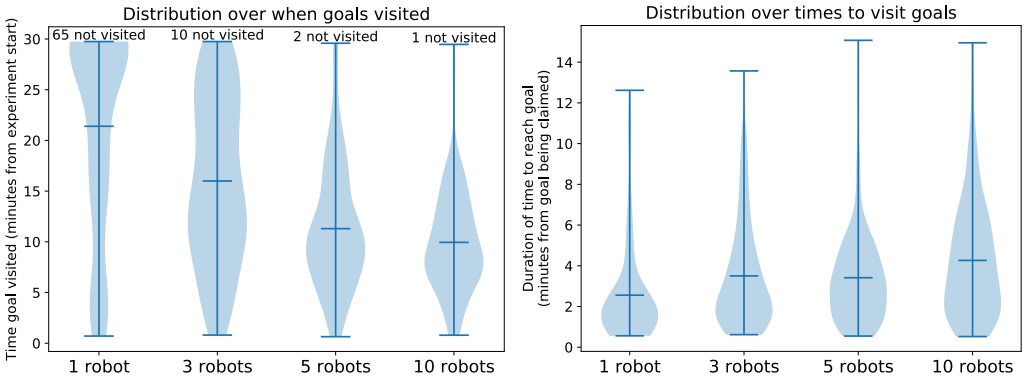

Figure 10: (Left): Distribution of the time taken to visit goals after mission start. Goals not visited are assigned a default visit time of 30 minutes for the purposes of visualization. (Right): Distribution of the time it takes between a given robot claiming and reaching a goal, for those goals which are visited (see Fig. 3)

.

The distributed communication system is also simulated. If two robots come within 50 meters of each other, they synchronize their last known positions. If they come within 30 meters, they synchronize everything. This approximates the behavior of the distributed communication system where the highest priority messages may start syncing first, and after some time or after the link quality improves the full synchronization is completed. There is a basestation at the starting point, which like the basestation in the real world experiments acts as a static node for the distributed communication system that all robots synchronize with opportunistically. All experiments are run for 30 minutes, and we run 10 copies of the same experiment for each configuration.

4.1 Scaling

We first investigate the performance of our system as the number of UGVs is increased. In all of these experiments, we have a single UAV. In Fig. 10 we visualize the times that goals are visited. Note that as the number of UGVs increases, the goals are visited earlier, and more goals are visited, suggesting that our system is able to utilize multiple robots to improve overall performance. Interestingly, increasing the number of robots also appears to increase the time taken to reach a goal, but this is likely mainly due to the further goals not being reached with fewer robots, skewing the distribution.

We also analyze how the robots are spending their time. Robots may be either headed to a goal or not. If they are headed to a goal, they may ultimately visit that goal, or they may time out or be preempted. In

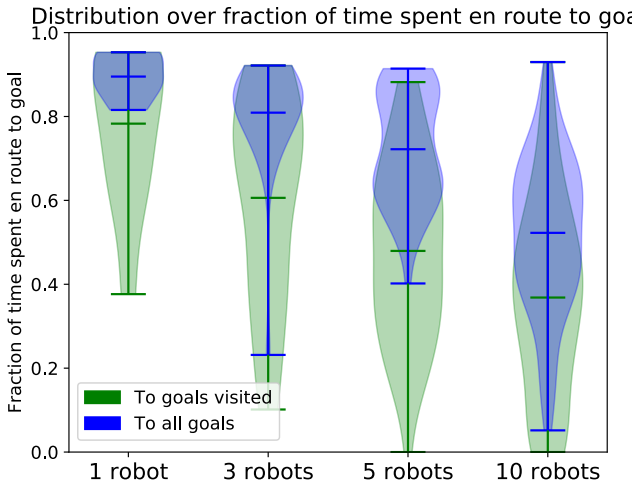

Figure 11: Distributions of the fraction of time robots spend actively navigating to a goal for different team sizes.

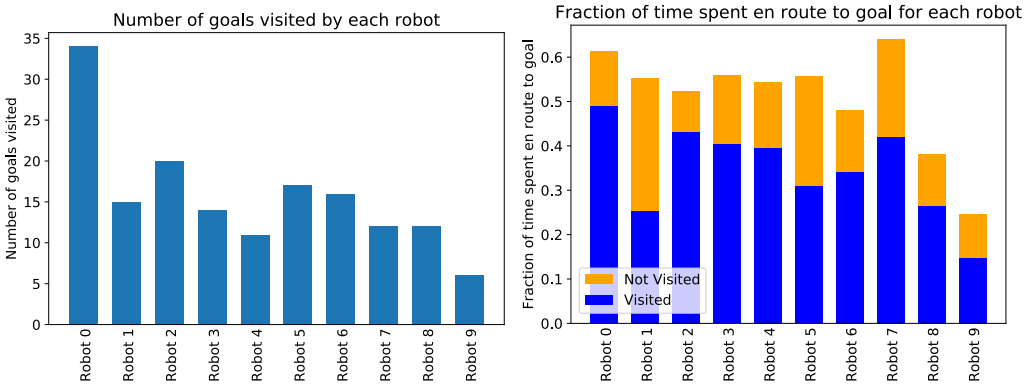

Figure 12: Robots' productivity on a per-robot basis for a 10-UGV team.

Fig. 11 we visualize the distribution over all robots and experiments. Note that the fraction of time that a robot spends headed to a goal decreases as the number of robots goes up. This suggests that while adding more robots helps, there are also diminishing returns because those robots spend less time doing useful work.

We further analyze this phenomenon on a per-robot basis in Fig. 12. Robots are organized in decreasing priority, and note that robots of lower priority generally visit fewer goals as well as spend less time visiting goals. This suggests that there simply are not enough goals being fed to all robots to use them all effectively. Given that there are approximately 15 goals, 10 robots, and the goals are gradually discovered over time, this is an intuitive result.

All of these experiments are considering only a single UAV. We propose two ways of scaling up the number of UAVs. In our prior work (Cladera et al., 2023), two UAVs are deployed, where the mission of one UAV is to perform only communications. This approach may lead to network congestion with most data travelling through a single node, decreasing the overall performance of the system. We can solve this problem by running multiple parallel teams, where each team independently explores a different area. This scales in a perfectly parallel fashion, avoiding network congestion, but we lose any benefits of interaction between teams beyond simply dividing up the environment. Under this architecture, the relevant scaling metric is the UAV/UGV ratio, as each UAV is assigned an independent UGV team. However, if inter-team communication and collaboration are allowed, the system becomes more complex, and we leave this for future work.

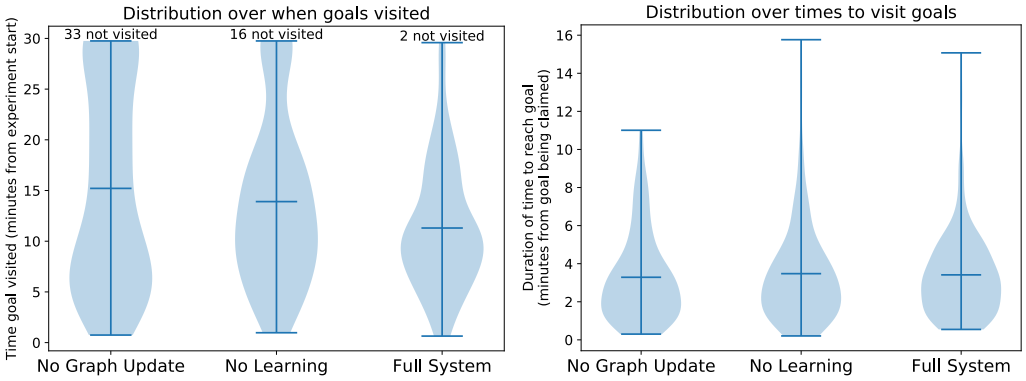

Figure 13: For different terrain learning ablations, (Left): Distribution of the time goals are visited from mission start. Goals not visited are assigned a visit time of 30 minutes for the purposes of visualization. (Right): Distribution of the time it takes a given robot to reach a goal once that robot has claimed the goal, for those goals which are visited.

4.2 Learning Traversability

In order to analyze the effectiveness of our terrain analysis and learning system, we perform several ablation experiments. For these experiments, we use a team size of 5 robots. In one set of tests, we disable the terrain learning system. Instead, we rank the classes by traversability and assign each edge a number T_{cls} corresponding to the most difficult class to traverse. We then weight the edge $W = 100^{T_{cls}} - 1 + D$, where D is the minimum distance from a point on the edge to the nearest next-highest class. This causes the planner to find a path entirely on the easiest class, and if that fails to include the next highest class, and so on.

We perform another set of tests where in addition to disabling the learning system, we also disable any updates of the traversability graph. In other words, the traversability graph is frozen from the beginning, and there is no global replanning at all. All global paths planned are therefore static.

In Fig. 13 we perform the same analyses as in Fig. 10, but instead with varying terrain learning modes. Note that removing updates to the graph (that is, disabling global replanning) significantly affects the team's performance. It does not have a large effect on the time to visit goals that are eventually reached, but it does decrease the number of goals that are ultimately visited. For all experiments, we erect several roadblocks in the environment which are labeled semantically as road. Without global replanning, the robots are unable to handle these obstacles, because they have no way of adapting their global plan.

Removing the terrain learning system also has an effect. Fig. 14 gives us some clues as to why. The environment has a number of curbs between the sidewalks and roads. There are ramps that the robots can use to drive up onto the sidewalk from the roads, but these have the same semantic class as sidewalk. Some of these areas are highlighted by pink boxes in Fig. 14. Note that the robot experiences ramps in one area of the map but is able to extrapolate them to a different region. Even though they are the same semantic class, the model is able to learn cues from the ramp colors. This means that for those goals which are on the sidewalks, robots are able to find paths up ramps to them much more quickly than without the terrain learning system.

We additionally analyze the robots' time as in Fig. 11 in Fig. 15. The effect here is less pronounced, which is unsurprising. Robots spend about the same amount of time going somewhere, but the full system spends a larger time going to a goal that they will eventually visit. This is due to the goals being reached instead of timing out, and correlates with the reduced number of goals that are never visited.

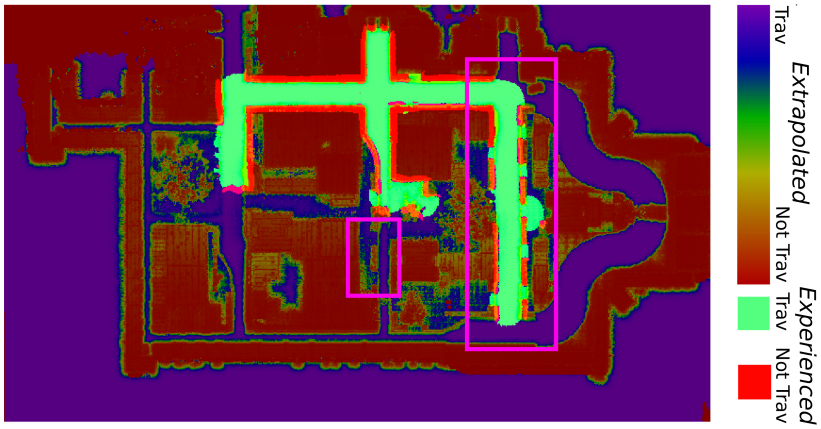

Figure 14: Visualization of the terrain analysis partway through a simulated experiment. The pink boxes highlight regions where the robots observed ramps up the curb, and areas where these ramp detections were extrapolated to regions seen only by the UAV.

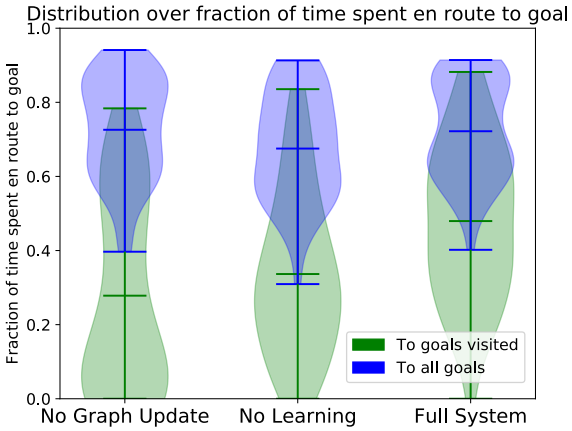

Figure 15: Distributions of the fraction of time robots spend actively navigating to a goal for different terrain learning ablations

	Distance (km)	Time (min)
UGV 1	0.88	20.6
UGV 2	0.30	6.4

Table 1: Two robots' times to a goal, where UGV 2 is given UGV 1's experience.

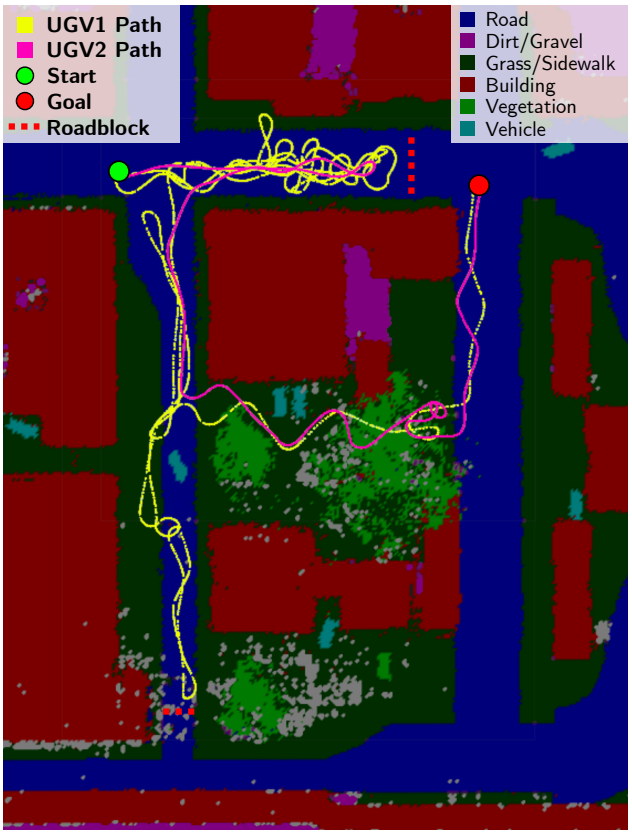

Figure 16: Trajectories of two robots navigating to the same goal. UGV 2 is given UGV 1's experience before beginning.

4.3 Communication

Along with ablating how the robot uses the terrain data, we also ablate how the robots share terrain information with each other. To demonstrate the utility of this, we first perform a simple experiment. Two robots are given the same map, and one is asked to navigate to a target position. Once it arrives there and the other robot has gotten all of the other robot's reachability scans, that other robot is given the same goal. The results of this experiment can be seen in Fig. 16. Notably, there are several roadblocks on the map, including one directly between the start and goal positions. The first robot must therefore explore several possible routes which turn out to not be traversable before finding a route. The second robot, by contrast, is able to use the first robot's experience to much more quickly find the correct route. It does not trust the first robot's data without question and does perform some exploring on its own, but overall is far more efficient by virtue of the other data. We compare the two trajectories in Table 1. Note that UGV 2 is more than 3 times faster and takes nearly 3 times less distance to reach the same goal, since it is able to leverage UGV 1's experience instead of taking the time to rediscover the same things.

In addition to this toy experiment, we also run large scale experiments, adjusting how much terrain information is shared. In one set of experiments we turn off communications about terrain completely, though

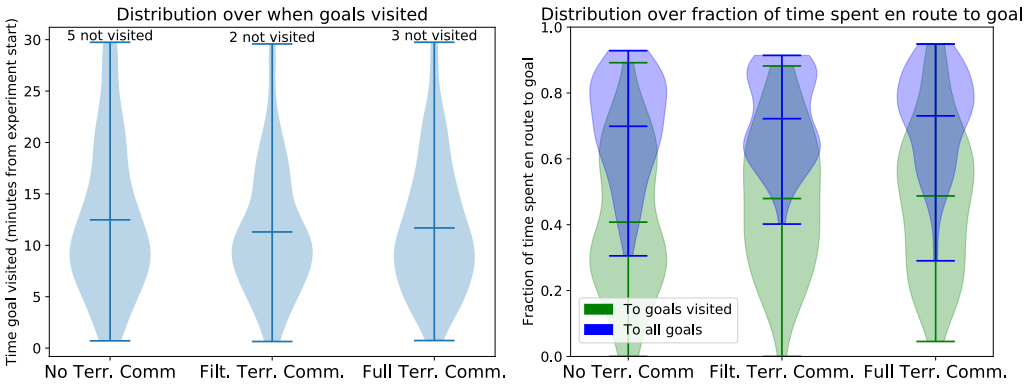

Figure 17: For varying communication, (Left): Distribution of the time goals are visited from mission start. Goals not visited are given a visit time of 30 minutes for the purposes of visualization. (Right): Distributions of the fraction of time robots spend actively navigating to a goal.

robots do still share information about their location and target goals for coordination. In another set the robots share all of their reachability scans, so in effect each robot has the same total information, assuming all robots are synchronized. Finally, we perform a set of experiments where the robots share a downsampled set of reachability scans. Each robot checks to see if a new reachability scan is at least 2 meters away from all prior scans. If it is, it is shared with other robots. This is the approach used in all other experiments, including those in the real world. These results are shown in Fig. 17. Sharing traversability information does appear to result in slightly increased team effectiveness, with robots visiting goals earlier and spending more time navigating to a goal they will ultimately visit. However, sharing the full information has minimal effect, and even slightly increases the average time goals are visited. Qualitatively, the differences are most notable at the very beginning of the experiment. After all the robots have traveled a little bit, their inferred traversability maps are fairly decent. Therefore, while sharing information helps the traversability estimation to converge to something reasonable faster, after it has done so the effect is relatively small, at least for this particular environment.

While it is possible that the small increase in average time goals are visited with full communication is due to statistical variations, we did also observe qualitatively one reason it could happen. Sometimes, a robot determines a path to be not traversable when it actually is. This is by design, since the system is tuned to be conservative, as it is better in most cases to be cautious than risk damage. However, if all information is shared and a robot makes a poor determination like this, then all other robots will more easily make that same mistake. Downsampling helps to regularize other robots' measurements, resulting in robots being more willing to explore regions others have not been able to. In summary, sharing traversability information is most valuable when robots are exploring the same region sequentially, and less helpful in simultaneous operation.

We note as well that our system fails gracefully as communication degrades. In other words, the task can be executed, even if no communication about terrain can occur at all. However, all of these experiments do assume some level of communication. Our system is dependent on the UAV being able to communicate the orthomap to the UGVs. If communication is so degraded, or the map so large, that this cannot happen, it will fail, though we could instead provide a static map *a priori*. In addition, UGVs must be able to communicate to deconflict goals. In the absence of communication, robots could simply pick goals at random, but this would result in significant redundant goal visits. We do assume a basic level of communication to be available, but anything above and beyond that, while helpful, is not strictly necessary.

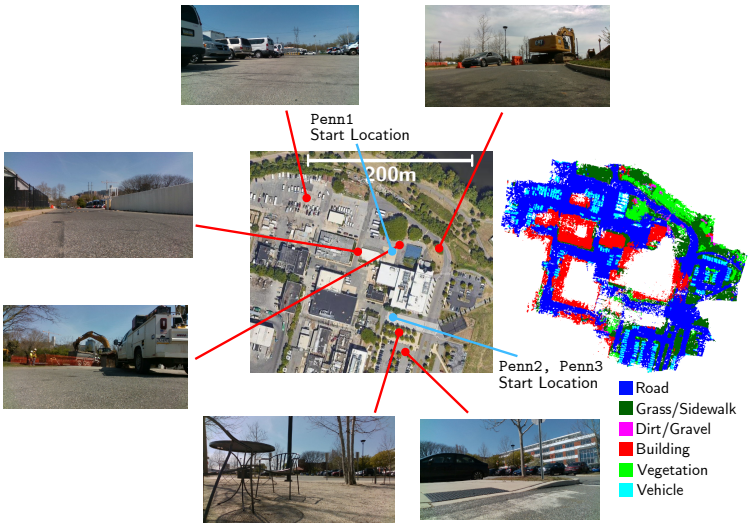

Figure 18: Satellite and semantic map of the Pennovation environment, with some robot's-eye views of some representative areas.

5 Real World Experiments

Finally, we perform experiments in the real world. We emphasize that these were not controlled environments. Cars were frequently going by, construction was ongoing with roadblocks and construction equipment, and there was a great deal of complicated structure. We test in two environments: Pennovation and the New Bolton Center (NBC). In all cases, our team consisted of 3 Jackal UGVs and a single UAV. The Jackals were equipped with an AMD Ryzen 3600 CPU, NVidia GTX 1650 GPU, Ouster OS1-64 LiDAR, and Realsense D435i camera (only used for visualization). The UAV was a custom PX4-based robot (Liu et al., 2022b), equipped with an Intel NUC i7-10710U, UBlox ZED-F9P GPS, and Open Vision Computer (OVC) (Quigley et al., 2019) global shutter RGB camera (we did not use the stereo pair or IMU). All robots used 5 GHz Rajant Breadcrumb DX2 mesh radios, with a Rajant ME4 basestation.

An overview of the Pennovation environment is shown in Fig. 18. This is a former factory refurbished to accommodate a combination of labs, offices, and storage. It offers a relatively urban setting, though there are some green spaces and trees. Particularly challenging regions include a cluttered patio with poles, tables, and chairs, as well as parking lots which contain many curbs and ramps. During our experiments there was ongoing road construction, resulting in roadblocks, construction equipment, and increased traffic due to the main road being reduced to one lane. We perform 3 experiments here, with `Penn1` starting the UAV and UGVs in a different area of the map from `Penn2` and `Penn3`, and therefore using a different exploration path for the UAV. The basestation in all cases was positioned at the start location.

An overview of the NBC environment is shown in Fig. 19. For this experiment we did not fly the UAV in real-time due to a hardware failure of the aerial robot. Instead, we fed the UGV a complete aerial map that was built earlier by ASOOM at the beginning of their mission. Robots were able to communicate with each-other and the basestation directly when in range, but not through the UAV in this case. We are therefore unable to evaluate the UAV's effect on this experiment, but nonetheless do still exercise the coordination and traversability-sharing parts of the system. This environment is somewhat larger scale and much more rural, consisting of roads between scattered buildings. There are some more significant elevation changes here as well as many more trees and much more grass compared to Pennovation.

Table 2 summarizes our 4 experiments. We ended an experiment either when all robots returned back to the start point with all identified goals visited, or after the UAV battery died and it appeared that not all goals would be visited soon. Over all experiments, we visited 49/51 goals, or 96%. Both times a goal was

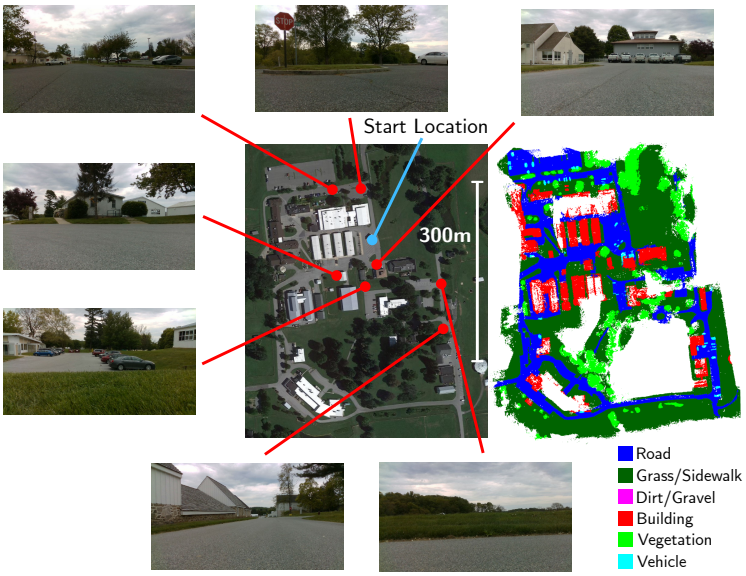

Figure 19: Satellite and semantic map of the NBC environment, with some robot's-eye views of some representative areas.

Dataset	Duration (min)	Goals Visited/Total Goals	Total Dist Travelled (km)
Penn1	28.5	10/11	3.9
Penn2	28.9	15/16	4.8
Penn3	28.0	16/16	5.0
NBC1	31.4	8/8	4.1

Table 2: Overview of the scale of 4 full-scale real-world experiments in terms of duration, goal count, and distance travelled.

not reached the goal was in a similar location, which was in a parking lot very challenging to reach due to construction on the access road to it. In `Penn1` it was particularly difficult because the UAV misclassified a portion of the road as building, causing the global planner to struggle. In `Penn2` the same goal was almost reached, but the goal timed out just before the robot got there. UGVs traveled a total of 17.8km autonomously across all the UGVs and experiments, constituting a total of 4.2 UGV-hours. Each individual UGV had on average 1.4 hours of autonomous operation and traveled 5.9km.

During our experiments, each robot was followed by a safety driver with the ability to take over manually if need be. In Table 3 we analyse the robustness of the autonomy system by looking at the fraction of time each robot was taken over manually. A large portion of these takeovers were due to dynamic obstacles, which our planner was not designed to handle. Most often these obstacles were vehicles since our experiments were conducted on active roads and parking lots. In addition, on occasion, the robots had to be prevented from colliding with each other. Finally, there were several speed bumps in the Pennovation environment, and the robots were manually driven over these to reduce mechanical wear and tear on the robots. Because dynamic obstacles and speed bumps were cases our system was not designed to handle, we refer to the fraction of time spent in manual mode, ignoring these cases, as *filtered* in Table 3.

Most of the time, robots spent only 1-3 % of the time in manual mode, corresponding to on the order of 5 takeovers per robot per experiment. These unfiltered takeovers were done to avoid collisions at the discretion of the human safety driver. The most common cause was curbs, particularly when approached from above with the curb as a negative obstacle. In this case, it was difficult for the robot to distinguish a curb from a relatively mild downslope. Other challenging obstacles included poles and other small objects like chairs. Due to the polar terrain analysis, the inflation radius of objects very close to the robot tends to be inaccurate,

Dataset	UGV1 % manual (raw/filtered)	UGV2 % manual	UGV3 % manual	Overall % manual
Penn1	32.2/23.1	8.1/3.9	15.7/11.8	17.9/11.9
Penn2	5.9/3.0	3.8/0.07	3.2/0.36	4.5/1.3
Penn3	18.8/7.6	7.7/1.0	9.7/0.81	12.0/3.0
NBC1	3.9/1.8	3.4/3.2	3.7/0	3.7/1.8

Table 3: Analysis of the fraction of time robots spent being driven manually across different real-world experiments.

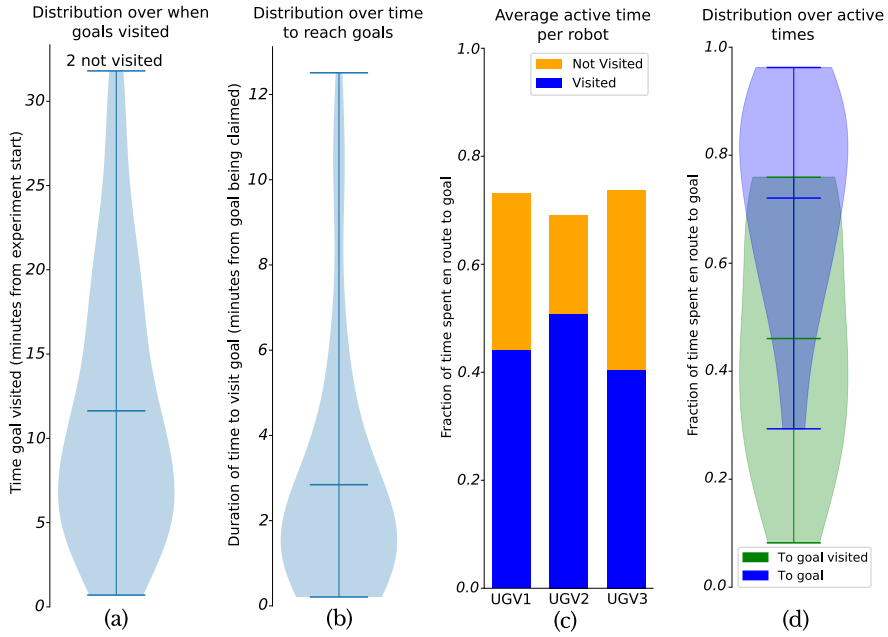

Figure 20: Analysis of goal timings and robot activity throughout the real-world experiments.

sometimes causing the local planner to drive into nearby narrow obstacles. `Penn1` had a significantly higher number of takeovers due to the experiment starting next to a parking lot with many curbs and a patio with many trees, poles, and chairs. Finally, on some rare occasions, the robots tried to drive under overhangs, such as large trucks and construction equipment. While it is possible that they would have been able to do so successfully, the operator intervened out of an abundance of caution. It is also worth noting that for `NBC1`, one UGV tried to drive over a particularly steep rut next to the road, and while it made it over, the impact caused the robot to shut down.

During all experiments, the communication system and UAV planner performed well. It was rare that UGVs did not have any active goals to visit, as can be seen in Fig. 20 (c) and (d). We can also see in Fig. 20 (a) that the majority of the goals were usually visited in the first half of the experiment, with the long tail at the end being often due to particularly challenging or distant goals to visit.

We also analyse the learned terrain model from `NBC1` over the course of the experiment in Fig. 21. We note that as time goes on and the team experiences more and more of the map, the model becomes increasingly good. Roads become more well-defined as highly traversable areas and buildings are defined as untraversable areas. Grassy regions take a value somewhere in between: sometimes drivable, sometimes not. This suggests that our modeling and analysis system is able to draw useful conclusions about the environment, even one that is much more complex than the simulation.

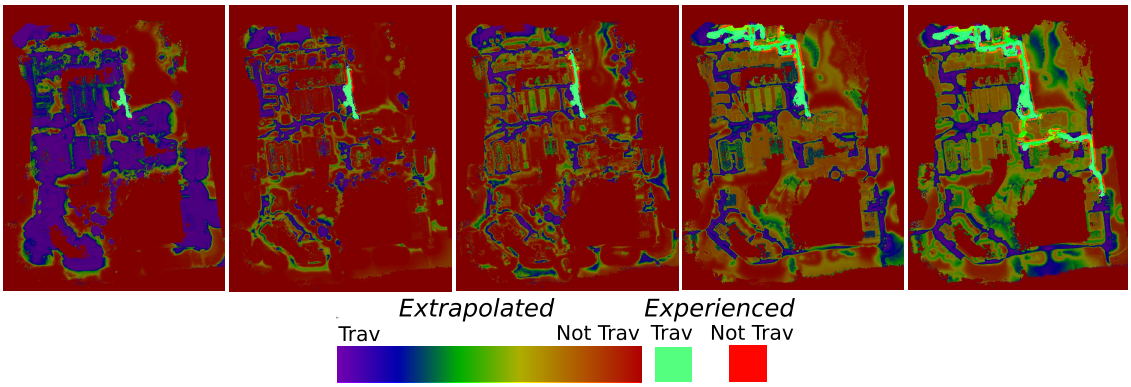

Figure 21: The learned terrain model and experienced terrain for one of the UGVs over the course of the `NBC1` experiment.

6 Conclusion

In this paper, we developed a system for air-ground collaboration, where robots are able to use information from each other to improve their own localization and planning in the absence of GPS for all robots except the UAV. Semantics enabled robots diverse in sensing and mobility to represent the world around them and communicate with one another in a common way, even with only intermittent communication. In addition, semantics proved to be a strong predictor of traversability, enabling UGVs to use the aerial map as a predictor of unvisited regions. We have also performed extensive ablation experiments in simulation, demonstrating the contribution of each component of our system. Finally, we have performed real-world experiments across two different challenging environments, urban and rural. These experiments have comprised nearly 18 kilometers driven autonomously between the 3 UGVs, demonstrating the large-scale operation and robustness of our system.

A key assumption in this work has also been that an aerial orthomap is representative of the environment. This is true for most outdoor cases but fails to hold in many environments such as those with overhangs or bridges. The single high-altitude UAV and multiple UGV architecture is not ideal for all applications. Additionally, dynamic obstacles remain a significant challenge, and there is currently no mechanism for robots to update previously mapped areas, or the UGVs to update the aerial map. These limitations provide exciting avenues for future work.

7 Acknowledgement

The authors would like to thank of Dr. Barbara Dallap Schaer at the Penn Vet New Bolton Center, as well as Dr. Ethan Stump at ARL for their help in finding places to test in the real world. We would also like to thank Dr. Priyanka Shah and Alice Kate Li for their invaluable time spent supporting our field experiments, as well as Alex Zhou and Jeremy Wang for building and repairing the many robots used in this work.

References

Borges, P., Peynot, T., Liang, S., Arain, B., Wildie, M., Minareci, M., Lichman, S., Samvedi, G., Sa, I., Hudson, N., et al. (2022). A survey on terrain traversability analysis for autonomous ground vehicles: Methods, sensors, and challenges. *Field Robotics*, 2(1):1567–1627.

Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M., and Tardós, J. D. (2021). Orb-slam3: An

accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890.

Chamseddine, A., Akhrif, O., Charland-Arcand, G., Gagnon, F., and Couillard, D. (2016). Communication relay for multiground units with unmanned aerial vehicle using only signal strength and angle of arrival. *IEEE Transactions on Control Systems Technology*, 25(1):286–293.

Chen, S. W., Nardari, G. V., Lee, E. S., Qu, C., Liu, X., Romero, R. A. F., and Kumar, V. (2020). Sloam: Semantic lidar odometry and mapping for forest inventory. *IEEE Robotics and Automation Letters*, 5(2):612–619.

Chung, T. H., Orekhov, V., and Maio, A. (2023). Into the robotic depths: Analysis and insights from the darpa subterranean challenge. *Annual Review of Control, Robotics, and Autonomous Systems*, 6:477–502.

Cladera, F., Ravichandran, Z., Miller, I. D., Hsieh, M. A., Taylor, C. J., and Kumar, V. (2023). Enabling large-scale heterogeneous collaboration with opportunistic communications. *arXiv preprint*.

Ding, Y., Xin, B., and Chen, J. (2019). Precedence-constrained path planning of messenger uav for air-ground coordination. *Control Theory and Technology*, 17(1):13–23.

Ding, Y., Xin, B., and Chen, J. (2021). A review of recent advances in coordination between unmanned aerial and ground vehicles. *Unmanned Systems*, 9(02):97–117.

Dong, L., He, Z., Song, C., and Sun, C. (2023). A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. *Journal of Systems Engineering and Electronics*, 34(2):439–459.

Fankhauser, P., Bloesch, M., Krüsi, P., Diethelm, R., Wermelinger, M., Schneider, T., Dymczyk, M., Hutter, M., and Siegwart, R. (2016). Collaborative navigation for flying and walking robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2859–2866. IEEE.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33.

Frey, J., Mattamala, M., Chebrolu, N., Cadena, C., Fallon, M., and Hutter, M. (2023). Fast Traversability Estimation for Wild Visual Navigation. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea.

Gabrlik, P., Lazna, T., Jilek, T., Sladek, P., and Zalud, L. (2021). An automated heterogeneous robotic system for radiation surveys: Design and field testing. *Journal of Field Robotics*, 38(5):657–683.

Garg, S., Sünderhauf, N., Dayoub, F., Morrison, D., Cosgun, A., Carneiro, G., Wu, Q., Chin, T.-J., Reid, I., Gould, S., et al. (2020). Semantics for robotic mapping, perception and interaction: A survey. *Foundations and Trends in Robotics*, 8(1–2):1–224.

Grocholsky, B., Keller, J., Kumar, V., and Pappas, G. (2006). Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Lecture 6d - a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning.

Ho, K., Peynot, T., and Sukkarieh, S. (2013). Traversability estimation for a planetary rover via experimental kernel learning in a gaussian process framework. In *2013 IEEE International Conference on Robotics and Automation*, pages 3475–3482. IEEE.

Hosseinpoor, S., Torresen, J., Mantelli, M., Pitto, D., Kolberg, M., Maffei, R., and Prestes, E. (2021). Traversability analysis by semantic terrain segmentation for mobile robots. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1407–1413. IEEE.

Howard, A. and Seraji, H. (2000). Real-time assessment of terrain traversability for autonomous rover navigation. In *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, volume 1, pages 58–63. IEEE.

Hudson, N., Talbot, F., Cox, M., Williams, J., Hines, T., Pitt, A., Wood, B., et al. (2022). Heterogeneous ground and air platforms, homogeneous sensing: Team csiro data61's approach to the darpa subterranean challenge. *Field Robotics*, 2(1):595–636.

Larson, J., Trivedi, M., and Bruch, M. (2011). Off-road terrain traversability analysis and hazard avoidance for ugvs. Technical report, California University San Diego Dept. of Electrical Engineering.

Lazna, T., Gabrlik, P., Jilek, T., and Zalud, L. (2018). Cooperation between an unmanned aerial vehicle and an unmanned ground vehicle in highly accurate localization of gamma radiation hotspots. *International Journal of Advanced Robotic Systems*, 15(1).

Lee, H., Kwak, K., and Jo, S. (2017). An incremental nonparametric bayesian clustering-based traversable region detection method. *Autonomous Robots*, 41:795–810.

Li, B., Zhao, S., Miao, R., and Zhang, R. (2021). A survey on unmanned aerial vehicle relaying networks. *IET Communications*, 15(10):1262–1272.

Lindqvist, B., Karlsson, S., Koval, A., Tevetzidis, I., Haluška, J., Kanellakis, C., Agha-mohammadi, A.-a., and Nikolakopoulos, G. (2022). Multimodality robotic systems: Integrated combined legged-aerial mobility for subterranean search-and-rescue. *Robotics and Autonomous Systems*, 154:104134.

Liu, C., Zhao, J., and Sun, N. (2022a). A review of collaborative air-ground robots research. *Journal of Intelligent & Robotic Systems*, 106(3):60.

Liu, X., Nardari, G. V., Ojeda, F. C., Tao, Y., Zhou, A., Donnelly, T., Qu, C., Chen, S. W., Romero, R. A., Taylor, C. J., et al. (2022b). Large-scale autonomous flight with real-time semantic slam under dense forest canopy. *IEEE Robotics and Automation Letters*, 7(2):5512–5519.

Liu, X., Prabhu, A., Cladera, F., Miller, I. D., Zhou, L., Taylor, C. J., and Kumar, V. (2023). Active metric-semantic mapping by multiple aerial robots. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3282–3288. IEEE.

Martínez, J. L., Morales, J., Sánchez, M., Morán, M., Reina, A. J., and Fernández-Lozano, J. J. (2020). Reactive navigation on natural environments by continuous classification of ground traversability. *Sensors*, 20(22):6423.

Milioto, A., Vizzo, I., Behley, J., and Stachniss, C. (2019). Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 4213–4220. IEEE.

Miller, I. D., Cladera, F., Cowley, A., Shivakumar, S. S., Lee, E. S., Jarin-Lipschitz, L., Bhat, A., Rodrigues, N., Zhou, A., Cohen, A., et al. (2020). Mine tunnel exploration using multiple quadrupedal robots. *IEEE Robotics and Automation Letters*, 5(2):2840–2847.

Miller, I. D., Cladera, F., Smith, T., Taylor, C. J., and Kumar, V. (2022). Stronger together: Air-ground robotic collaboration using semantics. *IEEE Robotics and Automation Letters*, 7(4):9643–9650.

Miller, I. D. et al. (2021). Any way you look at it: Semantic crossview localization and mapping with lidar. *IEEE Robotics and Automation Letters*, 6(2):2397–2404.

Morrell, B., Thakker, R., Santamaria Navarro, À., Bouman, A., Lei, X., Edlund, J., Pailevanian, T., Vaquero, T. S., Chang, Y. L., Touma, T., et al. (2022). Nebula: Team costar's robotic autonomy solution that won phase ii of darpa subterranean challenge. *Field robotics*, 2:1432–1506.

Mox, D., Kumar, V., and Ribeiro, A. (2022). Learning connectivity-maximizing network configurations. *IEEE Robotics and Automation Letters*, 7(2):5552–5559.

Neuhaus, F., Dillenberger, D., Pellenz, J., and Paulus, D. (2009). Terrain drivability analysis in 3d laser range data for autonomous robot navigation in unstructured environments. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–4. IEEE.

Papadakis, P. (2013). Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385.

Park, B., Choi, J., and Chung, W. K. (2012). An efficient mobile robot path planning using hierarchical roadmap representation in indoor environment. In *2012 IEEE International Conference on Robotics and Automation*, pages 180–186. IEEE.

Peterson, J., Chaudhry, H., Abdelatty, K., Bird, J., and Kochersberger, K. (2018). Online aerial terrain mapping for ground robot navigation. *Sensors*, 18(2):630.

Prágr, M., Bayer, J., and Faigl, J. (2023). Autonomous exploration with online learning of traversable yet visually rigid obstacles. *Autonomous Robots*, 47(2):161–180.

Prágr, M., Váňa, P., and Faigl, J. (2019). Aerial reconnaissance and ground robot terrain learning in traversal cost assessment. In *International Conference on Modelling and Simulation for Autonomous Systems*, pages 3–10. Springer.

Qu, C., Shivakumar, S. S., Liu, W., and Taylor, C. J. (2022). Llol: Low-latency odometry for spinning lidars. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4149–4155.

Quigley, M., Mohta, K., Shivakumar, S. S., Watterson, M., Mulgaonkar, Y., Arguedas, M., Sun, K., Liu, S., Pfrommer, B., Kumar, V., et al. (2019). The open vision computer: An integrated sensing and compute system for mobile robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1834–1840. IEEE.

Reddy, S. K. and Pal, P. K. (2016). Computing an unevenness field from 3d laser range data to obtain traversable region around a mobile robot. *Robotics and Autonomous Systems*, 84:48–63.

Ribeiro, A. and Conesa-Muñoz, J. (2021). Multi-robot systems for precision agriculture. *Innovation in Agricultural Robotics for Precision Agriculture: A Roadmap for Integrating Robots in Precision Agriculture*, pages 151–175.

Rizk, Y., Awad, M., and Tunstel, E. W. (2019). Cooperative heterogeneous multi-robot systems: A survey. *ACM Computing Surveys (CSUR)*, 52(2):1–31.

Romera, E., Alvarez, J. M., Bergasa, L. M., and Arroyo, R. (2017). Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272.

Scherer, S., Agrawal, V., Best, G., Cao, C., Cujic, K., Darnley, R., DeBortoli, R., Dexheimer, E., Drozd, B., Garg, R., et al. (2021). Resilient and modular subterranean exploration with a team of roving and flying robots. *Submitted to the Journal of Field Robotics*, 2(3):6.

Schilling, F., Chen, X., Folkesson, J., and Jensfelt, P. (2017). Geometric and visual terrain classification for autonomous mobile navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2684. IEEE.

Sevastopoulos, C. and Konstantopoulos, S. (2022). A survey of traversability estimation for mobile robots. *IEEE Access*, 10:96331–96347.

Shaban, A., Meng, X., Lee, J., Boots, B., and Fox, D. (2022). Semantic terrain classification for off-road autonomous driving. In *Conference on Robot Learning*, pages 619–629. PMLR.

Shah, D. and Levine, S. (2022). ViKiNG: Vision-Based Kilometer-Scale Navigation with Geographic Hints. In *Proceedings of Robotics: Science and Systems*.

Stavens, D. and Thrun, S. (2006). A self-supervised terrain roughness estimator for off-road autonomous driving. In *Proceedings of the Conference on Uncertainty in AI (UAI)*. AUAI Press.

Su, Z., Wang, C., Wu, X., Dong, Y., Ni, J., and He, B. (2021). A framework of cooperative uav-ugv system for target tracking. In *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 1260–1265. IEEE.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692.

Tranzatto, M., Miki, T., Dharmadhikari, M., Bernreiter, L., Kulkarni, M., Mascarich, F., Andersson, O., Khattak, S., Hutter, M., Siegwart, R., et al. (2022). Cerberus in the darpa subterranean challenge. *Science Robotics*, 7(66):eabp9742.

Vandapel, N., Donamukkala, R. R., and Hebert, M. (2006). Unmanned ground vehicle navigation using aerial ladar data. *The International Journal of Robotics Research*, 25(1):31–51.

Wellhausen, L., Dosovitskiy, A., Ranftl, R., Walas, K., Cadena, C., and Hutter, M. (2019). Where should i walk? predicting terrain properties from images via self-supervised learning. *IEEE Robotics and Automation Letters*, 4(2):1509–1516.

Wu, G., Gao, X., and Wan, K. (2020). Mobility control of unmanned aerial vehicle as communication relay to optimize ground-to-air uplinks. *Sensors*, 20(8):2332.

Yue, Y. et al. (2020). Collaborative semantic understanding and mapping framework for autonomous systems. *IEEE/ASME Transactions on Mechatronics*, 26(2):978–989.

Zhou, L., Wang, J., Lin, S., and Chen, Z. (2022). Terrain traversability mapping based on lidar and camera fusion. In *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*, pages 217–222. IEEE.