SPINE: Online Semantic Planning for Missions with Incomplete Natural Language Specifications in Unstructured Environments

Zachary Ravichandran, Varun Murali, Mariliza Tzes, George J. Pappas, and Vijay Kumar

***Abstract*— As robots become increasingly capable, users will want to describe high-level missions and have robots fill in the gaps. In many realistic settings, pre-built maps are difficult to obtain, so execution requires exploration and mapping that are necessary and specific to the mission. Consider an emergency response scenario where a user commands a robot, "triage impacted regions." The robot must infer relevant semantics (victims, etc.) and exploration targets (damaged regions) based on priors or other context, then explore and refine its plan online. These missions are *incompletely specified*, meaning they imply subtasks and semantics. While many semantic planning methods operate online, they are typically designed for well specified tasks such as object search or exploration. Recently, Large Language Models (LLMs) have demonstrated powerful contextual reasoning over a range of robotic tasks described in natural language. However, existing LLM planners typically do not consider online planning or complex missions; rather, relevant subtasks are provided by a pre-built map or a user. We address these limitations via SPINE (online Semantic Planner for missions with Incomplete Natural language specifications in unstructured Environments). SPINE uses an LLM to reason about subtasks implied by the mission then realizes these subtasks in a receding horizon framework. Tasks are automatically validated for safety and refined online with new observations. We evaluate SPINE in simulation and real-world settings. Evaluation missions require multiple steps of semantic reasoning and exploration in cluttered outdoor environments of over 20,000m^2 area. We evaluate SPINE against competitive baselines in single-agent and air-ground teaming applications. Please find videos and software on our project page: https://zacravichandran.github.io/SPINE**

I. Introduction

Consider an inspection robot operating after a heavy storm. A user may provide the following mission: "Communications are down. Why?" The robot will have to explore missing or changed regions of the map, locate relevant semantic entities (*i.e.,* communication infrastructure), and collect precise mission-relevant information to assess infrastructure damage. We refer to these mission specifications as *incomplete*. They imply sugboals and semantic targets that are not directly given to the robot; rather, they must be inferred from context. Furthermore, in many real-world scenarios, environments are dynamic and data is hard to collect, so the robot must actively map its environment and plan online.

Semantic planning methods have made progress on tasks such as object search, inspection, exploration, and mobile manipulation [1]–[9]. These methods typically maintain a

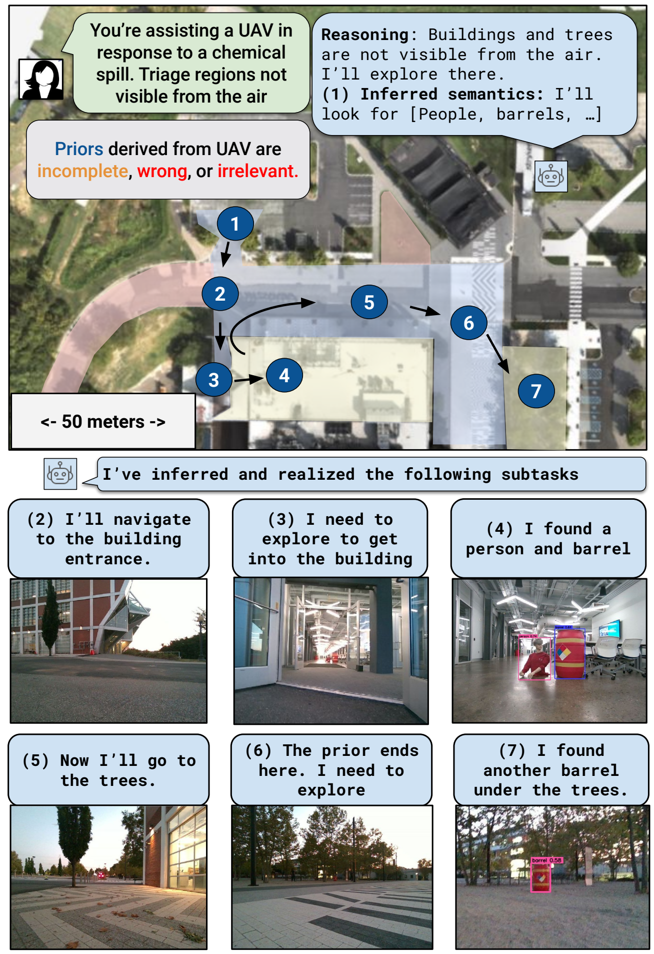

Fig. 1. Representative mission: User provides incomplete specification of a mission to planner, and the planner receives an incomplete prior map where many regions are either missing or incorrect. Planner infers relevant semantics and subtasks, which require mapping and exploration. The planner reacts to information acquired online and completes the mission.

semantic map of the environment such as a metric-semantic grid, object-oriented map, or scene graph, which the planner reasons over in search of its goal [4], [7], [10]. With advances in semantic mapping, these representations can be built online, which enables downstream planning [11]–[14]. Semantic planning methods have been designed for multi-robot systems [4], [15]–[17], and some approaches are robust enough to be fielded in large scale environments [16], [18], [19]. However, these methods are designed for well specified tasks which must be explicitly given to the robot (*e.g.*,"inspect all the antennas in Zone A"); they cannot reason over more general requirements. Formal and structured

All authors are with are with the GRASP Laboratory, University of Pennsylvania. Corresponding author: zacravi@seas.upenn.edu.
We gratefully acknowledge support from ARL DCIST CRA W911NF-17-2-0181, NSF Grant CCR-2112665, and the NSF Graduate Research Fellowship Program

planning languages provide mission-level specifications for one or multiple robots [20]–[22]. However, these methods require a human operator to explicitly compose subtasks.

Recent work has addressed these limitations by using Large Language Models (LLMs), which have demonstrated powerful contextual reasoning over many domains, to plan over tasks described in natural language [23]–[26]. Research has applied LLM-enabled planners to problems including mobile manipulation, navigation, and fault detection [25], [27]–[32]. However, LLM-enabled planners typically require a pre-built map, which is unavailable in many environments [27], [28], [31], [33], [34]. Furthermore, current LLM-planning methods generally consider well-specified tasks or missions [25], [31], [35]–[37]. These assumptions break in large-scale environments such as outdoor settings.

To address these limitations, we present SPINE (online Semantic Planner for missions with Incomplete Natural language specifications in unstructed Environments). The planner can operate in partially-known, unstructured environments, and it leverages priors from a variety of data soruces. Given incomplete mission specifications in natural language, the planner uses an LLM to infer subtasks consisting of navigation, active sensing, and user interaction behaviors. LLM-generated plans are validated using a module that checks for semantic and spatial correctness, which prevents potentially unsafe tasks from being sent to the downstream controller. Validated subtasks are realized in a receding horizon manner and are refined online. In summary, the contributions of the paper are:

- An online semantic planning method for language-specified missions in partially-known environments.
- A process to infer subtasks from incomplete mission specifications and refine the subtasks online.
- A verification module that enables an LLM to safely propose navigation and exploration goals in unstructured and partially-known environments.

We validate contributions via experiments in large-scale outdoor environments in simulation and the real-world. We compare against competitive baselines and apply our method to missions involving semantic route inspection, multi-object search, and air-ground teaming.

II. Related Work

Representations for Semantic Planning. Effective semantic representations capture traversability, semantics, and spatial relationships needed for reasoning over contextual goals. Advances in semantic mapping have enabled online planning tasks such as active exploration or air-ground teaming for object search [1], [4], [16], [19], [38].

Scene graphs are a popular representation for semantic planning, as they concisely represent objects, topology, and traversable regions. Hydra provides a real-time scene graph engine [11] designed for indoor environments. Strader *et al*. [14] relax this assumption. Topological maps are similar, but do not include a hierarchy [39], [40]. Recent work incorporates foundation models into mapping pipelines in order to create open-vocabulary representations. Mappers including

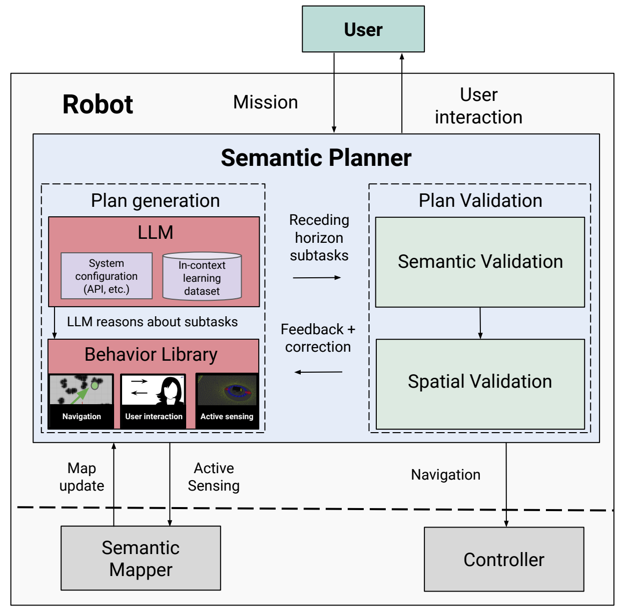

Fig. 2. System overview. A user provides a mission to the planner. The planner uses an LLM to decompose the task into subgoals using the provided action set consisting of mobility, and active sensing, and user interaction behaviors. The LLM generates a receding horizon plan, and the validation module checks for semantic and spatial correctness; if necessary, feedback and correction are provided in real-time. Actions are sent to the appropriate module, and the planner refines its plan as new information is aquired.

ConceptGraphs [41], HOVSG [42], and Clio [43] assign semantic feature vectors to entities in the map. Semantic labels are then produced at runtime, depending on the task. Our architecture is compatible with state of the art mapping methods. In particular, we use an open-vocabulary semantic-topological mapper extended from SlideSlam [12], which allows the planner to configure task-specific semantics at runtime and operate in unstructured outdoor environments.

Online Semantic Planning. Semantic planners reason over objects, regions, or other contextual information to address problems such as object search and inspection, and semantic exploration [1], [3], [5], [6], [44]. Many works address online planning in partially-known environments. In these settings, prior scene information may be obtained from previous mission data [3], and the planner augments priors with information acquired online [3], [7]. Beyond object-level reasoning, semantic information also accelerates exploration of partially-known or unknown environments [1], [45], [46]. Fusing semantic knowledge from foundation models with classical search methods such as frontier exploration has been shown to an expecially effective exploration strategy [8], [45]. Structured or formal planning languages, such as Linear Temporal Logic (LTL), may be used to compose more complex missions [10], [21], [22], [47], [48]. Notably, these methods require detailed mission specifications from a user, whereas our method infers specifications.

LLMs for Planning. Language has emerged as a powerful representation for specifying tasks, and LLM-enabled planners have been applied to domains including mobile manipulation [27], [28], [49], service robotics [50], autonomous driving [51], navigation [31], [52]–[54], and fault detection [29], [30] These methods typically configure the LLM via in-context prompts, which channel the LLM's common sense into the given problem domain without fine-tuning [32]. The LLM is then given a set of action primitives such as graph navigation goals [51], predicates in a formal planning language [35], lower-level APIs for code generation [25], [55], [56], or learned behaviors [27]. At runtime, the LLM is given an environment map, such as a graph [28] or semantic regions [32].

While these works typically consider well-specified tasks, a line of research develops LLM-enabled planners that translate missions specifications to a formal language such as Linear Temporal Logic (LTL) or Planning Domain Definition Language (PDDL) [33]–[35], [37], [57]–[59].

While these instructions are complex, they explicitly state subtasks and semantic referents [36]. While works such as SayPlan consider less well-specified tasks ("find me something to drink"), they still assume pre-mapped or highly-structured indoor scenes [28], [49]. Other research relaxes the requirement of a pre-built semantic map by incorporating feedback from perception systems [29], [32], [49] or specifying semantics at runtime [26], [36]. However, perception is limited to object detection or designed for small room-centric environments where the planner can leverage clear hierarchy and natural bounds on the environment.

In summary, prior work generally assumes a pre-built map, a strict room-oriented environment structure, or studies explicit tasks such as object search or goal navigation. These assumptions translate poorly to many problem domains (*e.g.*, outdoors) and missions where a user only provides a high-level description of requirements.

III. Online Semantic Planning Approach

A. Problem Statement

We consider a scenario where a robot operates in a partially-known, unstructured environment and is assigned a mission with incomplete natural language specifications. The semantic planner has access to a set of behaviors for navigation, active sensing, and user interaction. At runtime, the planner receives an incomplete prior map and utilizes a semantic mapper to obtain real-time semantic updates of the environment. Given the incomplete mission specification, a fixed planning horizon, and online semantic feedback, the planner must infer and generate appropriate subtasks within a receding horizon framework to fulfill the mission.

B. System Overview

We outline SPINE's architecture in Fig. 2. The planner consists of an LLM, a behavior library, and a validation module. The planner receives a mission from the user and a prior from the Semantic Mapper (Sec. III-C). The planner composes a subtask sequence in a receding horizon manner (Sec. III-D), which is validated for syntactic and semantic correctness (Sec III-E). Verified subtasks are sent to their relevant module (mobility, active sensing, or user interaction, Sec. III-D). At each iteration, the Semantic Mapper provides updates which are stored in-context, and the planner refines its subtask sequence online (Sec. III-D).

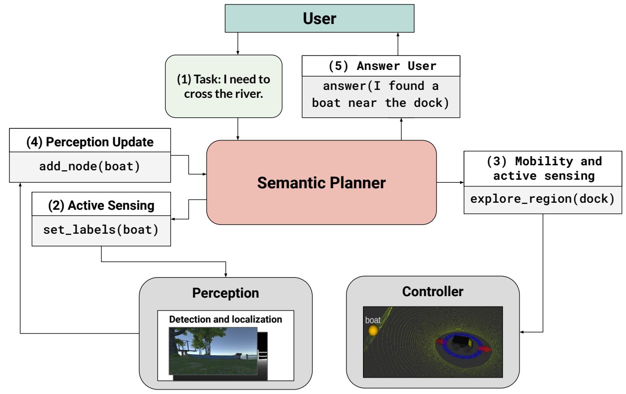

Fig. 3. Example API use during mission. (1) user provides an incomplete task. (2) Planner infers relevant semantic categories and configures perception. (3) Planner then reasons about exploration targets. (4) Perception finds a boat and notifies planner. (5) Planner informs user.

C. Semantic Mapper

Our architecture assumes a topological graph-based semantic mapper. Each node is either a *region* or *object*. As is standard in the semantic mapping literature, regions are traversable points in freespace [11], [42]. An edge between two regions indicate that there is an obstacle-free path. Object nodes represent localized objects in freespace. An edge between an object and region node indicate that the object can be observed from that region. Regions and objects may be enriched with additional semantic information (*e.g.*, this region is in a busy parking lot), which provides additional cues for planning. The mapper also maintains a local occupancy map, which the planner uses for Spatial Validation (Sec III-E). The mapper is initialized with priors from satellite imagery, UAV maps, or previous mission data. At each planning update it will provide updates to the planner via the API described in Sec. III-D).

D. LLM Planner with Behavior Library

We configure the LLM via a system prompt with three primary components: role description, mapping interface, and behavior library. Role description is described in the Problem Statement (Sec III-A), and the following componenets are described below.

Mapping interface: At each planning iteration, all map updates are provided to the LLM in-context via the following API which captures high-level graph manipulations: `add_nodes`, `remove_nodes`, `add_edges`, `remove_edges`, `update_nodes`. The nodes are defined as a dictionary of attributes, which allows for providing nodes with rich semantic descriptions (example in Fig. 6).

Behavior Library: The planner has access to atomic behaviors for mobility, active sensing, and user interaction (see Table I). At each planning iteration, the LLM generates a

Purpose	Function	Arguments	Behavior	Constraints
Navigation	`map_region`	region node	navigate to goal and find objects	syntax, reachable
	`exlore_region`	goal region, exploration radius	explore around goal	syntax, reachable, explorable
	`extend_map`	2D coordinate	add frontier at coordinate	syntax, explorable
	`goto`	region	navigate to region	syntax, reachable
Active Sensing	`inspect`	object and query	Inspect object	syntax, reachable
	`set_labels`	list of labels	Configure object detection	syntax
User interaction	`clarify`	question	ask for clarification from user	syntax
	`answer`	provides answer	denotes task is complete	syntax

TABLE I

AVAILABLE BEHAVIORS USED BY THE SEMANTIC PLANNER TO COMPOSE ACTION SEQUENCES.

sequence of behaviors and corresponding arguments. This sequence is then provided to the validation module. The planner uses the `answer` behavior to terminate a mission and notify the user of results. The `clarify` behavior is used gain further instructions, if needed.

Online in-context updates: The LLM uses chain-of-thought reasoning to compose an action sequence from available behaviors [60]. All inputs to the planner and action history are maintained in-context via the provided APIs. See Fig. 3 for an example action-control-perception loop.

E. Semantic and Spatial Validation

To create subtasks, the planner must correctly invoke its behavior library while reasoning over constraints such as traversability (see Table I). LLMs are prone to hallucinate this information, thus we filter LLM-generated plans through a validation module, which is outlined in Algorithm 1.

The validation module first ensures that the task sequence is composed of behaviors which are invoked with semantically correct arguments. Validation also checks reachability; meaning, for a given goal there must exists a path to that goal in the current map. If a given task is invalid, the validator forms state-specific feedback to the LLM. For example, because the `goto` behavior requires reachability, if the LLM tries to call `goto` on an unreachable node, `node x`, the verifier will provide the feedback "node x is unreachable from your current location. Consider exploring, and update your plan accordingly."

Given a semantically valid plan, tasks with exploration constraints are spatially validated to prevent hazardous or unreachable goals from being sent to the controller. Spatial validation uses frontier-style exploration to iteratively search for a traversable path towards a given goal. The algorithm terminates after reaching the goal or encountering an obstacle. For each breaking condition, semantic feedback is provided to the planner such as "exploration terminated after encountering an obstacle."

Algorithm 1: VALIDATION MODULE

```
Input: TaskSequence, Map
Initialize: ValidSequence = [], LLMFeedback= []
for Task in TaskSequence do
    if SemanticallyValid(Task, Map) then
        ValidSequence ← ValidSequence + [Task]
    else
        LLMFeedback ← GetFeedback(Task, Map)
        return ValidSequence, LLMFeedback
for Task in TaskSequence do
    if Exploring not in constraints(Task) then
        continue
    Task ← SpatiallyValidate(Task, M)
    ValidSequence ← ValidSequence + [Task]
    LLMFeedback ← GetFeedback(Task, M)
return ValidSequence, LLMFeedback
```

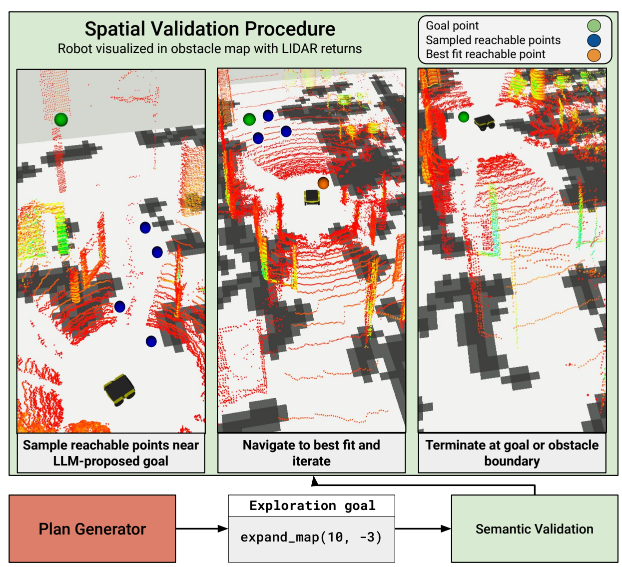

Fig. 4. Example spatial validation on real system. LLM generates exploration goal outside robot's obstacle map, which may not be reachable. Spatial validation iteratively finds best reachable fit, and the robot navigates to that point. Procedure terminates once robot reaches goal

IV. EXPERIMENTS

We design experiments to assess our three contributions (Sec I):

- **Q1**: How much time and distance savings does SPINE provide compared to mapping then planning approaches?
- **Q2**: Can SPINE, which infers semantics and subtasks online, achieve tasks competitively compared to methods that are explicitly given those via a prior map and

full mission specifications?

- **Q3**: How important is validation for online planning?

We use simulation and real robot experiments to answer **Q1** and **Q2**, and we design an ablation study to answer **Q3**.

A. Implementation Details

Both simulated and real robot experiments assume a mobile robot, a Lidar and RGB-D camera. Our planner uses GPT-4 [61]. Our graph-based semantic mapper enriches the object-oriented map from SlideSLAM [12] with a traversability graph as described in Sec. III-C. The mapper uses GroundGrid [62] for traversability estimation; these points are used to establish region nodes. The mapper performs open-vocabulary object detection with GroundingDino [63], and the detected objects are grounded using a multiple-hypothesis tracker. The mapper uses the LLaVA vision-language model for object inspection and region description [64]. Faster-LIO provides odometry [65]. We use ROS MoveBase for control. Simulation experiments employ a photorealistic Unity testbed, which provides realistic sensor and control feeds on a ClearPath Husky. We then perform real robot experiments on Clearpath Jackal equipped with a Ouster Lidar, Realsense RGB-D Camera, Nvidia RTX 4000 GPU, and Ryzen 5 3600 CPU.

B. Baselines and Metrics

We compare against two baselines. First, we compare against **Explicit Tasking**, where the planner is given a step-by-step instruction by the user. The planner does not have to infer subtasks or exploration objectives. While this method does not use a formal planning language (eg Linar Temporal Logic), we note the similarity to existing formal planning methods where the user provides explicit mission instructions [35], [57]. We then consider the **Mapping then LLM-as-Planner** (henceforth referred to as Two Stage) approach, which is a common LLM planning paradigm [25], [27], [28], [36]. In this baseline, a map is built, then the planner is given the mission. Following previous work, all spatial regions are provided in the map given to the planner, but the planner can still discover new objects in the scene. [27], [32].

For SPINE and Mapping then LLM-as-Planner approach, the operator provides an initial mission. If the planner stops prematurely, the operator may intervene to provide subsequent instructions, which we report as *interactions*. However, a successful trial requires the planner to complete the mission without user intervention. We also report distance traveled, time elapsed, and LLM API calls (queries) required to complete a mission. When explicitly tasking the robot, more complex tasks will have more user iterations. Ideally, our method will infer these tasks with minimal interactions. We normalize all results against Explicit Tasking, as that provides nominal mission performance, and we refer to our method as Online.

C. Missions with Incomplete Specifications

We consider the missions 1) *there was a storm last night. I am worried that impacted logistics, because I need to drop off supplies today. Can I still do that?*, 2) *I sent a robot out to collect supplies from an incoming boat. I have not heard back. What happened?*, 3) *communications are down, why?* 4) *I need to gather supplies from my boat. Has recent construction impacted that*, and 5) *You're assisting a UAV in response to a chemical spill. Triage regions not visible from the air.* Each mission requires completing 2-8 subtasks of semantic reasoning and exploration. Our method is given an incomplete prior and must infer subtasks and semantics. The Explicit Tasking method receives subtasks directly. The Mapping then LLM-as-Planner receives a map with relevant semantics, but must identify and realize subtasks. We run each mission one to three times and vary the prior map and initial conditions. See Fig. 1, Fig. 5, and Fig. 6 for example priors and tasks.

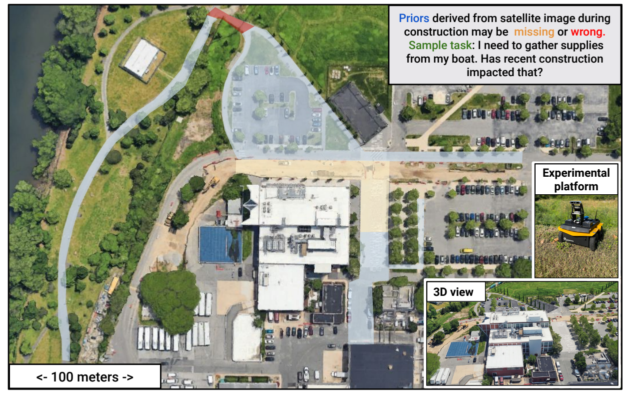

Fig. 5. Experimental platform, 3D view of environment, and example prior and corresponding task used for real-world experiments. The prior map is derived from outdated satellite imagery or obstructed due to trees and other coverings. The prior map is thus incomplete and partially incorrect, which requires the planner to reason about information acquired online.

D. Simulation Results

We run simulation experiments in outdoor environments of over 40,000m^2, where missions require the robot to travel up to 400m. Results are provided in Table II. Averaged over all scenarios, Explicit Tasking takes 532s, travels 292m, 8.6 API calls, and 4 user interactions to complete a mission. Despite only receiving partial knowledge of the mission and environment, our method performs competitively to the Explicit Tasking approach. Compared to Explicit Tasking, our method requires less user interaction but makes a similar amount of LLM queries, which implies that it reasons about the subgoals required for a mission online (for **Q2**). Imperfect success rate comes from the third mission, where the planner must inspect multiple communication towers for damage. After finding that the first tower is damaged, instead of inspecting the next tower, it declares the task complete. Because SPINE performs online mapping, it provides rich mission-specific detail. For example, when assessing a communication outage, the planner finds that the radio towers are rusted and in a state of disrepair (Fig. 6). While the Two Stage approach is also competitive in success, this method requires over twice the distance and time required to complete missions (for **Q1**). Note that because the Two Stage approach receives the full map, it is able to formulate a plan upfront in only a few LLM queries.

TABLE II
SIMULATION RESULTS NORMALIZED BY EXPLICIT TASKING.

Method	Metrics				
	Success	Time	Distance	Interactions	Queries
SPINE	94.3%	102.6%	107.0%	33.3%	77.3%
Two Stage	100%	238.0%	232.0%	30.6%	17.0%
Explicit Tasking	100%	100%	100%	100%	100%

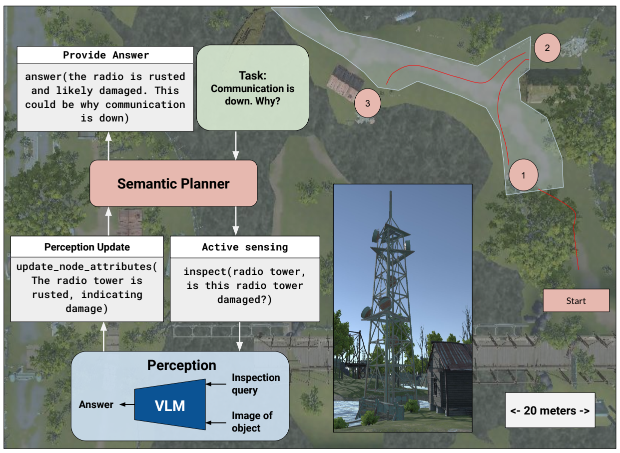

Fig. 6. The robot is given a mission (upper left), and a prior map (shaded blue). The robot must (1) explore until it gets inside the prior map then (2, 3) visit and inspect communication infrastructure. The planner then forms an appropriate inspection query for the mapper's vision language model (VLM), and it uses the acquired information to solve the task.

E. Real Robot Results

We evaluate SPINE on a ClearPath Jackal in a semi-urban office park. The results are shown in Table III. Averaged over all scenarios, Explicit Tasking takes 1035s, travels 202m, 8.6 API calls, and requires 5 user interactions to complete a mission. In the real world, the planner must adapt to more complex perception input, greater actuation noise, and avoid more obstacles as compared to simulation, thus it travels slower on average. Results show that SPINE still compares favorably to Explicit Tasking (for **Q2**). Interestingly, the real robot success rate was higher than in simulation, which is likely due to the increased scale of the simulated environment. Due to the environment scale and real constraints of the robot, there is a comparatively larger gap between the Mapping then LLM-as-Planner approach and our method (for **Q1**). See Fig. 1 for an example mission.

F. Importance of Validation for Online Planning

In order to measure the importance of validation for online mapping, we compare SPINE to an ablated version without the validation module (for **Q3**). We provide identical missions to each method, and we measure mission success rate as we randomly remove portions of the prior map. Results, shown in Fig. 7, indicate that verification is increasingly important as the environment becomes less certain.

TABLE III
REAL-WORLD EXPERIMENTS NORMALIZED BY EXPLICIT TASKING

Method	Metrics				
	Success	Time	Distance	Interactions	Queries
SPINE	100%	108.8%	110.9 %	26.75%	96.4%
Two Stage	100%	357.6%	282.6%	25%	16.6%
Explicit Tasking	100%	100%	100%	100%	100%

Qualitatively, the LLM is prone to hallucinate connections and exploration goals. Validation prevents hallucinated goals from being realized on the robot and offers an alternative plan instead (See Fig. 4).

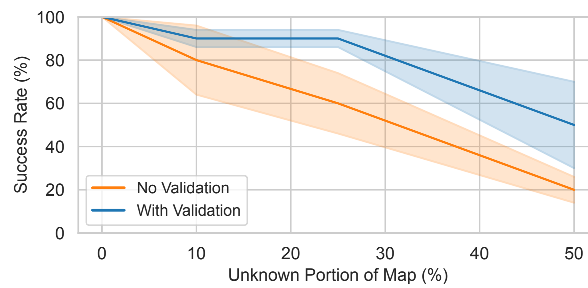

Fig. 7. Validation boosts mission success rate as prior knowledge decreases.

V. CONCLUSION

We present SPINE, a method for online semantic planning in partially-known, unstructured environments. We consider missions with incomplete specifications given in natural language. Our planner uses an LLM to decompose these specifications into a sequence of navigation, active sensing, and user interaction subtasks in a receding horizon framework. These subtasks are automatically validated and refined online.

Simulation and real-world experiments demonstrate that the performance of our method is comparable to *explicit tasking*, where an expert user goes through a tedious process of providing detailed subgoals to the planner. Our method is also significantly more efficient in terms of distance and time required to complete a mission as compared to the two step process of first mapping, and then using a LLM-as-Planner approach, without requiring full *a priori* knowledge of the environment.

Future work may take several directions. While the planner accomplishes missions with incomplete specifications, we find that the method may pick regions that are suboptimal given the complete environment knowledge. For example, if a task requires visiting multiple regions, the LLM may choose to visit the farther regions first, leading to inefficient behavior. This limitation of the LLM was one of driving factors in the validation module design, and we would like to design better techniques. Going forward, we would also like to explore the use of open-sourced LLMs such as Llama [66] or Gemma [67]. Finally, we are interested in extending our method to distributed multi-robot planning missions.

VI. Acknowledgements

We thank Fernando Cladera for feedback on the manuscript and support for the experiments.

References

[1] Y. Tao, X. Liu, I. Spasojevic, S. Agarwal, and V. Kumar, "3d active metric-semantic slam," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2989–2996, 2024.

[2] S.-K. Kim, A. Bouman, G. Salhotra, D. D. Fan, K. Otsu, J. W. Burdick, and A. akbar Agha-mohammadi, "Plgrim: Hierarchical value learning for large-scale exploration in unknown environments," *ArXiv*, vol. abs/2102.05633, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231861864

[3] M. F. Ginting, S.-K. Kim, D. D. Fan, M. Palieri, M. J. Kochenderfer, and A. akbar Agha-Mohammadi, "SEEK: Semantic reasoning for object goal navigation in real world inspection tasks," 2024. [Online]. Available: https://arxiv.org/abs/2405.09822

[4] I. D. Miller, F. Cladera, T. Smith, C. J. Taylor, and V. Kumar, "Stronger together: Air-ground robotic collaboration using semantics," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9643–9650, 2022.

[5] X. Liu, G. V. Nardari, F. Cladera, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. F. Romero, C. J. Taylor, and V. Kumar, "Large-scale autonomous flight with real-time semantic slam under dense forest canopy," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5512–5519, 2022.

[6] M. F. Ginting, S.-K. Kim, O. Peltzer, J. Ott, S. Jung, M. J. Kochenderfer, and A. akbar Agha-mohammadi, "Safe and efficient navigation in extreme environments using semantic belief graphs," 2023. [Online]. Available: https://arxiv.org/pdf/2304.00645.pdf

[7] V. Vasilopoulos, G. Pavlakos, S. L. Bowman, J. D. Caporale, K. Daniilidis, G. J. Pappas, and D. E. Koditschek, "Reactive Semantic Planning in Unexplored Semantic Environments Using Deep Perceptual Feedback," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4455–4462, July 2020.

[8] N. Yokoyama, S. Ha, D. Batra, J. Wang, and B. Bucher, "Vlfm: Vision-language frontier maps for zero-shot semantic navigation," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[9] M. Tzes, V. Vasilopoulos, Y. Kantaros, and G. J. Pappas, "Reactive informative planning for mobile manipulation tasks under sensing and environmental uncertainty," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7320–7326.

[10] A. Ray, C. Bradley, L. Carlone, and N. Roy, "Task and motion planning in hierarchical 3d scene graphs," 2024. [Online]. Available: https://arxiv.org/abs/2403.08094

[11] N. Hughes, Y. Chang, S. Hu, R. Talak, R. Abdulhai, J. Strader, and L. Carlone, "Foundations of spatial perception for robotics: Hierarchical representations and real-time systems," *The International Journal of Robotics Research*, 2024. [Online]. Available: https://doi.org/10.1177/02783649241229725

[12] X. Liu, J. Lei, A. Prabhu, Y. Tao, I. Spasojevic, P. Chaudhari, N. Atanasov, and V. Kumar, "Slideslam: Sparse, lightweight, decentralized metric-semantic slam for multi-robot navigation," *arXiv preprint arXiv:2406.17249*, 2024.

[13] F. Furrer, T. Novkovic, M. Fehr, A. Gawel, M. Grinvald, T. Sattler, R. Siegwart, and J. Nieto, "Incremental Object Database: Building 3D Models from Multiple Partial Observations," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 6835–6842.

[14] J. Strader, N. Hughes, W. Chen, A. Speranzon, and L. Carlone, "Indoor and outdoor 3d scene graph generation via language-enabled spatial ontologies," *IEEE Robotics and Automation Letters*, vol. 9, no. 6, pp. 4886–4893, 2024.

[15] A. Prasad, A. Koller, M. Hartmann, P. Clark, A. Sabharwal, M. Bansal, and T. Khot, "Adapt: As-needed decomposition and planning with language models," 2024. [Online]. Available: https://arxiv.org/abs/2311.05772

[16] I. D. Miller, F. Cladera, T. Smith, C. J. Taylor, and V. Kumar, "Air-ground collaboration with spomp: Semantic panoramic online mapping and planning," *IEEE Transactions on Field Robotics*, vol. 1, pp. 93–112, 2024.

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

[18] Y. Chang, K. Ebadi, C. E. Denniston, M. F. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A. akbar Agha-mohammadi, and L. Carlone, "Lamp 2.0: A robust multi-robot slam system for operation in challenging large-scale underground environments," 2022. [Online]. Available: https://arxiv.org/abs/2205.13135

[19] M. S. Kurtz, S. Prentice, Y. Veys, L. Quang, C. Nieto-Granda, M. Novitzky, E. Stump, and N. Roy, "Real-world deployment of a hierarchical uncertainty-aware collaborative multiagent planning system," 2024. [Online]. Available: https://arxiv.org/abs/2404.17438

[20] W. Gosrich, S. Mayya, S. Narayan, M. Malencia, S. Agarwal, and V. Kumar, "Multi-robot coordination and cooperation with task precedence relationships," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5800–5806.

[21] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Multi-robot mission planning in dynamic semantic environments," 2023. [Online]. Available: https://arxiv.org/pdf/2304.00645.pdf

[22] C. K. Verginis, Y. Kantaros, and D. V. Dimarogonas, "Planning and control of multi-robot-object systems under temporal logic tasks and uncertain dynamics," 2022. [Online]. Available: https://arxiv.org/abs/2204.11783

[23] M. Omama, P. Inani, P. Paul, S. C. Yellapragada, K. M. Jatavallabhula, S. Chinchali, and M. Krishna, "Alt-pilot: Autonomous navigation with language augmented topometric maps," 2023. [Online]. Available: https://arxiv.org/abs/2310.02324

[24] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.16291

[25] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *arXiv preprint arXiv:2209.07753*, 2022.

[26] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and y. Daniel Kappler booktitle=arXiv preprint arXiv:2209.09874, "Open-vocabulary queryable scene representations for real world planning."

[27] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can and not as i say: Grounding language in robotic affordances," in *arXiv preprint arXiv:2204.01691*, 2022.

[28] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable task planning," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=wMpOMO0Ss7a

[29] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone, "Real-time anomaly detection and reactive planning with large language models," in *Robotics: Science and Systems*, 2024.

[30] A. Tagliabue, K. Kondo, T. Zhao, M. Peterson, C. T. Tewari, and J. P. How, "Real: Resilience and adaptation using large language models on autonomous aerial robots," *Conference on Robot Learning (CoRL)*, 2023.

[31] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Visual language maps for robot navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 10 608–10 615.

[32] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," in *arXiv preprint arXiv:2207.05608*, 2022.

[33] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "Nl2tl: Transforming natural languages to temporal logics using large language models," *arXiv preprint arXiv:2305.07766*, 2023.

[34] Z. Dai, A. Asgharivaskasi, T. Duong, S. Lin, M.-E. Tzes, G. Pappas, and N. Atanasov, "Optimal scene graph planning with large language model guidance," 2024. [Online]. Available: https://arxiv.org/abs/2309.09182

[35] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah, "Lang2ltl: Translating natural language commands to

temporal robot task specification," in *Conference on Robbot Learning (CoRL)*, 2023. [Online]. Available: https://arxiv.org/abs/2302.11649

[36] B. Quartey, E. Rosen, S. Tellex, and G. Konidaris, "Verifiably following complex robot instructions with foundation models," vol. 1, 2024.

[37] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, "Autotamp: Autoregressive task and motion planning with llms as translators and checkers," *arXiv preprint arXiv:2306.06531*, 2023.

[38] A. Asgharivaskasi and N. Atanasov, "Semantic octree mapping and shannon mutual information computation for robot exploration," *IEEE Transactions on Robotics*, 2023. [Online]. Available: https://arashasgharivaskasi-bc.github.io/SSMI_webpage/

[39] S. Garg, K. Rana, M. Hosseinzadeh, L. Mares, N. Suenderhauf, F. Dayoub, and I. Reid, "Robohop: Segment-based topological map representation for open-world visual navigation," *arXiv*, 2023.

[40] H.-T. L. Chiang, Z. Xu, Z. Fu, M. G. Jacob, T. Zhang, T.-W. E. Lee, W. Yu, C. Schenck, D. Rendleman, D. Shah, F. Xia, J. Hsu, J. Hoech, P. Florence, S. Kirmani, S. Singh, V. Sindhwani, C. Parada, C. Finn, P. Xu, S. Levine, and J. Tan, "Mobility vla: Multimodal instruction navigation with long-context vlms and topological graphs," 2024. [Online]. Available: https://arxiv.org/abs/2407.07775

[41] Q. Gu, A. Kuwajerwala, S. Morin, K. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. de Melo, J. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," *International Conference on Robotics and Automation*, 2024.

[42] A. Werby, C. Huang, M. Büchner, A. Valada, and W. Burgard, "Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation," *Robotics: Science and Systems*, 2024.

[43] D. Maggio, Y. Chang, N. Hughes, M. Trang, D. Griffith, C. Dougherty, E. Cristofalo, L. Schmid, and L. Carlone, "Clio: Real-time task-driven open-set 3d scene graphs," 2024.

[44] M. Ryll, J. Ware, J. Carter, and N. Roy, "Semantic trajectory planning for long-distant unmanned aerial vehicle navigation in urban environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 1551–1558.

[45] K. Zhou, K. Zheng, C. Pryor, Y. Shen, H. Jin, L. Getoor, and X. E. Wang, "Esc: Exploration with soft commonsense constraints for zero-shot object navigation," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[46] B. Yu, H. Kasaei, and M. Cao, "Frontier semantic exploration for visual target navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 4099–4105.

[47] A. Pacheck and H. Kress-Gazit, "Physically feasible repair of reactive, linear temporal logic-based, high-level tasks," *IEEE Transactions on Robotics*, 2023.

[48] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," 2019. [Online]. Available: https://arxiv.org/abs/1901.02077

[49] D. Honerkamp, M. Büchner, F. Despinoy, T. Welschehold, and A. Valada, "Language-grounded dynamic scene graphs for interactive object search with mobile manipulation," *arXiv preprint arXiv:2403.08605*, 2024.

[50] Z. Hu, F. Lucchetti, C. Schlesinger, Y. Saxena, A. Freeman, S. Modak, A. Guha, and J. Biswas, "Deploying and evaluating llms to program service mobile robots," *IEEE Robotics and Automation Letters*, 2024.

[51] S. Sharan, F. Pittaluga, V. Kumar B G, and M. Chandraker, "Llm-assist: Enhancing closed-loop planning with language-based reasoning," *arXiv preprint arXiv:2401.00125*, 2023.

[52] Q. Xie, T. Zhang, K. Xu, M. Johnson-Roberson, and Y. Bisk, "Reasoning about the unseen for efficient outdoor object navigation," 2023.

[53] D. Shah, M. R. Equi, B. Osiński, F. Xia, B. Ichter, and S. Levine, "Navigation with large language models: Semantic guesswork as a heuristic for planning," in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Tan, M. Toussaint, and K. Darvish, Eds., vol. 229. PMLR, 06–09 Nov 2023, pp. 2683–2699. [Online]. Available: https://proceedings.mlr.press/v229/shah23c.html

[54] D. Shah, B. Osiński, b. ichter, and S. Levine, "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 492–504. [Online]. Available: https://proceedings.mlr.press/v205/shah23b.html

[55] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.

[56] Y. J. Ma, W. Liang, H. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "Dreureka: Language model guided sim-to-real transfer," in *Robotics: Science and Systems (RSS)*, 2024.

[57] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and Shah, "Grounding complex natural language commands for temporal tasks in unseen environments," in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Tan, M. Toussaint, and K. Darvish, Eds., vol. 229. PMLR, 06–09 Nov 2023, pp. 1084–1110. [Online]. Available: https://proceedings.mlr.press/v229/liu23d.html

[58] K. Garg, J. Arkin, S. Zhang, N. Roy, and C. Fan, "Large language models to the rescue: Deadlock resolution in multi-robot systems," 2024. [Online]. Available: https://arxiv.org/abs/2404.06413

[59] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[60] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[61] OpenAI et al., "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[62] N. Steinke, D. Goehring, and R. Rojas, "Groundgrid: Lidar point cloud ground segmentation and terrain estimation," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 420–426, 2024.

[63] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499*, 2023.

[64] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," in *NeurIPS*, 2023.

[65] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-lio: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4861–4868, 2022.

[66] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[67] Gemma Team et al., "Gemma: Open models based on gemini research and technology," 2024. [Online]. Available: https://arxiv.org/abs/2403.08295

APPENDIX I
SUMMARY

In this appendix we provide further detail on our proposed method. Subsection A2-A provides details on the LLM system prompt, including the perception api and planning interface. Subsection A2-B describes the behavior library implementation including the controller used. Subsection A2-C provides more details and visualizations on teh semantic mapping components traversability estimation, object detection, and VLM results. Section A3 provides details on the experimental setup. We provide more details on the experimental missions, including subtasks required and prior maps given to the planner and provide further discussion on results (Subsection A3-F), including why the performance of SPINE was 6% lower than baselines in the simulation experiments (see Tab. II)

APPENDIX II
FURTHER METHOD DETAILS

We provide details on the implementation of the LLM configuration, semantic mapper, and behavior library.

A. LLM configuration

The LLM configuration consists of four main parts: main system configuration, perception API, planning API, and planning advice. The system configuration provides an overview of the LLM's role in the planning framework and defines interfaces (see Listing A2-C). The perception API defines how the LLM will receive updates from the semantic mapper (see Listing A2-C). The planning API defines how the LLM will compose subtasks sequences (see Listing A2-C). Finally, the advice portion of the configuration preempts common mistakes we observed the LLM making during development (see Listing A2-C). We also provide five in-context examples of canonical planning behavior, and example of which is detailed in Listing A2-C, and we refer the reader to our software for a complete list. At runtime, the user-provided mission and current scene graph is appended to the context.

B. Behavior library and Constraint Feedback

We provide further details on the behaviors listed in Tab. I. `goto` takes a string, which is interpreted as a region node. The planner with find the shortest path to that node over the current graph, and it will then navigate to that node. The following behaviors call `goto` for navigation to a particular node, where applicable. `map_region` takes a string, which is interpreted as a region node. The robot will navigate to that node and report any objects detected along the way. `explore_region` takes a string and float, which is interpreted as a region node and exploration radius, r, in meters. The robot will navigate to that node, then explore the circle of radius r around that region node. `extend_map` takes two floats, which is interpreted a 2D coordinate. The robot will attempt to navigate to that coordinate. `inspect` takes two strings, which is interpreted as an object node and inspection query. The robot will navigate to that object, which is obtain an image of that object, pass that image and query to a VLM, and report the VLM ansewr. `set_labels` takes a set of strings, which is interpreted as class labels. These labels are used to configure the object detector. `clarify` takes a string, interpreted as a question and provided to the user. The user can respond. `answer` takes a string, which is interpreted as an answer to the user's mission. This terminates the mission. For all navigation behaviors, we use the controller implemented by ROS Move Base [1] with a target velocity of $0.5m/s$.

Each constraint provides tailored feedback, if violated. **syntax** is defined over the previously described behaviors. The feedback associated with this constraint highlights offending variables and function spelling.
reachable is defined over region nodes. There must be a path to the region node in the current map. Feedback associated with this constraint lists unreachable nodes. Feedback will then suggest exploration objectives based on the closest reachable node to the goal point.
explorable is defined over exploration goals. There must be a obstacle-free path between the robot's current location and the goal. If such a path cannot be found, feedback will provide the reason why (*e.g.*,, exploration hit an obstacle boundary).

C. Semantic Mapper

The architecture for the semantic mapper used by SPINE is shown in Fig. A1. The mapper takes RGB + Depth, LiDAR, and semantic configuration as inputs. LiDAR is used for odometry estimation (Faser-LIO [65]) and local occupancy map construction (GroundGrid [62]). The occupancy map is used to add and remove regions and edges from the map based on connectivity. RGB+D is used for object localization and captioning. Objects are detected using GroundingDino [63]). Detections are then clustered and localized with a multiple-hypothesis tracker. A vision-language model (LLaVA [64]) provides enriches the semantic information available to the planner (see Fig. A2, Fig. 6). Outputs from these modules are used to add and remove nodes and enrich them with semantic information. Semantic configuration is provided by the planner and is used to set the labels of the object detector and provide queries to the vision language model. The detection and tracking modules runs at roughly 5Hz, and the vision-language model runs at roughly 1Hz, and occupancy map construction runs well over 10Hz, all onboard. Taken together, the semantic mapper runs sufficiently fast for real-time planning and control.

[1] http://wiki.ros.org/move_base

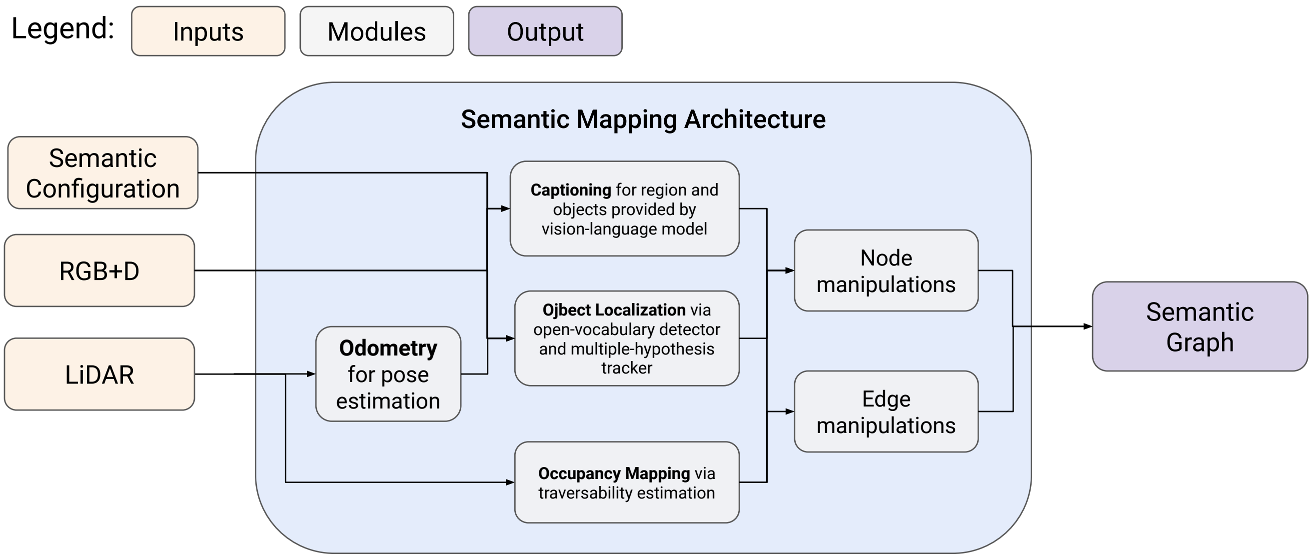

Fig. A1. Semantic mapping architecture used by SPINE. The mapper takes LiDAR, RGB + Depth (RGB+D) sensor streams, and semantic configuration provided by the semantic planner. Odometry provides pose estimation. Occupancy mapping uses a traversability estimator to build a local map of obstacles, which is used to add and remove regions or edges to the map. Object localization uses an open-vocabulary object detector and multiple-hypothesis tracker to identify and ground objects in physical space. The captioning module provides further semantic detail to detected objects or regions. Information from these modules is uses to added and remove nodes and edges. Semantic configuration is used to set labels for the Object Localization module or provide queries for the Captioning module.

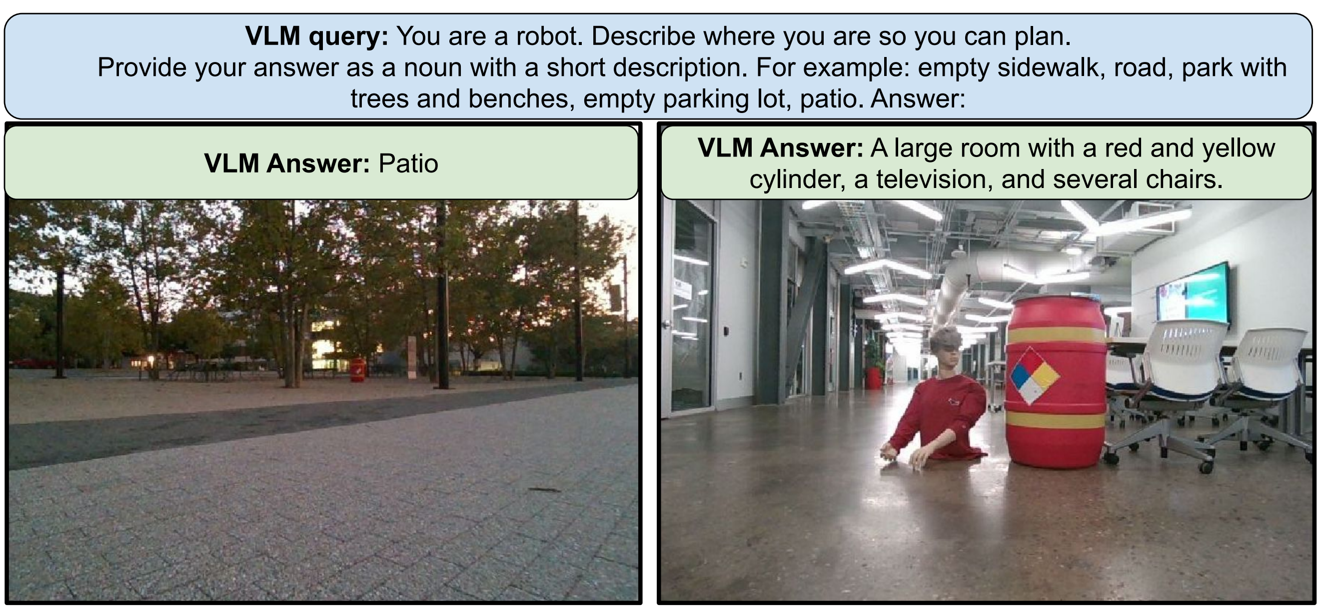

Fig. A2. Examples of Vision-Language Model captioning during exploration. Captions provide brief semantic descriptions of the scene which may be useful for planning.

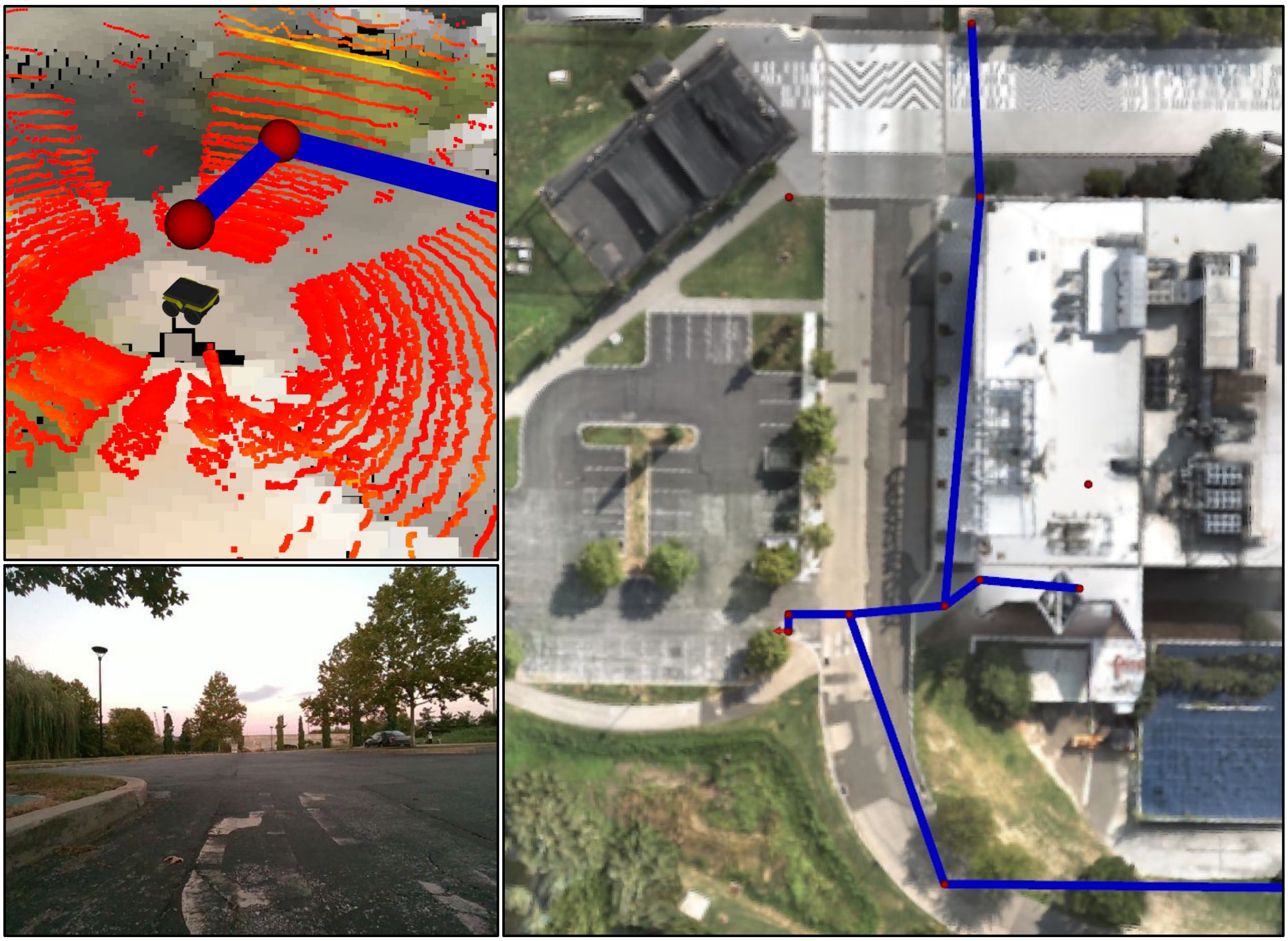

Fig. A3. Example prior graph used by SPINE (right). Edges in blue and nodes in red. Semantic labels omitted for clarity. Third person view of robot is overlaid on overhead imagery (top left). Camera view from the robot is shown in the bottom left. Because this graph was derived from overhead imagery, registration was imperfect, and the planner must adjust in real-time (note edges that cross the building intersection).

```
Agent Role: You are an excellent graph planner. You must fulfill a given task provided by
the user given an incomplete graph representation of an environment.

You will generate a step-by-step plan that a robot can follow to solve a given task.
You are only allowed to use the defined API and nodes observed in the scene graph for planning.
Your plan will provide a list of actions, which will be realized in a receding-horizon manner.
At each step, only the first action in the plan will be executed.
You will then receive updates, and you have the opportunity to replan.
Updates may include discovered objects or new regions in the scene graph.
The graph may be missing objects and connections, so some tasks may require you to explore.
Exploration means mapping existing regions to find objects, or adding a new region to find paths.

The graph is given the in the following json format:
```
{
        "objects": [{"name": "object_1_name",
        "coords": [west_east_coordinate, south_north_coordinate]}, ...],
        "regions": [{"name": "region_1_name",
        "coords": [west_east_coordinate, south_north_coordinate]}, ...],
        "object_connections: [["object_name", "region_name"], ...],
        "region_connections": [["some_region_name", "other_region_name"], ...]
        "robot_location": "region_of_robot_location
}
```

Each entry of the graph contains the following types:
- "regions" is a list of spatial regions.
The regions are traversable ONLY IF they appear in the "region_connections" list
- "object_connections" is a list of edges connecting objects to regions in the graph.
An edge between an object and a region implies that the robot can see
the given object from the given region
- "region_connections" is list of edges connecting regions in the graph.
An edge between two regions implies that the robot can traverse between those regions.


Provide you plan as a valid JSON string (it will be parsed by the `json.loads` function in python):
```
{
"primary_goal": "Explain your primary goal as provided by the user.
Reference portions of graph, coordinates, user hints, or anything else that may be useful.",
"relevant_graph": "List nodes or connections in the graph needed to complete your goal.
If you need to explore, say unobserved_node(description). List ALL relevant nodes.",
"reasoning": "Explain how you are trying to accomplish this task in detail.",
"plan": "Your intended sequence of actions.",
}
```
```

Listing 1: LLM system prompt: role description

```python
def remove(node: str) -> None:
    """Remove `node` and associated edges from graph."""


def add_node(type: str, name: str) -> None:
    """Add `node` of `type` to graph."""


def add_connection(type: str, node_1: str, node_2: str) -> None:
    """Add connection of `type`
    (either `region_connection` or `object_connection`) between `node_1` and `node_2`."""


def update_robot_location(region_node: str) -> None:
    """Update robot's location in the graph to `region_node`."""


def update_node_attributes(region_node, **attributes) -> None:
    """Update node's attributes, where attributes are key-value pairs of attributes
    and updated values."""


def no_updates() -> None:
    """There have been no updates."""
```

Listing 2: LLM system prompt: perception API

```python
def goto(region_node: str) -> None:
    """Navigate to `region_node`."""


def map_region(region_node: str) -> List[str]:
    """Navigate to region in the graph and look for new objects.
    - region_node must be currently observed in graph and reachable from the robot's location.
    - This CANNOT be used to add connections in the graph.

    Will return updates to graph (if any).
    """


def extend_map(x_coordinate: int, y_coordinate: int) -> List[str]:
    """Try to add region node to graph at the coordinates (x_coordinate, y_coordinate).

    You should call this when your goal is far away (over 10 meters, for example).

    NOTE: if the proposed region is not physically feasible
    (because of an obstacle, for example), the closest feasible region will
    be added instead.

    Will return updates to graph (if any).
    """


def explore_region(region_node: str, exploration_radius_meters: float) -> List[str]:
    """Explore within `exploration_radius_meters` around `region_node`
    If (x, y) are the coordinates of `region_node` and `r` is the exploration radius.
    This will try to add regions at (x + r, y), (x - r, y), (x, y + r), (x, y - r).
    The robot will then map the discovered regions to find any unobserved objects.

    You should only call this if you are close to your goal (within exploration radius).

    Will return updates to graph (if any).
    """


def replan() -> None:
    """You will update your plan with newly acquired information.
    This is a placeholder command, and cannot be directly executed.
    """


def inspect(object_node: str, vlm_query: str) -> List[str]:
    """Gather more information about `object_node` by
    querying a vision-language model with `vlm_query`. Be concise in
    your query. The robot will also navigate to the
    region connected to `object_node`.

    Will return updates to graph (if any).
    """


def answer(answer: str) -> None:
    """Provide an answer to the instruction"""


def clarify(question: str) -> None:
    """Ask for clarification. Only ask if the instruction is too vague to make a plan."""
```

Listing 3: LLM system prompt: planning API

```
The user given task with be prefaced by `task: `, and updates will be prefaced by `updates: `.

Remember the following when constructing a plan:
- You will receive feedback if your plan is infeasible.
The feedback will discuss the problematic parts of your plan and reference specific regions of the
graph. You will be expected to replan.

Remember the following at each planning iteration:
- When given an update, replan over the most recent instruction and updated scene graph.
- When given feedback, you must provide a plan that corrects the issues with your previous plan.


Planning Advice:
- Carefully explain your reasoning and all information used to
create your plan in a step-by-step manner.
- Recall the scene may be incomplete.
You may need to add regions or map existing regions to complete your task.
- Reason over connections, coordinates, and semantic relationships between objects and regions
in the scene. For example, if asked to find a car, look near the roads.
- Coordinates are given west to east and south to north.

Before calling extend_map, consider this:
- If you need to find a path but there are NO existing connections,
you should call extend_map in the direction of that region.
- Before you call extend_map ask:
is there an existing connection I can use to get to my goal region? If so, use that.

Before calling explore_region, consider this:
- If you need to check if a path is clear, do not call explore.
Rather, map the region to find obstacles.

Before calling goto, consider this:
- goto uses a graph-search algorithm to find an efficient path,
so avoid calling goto on intermediate nodes.
- For example, if you path is ground_2 ->
ground_7 -> ground_10. Call goto(ground_10) instead of goto(ground_7)
```

Listing 4: LLM system prompt: planning advice

```
EXAMPLE_GRAPH_1 = {
    "objects": [
        {"name": "house_1", "coords": [-1, -1]},
        {"name": "house_2", "coords": [-3, -1]},
        {"name": "grocery_store_1", "coords": [-5, -1]},
        {"name": "shed_1", "coords": [1, 3]},
        {"name": "shed_1", "coords": [1, 5]},
    ],
    "regions": [
        {"name": "example_road_1", "coords": [-1, 0]},
        {"name": "example_road_2", "coords": [-2, 0]},
        {"name": "field_11", "coords": [0, 1]},
        {"name": "field_13", "coords": [2, 3]},
    ],
    "object_connections": [
        ["house_1", "example_road_1"],
        ["house_2", "example_road_2"],
        ["shed_1", "field_11"],
        ["shed_2", "field_13"],
    ],
    "region_connections": [
        ["example_road_1", "example_road_2"],
        ["example_road_1", "field_11"],
        ["field_11", "field_13"],
    ],
    "robot_location": "example_road_1",
}

EXAMPLE_1 = [
    {
        "role": "user",
        "content": "task: I need a shovel. Is there one in the scene?"
        + f"Scene graph:{str(EXAMPLE_GRAPH_1)}",
    },
    {
        "role": "assistant",
        "content": '{\n\
        "primary_goal": "find a shovel for the user."\
        "relevant_graph": "field_11, field_13, unobserved_node(shovel)", \
        "reasoning": "The graph does not contain any shovels.
        However, I know that the graph may be incomplete, so I will explore before providing a definitive answer.
        I will first map, then if needed I will add regions.
        There are two sheds in the scene, and shovels are often found near sheds.
        Therefore, for each of the sheds, I will navigate to the nearby region and map.",\
        "plan": "[goto(field_11), map_region(field_11), goto(field_13), map_region(field_13)]"\n}',
    },
    {
        "role": "user",
        "content": "updates:[no_updates()]",
    },
    {
        "role": "assistant",
        "content": '{\n\
            "primary_goal": "find a shovel for the user."\
            "relevant_graph": "field_13, unobserved_node(shovel)",\
            "reasoning": "After mapping field_11, which is next to shed_1, I did not find a shovel.
            I also planned on going to example_node_14 to map near shed_2, so I will continue my plan.",\
            "plan": "[goto(field_13), map_region(field_13)]"\n}',
    },
    {
        "role": "user",
        "content": "updates:[add_nodes({ name: shovel_1, type: object}), add_connections([shovel_1, field_13])]",
    },
    {
        "role": "assistant",
        "content": '{\n\
        "primary_goal": "find a shovel for the user."\
        "relevant_graph": "shovel_1", \
        "reasoning": "After exploring field_13, which is next to shed_2, I found a nearby shovel, shovel_1,
        that is connected to field_13. This fulfills the users request.",\
        "plan": "[answer(There is a shovel, shovel_1, that is near shed_2 and connected to region_13.)]"\n}',
    },
    {
        "role": "user",
        "content": "task: How many grocery stores are currently in the scene?",
    },
    {
        "role": "assistant",
        "content": '{ \
        "primary_goal": "find a shovel for the user."\
        "relevant_graph": "grocery_store_1",\
        "reasoning": "There is one grocery store in the scene. Although the scene may be incomplete,
        I am only asked about currently observed grocery stores. Thus the answer is one",\
        "plan": "[answer(There is one grocery store in the scene, grocery_store_1.)]"}',
    },
```

Listing 5: In context learning example provided to the LLM.

APPENDIX III
EXPERIMENTAL DETAILS

This section provides details on the experimental tasks reported in Section IV. We describe the mission, instruction given the SPINE, and the subtasks required. We then provide further discussion on experiments.

A. Semantic Route inspection

Mission provided to SPINE: "There was a storm last night. I am worried that impacted logistics, because I need to drop off supplies today. Can I still do that¿'

Implied subtasks: The planner must recognize that the delivery depot is the most likely place for supply delivery. The user wants to make sure the path between the current location and delivery depot is free. These subtasks are

1) Recognize semantics. Primarily current location and delivery depot. Bonus: recognize that debris, puddles, fallen trees, etc, will give information about the extend of the storm.
2) Navigate along path path to delivery depot. At each step, if the robot cannot traverse an edge, it is likely blocked.

Map is shown in Figure A4, which provided semantics: ground, road, cabin, radio tower, truck, light pole, bridge, supply depot.

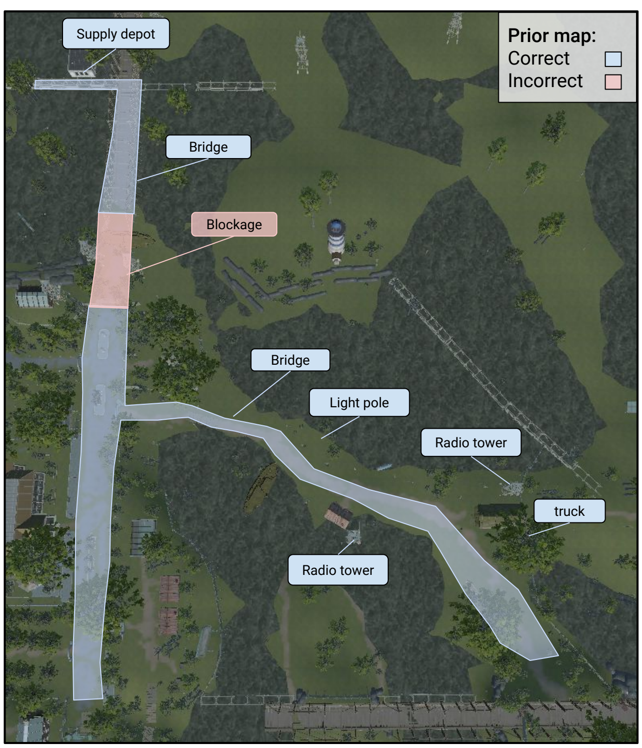

Fig. A4. The semantic route inspection mission requires the planner to check if the path to the supply depot is free. Red is blocked by storm. The extent of prior is roughly 260m x 225m

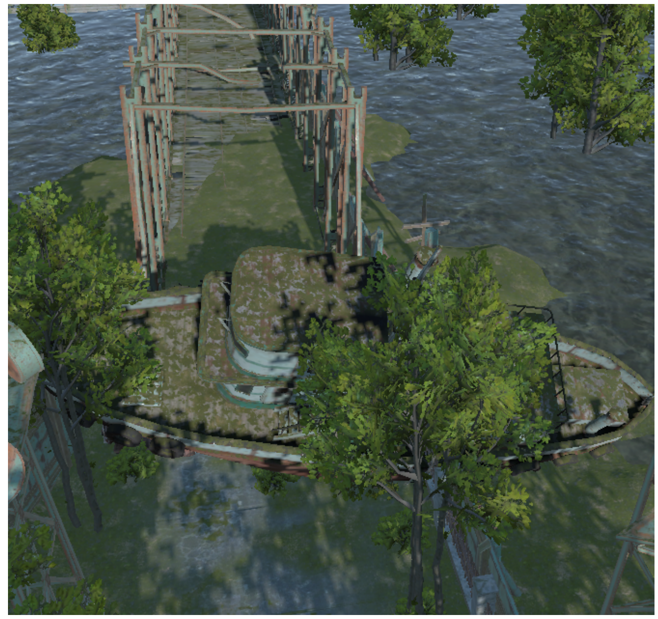

Fig. A5. The semantic route inspection mission requires the robot to infer a path to the supply depot (across the bridge shown in the figure). During route inspection, the robot must recognize that the bridge is physically blocked.

B. Search and inspection with implicit goals

Mission provided to SPINE: "I sent a robot out to collect supplies from an incoming boat. I have not heard back. What happened?"

Implied subtasks: The planner must recognize that it is looking for a robot, and use the contextual information provided to infer the robot is likely near one of the three docks in the scene. The map does not provide a direct path to these docks, so the planner must explore in order to reach its goal locations. The planner must then find the mission robot, which is near the third dock.

The implied subtasks are:

1) Infer correct semantic labels (robot) and best search locations (three docks)
2) Understand gaps in map (three major gaps)
3) Navigate to the map boundary
4) Extend map to the first dock
5) Extend map to the second dock
6) Extend map to the third dock
7) Find and inspect robot
8) Report findings to user

Map is shown in Fig. A6 with semantics dock, ground, road, cabin, radio tower, truck, light pole. Not all regions or semantics in prior are relevant to task.

C. Multi-object inspection with implied semantics

Instruction provided to SPINE: "Communications are down. Can you figure out why?"

Implied subtasks: There are two radio towers provided in the prior map. The planner must infer that radio towers are relevant for communication, so it should inspect those. There is no direct path between the planners start locations and the radio towers, so the planner must explore.

Fig. A6. The search and inspection mission requires the planner to search near docks for another missing robot. The prior information provided to the planner and missing componenets in the map are illustrated.

Fig. A7. The search and inspection mission requires the planner to locate the robot shown in the figure and report the robot's position (eg "robot is at location (x,y) and appears to be stationary"

The implied subtasks are:

1) Identify inspection targets (radio towers)
2) Go to region boundary
3) Explore a path to the first radio tower
4) Inspect the first radio tower by forming appropriate query (eg, "is this radio tower damaged") and reason over response
5) Navigate to second radio tower
6) Inspect the first radio tower by forming appropriate query (eg, "is this radio tower damaged") and reason over response
7) Provide information to user

Map is shown in Fig. A3-C, with provided semantics, ground, road, cabin, radio tower, truck, light pole.

D. Semantic route inspection on real robot

Instruction given the SPINE I am worried that recent construction on roads and fences impacted maritime supply logistics. Can you check?

Implied subtasks: The planner must recognize that the user is concerned about a path to the dock, which is provided

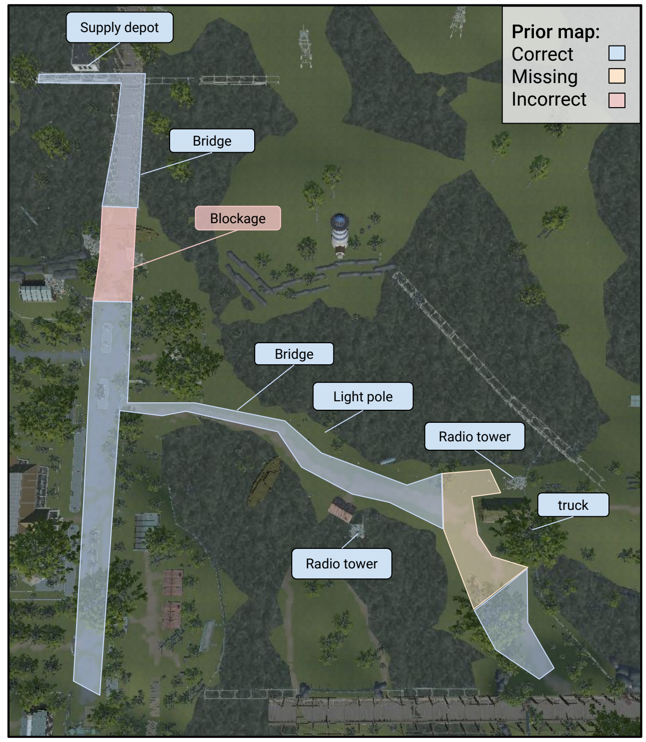

Fig. A8. The multi-object inspection mission requires the planner to infer inspection targets (radio towers). There is no direct path provided in the prior, so the robot must explore to find a path. Furthermore, there are some distractors inclugin the bridge and supply depot (top of figure).

in the prior. The prior is outdated; there is a newly built fence which obstructs the path. Furthermore, some of the path between the robots starting location and dock is missing. Thus, the planner must inspect the path towards the dock, recognized blockage, and report findings to the user. A successful mission terminated when the the discovered the fence recently constructed, and the planner notifies the user. See Fig. A9

The implied subtasks are:

1) Specify correct semantics (roads, fences)
2) Identify goal location (dock)
3) Go to map boundary
4) Fill in missing portion of path
5) Use valid priors to navigate towards the dock
6) Recognize blockage
7) Report to user

Map is shown in Fig. 5 with semantics courtyard, tree, parking lot, road, dock, path.

E. Air-ground teaming on real robot

Mission provided to SPINE You are assisting a high-altitude UAV in responding to an emergency chemical spill. Triage regions that are not visible from the air.

Implied subtasks: The planner must recognize that inside buildings and under trees cannot be observed from high-altitude UAVs, thus the planner should explore those regions. There are regions of the map that are not provided in the prior, so the planner must explore. The planner must also

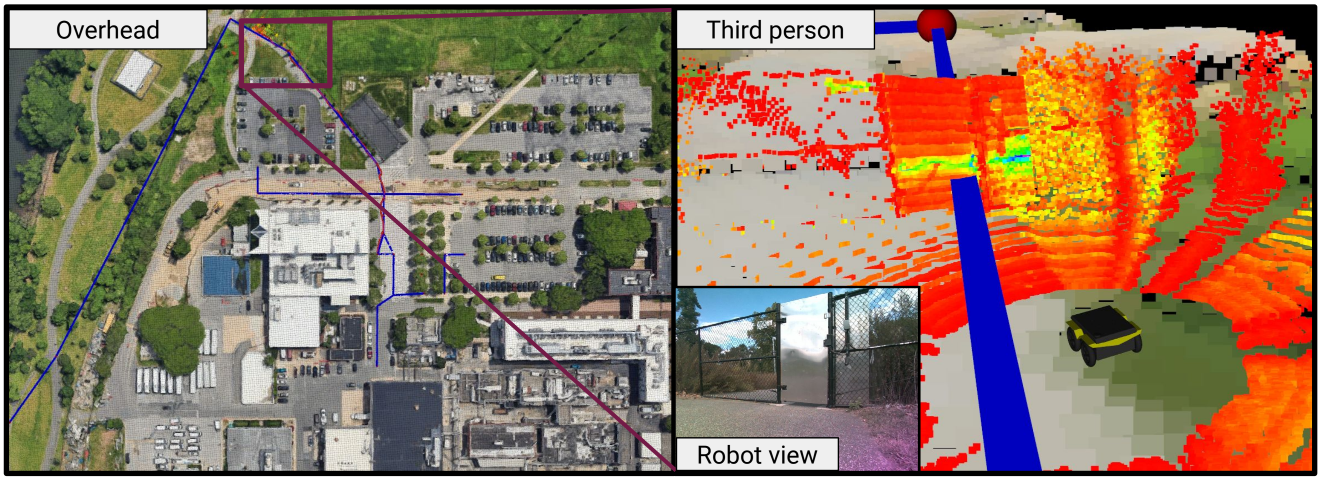

Fig. A9. Example outcome on semantic route inspection mission. The mission implies that recent construction may have impacted the user's intended route to the dock (bottom left, off image). The planner searches along route until it finds a blockage (right). The planner then reports its findings to the user.

look for relevant semantics, including people and chemical barrels.

The implied subtasks are:

1) Configure semantics (people, barrels)
2) Go to the building entrace
3) Explore to find a path inside
4) recognize task-relevant objects
5) Navigate to tree cover, which requiures going to boundary of prior map
6) Explore to tree cover
7) Itentify task-relevant objects.

Map is shown in Fig. 1 with semantics: parking lot, road, field, sidewalk, building, trees

F. Discussion of results

We observed comparative performance drop in SPINE (see Table II) during multi-object inspection missions (Subsection A3-C). This mission required the planner to inspect two radio towers in the scene. During some runs, the planner would inspect the first tower, learn that the tower was damaged, and terminate the mission. While this behavior is correct, it is not complete.

For both the explicit tasking baseline and SPINE, there was one manual takeover for each experiment. These takeovers were both due to the minimum range of the obstacle detector, which was around 1 meters. If the robot came closer to one meter to an obstacle, that obstacle would not be registered in the perception costmap, thus the robot would try to drive into the obstacle. See Fig. A10 for an illustration.

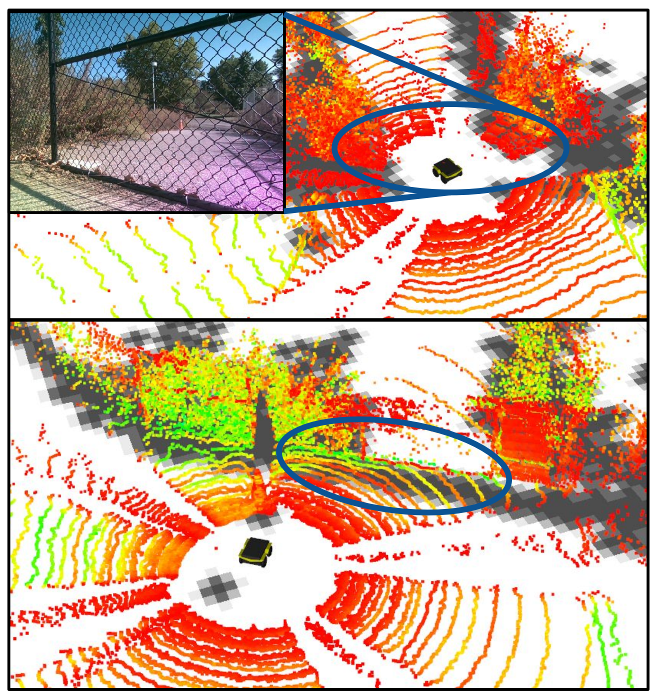

Fig. A10. Cause of manual takeover during experiment. The LiDAR's minimum return distance was roughly 1 meter, so obstacles closer than this were not detected (top). When obstacles were not registered in the occupancy map, the robot tried to drive through them, which required manual takeover. The obstacles were picked back up again when robot moves farther away (bottom).