

**Natural Language-Driven Situational Awareness for
Autonomous Robots**

by

Harel Biggie

B.S., University of Rochester, 2018

M.S., University of Colorado Boulder, 2023

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
2024

Committee Members:

Christoffer Heckman, Chair

Bradley Hayes

Thomas Howard

James H. Martin

Nima Keivan

Biggie, Harel (Ph.D., Computer Science)

Natural Language-Driven Situational Awareness for Autonomous Robots

Thesis directed by Dr. Christoffer Heckman

Autonomous mobile robots are essential for applications like search and rescue, but navigating unfamiliar environments requires sophisticated decision-making based on semantics which are typically obtained from human input. A key limitation in traditional navigation approaches includes a lack of semantic understanding core to human reasoning. This work aims to bridge this gap by leveraging the power of large language models (LLMs) and traditional navigation techniques to enhance robotic situational awareness.

While LLMs excel at language tasks, integrating them into robotic systems presents challenges. The “world knowledge” within LLMs must be grounded in the robot’s physical sensor data for meaningful use. Additionally, safety in robotic systems demands explainable outputs. This dissertation presents two frameworks that address these limitations: code generation from natural language and a contextual reasoning framework that encodes environmental states as natural language. These frameworks build upon a state-of-the-art exploration system developed for the DARPA Subterranean Challenge, which enabled successful autonomous navigation in GPS-denied environments.

This work’s contributions include a flexible exploration system for mobile robots that emphasizes autonomy with high-level human input, along with innovations in subterranean communications. Most importantly, this research introduces two LLM-based frameworks – code generation and contextual reasoning – to improve robots’ understanding and navigation abilities within their environments.

Dedication

With heartfelt thanks to my family and friends, who have always been there for me on this journey. In loving memory of my Dad.

Acknowledgements

I would like to express my sincere thanks to my advisor Christoffer Heckman, for his invaluable guidance and support throughout this transformative journey. His insights, expertise, and insightful conversations played a critical role in shaping this work. Additionally, I extend my appreciation to the other principal investigators on the DARPA Subterranean Challenge, Sean Humbert and Eric Frew, for their unwavering support on the Subterranean challenge. I would also like to thank Steve McGuire for his instrumental technical guidance and the stimulating discussions that greatly enriched this process. My heartfelt thanks go to my committee members for their valuable time and feedback. I am also grateful to all my labmates in ARPG and the dedicated members of Team MARBLE without whom this work would not have been possible

Finally, I would like to acknowledge the funding provided by the following sources: DARPA Subterranean Challenge cooperative agreement HR0011-18-2-0043, USDA-NIFA #2021-67021-33450, and NSF #1764092

Contents

Chapter

1	Introduction	1
1.1	Contributions	2
1.2	Publications	3
1.2.1	Journal Articles	3
1.2.2	Conference Proceedings	4
2	Flexible Supervised Autonomy For Exploration in the Context of Search and Rescue	6
2.1	Background and Problem Definition	6
2.2	Related Works	7
2.3	Overview of Underground Exploration Approach	8
2.3.1	Concept of Operations	9
2.3.2	Perception	11
2.3.3	Localization	11
2.3.4	Exploration	11
2.3.5	Mapping	12
2.3.6	Artifact Detection	12
2.3.7	Communication Systems	13
2.3.8	Multi-Agent Coordination	13
2.3.9	Mission Management	13

2.4	Platform Development	14
2.4.1	Communication Beacons	15
2.4.2	Power and Platform Control Systems	17
2.5	Object Detection	18
2.5.1	Sensing Modalities	18
2.5.2	Visual Detection	19
2.5.3	Non-Visual Detection	21
2.6	Performance of the System	22
2.6.1	Overview	22
2.6.2	Localization & Mapping	24
2.7	Planning	27
2.7.1	Exploration	28
2.7.2	Treacherous Terrain	29
2.7.3	Dynamic Environment	30
2.7.4	Limitations	32
2.8	Artifact Detection	33
2.8.1	Visual Detection	33
2.8.2	Non-Visual Detection	35
2.9	Mission Management	36
2.9.1	Robot Launches	37
2.9.2	Robot Interventions	38
2.10	Lessons Learned	39
2.10.1	Platform Mobility	40
2.10.2	Testing and Validation	40
2.10.3	System Adaptability	41
2.10.4	Autonomy and Human Robot Interaction	42
2.10.5	Retrospective	43

2.11	Conclusions	44
3	Communication Systems	45
3.1	Introduction	45
3.2	Problem Requirements	46
3.3	Background	47
3.3.1	Physical Layer	47
3.3.2	Logical Layer	47
3.3.3	Transport Layer	48
3.4	Methods	50
3.4.1	Physical	50
3.4.2	Logical	51
3.4.3	Transport	51
3.4.4	Discovery and Address Resolution	52
3.4.5	ROS Message Encapsulation	53
3.4.6	Point to Point Transport	53
3.4.7	Quality of Service	54
3.4.8	Point-to-Multipoint Transport	54
3.5	Functional Test	55
3.5.1	Case Study: Differential Maps	56
3.5.2	Case Study: Adding Point-Of-View Imagery	58
3.6	Discussion	58
3.6.1	Porting to arbitrary publish / subscribe architectures	58
3.6.2	Scaling to larger team sizes	59
3.6.3	Commodity Hardware	59
3.6.4	Interoperability with Existing Systems	60
3.6.5	Fast Reconnect Times	60

3.6.6	Prioritization Controls	61
3.7	Conclusions	61
4	Background On Language In Robotics	62
4.1	Language grounding:	63
4.2	How Can Language Be Represented	63
4.3	Probabilistic Models	65
4.4	Large Language Models	67
5	Code Generation For Language Grounding	71
5.1	Introduction	71
5.2	Related Works	73
5.2.1	Modular Neural Network Frameworks and Code Synthesis	73
5.2.2	Path Planning for Robotic Navigation	74
5.2.3	Embodied Language-Based Navigation	74
5.3	Methodology	75
5.3.1	Input Representations	75
5.3.2	Intermediate Layer	76
5.3.3	Planning	76
5.4	Experimental Results	77
5.4.1	Real-World Environments	84
5.4.2	Qualitative Examples:	86
5.5	Conclusion and Limitations	91
6	CogExplore: Planning Based On Contextual Cues with Temporal Reasoning	93
6.1	Related Works	95
6.2	Methodology	98
6.2.1	Exploration Planner	99

6.2.2	Foundation Model Prompting Schemes	100
6.3	Experimental Setup	103
6.4	Experimental Results	104
6.5	Discussion	107
6.5.1	Geometric Reasoning	108
6.5.2	Semantic Reasoning	111
6.5.3	Temporal Reasoning	112
6.5.4	Limitations of CogExplore	114
6.6	Conclusions	115
7	Future Work and Conclusions	117
7.1	Future Work	117
7.2	Conclusions	118
	Bibliography	120
	Appendix	
A	Full Language Model Prompts	133
A.1	NavCon: Code Generation Language Prompt	133
A.2	Cog Explore Language Model Prompts:	147
A.2.1	VQA Prompting	147
A.2.2	Object Detection Label Prompting	149
A.2.3	Explore Iteration Prompting	150
A.2.4	GPT-3.5 Turbo Specific Prompt Additions	155
A.2.5	Claude Haiku Specific Prompt Additions	156

Tables

Table

2.1	Artifact Sensing Modalities	19
2.2	Scored Artifacts	25
2.3	Missed Artifact Scores	36
2.4	Artifact Report Statistics	37
2.5	Final Event Robot Launch Sequences	38
2.6	Human Intervention List	40
5.1	Code Generation Framework Results (NavCon	79
5.2	Scene-05: Classroom Full Sentence List	80
5.3	Category Wise Success Rate	85
5.4	Scene-wise Success Rate	86
5.5	Scene-01: Theater	88
5.6	Scene-02: Lobby	89
5.7	Scene-03: Outdoor	90
5.8	Scene-04: Courtyard	92
6.1	Simulation Environments and Task Overview	104
6.2	Path lengths for FE1 in Office 1	106
6.3	Path lengths for FE2 in Office 1	106
6.4	Path lengths for FE3 in Office 1	107

6.5	Path lengths for CT in Office 2	113
6.6	Path lengths for OC in Office 2	113
6.7	Path lengths for WB in the School	114
6.8	Path lengths for BS in the School	115

Figures

Figure

2.1	Robot Fleet	9
2.2	System Block Diagram	10
2.3	Human Supervisor Interface	14
2.4	Power and Signal Routing Diagrams	16
2.5	Final Robot Configurations	16
2.6	Overview of the artifact detection system. Sensor inputs are shown in red and outputs are shown in green.	21
2.7	Final Maps From Individual Robots	23
2.8	Final Event Course Map	24
2.9	Final Point Cloud Map	26
2.10	Map Coverage Error	26
2.11	Subway Platform Exploration	27
2.12	Cavern Exploration Example	27
2.13	Pose Graph Coordination Example	28
2.14	Testing Sites	29
2.15	Volumetric Gain For Exploration	30
2.16	Spot Stair Planning	31
2.17	Dynamic Obstacle Example	31
2.18	Mission Management Timeline	32

2.19	Artifact Detection System Diagram	34
2.20	Ground Truth Artifact Locations	35
2.21	Robot Intervention Imagery	39
3.1	ROS Master Network Graph	49
3.2	Multi-Ros Network Graph	50
3.3	Network Architecture For Husky	52
3.4	UDP Mesh Graph	55
3.5	Map Transmission Example	55
3.6	PDF of Message Arrival Times	57
3.7	Final Comms Map	57
3.8	First Person Video Feed	59
4.1	Visual Language Navigation in Robotics	64
5.1	Contextual Navigation Example	71
5.2	Testing Environments	74
5.3	NavCon Full System Diagram	77
5.4	Example Sentences	78
5.5	Sentence Result	87
5.6	Run Upstairs Example	91
6.1	CogExplore Inputs Overview	93
6.2	Renderings from Unreal Engine environments. An example of a simulated spot can be seen in the rightmost image.	97
6.3	CogExplore System Diagram	99
6.4	Example Runs demonstrating geometric reasoning (left), semantic reasoning (cen- ter), and temporal reasoning (right).	105
6.5	Path Length Comparisons	105

6.6	Overhead Path Examples for Office Environment 1	109
6.7	Office 1, Fire Extinguisher 1	109
6.8	Office 1, Fire Extinguisher 2	109
6.9	Office 1, Fire Extinguisher 3	109
6.10	Office Environment 2 Overhead Path Examples	111
6.11	Office 2, Coffee Table	111
6.12	Office 2, Office Chair	111
6.13	School Environment Overhead Paths	114
6.14	School, Whiteboard Eraser	114
6.15	School, Bookshelf	114

Chapter 1

Introduction

Autonomous mobile robots are increasingly being used in critical applications such as search and rescue, where their success depends on their ability to explore and search for objects in unfamiliar environments with minimal human intervention. Traditional approaches usually rely on a navigation stack consisting of a metric-based planner, simultaneous localization and mapping package, and a state machine or behavior tree. While these conventional pipelines have shown promising performance, they struggle to make high-level decisions similar to those of humans. To make these kinds of decisions autonomous systems need to be able to both **detect** and **reason** over the semantics of an environment.

Humans have accumulated and shared knowledge for centuries through language, reading, and writing. Recent advancements in deep learning, particularly the transformer architecture [133] and natural language processing have led to powerful new tools in textual processing, large language models (LLMs), or foundation models [35]. These models excel at next-token prediction, e.g., outputting the next most likely character in a sequence. In fact, these models can even pass rigorous exams such as the bar exam. Yet, several key challenges exist to deploy these in robotic systems.

Foremost LLMs operate on text whereas robots are situated in the physical world and rely on both 2D and 3D sensor data. To harness the rich **world knowledge** present in these models, text needs to be grounded, or associated with physical objects and concepts. This work provides two paradigms for this, code generation, and a contextual reasoning framework based on encoding

environmental states as natural language. Another key challenge for physical systems is explainability and verification. It’s one thing to output the wrong chat response but physical actions can have serious consequences. Both of the proposed frameworks in this work utilize human-readable outputs to aid in explainability.

These frameworks build upon a state-of-the-art subterranean exploration system developed as part of the DARPA Subterranean Challenge [28]. The challenge spurred innovation in the areas of autonomy, perception, mobility, and networking in subterranean environments. Teams of robots were required to perform mock search and rescue missions in GPS-denied environments with austere lighting conditions. The solution presented in this work focuses on flexibility between fully autonomous operations and high-level human input. The final software stack was successfully deployed on Boston Dynamics Spot platforms and Clearpath Husky A200 platforms, ultimately placing 3rd overall in the Subterranean Challenge.

1.1 Contributions

The contributions of this work cover a few distinct areas which are all aimed at improving situational awareness for both robots and human teammates. A state-of-the-art framework for exploration in GPS-Denied areas is introduced in Chapter 2. The system is capable of autonomously exploring kilometer-scale subsurface environments with additional performance improvements gained from high-level human input. Further details on the communication system are provided in Chapter 3 which emphasizes fast reconnection times, and data prioritization. The communication backbone is critical for both subterranean navigation and further developments in natural language-based representations which need to be transmitted efficiently.

A key limitation of the system developed for the DARPA Subterranean challenge was the inability of the robot to reason over contextual cues without aid from a human supervisor. When communication was poor the performance of the system was suboptimal due to a reliance on hand-engineered heuristics. Moreover, human supervision doesn’t scale to large fleets of robots due to fanout. This work provides two key frameworks for incorporating the world knowledge present in

foundation models into classical planning frameworks. A method based on code generation from natural language is presented in Chapter 5. The final core contribution is a framework that encodes environmental descriptions, object locations and spatial points into natural language and performs navigation based on any available contextual cues which is presented in Chapter 6. Ultimately this work proposes and validates the following thesis statement:

Integrating natural language encodings of key semantic features in a robot’s environment with classical planning algorithms and foundation models can enable more efficient search and exploration paths by leveraging semantic, geometric, and temporal reasoning.

Where we define efficient paths as the shortest distance traversed by the robot to the target of the navigational instruction.

1.2 Publications

The following publications were a result of work conducted during the course of the doctoral degree:

1.2.1 Journal Articles

- (1) Zhaozhong Chen, **Harel Biggie**, Nisar Ahmed, Simon Julier, and Christoffer Heckman. "Kalman Filter Auto-tuning through Enforcing Chi-Squared Normalized Error Distributions with Bayesian Optimization." *IEEE Transactions on Aerospace and Electronic Systems*, 2024. [Online]. DOI: 10.1109/TAES.2024.3350587
- (2) Kamak Ebadi, Lukas Bernreiter, **Harel Biggie**, Gavin Catt, Yun Chang, Arghya Chatterjee, Christopher E Denniston, Simon-Pierre Deschênes, Kyle Harlow, Shehryar Khattak, et al. "Present and Future of SLAM in Extreme Underground Environments." *IEEE Transactions on Robotics*, 2023. [Online]. DOI: 10.1109/TR0.2023.3323938
- (3) M. Ohradzansky, E. Rush, D. Riley, A. Mills, S. Ahmad, S. McGuire, **Harel Biggie**, K. Harlow, M. Miles, E. Frew, C. Heckman, and J. Humbert. "Multi-Agent Autonomy:

Advancements and Challenges in Subterranean Exploration.” *Field Robotics*, vol. 2, pp. 1068-1104, 2022. [Online]. DOI: 10.55417/fr.2022035

- (4) **Harel Biggie**, E. Rush, D. Riley, S. Ahmad, M. Ohradzansky, K. Harlow, M. Miles, D. Torres, S. McGuire, E. Frew, C. Heckman, and J. Humbert. ”Flexible Supervised Autonomy for Exploration in Subterranean Environments.” *Field Robotics*, vol. 3, pp. 125-189, 2023. [Online]. DOI: 10.55417/fr.2023004
- (5) Michael J Miles, **Harel Biggie**, and Christoffer Heckman. ”Terrain-Aware Semantic Mapping for Cooperative Subterranean Exploration.” *Frontiers in Robotics and AI*, vol. 10, 2023. [Online]. DOI: 10.3389/frobt.2023.1249586

1.2.2 Conference Proceedings

- (1) **Harel Biggie**, Patrick Cooper, Doncey Albin, Kristen Such and Christoffer Heckman. ”CogExplore: Contextual Exploration with Language-Encoded Environment Representations.” In *Conference on Robot Learning*, 2024. [Online]. (Under Review)
- (2) **Harel Biggie**, Ajay Narasimha Mopidevi, Dusty Woods, and Christoffer Heckman. ”Tell Me Where to Go: A Composable Framework for Context-Aware Embodied Robot Navigation.” In *Conference on Robot Learning*, 2023. [Online]. Available: <https://proceedings.mlr.press/v229/biggie23a/biggie23a.pdf>
- (3) **Harel Biggie**, Andrew Beathard, and Christoffer Heckman. ”BO-ICP: Initialization of Iterative Closest Point Based on Bayesian Optimization.” In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023. [Online]. DOI: 10.1109/ICRA48891.2023.10160570
- (4) **Harel Biggie** and Steve McGuire. ”Heterogeneous Ground-Air Autonomous Vehicle Networking in Austere Environments: Practical Implementation of a Mesh Network in the DARPA Subterranean Challenge.” In *2022 18th International Conference on Distributed*

Computing in Sensor Systems (DCOSS), 2022. [Online]. DOI: 10.1109/DCOSS54816.2022.00051

Chapter 2

Flexible Supervised Autonomy For Exploration in the Context of Search and Rescue

2.1 Background and Problem Definition

Despite a myriad of developments in sensing, planning, control, and state estimation over the last few decades, deploying robots in harsh subterranean environments for the purpose of rapid situational awareness presents a number of new challenges to robot autonomy. Traditionally, robots rely on a number of complex, interconnected sub-processes, such as localization, mapping, and planning, to navigate unknown environments. Maintaining accurate state estimates, a process critical to mapping and exploration, is exceptionally challenging in subterranean environments. GPS is unavailable for obtaining position estimates and visual-based localization methods can be affected by varied lighting conditions and environmental factors such as heavy dust, fog, or smoke. Subterranean environments, such as mines and caves, are often unstructured and contain hazardous obstacles, making navigation with ground vehicles challenging.

The DARPA Subterranean Challenge (SubT) [28] aimed to spark new developments in the areas of autonomy, perception, mobility, and networking in subterranean environments. In the following chapter, a scalable multi-agent autonomy solution for subterranean exploration developed by the University of Colorado’s Team MARBLE for the SubT Challenge is presented, along with critical lessons learned and developments made along the way. The system was tasked with exploring and looking for objects that indicated human presence in three categories of subterranean environments: tunnels, urban underground settings, and natural caves.

Specifically, DARPA designed the SubT Challenge to simulate search-and-rescue scenarios in unknown subterranean environments, and consisted of three domain-specific circuit events, Tunnel Circuit, Urban Circuit, and Cave Circuit, followed by the Final Event, which was a combination of three subterranean domains. Teams were challenged with developing robot platforms to deploy in each of the events in search of sets of predefined artifacts, such as backpacks or a Bluetooth signal produced by a cell phone. Correct identifications, consisting of an artifact classification and location to within a 5m sphere of the ground truth location, resulted in a point scored. Placement in the competition was determined by the team which could score the most points over a series of one-hour deployments. For the Final Event, teams were limited to a single “human supervisor” who was able to interact with the systems and visualize any incoming data. Adding to the challenge, teams had a limited window of time in which five team members could set up and initialize robots at the entrance to the course.

2.2 Related Works

Other teams developed impressive solutions to the initial Tunnel and Urban Circuit challenges. Team CSIRO, a collaboration between the Commonwealth Scientific and Industrial Research Organization (CSIRO), Emesent, and Georgia Tech, presents a unique homogeneous sensing solution [51]. In this approach, heterogeneous teams of robots, including both ground and aerial platforms, share sensor information as a part of a decentralized multi-agent SLAM system. Initially, exploration was handled through manual waypoints commanded by the human supervisor, but eventually, an autonomous exploration algorithm was implemented [138]. A common perception module, the CatPack, is used across all ground vehicles for easy reuse of the autonomy stack across different platforms. Similar to Team CSIRO, Team CERBERUS also used a heterogeneous team of ground and aerial platforms [128, 93, 129]. In their approach, map information from different agents is fused into an optimized global map that is shared back to the agents [57]. Similar to the work presented in [87], other teams used graph-based planning approaches for global navigation [25, 26]. Other noteworthy teams and their approaches to autonomous subterranean navigation in-

clude Team CoSTAR [107, 32, 2, 90], CTU-CRAS [105], CMU (Carnegie-Mellon University) [110], and NCTU (National Chiao Tung University) [50]. Additional discussions on the challenges, novel developments, and lessons learned from the Tunnel and Urban circuit events are included in the following works [82, 64]. One common theme common to many of these approaches is the use of heterogeneous teams of agents with multi-modal sensing solutions. By diversifying the sizes, types of locomotion, and sensor modalities of individual robots, teams can be more versatile when faced with varied environments, each with a unique set of challenges. This ability to be flexible and adapt to the needs of the mission is one of Team MARBLE’s driving philosophies.

2.3 Overview of Underground Exploration Approach

This section presents a high-level approach to the flexible autonomy pipeline which will be the foundation for future natural language-based frameworks described in Chapter 6. First, a rationale for the solution is provided followed by the general concept of operation is described, followed by an overview of each of the major components of the developed autonomy solution. A full high-level summary of the autonomy system can be seen in Figure 2.2.

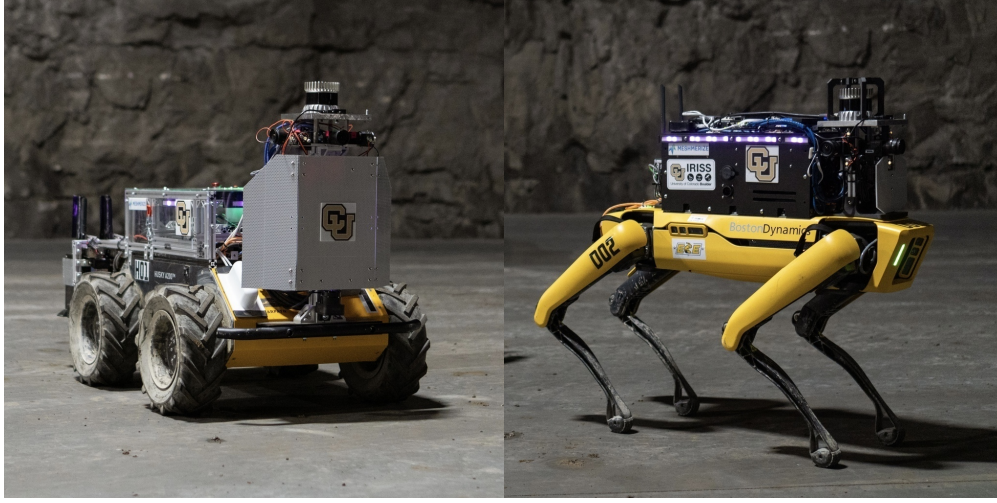
The initial approach to subterranean exploration for the Tunnel and Urban circuit events is presented in [88]. Initially, a graph-based planning and exploration strategy was implemented, the details of which are presented in [87]. This solution is suitable for tunnel-like mines that have mostly planar corridor-junction structures because the environment can be easily represented by a graph of nodes and edges. A scanning lidar was used to center robots in corridors while navigating edge sections as well as avoid obstacles in the environment. However, this approach lacked multi-agent coordination, resulting in significant overlap of explored regions by different agents. For the Urban and Cave circuit events, a three-dimensional volumetric map representation of the environment was generated and used in a frontier-based exploration strategy [4, 5]. In this approach, the exploration rate of the robot is maximized using a frontier-based [140] sampling technique and a fast marching cost-to-go calculation [114] to select goal poses and plan paths to them in three-dimensional space. An artificial potential function-based obstacle avoidance algorithm enables the robot to follow paths

while avoiding small obstacles in the environment. Our initial approach also implemented limited forms of multi-agent coordination in the form of agents sharing goal points and paths.

Figure 2.1: The fleet is composed of two classes of robotic agents: (a) Clearpath Husky A200, (b) Boston Dynamics Spot. Each platform carries a common sensor modular sensor suite.

(a) Clearpath Husky A200.

(b) Boston Dynamics Spot.



2.3.1 Concept of Operations

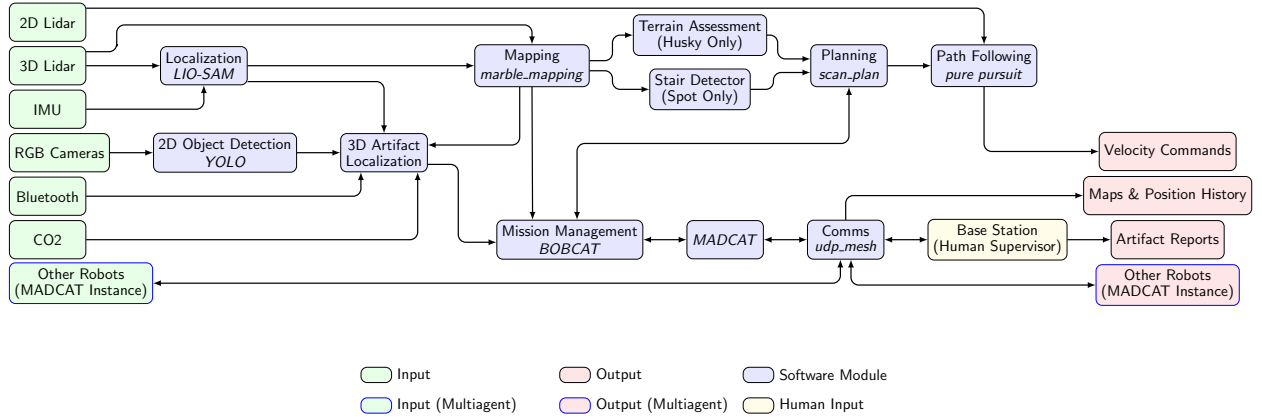
Team MARBLE has emphasized the development of multi-agent autonomy solutions that are able to operate without requiring intervention from a human operator. This aligns with the goals of the DARPA Subterranean Challenge where intermittent or unavailable communications with agents from a base station or between agents is expected. Therefore, our solution is centered around robust single-agent autonomy, where independent robots are able to explore unknown environments and report back to a base station with collected information about the environment including map data and detected artifact locations. In communication-limited environments where information sharing with other agents and the human supervisor may not be available, our agents persist and continue to execute the mission.

While it is important for single, independent agents to be able to explore autonomously, our solution incorporates several multi-agent components to improve exploration efficiency. Our fleet

is also designed to be opportunistic, capitalizing on communication links when they are available to amplify fleet performance. Multi-agent coordination is an auxiliary capability that reduces redundant efforts when agents enter communication range with one another, and inform each other where they have been and where they plan to go next.

Our system’s performance can further be improved when communications are available which enables the human supervisor to have a holistic perspective of the specific search-and-rescue scenario. This holistic perspective empowers the human to make high-level contextual decisions through two types of intervention: directing the agent to a specific location by commanding a high-level waypoint, or teleoperating the agent by commanding low-level velocity signals. During the 60-minute final event prize run, Team MARBLE’s robot fleet was completely autonomous, with the exception of five strategic low-level human supervisor interventions.

Figure 2.2: Overall block diagram showing the high level functionality of the autonomy stack. Inputs are shown in green and outputs are shown in red. Software package names are **italicized** and inputs & outputs which are shared between agents are outlined in blue. Terrain assessment and stair detection both add semantic information to the map but are only run on the Husky and Spot respectively.



2.3.2 Perception

A modular perception suite, shown in Figure 2.5b, was designed as the basis for the autonomy stack. The primary sensor is the Ouster OS1-64 lidar (Light Detection and Ranging), which provides 3D point clouds for mapping, localization, semantic mapping, and obstacle avoidance. A LORD Microstrain 3DM-GX5-15 IMU (Inertial Measurement Unit) is used to measure linear and angular acceleration of the sensor head for use in lidar-inertial state estimation. To identify visual artifacts, the systems are equipped with several FLIR Blackfly PGE-05S2C-CS cameras and an array of 5W dimmable LEDs for self-illumination. The Husky platforms were equipped with four cameras facing forward, backward, to the left, and to the right. The Spot robots had a similar configuration, save omitting the rear camera due to occlusions caused by the custom-built compute and power management system.

2.3.3 Localization

Localization provides consistent pose information for many downstream autonomy processes including volumetric mapping, path planning, artifact detection, and multi-agent coordination. However, ensuring reliable localization is difficult in austere underground environments. Because conventional vision-based solutions can be unreliable due to dark, feature-poor settings, Team MARBLE utilized lidar-based methods and specifically tested and integrated LIO-SAM [118], which has fast online loop closures during long-duration missions.

2.3.4 Exploration

The exploration algorithm generates safe and traversable paths that lead agents toward unexplored areas of previously unseen environments. The developed sampling-based path planning algorithm is designed to be lightweight, so that it can operate on rapid exploration timescales, regardless of the extent of the environment. Computational efficiency is achieved by employing a bifurcated local-global graph for sampling unseen frontiers as well as a **good enough** strategy for final selection. In such time-constrained search and rescue settings, even humans will often make

rapid decisions rather than dwell for long periods of time to make globally optimal ones.

The planner is also constructed to be flexible so that additional capabilities could be scaffolded on top of the core algorithm. Integration of the planning algorithm with semantic mapping was critical for rough terrain and stair traversal. The planning algorithm re-plans in dynamic environments, whether due to doorways that are being opened or closed, or fellow agents that are passing by. This capability is crucial for robust operation in real-world environments, which cannot be assumed to be static.

Agents follow paths via a modified pure pursuit controller [23]. The yaw rate command is computed by comparing the current agent’s heading against a lookahead point that is a fixed distance along the path. Forward speed is regulated based on local proximity to obstacles in the environment and the relative heading error to the look-ahead point. This results in slower speeds when agents are in cluttered environments or experiencing large heading errors. A 2D (RP Lidar) was used for local obstacle avoidance on the Husky platform and the Spot platform had built-in obstacle avoidance.

2.3.5 Mapping

Team MARBLE’s mapping framework is based on the open source **Octomap** package [45], and has been customized with additional capabilities including map merging, transmission of difference maps, and encoding of semantic information. The core of the mapping framework is a log-odds-based probability metric for occupied and unoccupied voxels or cells. These cells provide a 3D representation of the environment that is later used for navigation. This flexible framework enabled the transmission of key environmental features such as rough terrain and the location of stairs efficiently through a bandwidth-limited communication system.

2.3.6 Artifact Detection

The artifact detection system precisely localizes visual artifacts using RGB sensors for visual classification and detection and lidar for depth estimation. Non-visual artifacts such as cellphones

and gas are localized based on the position of the robot. A weighted median filter fuses together all detections which are sent to the human supervisor for final validation as described in Chapter 2.5.

2.3.7 Communication Systems

A mesh networking solution transmits data between robots and back to the base station for the human supervisor to review. Standard 2.4ghz 802.11 wireless radios based on the *ath9k* chipset are used for the physical layer. The wireless radios are embedded in beacons that can be deployed from the back of the Husky platforms, allowing for ad-hoc mesh networks to be established. Meshing technology was provided by Meshmerize [92] and a custom UDP based transport layer (*udp_mesh*) was developed.

2.3.8 Multi-Agent Coordination

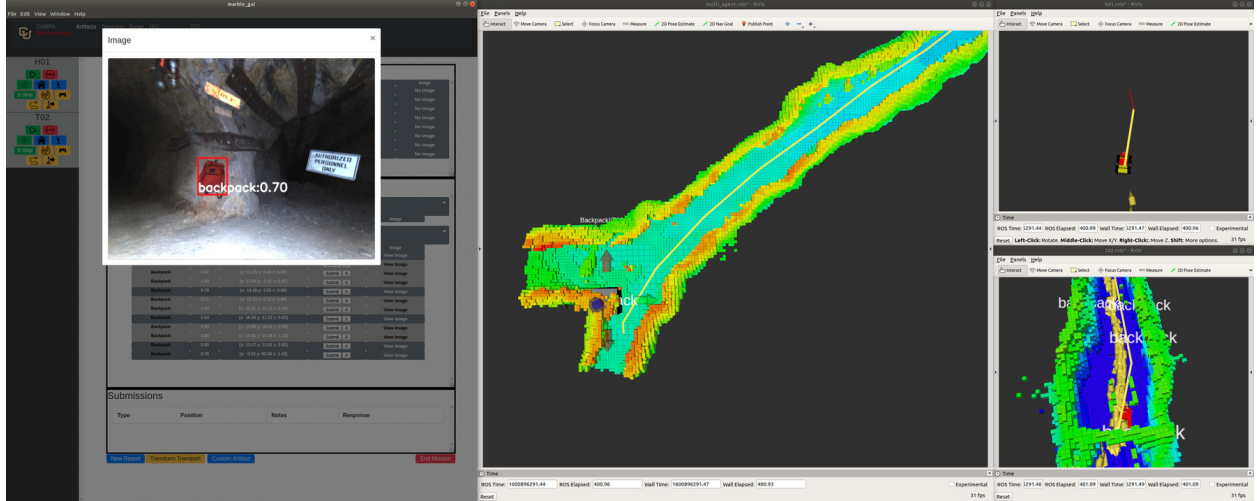
Exploring unknown environments with multiple agents can be made more efficient through coordination, especially when agents are not within communication range. Sharing information across agents, such as explored regions, discovered artifacts, and current behavioral states, allows for more intelligent management of multi-agent exploration. The framework called Multi-Agent Data Collaboration for Autonomous Teams (MADCAT), provides the multi-agent data sharing capabilities required for the Subterranean Challenge mission [103], including the transmission of relevant coordination data and maps, as well as map merging functionality and decision-making for each agent. Additionally, MADCAT implements Behaviors, Objectives and Binary states for Cooperative Autonomous Tasks (BOBCAT), originally presented in [104], for high-level autonomy, decision-making, and interfacing with the human supervisor.

2.3.9 Mission Management

The human supervisor is able to monitor the fleet’s progression through the unknown subterranean environment using a custom GUI operating on a computer at the entrance to the environ-

ment (base station). The current mission status of all agents in the field (‘Reporting’, ‘Exploring’, ‘Home’, etc.), as well as their location in the global map, are displayed whenever robots are within communication range of the base station. Additionally, the goal point and goal path for each robot is visible, allowing the supervisor to see the intent of each robot. The human supervisor is able to take over control of a given agent by either sending manual goal points or tele-operating the vehicle using a joystick interface with the base station. Reported artifacts are displayed per robot, with the type (survivor, cell phone, backpack, helmet, rope), position, confidence, submission result, and an image. The GUI also enables the modification of artifact classes and locations prior to submission to the DARPA scoring server. An example of the GUI interface is shown in Figure 2.3.

Figure 2.3: Example of the Human Supervisor’s interface. The custom GUI is on the left, showing a received artifact image. The middle is the multi-agent RViz view, with all robots and the complete merged map. The right is a third-person follower for each robot (two in this case) with that robot’s original unmerged map.



2.4 Platform Development

For the final event of the SubT Challenge, Team MARBLE deployed a heterogeneous fleet consisting of two Clearpath Husky A200s (H01, H02) and two Boston Dynamics Spots (D01, D02). Examples of each platform are shown in Figure 2.1. The Husky platforms are four-wheeled skid-steer

ground vehicles capable of carrying heavy payloads, while the Spot quadrupedal “dog” platforms are agile, capable of climbing staircases and traversing uneven terrain. The Husky platform is robust and stable, with its generous payload budget allowing it to carry six communication beacons. The intended deployment strategy is to first deploy the Spot platforms to maximize exploration, and then the Husky platforms to establish the mesh communication network.

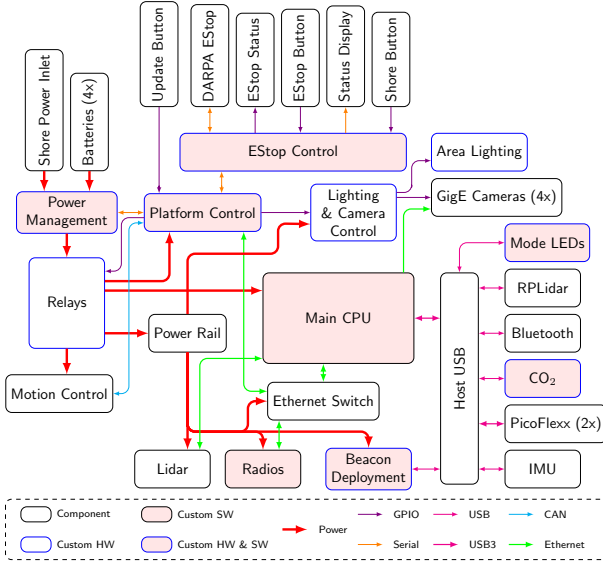
For processing power, each Husky is equipped with a 32-core AMD Ryzen CPU equipped with 128GB of RAM and 4TB of SSD storage, integrated into a complete platform as shown in Figure 2.4a. Dual NVIDIA GTX 1650 GPUs were used to accelerate object detection inference speed. The primary computer on the Spot platforms is an AMD Ryzen 5800U with 64GB of RAM and 2TB of storage which is paired with a Jetson Xavier AGX to process the camera streams and perform artifact detection, following a similar integration as shown in Figure 2.4b. Many purpose-built components are common between platforms to reduce field maintenance efforts and platform-specific code. Each system is outfitted with a custom power system, discussed in Section 2.4.2, which enables the ability to switch from a wired shore power supply to the onboard computer batteries. This leads to more efficient use of the onboard batteries, which are a limiting factor in the duration of field testing deployments.

2.4.1 Communication Beacons

Underground environments provide limited line-of-sight capabilities for wireless communications. As a result, Team MARBLE developed custom communication beacons to complement the custom multi-robot coordination solution. This allows for robots to share information with the base station and other robots in the field. Each Husky platform is capable of carrying six beacons, each containing a single 2W Doodlelabs 802.11n radio as seen in Figure 2.5a. The autonomous beacon deployment mechanism relies on a latching solenoid release coupled with a novel passive system to gently lower each beacon to the ground to ensure maximum antenna height.

Figure 2.4: Power and signal routing diagrams of customized (a) Husky and (b) Spot platforms.

(a) Husky platform power and signal routing diagram.



(b) Spot platform power and signal routing diagram.

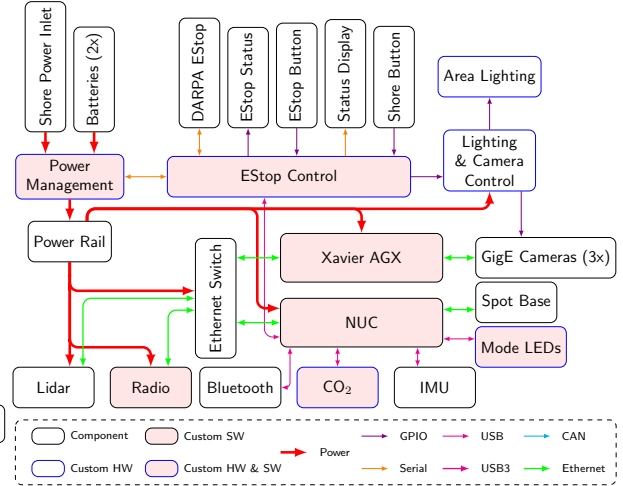
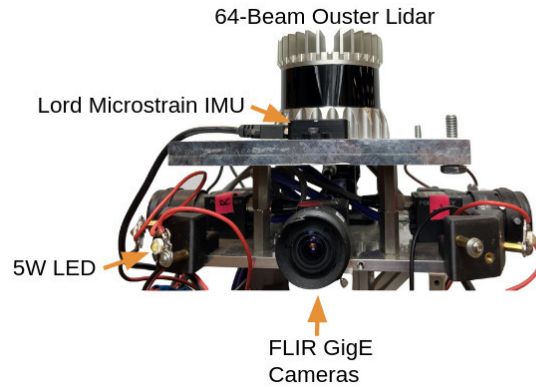
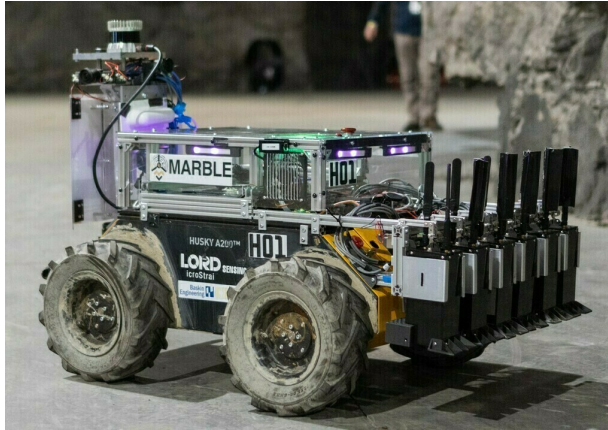


Figure 2.5: Final robot configurations, with (a) a deployment mechanism loaded with six communication beacons on Husky vehicles at Final Event and (b) a modular perception suite installed on both Huskies and Spots.

(a) Deployment mechanism loaded with six communication beacons on Husky vehicles at Final Event. (b) Modular perception suite installed on both Huskies and Spots.



2.4.2 Power and Platform Control Systems

In order to support each platform’s sensor and processing needs, as well as meet DARPA equipment requirements for emergency stop systems, the systems integration efforts relied on several custom hardware and software components. Where feasible, these components are shared between the Husky, shown in Figure 2.1a, and the Spot platforms, shown in Figure 2.1b, reducing the development and validation efforts, as well as team member operational training requirements.

Of the custom capabilities developed, the power management subsystem deserves special mention. This component implements a hardware-interlocked, ideal diode system to permit downstream electronics to source power from either a wall-connected power supply, referred to as shore power, or onboard batteries. By switching to shore power, onboard systems can remain powered for development, testing, and analysis while the batteries are charged without carrying load. Further, the ideal diode component allows the battery packs to load share and charge independently. In contrast to a bus-tied battery system, this ideal diode design prevents high-energy charge equalization between packs and allows each battery pack’s onboard management board to function independently. The system also enables live monitoring of current consumption and battery voltage, as well as intelligent e-stop management which ensures the robot cannot exit the emergency stopped state while connected to shore power.

Emergency stop requirements dictated that each platform needed the ability to be stopped by a physical button, software, and via a DARPA-deployed Xbee network. On the Husky control system, the emergency stop system was integrated directly into the base controller. In contrast, the Spot platform’s emergency stop tied into the available API to issue the robot a “sit” command before terminating power to the motors which allowed the robot to be stopped gracefully.

Several important lessons learned emerged after three years of platform architecture development. We have highlighted the most critical lessons here.

As part of our field testing campaign, we uncovered an issue where our USB-connected IMU was delayed in delivering measurements critical to localization performance. From an integration

perspective, our IMU’s USB interface was implemented using a standard USB Communications Data Class Abstract Control Model (CDC-ACM) interface. Using CDC-ACM for IMU measurements was particularly problematic due to the way in which CDC-ACM uses *bulk* transport. USB has several methods of transferring data from device to host, including *interrupt*, *isochronous*, and *bulk* transport. CDC-ACM uses *bulk* transport, which does not include any guarantee for on-time delivery of data. As a consequence, during high CPU load, IMU measurements were occasionally delayed and resulted in localization errors. In contrast, *interrupt* and *isochronous* transports are regularly serviced and can deliver on-time data. This problem could be solved in future deployments by either replacing the *bulk* interface with an *interrupt* interface or by using legacy serial interfaces such as RS-232 (not typically available on small form factor computing units). However, in practice, we found adjusting the IMU timing was sufficient and did not require engineering new firmware.

Another critical piece for reliable localization and consequently navigation is sensor synchronization. Synchronization is fundamentally necessary in order for our onboard sensors to communicate with their respective computers, and for those computers onboard individual agents to communicate with each other and the base station computer.

2.5 Object Detection

A core component of the SubT Challenge is the detection and localization of objects that could potentially indicate human presence. Each artifact needed to be reported within a 5m radius of the ground truth location. To achieve this requirement a lidar-inertial based state estimator is used. Robots are put into a common reference frame based on survey-grade measurements from a Leica Total Station (LTS) and objects are projected using the marble mapping framework.

2.5.1 Sensing Modalities

Table 2.1 shows the classes of artifacts present at the final event along with the types of sensing modalities capable of detecting each artifact. Each robot in the fleet is equipped with RGB

cameras, Bluetooth modules, and CO₂ sensors which enable the detection of all classes of artifacts using a minimal sensor suite. The visual detection system is not trained to detect either the cell phone, due to its small form factor, or the cube artifact which was detectable using Bluetooth. The cube artifact had rotating colors which pose significant challenges for visual detection methods.

Table 2.1: Sensing modalities, that Team MARBLE utilized (+) and did not utilize (−) for detecting the ten artifact classes. Blank entries indicate sensing modalities that are not useful for detecting specific artifact classes.

	Artifact Class	Visual	Thermal	Wireless	CO ₂
	Survivor	+	−		
	Cell Phone	−		+	
	Backpack	+			
	Drill	+			
	Fire Extinguisher	+			
	Gas				+
	Vent	+	−		
	Helmet	+			
	Rope	+			
	Cube	−		+	

2.5.2 Visual Detection

Visual object detection is a well-researched problem in computer vision and state of the art detectors are capable of identifying objects in both 2D and 3D. Common 2D detectors are typically based on Convolutional Neural Networks (CNN) [148], such as region proposal-based networks like Fast R-CNN [38]. Typically these networks require multiple passes over an image to classify an object and then detect where the object is in the image. In contrast, YOLO [101] performs both classification and detection in a single regression making it a significantly faster detection: 0.5 FPS for Fast R-CNN and 45 FPS for YOLO. Object detectors operating in 3D typically use point clouds obtained from a lidar and until recently were limited to classification rather than full

detection [78, 97]. Extensions to 3D classifiers such as Voxelnet [147] and PointRCNN [120] are capable of performing object detection on powerful GPUs. These GPUs are impractical from both a size and power consumption standpoint for mobile robots. We selected the Yolo V3 [102] model due to the fast and accurate nature of the YOLO [101] family of networks.

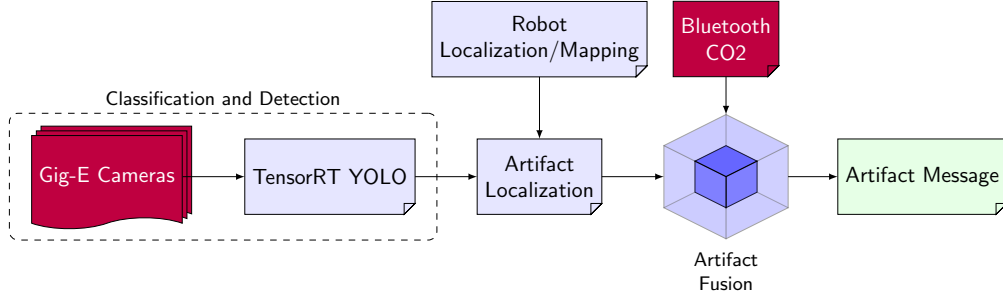
Specifically, for classification and detection, the visual pipeline utilizes a YOLO V3 Tiny [101] model with custom trained weights. The model is optimized for Nvidia TensorRT acceleration and we infer images at a resolution of 608x608. The Husky platforms are able to perform inference at 60FPS on a GTX 1650 based on Nvidia’s Turing architecture with 896 CUDA cores and 112 RT cores. The Spot platform uses a Nvidia Jetson Xavier AGX based on the Volta architecture with 512 CUDA Cores and 64 Tensor cores to perform inference at 40 FPS. These GPUs were chosen to balance performance against size and power constraints for on-board compute. The TensorRT YOLO detector outputs a message containing the detected artifacts as well as the coordinates of their bounding boxes.

A systematic procedure targeted at low-light conditions is used to train the model. At each location, data was collected using three different brightness levels to minimize the impact of lighting conditions on the model’s performance. Specifically, images were taken from past circuit events as well as separate field exercises. Images with excessive motion blur were subsequently filtered out and the data was later augmented with images that contained false positive defections.

Depth registration is performed using **marble_mapping** which is generated by the Ouster 64-beam lidar. Utilizing an Octomap based framework allowed us to avoid implementing any additional filtering due to the probabilistic nature of the map. Additionally, the Octomap structure aggregates scans into the map with temporal memory. This important feature resolves the inconsistency between the 33.2° vertical field of view of the Ouster and the 68° vertical field of view of the cameras. At further distances, the agent is able to incrementally build out regions near ceilings and floors, overcoming the vertical blind spots of the Ouster. Essentially, this temporal memory allows us to decouple the depth measurement from the visual artifact detection. The biggest drawback of this approach is the potential for an additional 0.15m of error on each detection due to the voxel

resolution. However, this error figure still falls within the design constraints of localizing an object to within 5m of its desired location.

Figure 2.6: Overview of the artifact detection system. Sensor inputs are shown in red and outputs are shown in green.



After 3D coordinates are obtained via the Artifact Localization node, we run a weighted median filter in the world coordinate frame to de-noise the projected location within the Artifact Fusion node in 2.6. Each localized artifact is considered to be part of the same measurement if it is the same class as a previous measurement and within 5m of that measurement. We then require five to 10 positive detection events and use the median position as the reported position to the human supervisor. The final detection is published in a custom ROS message which contains this position as well as a compressed version of a corresponding camera image and associated bounding box. The full overview of the artifact system can be seen in Figure 2.6.

2.5.3 Non-Visual Detection

Cell phone, cube, and gas reports are also fused using a weighted median filter. The Bluetooth and CO₂ detections are simply localized to the position of the robot at the time of detection. Bluetooth detections are also grouped together by unique SSIDs and gas detections within 10m of another detection are assumed to have originated from the same source. The final positioning of these non-visual artifacts relies on input from the human supervisor. Our human supervisor interface was designed to easily allow for movement of reported artifacts based on features observed in the map by the human operator. The details regarding the accuracy and success rate of these

reports can be found in Section 2.8.

2.6 Performance of the System

The SubT Final Event Prize Run on September 23, 2021 provided Team MARBLE an excellent opportunity to evaluate the performance of our complete supervised autonomy solution, and we share our results in this section. First, an overview is provided in Section 2.6.1, which includes an outline of the mission objectives, a description of the previously unknown course in Section, as well as a high-level summary of the results. Localization and mapping results are presented in Section 2.6.2. Further analysis of the planner and resulting exploration effort is described in Section 2.7. Artifact detection results are thoroughly analyzed in Section 2.8. Mission management results are detailed in Section 2.9, including the five instances where the human supervisor manual intervened, as well as the seven artifacts that were scored via FPV imagery. Together, this section elucidates how our systems worked together to score 18 artifacts, while also discussing the areas that limited even higher performance.

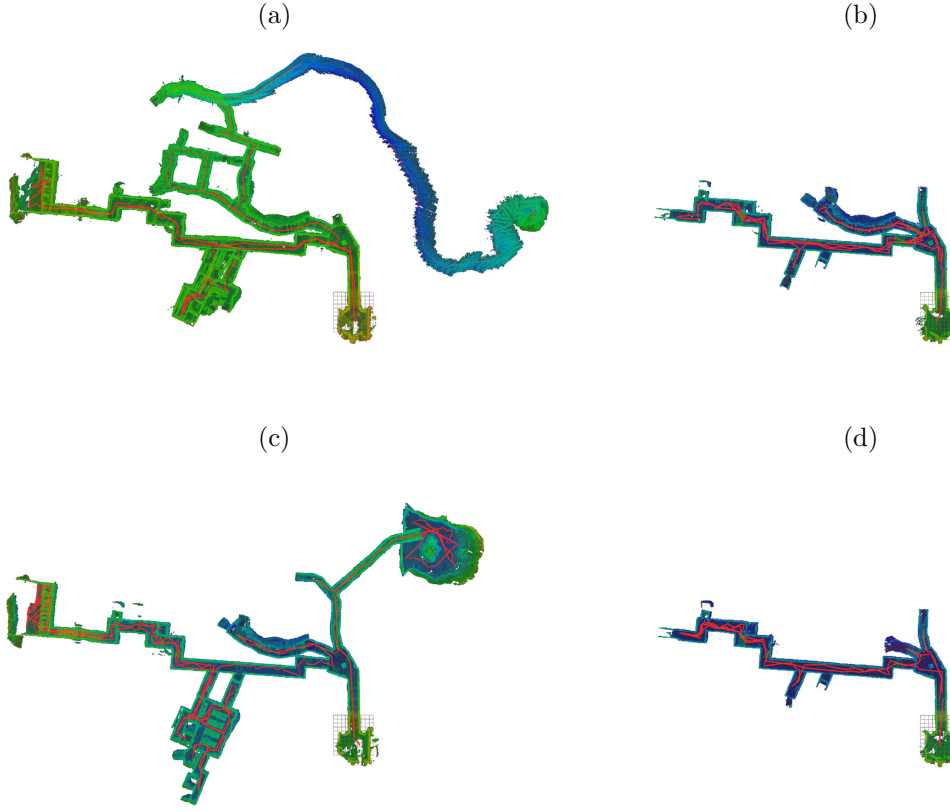
2.6.1 Overview

The mission objectives were to accurately report as many of the 40 artifacts in the course as possible during the mission. There are three hard constraints: the mission is 60 minutes long, there are a total of 40 attempts to report artifacts, and only one human, the human supervisor, is permitted to supervise the mission, manipulate robotic agents, and submit artifact reports.

The final course was custom constructed as illustrated in Figure 2.8, which breaks the course out into distinct tunnel, urban, and cave environments. The course contains numerous hazards and challenges: rough terrain, railroad tracks, slippery surfaces, ramps, stairs, large drop-offs, rocky cliffs, narrow hallways, low-to-the-ground corridors, wide-open caverns, fog, standing water, dynamic obstacles, trap doors, and a degraded communications environment. These challenges are discussed further in Section 2.7.

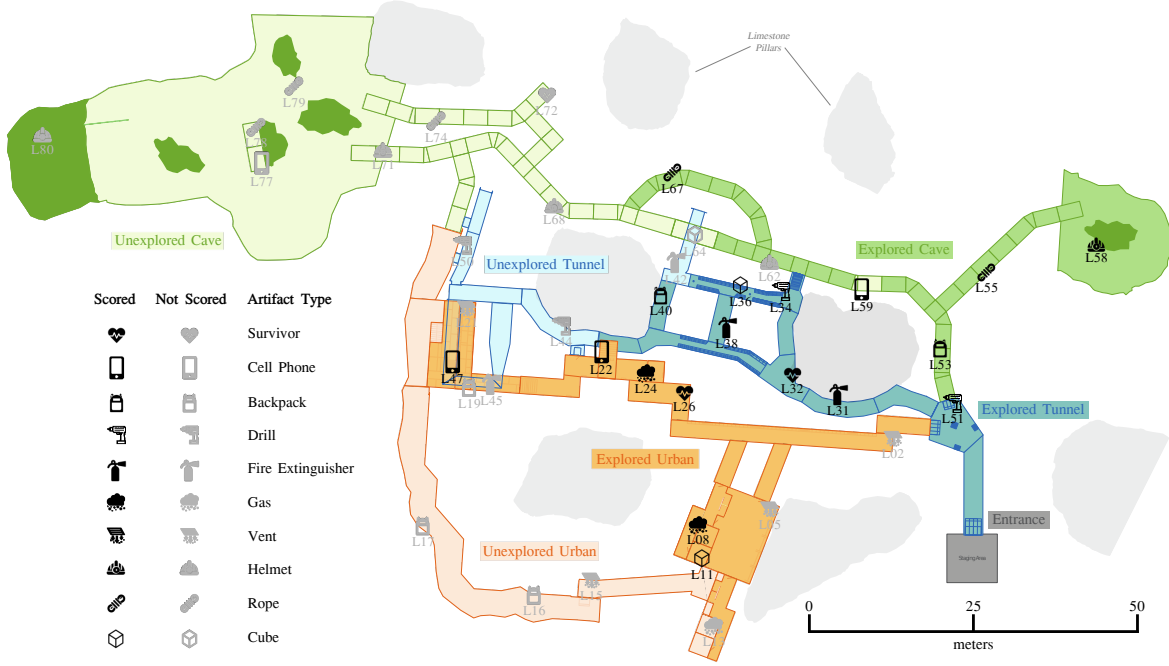
Here, we provide a brief high-level summary of the artifacts scored and the extent of the

Figure 2.7: The final maps from (a) D01, (b) H01, (c) D02, and (d) H02.



environment explored. Team MARBLE scored 18 of the 40 artifacts and explored roughly half of the environment. For reference, the top-scoring team scored 23 artifacts. The location and class of all 40 artifacts can be visualized in the context of the course map shown in Figure 2.8. This map also indicates which regions of the course that Team MARBLE explored as well as the 18 scored artifacts. The artifacts that Team MARBLE scored are also listed in Table 2.2, ordered chronologically from mission start to mission end. Each of the 18 artifacts in Table 2.2 correspond by ID to the scored artifacts in Figure 2.8. Further analysis of artifact detection results are detailed in Section 2.8.

Figure 2.8: Course map of the 60-minute Final Event Prize Run designed by DARPA, highlighting the 18 of 40 artifacts scored by Team MARBLE, along with the areas of the tunnel, urban, and cave sections were explored by agents.



2.6.2 Localization & Mapping

A secondary objective of the 60-minute Final Event Prize Run was to rapidly map the environment and transmit the real-time map back to DARPA every ten seconds. The map takes the form of a point cloud, or a collection of three-dimensional points that represent occupied space in the environment, e.g. floors, walls, ceilings. Figure 2.9 provides a comparison of the DARPA-generated ground truth map in black against the final map submitted by Team MARBLE, split into inliers in green and outliers in red.

DARPA has generously collected, processed, and shared this map data with participating teams [109]. Inliers are defined as points within 1m of ground truth map points, and outliers are defined as points outside 1m. Map coverage is a metric representing the ratio of the environment explored, and is defined as

Table 2.2: List of all artifacts scored by Team MARBLE during the 60-minute Final Event Prize Run, along with the corresponding mission time when artifacts were reported and scored, artifact type, unique DARPA-assigned ID numbers, and Euclidean distance error between the reported and ground truth location of the artifact.

Score	Time [mm:ss]	Type	ID	Error [m]
1	01:08	Drill	L51	0.57
2	01:23	Backpack	L53	2.23
3	06:23	Rope	L55	0.84
4	12:03	Survivor	L26	0.62
5	16:35	Survivor	L32	1.40
6	17:23	Gas	L08	1.80
7	28:08	Fire Extinguisher	L31	1.31
8	35:51	Drill	L34	1.43
9	36:53	Fire Extinguisher	L38	2.82
10	37:08	Cube	L36	3.94
11	37:58	Backpack	L40	1.40
12	38:47	Rope	L67	2.87
13	47:53	Cube	L11	1.83
14	50:33	Cell Phone	L22	4.06
15	50:45	Cell Phone	L47	4.00
16	51:48	Cell Phone	L59	2.15
17	52:45	Gas	L24	2.55
18	56:33	Helmet	L58	1.74

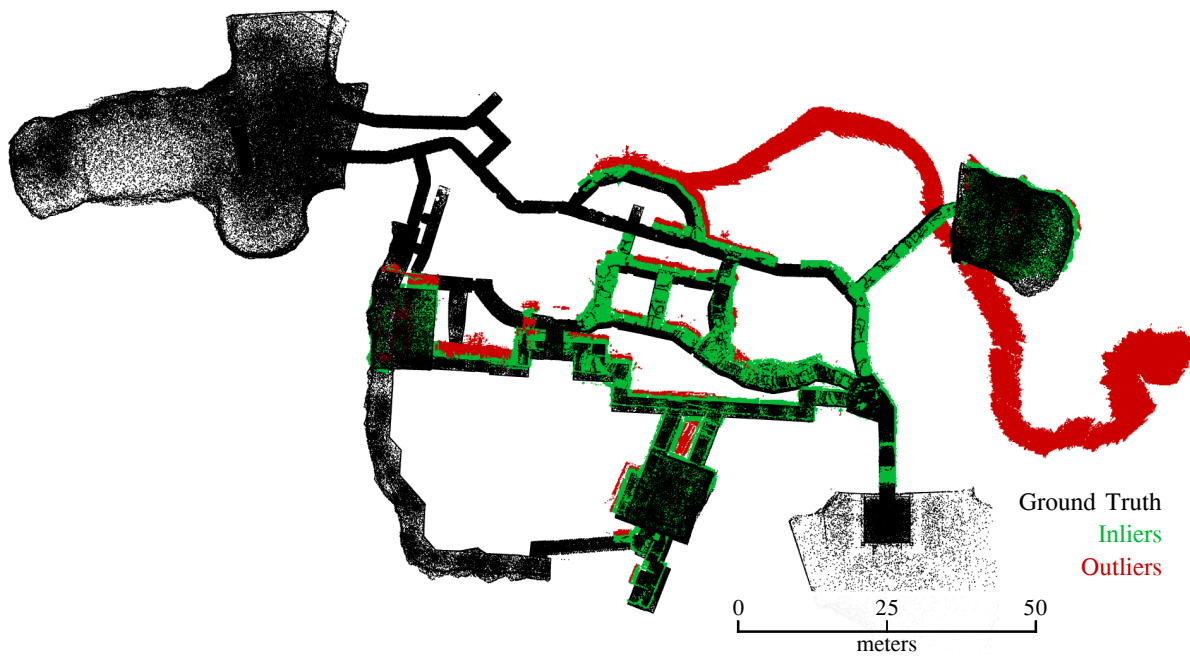
$$\text{map coverage} = \frac{\text{ground truth points within 1m of an inlier point}}{\text{total ground truth points}}. \quad (2.1)$$

Map error or deviation is a metric representing the ratio of the submitted map that is inaccurate relative to the ground truth map, and is defined as

$$\text{map deviation} = \frac{\text{outlier points}}{\text{total submitted points}}. \quad (2.2)$$

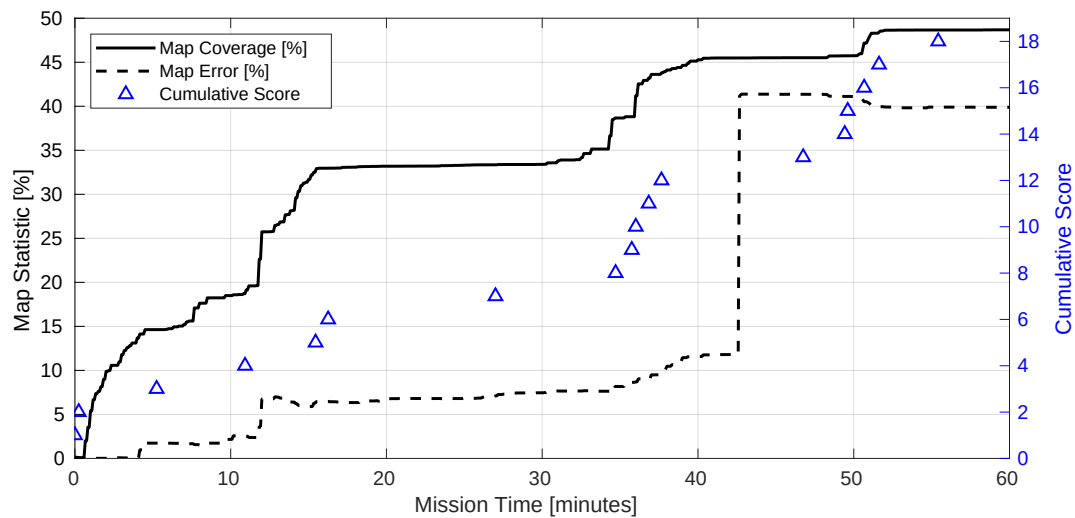
Figure 2.10 shows that map coverage steadily increases throughout the mission, with some periods of rapid exploration, and by the end of the mission, nearly 50% of the environment has been mapped. Map error on the other hand, increases modestly throughout the mission due to localization drift. However, it increases significantly due to a localization failure on D01, which is discussed further in Section 2.7.4. This failure generated erroneous sections of map which mistakenly

Figure 2.9: Final point cloud map submitted by Team MARBLE to DARPA staff during the Final Event Prize Run.



appear as a long winding corridor in Figure 2.9.

Figure 2.10: Map coverage, error, and cumulative score throughout Team MARBLE's deployment during the DARPA SubT Final Event Prize Run.



2.7 Planning

During the entire 60-minute mission and across a diversity of environments, none of the robots were teleoperated due to a planner failure or to improve the volumetric gain, which is seen as a successful demonstration of the planning reliability and flexibility. The human supervisor only intervened to complement autonomy with human-level cognition and intelligence, as discussed further in Section 2.9.

Figure 2.11: Instance of (a) D02 thoroughly exploring the subway platform without planning over the edge, with green edges of the graph, pink planned path, and associated map, along with (b) FPV imagery.

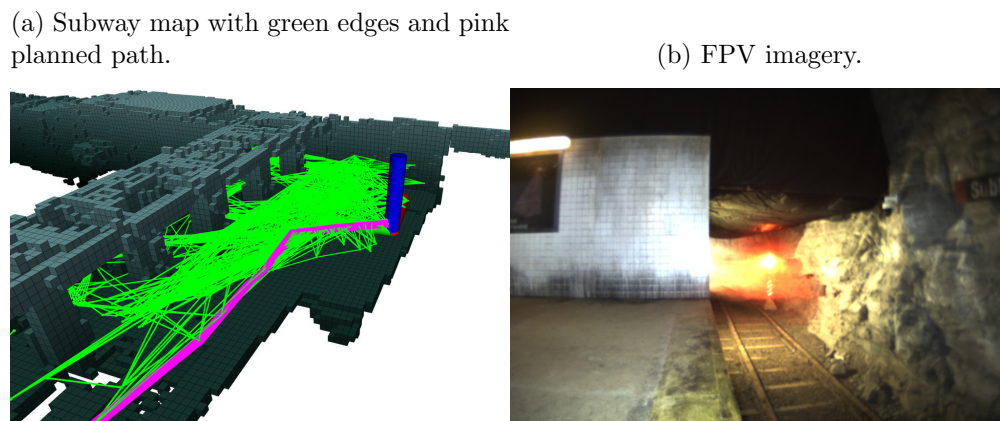
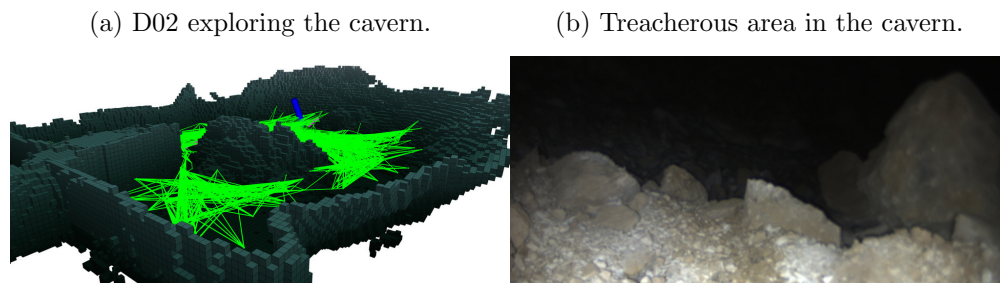


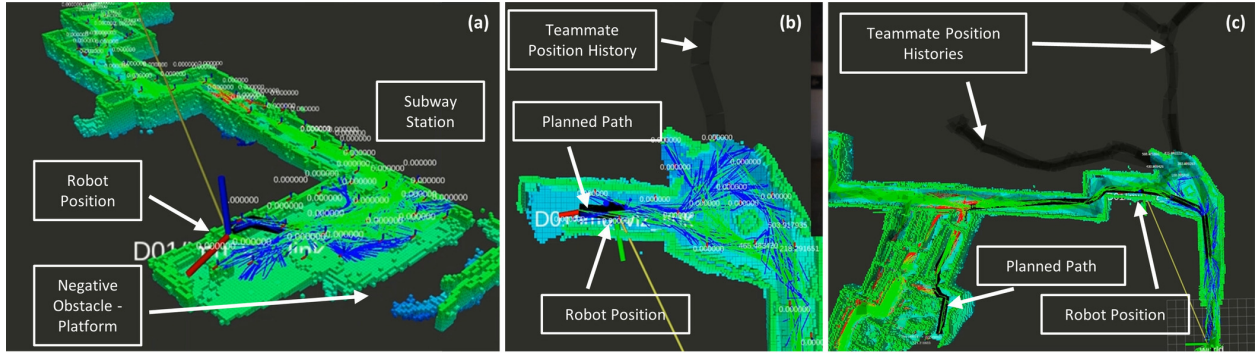
Figure 2.12: Instance of (a) D02 autonomously exploring the entire cavern, traversing the safer surfaces and avoiding treacherous areas, such as the one shown in (b).



As an overview, Figure 2.13 presents three scenarios from the Final Event Prize Run, which demonstrate negative obstacle avoidance, multi-agent coordination based on teammate position

histories, and teleoperation initiation by the human supervisor. Below, Section 2.7.1 highlights the exploration performance, Section 2.7.2 shares examples of agents avoiding treacherous terrain, and Section 2.7.3 demonstrates the planner adapting to dynamic changes in the environment. Section 2.7.4 discusses system limitations that prevented the agents from exploring the entire course.

Figure 2.13: The figure shows three snapshots from the SubT Final Event Prize Round run. The blue lines in the figure represents the locally sampled RRT* tree and the green lines represent the global graph. (a) The robot can be seen to avoid sampling over the negative obstacles i.e., the edge of the subway platform, owing to the settling-based collision-checks. (b) The agent planned away from the position history of a teammate robot that was launched before it, demonstrating effective multi-agent coordination. (c) This snapshot shows an instance where teleoperation was initialized on one of the robots. The robot can be seen to plan a path that had sufficient volumetric gain leading the robot toward a frontier in the urban area that had not been seen by any other robots. However, the human supervisor decided to teleoperate the robot through the initial section of the tunnel area and then let it autonomously explore the tunnel.



2.7.1 Exploration

The volume of unseen area explored by each agent across the mission is illustrated in Figure 2.15. Unlike the statistics of the global map submitted to DARPA in Figure 2.9, Figure 2.15 presents agent-specific exploration information stored onboard each robot. D02 was the largest contributor to overall exploration, partly because it was launched first and had the most time to explore. D01 played a complementary role by exploring most of the tunnel environment, which remained unexplored by D02. Because the subset of the environment safely traversable by the wheel robots was relatively more constrained, it naturally led to less exploration from the Huskies

across the mission.

Figure 2.14: Comprehensive field deployments were conducted at various sites including (a) the Edgar Experimental Mine operated by Colorado School of Mines, located in Idaho Springs, CO. which tested terrain, distance travelled, and communications; (b) the Engineering Center complex also located on University of Colorado Boulder Main Campus which tested multi-story navigation and repeated features from urban environments; and (c) the Folsom Parking Garage located on University of Colorado Boulder Main Campus in Boulder, CO. which tested planning in open spaces and vertical localization across multiple stories.



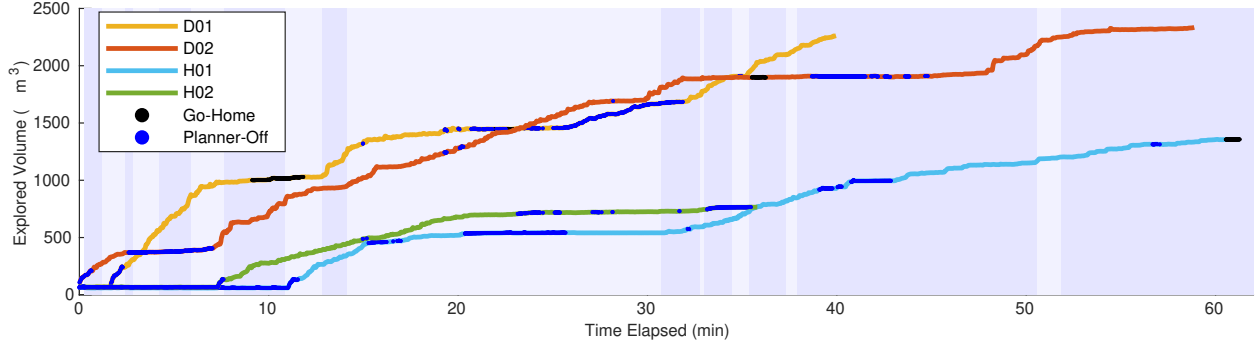
Besides showing the volume explored by each of the robots, Figure 2.15 also shows time periods during which the default planner exploration behavior was paused for higher-level mission management and teleoperation commands. These interjections by the autonomous mission management system were primarily triggered when agents approached each other, and resulted in lower-priority agents pausing and higher-priority agents resuming their task. During these encounters, agents appear as fast-approaching dynamic objects, and this simple procedure was employed rather than incorporating reactive obstacle avoidance into the planner.

2.7.2 Treacherous Terrain

The path planning solution successfully kept each agent safe from collision throughout the entire mission. The Spot robots, which explored more challenging features in the environment, fully avoided negative obstacles, such as shear drop-offs and rocky slopes, and traversed up and down stairs.

The Spot robots successfully traversed up and down the small set of stairs leading up to the subway platform, as shown in Figure 2.16. However, when stairs were first encountered from

Figure 2.15: Volumetric gain explored by each robot across the 60-minute Final Event Prize Run. Planner-Off, denoted by blue lines, represents instances when the planner was paused to allow the autonomous mission-management system to take over when robots are in close proximity, as well as the several interventions when the human supervisor manually teleoperated robots. Go-Home, denoted by black lines, represents instances when the planner began returning home to reconnect to the network and report new artifact reports and map data.



above, agents did not plan down them due to the limited $\pm 16.5^\circ$ vertical field of view of onboard Ouster OS-1-64 lidar sensors, as shown in Figure 2.11a. In addition, the Spots can only safely walk down stairs backwards, and therefore additional logic would be required to autonomously traverse those stairs.

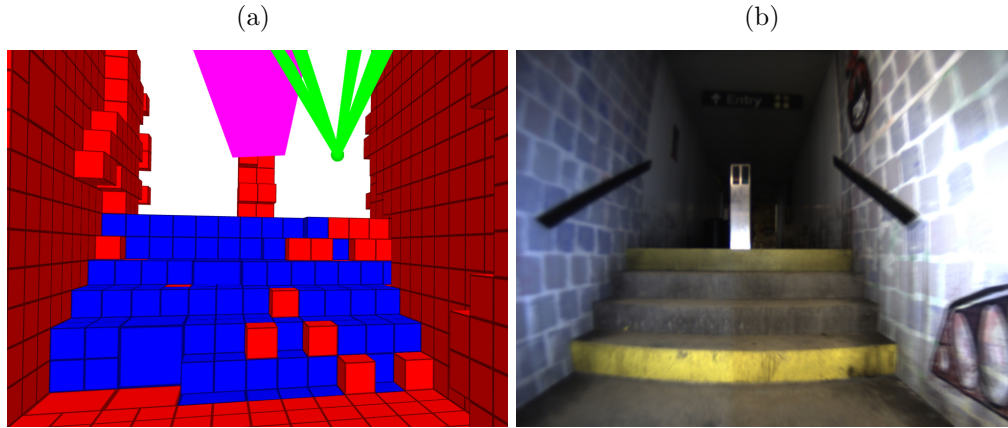
Both Spot robots thoroughly explored the subway platform, approaching the edge, but never stepping and falling over, as demonstrated by D02 in Figure 2.11. Additionally, the same Spot robot, D02, explored the entire cavern autonomously without entering treacherous terrain, as shown in Figure 2.12.

2.7.3 Dynamic Environment

The planner has the ability to adapt to dynamic environments, such as closing or opening of doors, falling rubble, as well as other situations also lead to dynamic changes in the map, including other nearby mobile agents, and localization and mapping error.

During the Final Event Prize Run, D01 traveled through a side branch of the urban environment, triggering a trap door. Figure 2.17 shows the planner adapting to the dynamic environment by re-assigning previously traversable edges as untraversable.

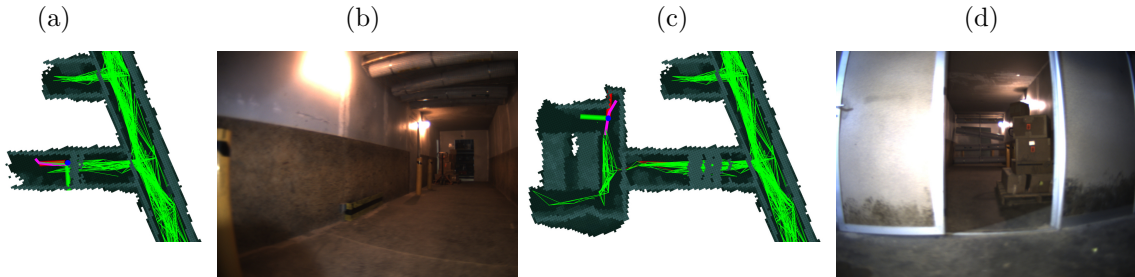
Figure 2.16: Instance of (a) D01 planning up stairs with the green edges of the graph, pink planned path, and associated semantic map with blue voxels representing stairs, along with (b) FPV imagery.



In addition, there were several instances of temporary localization and mapping error, which caused erroneous new map data to change previously traversable edges of the planning graph to untraversable. In each case, the planner adapted to the new scenario, and continuously operated throughout the localization drift and loop closure correction.

When agents pass each other, they appear as fast-moving dynamic obstacles, and cannot re-plan around one another fast enough. Therefore, an agent-based prioritization scheme prevents both collision and deadlock, by enforcing one agent to wait while the other passes.

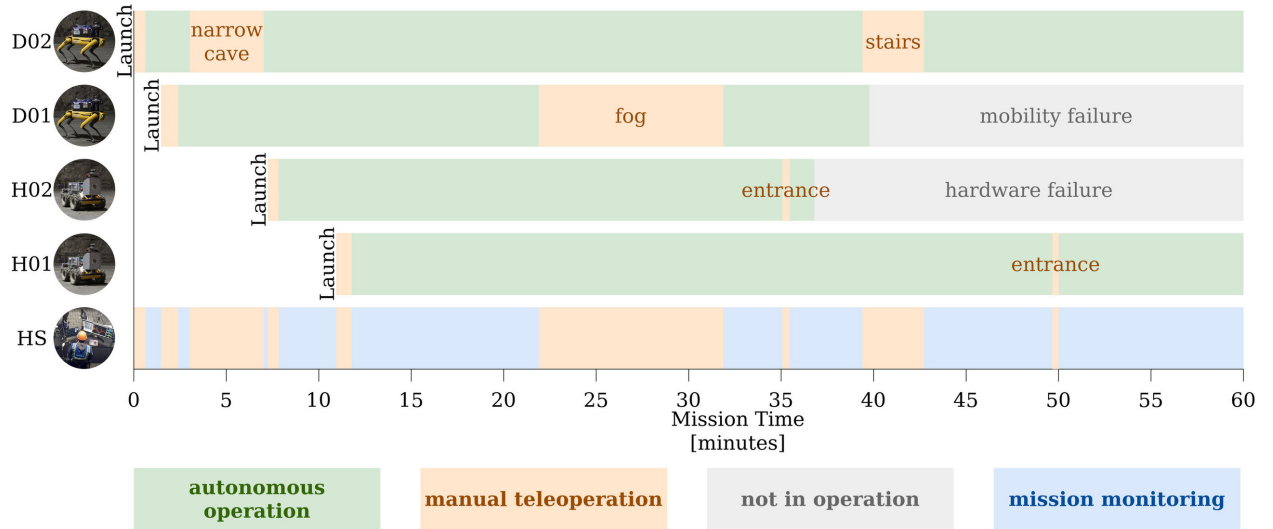
Figure 2.17: Early in the mission, D01 walked by the corridor with the trap door, as shown by (b) the left camera (1:47). The agent later returns to the corridor, (a) walks under the trap door (10:34), and soon after sees it has closed, as shown by (d) the right camera (11:01). After seeing the trap door close, (c) the updated map and graph show (red) edges as untraversable (11:11). Had the agent moved closer to the trap door, and fully been within mapping range, all edges would have updated as untraversable.



2.7.4 Limitations

However, several limitations did prevent agents from exploring roughly half of the course. In total, there were three types of bottlenecks: constrained corridors, slippery surfaces, and downward sets of stairs, each exposing unique limitations within the autonomy system. None of these are limitations of the planner itself, but rather limitations of mapping, mobility, and perception.

Figure 2.18: Mission management timeline for all robots and human supervisor during the 60-minute Final Event Prize Run. When not manually teleoperating a robot, the human supervisor was monitoring the mission, which includes watching live FPV streams from D01 and D02, reviewing incoming artifact reports from agents, reporting artifacts to DARPA, and in the last five minutes of the mission, checking archived FPV images from the Spots for previously missed artifacts.



The planner did not plan through all constrained spaces because the selected planning parameters for agent width and height did not allow the graph to propagate through especially short and narrow spaces. These parameters were intentionally chosen to be conservative to prevent the agent from moving along an unsafe trajectory. Utilizing a higher-resolution local map and planner could result in a more agile robot that could safely traverse those spaces. Additionally, autonomously transitioning into a crouching gait could improve the Spots ability to traverse spaces with low ceilings.

The second limitation is slippery surfaces, and led to D01 slipping and falling in the cave

section. Some of these rocky surfaces were intentionally designed to be slippery, and plenty of humans walking through the course after the event also slipped and fell. After D01 fell, it also experienced a localization failure. Methods to recovery the system from an event such as this one would involve implementing a fall detection algorithm, as well as autonomous self-righting and localization reset logic.

2.8 Artifact Detection

In this section, we present performance results of the artifact detection and reporting system. During the 60-minute Final Event Prize Run, Team MARBLE scored a total of 18 artifacts out of the 40 artifacts in the environment. Figure 2.19 presents a flow diagram that summarizes how our team scored 18 artifacts and the limitations that resulted in the remaining 22 from being scored. Of all 40 artifacts, our agents explored enough of the environment that they were in the vicinity of 25 artifacts, leaving 15 unexplored due to mobility challenges discussed in Section 2.7. Team MARBLE reported 19 of the 25 artifacts that were explored, and successfully scored 18 of those 19 reported. A map of the area explored and scored artifacts is shown in Figure 2.20.

Of the 18 artifacts that Team MARBLE successfully scored, 11 were scored by autonomous robot reports, two were scored by the human supervisor modifying the position of autonomous reports, and five were scored by the human supervisor via robot FPV imagery. One artifact was reported but did not score due to localization error in excess of 5m. There were six artifacts that agents saw, but did not report due to errors in the autonomous artifact detection system. The human supervisor received information regarding three of these six artifacts but missed them due to high workload demands during the mission. The other three artifacts were located in areas that prevented agents from communicating back to the human supervisor.

2.8.1 Visual Detection

The focus of this section is to quantify the performance of the visual artifact detection system. Of the 18 artifacts that Team MARBLE scored, 11 of them were visual artifacts, as shown by

Figure 2.19: Flow diagram illustrating how Team MARBLE scored 18 of the 40 artifacts in the course, and the limitations preventing the remaining 22 artifacts from being scored.

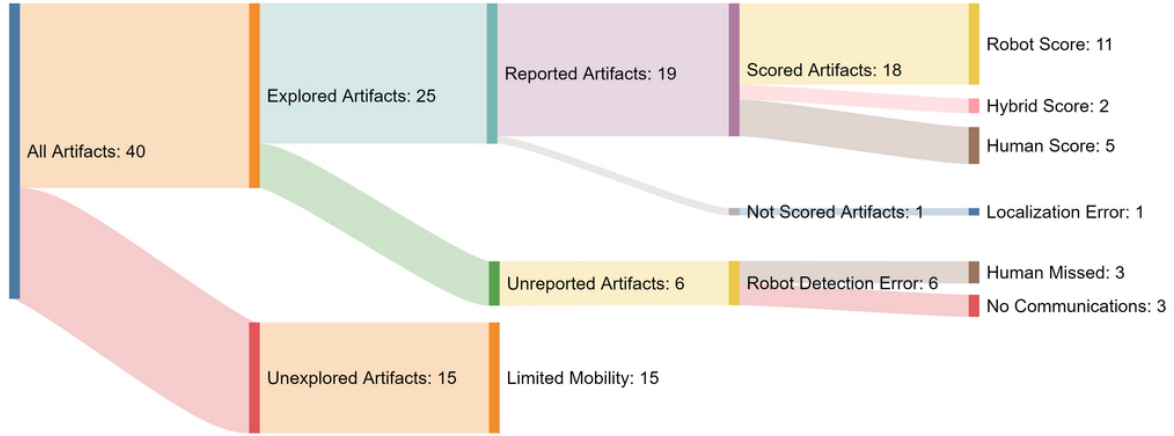


Table 2.3. First, we focus on the six artifacts (L51, L53, L26, L34, L40, L67) that were successfully reported by the agents’ autonomous artifact detection systems in the course. All six artifacts were accurately localized to within 5m. In fact, the largest error for a visual artifact was 2.87m (L67). This eliminated the need for the human supervisor to spend time trying to correctly localize artifacts.

The autonomous artifact detection system filters raw frame-to-frame detections onboard the agent, with the aim to reduce the number of false positive and redundant artifact reports. Because the human supervisor has limited bandwidth, unnecessary distractions detract from the completing other mission-related tasks. In the process of scoring these six visual artifacts, the human supervisor had to process 21 artifact reports from the automated artifact detection systems onboard agents. Of these, 11 were true positives, six were approved by the human supervisor and successfully reported, and five were ignored because they were redundant reports that were previously scored. The other 10 reports were also ignored by the human supervisor because they were false positives.

In total, there were only four false reports that the human supervisor submitted. One was caused by human error, the other three were caused by erroneous CO₂ reports.

Agents in the course failed to autonomously report the other five artifacts (L55, L32, L31,

L38, L58), but did transmit FPV imagery back to the human supervisor, who manually reported and scored them. The fact that the autonomous artifact detection system did not detect five of the 11 visual artifacts it saw, indicates that certain reliability limitations exist. Team MARBLE acknowledged this limitation and relied on the the human supervisor and FPV system to fill in that void, which is further in Section 2.9.

Figure 2.20: Locations of ground truth artifacts in white, reports that scored in green, and reports that did not score in red, overlaid on ground truth map of the course.



2.8.2 Non-Visual Detection

Team MARBLE scored seven non-visual artifacts, i.e. cell phone, cube, and gas, but as shown in Table 2.4, required submitting more reports due to difficulty around accurately localizing the source. The main limitation is that the detection scheme relies on threshold-based logic for RF and CO₂ levels, and when triggered, simply reports the current location of the agent. The thresholds were intentionally set low to increase the probability of detection when agents pass by the vicinity

of these non-visual artifacts. Of the seven non-visual artifacts scored, five of them (L08, L36, L22, L47, L24) were scored via the autonomous robot reports, with an average error of 3.27m. The remaining two artifacts (L11, L59) were scored by the human supervisor manually adjusting the reported artifact locations.

Table 2.3: Artifact statistics for Team MARBLE during the Final Event Prize Run. A total of 18 artifacts were *scored*. Only one artifact was reported but *not scored*, which was due to localization error greater than 5m. Agents were in the vicinity of six artifacts, but they went *unreported* due to autonomous artifact detection failure and in some cases, also missed by the human supervisor due to excess workload. The remaining 15 *unexplored* artifacts were never seen by agents because they were located in parts of the course that were never reached.

Artifact Type	Scored	Not Scored	Unreported	Unexplored	Total
Survivor	2	0	0	1	3
Cell Phone	3	0	0	1	4
Backpack	2	0	0	3	5
Drill	2	0	0	2	4
Fire Extinguisher	2	0	1	1	4
Gas	2	0	1	0	3
Vent	0	0	3	1	4
Helmet	1	0	1	3	5
Rope	2	0	0	3	5
Cube	2	1	0	0	3
Total	18	1	6	15	40

2.9 Mission Management

Overall, the mission management system was able to keep the robots on task with minimal human supervisor intervention. However, when needed, the interventions were crucial for both the exploration capabilities of the system and the final event performance. Figure 2.18 presents a detailed timeline of the four agents in the field, as well as the human supervisor. The four robot launches and five robot interventions were the only times when the human supervisor used teleoperation. There were only two other types of instructions agents received from the human supervisor. One was commanding H02 to drop two communication beacons. The other was commanding D01 to return home three times, each occurring while the human supervisor was also teleoperating D01

Table 2.4: Artifact report statistics for Team MARBLE during the Final Event Prize Run. A total of 34 reports, or attempts, were made throughout the mission. A total of 18 attempts resulted in *scores*, 11 attempts were *misses* and did not result in a score due to localization error greater than 5m, and five attempts were *false* attempts in that they were false positives and no artifact of that class was in the vicinity.

Attempt Type	Scored	Missed	False	Total
Survivor	2	0	0	2
Cell Phone	3	5	0	8
Backpack	2	0	1	3
Drill	2	0	0	2
Fire Extinguisher	2	0	1	3
Gas	2	1	3	6
Vent	0	0	0	0
Helmet	1	0	0	1
Rope	2	0	0	2
Cube	2	5	0	7
Total	18	11	5	34

through the fog.

2.9.1 Robot Launches

The human supervisor was under immense pressure to optimally balance many competing tasks during the 60-minute Final Event Prize Run. With such limited time, the primary objective at the beginning of the mission is to launch robots into the environment as fast as possible. Through extensive practice and full-scale comprehensive field deployments, discussed further in Section 2.10.2, Team MARBLE launched all four robots, with a mean launch time of 41 seconds, as shown in Table 2.5.

Two agents experienced failures late in the mission, reducing overall fleet utilization rate from 92% to 73%. H02 experienced a hardware failure (36:48), and post-event inspection revealed better vibration isolation of the computing system would reduce the likelihood of such a failure in the future. D01 experienced a mobility failure (39:46), in which the Spot slipped on a slick rock and fell over. The agent then experienced a localization instability due to the large induced velocity. To recover from such an incident in the future, Team MARBLE could implement an autonomous

self-righting maneuver and localization reset logic.

Table 2.5: List of the four robot launch (RL) sequences executed by the human supervisor during the 60-minute Final Event Prize Run. Extensive process streamlining and repeated practice deployments resulted in quick and repeatable launch sequences. The course entry column represents the mission time at which the agent crossed into the course.

Launch	Agent	Duration	Teleoperation Window	Course Entry
		[s]	[mm:ss - mm:ss]	[mm:ss]
RL1	D02	75	-00:37 - 00:38	00:02
RL2	D01	55	01:29 - 02:24	01:44
RL3	H02	35	07:15 - 07:50	07:25
RL4	H01	50	10:57 - 11:32	11:08
Mean		41		

Despite the fact that some artifacts were not detected and reported by the autonomous board artifact detection system, the human supervisor filled in the void. The human supervisor reported and scored five artifacts that were seen via robot FPV imagery, but not autonomously detected. Of these, the human supervisor saw four (L55, L32, L31, L38) from live FPV streams, while one (L58) was found while reviewing archived FPV images near the end of the mission.

2.9.2 Robot Interventions

In total, the human supervisor intervened in the autonomous fleet five times during the 60-minute mission. Table 2.6 presents the duration of each intervention, with a mean length of 217 seconds, as well as the reason for intervening. Each of these interventions was in the form of manual teleoperation, in which FPV streams were available at the base station, and velocity commands from the human supervisor was transmitted to the remote agent in the field. Of the five manual teleoperation interventions, only two had significant impact on the mission. The first (RI1) was navigating D02 through the narrow cave corridor early in the mission (3:02 - 7:02), which led the agent to the small cavern and two artifact scores. The second (RI2) was navigating D01 through fog in the tunnel section midway through the mission (21:54 - 31:52), leading to six artifact scores. Figure 2.21 provides a side-by-side comparison of the full-resolution FPV imagery processed onboard by the autonomous artifact detection system and the compressed FPV imagery

transmitted to the human supervisor.

Two interventions (RI3, RI5) commanding agents back into the course, was out of a sheer abundance of caution. The planning algorithm is configured so that the staging area is treated as explored, so agents should not attempt to explore it. One intervention failed (RI4), in which the human supervisor attempted to teleoperate D02 down the stairs by the subway platform. This attempt failed because communication to and from the robot was intermittent.

Figure 2.21: Imagery during robot intervention 2 (RI2). when human supervisor was pushing D01 through the foggy area in the tunnel environment, eventually leading to six additional points. Shown is a comparison of (a) full-resolution FPV imagery onboard the agent and (b) compressed low-resolution FPV imagery transmitted to the human supervisor.



The main takeaways from these results is that our agents are highly autonomous, leaving the human supervisor to focus on mission monitoring and targeting strategic, high-value intervention opportunities. The mission management system enabled convenient transition between autonomy and manual operation, while the communication system enabled visibility and control over the agents in the field.

2.10 Lessons Learned

Underground exploration of previously unknown environments, especially in a time and resource-constrained search-and-rescue context, requires a highly adaptable human-robot team. The lessons learned presented in the following sections enhance our proposed system's flexibility

Table 2.6: List of the five robot interventions (RI) executed by the human supervisor during the 60-minute Final Event Prize Run. Our concept of operations relies on autonomous multi-agent exploration and does not necessitate manual waypoints or teleoperation from the human supervisor. Therefore, agents were completely autonomous, except for the human supervisor’s input during these five instances of teleoperation. The intervention goals varied but were all specific scenarios where human intervention would augment autonomous agent capabilities in a mission-relevant manner.

Intervention	Agent	Duration [s]	Window [mm:ss]	Goal	Success	Points
RI1	D02	240	03:02 - 07:02	Enter narrow cave corridor	+	6
RI2	D01	598	21:54 - 31:52	Enter foggy tunnel area	+	2
RI3	H02	24	35:04 - 35:28	Avoid course exit	+	0
RI4	D02	200	39:24 - 42:44	Walk down stairs	–	0
RI5	H01	21	49:40 - 50:01	Avoid course exit	+	0
Mean		217				1.6

across mobility, communications, human-robot teaming, and multi-agent coordination.

2.10.1 Platform Mobility

Systems with heterogeneous platforms allow for specialization by each platform for both specific environments and roles which benefits the entire mission. Specifically, in Team MARBLE’s case, the addition of Spot platforms into an exploration role enabled rapid multi-story expeditions. This capability was further augmented with the utility of higher-payload, wheeled Huskies, carrying communication beacons. When deployed, these beacons allowed the Spot platforms to report artifacts without the need to traverse “home,” leading to much more efficient exploration. No single, platform is as performant as the combination of platforms operating in different roles based on each one of their strengths.

2.10.2 Testing and Validation

Significant testing and validation was conducted for every element of the Team MARBLE system. These tests taken over a variety of environments, shown in Figure 2.14, reduced mission-to-mission variability, and increased system-wide adaptability. Through these tests, Team MARBLE locked in well tested solutions with minimal unexplained errors. This reduced changes on the

system as we approached the final competition, given that new solutions had to be verified against similarly stringent tests. To our knowledge our team made the fewest hardware and software based adjustments from the preliminary runs to the final run to achieve our score, instead relying on our testing having eliminated all significant errors outside of a so-called "Poisson distributed fatal error." These types of errors were mission ending to any individual robot but difficult to predict or adapt to in a meaningful sense. These can be seen in our final competition run where D01 fell over on rough steep terrain, and where H02 suffered a hardware error.

A secondary goal of these full-scale deployments was to reduce human-based variability in performance. They helped both the human supervisor and pit crew prepare for the stressors of operating and responding quickly to complex interactions between robots, the environment, and potential failure modes. At no point was a single test considered sufficient for validating a solution as sufficient; instead all solutions released demonstrated repeatability across the same and varied environments.

2.10.3 System Adaptability

The challenges posed by operating a system in an unknown environment necessitate a high level of system adaptability. Predicting every capability that a system will need for a given mission, such as search and rescue, is impractical. Instead, having a highly flexible system capable of adapting to unknown situations is crucial. Having already addressed the mobility considerations in Section 2.10.1, we also found software adaptability key to our success. For instance, the flexible communication network enabled the ability to pass on FPV to the human supervisor with a simple configuration change. While our system was designed with autonomy in mind and the capability was not previously planned for, the adaptation proved invaluable. Minimal human supervision directly impacted the final score and exploration capabilities of the system as a whole.

The notion of adaptability extends to other software architecture as well. The artifact detection system was adjusted during the final competition to include SSID information for Bluetooth artifacts. These artifacts could then be more accurately merged between agents and by the human

supervisor, despite inaccurate positioning from wireless signals.

In some cases, adaptability is explicitly accounted for in our design, most notably in the mission management system. BOBCAT has many parameters which allowed the human supervisor to adjust exploration activity, including how long a robot can explore before reporting detected artifacts, whether a robot should find multiple artifacts before reporting, and how the system should adjust to time constraints in the mission.

2.10.4 Autonomy and Human Robot Interaction

Team MARBLE emphasized autonomous system design as our primary goal. Since only one person, the human supervisor, was permitted to interact with agents while they were deployed, having robots controlling their own paths and decision making was key to reducing the cognitive load required to manage mission objectives. Despite the focus on autonomy, the human supervisor is still necessary to maximize system performance. As Team MARBLE approached the final competition, we found targeted areas where direct human supervisor control improved results. This influenced decisions regarding the number of deployed platforms, how artifacts were passed from agents to the human supervisor, and how we recovered from anomalous robot behavior.

To maximize the human supervisor's ability to track the data streams present across the fleet, only four robots were deployed. While a larger fleet might have enabled a rapid exploration of the environment, it could reduce the human supervisor's ability to meaningfully address problems arising from any specific robot.

The artifact detection system was designed to filter against false positives before passing information to the human supervisor. Artifacts had to consistently be detected in multiple frames, and be a sufficient distance from existing artifact estimates. Where possible, artifacts were returned with additional information including SSID for Bluetooth artifacts and images for visual artifacts. This data allowed the human supervisor to sort remaining false positives quickly without being distracting from the core mission objectives.

Specifically, the human supervisor spent most of their operational load solving problems the

robots were incapable of correcting through their autonomy stack. For example, during the Final Event Prize Run, the human supervisor used the first person vision to navigate through dense fog, which the robot was incapable of planning through autonomously. After this intervention, the human supervisor allowed the robot to return to autonomous operations, where it explored several new areas, and reported a total of six new artifacts. The human supervisor was able to opportunistically intervene like this, only because the other agents were operating autonomously without supervision.

2.10.5 Retrospective

It's important, after a three-year effort of this size and scope, to examine some of the bigger picture questions. What would we do differently next time? What did we wish we knew at the start, that we know now? The answer to the first two questions is that we would have spent more time at the onset to scope out short-term and long-term development goals. We excelled at the Tunnel Event, placing fourth amongst a large group of competitors. This was mostly due to meeting well-scoped short-term goals within the one year cycle. However, during the six-month development period for the Urban Event, we focused on long-term goals that we ultimately only partially validated, leading to a disappointing performance. In hindsight, we should have focused on making our already capable platforms more capable, rather than spreading our resources thin across many thrusts. However, what our team excelled at after this experience, was pivoting and adopting two new strategies. First, we leaned into student-led project management, which led to greater team coordination, a key component of rapid and effective system development. Secondly, we focused on long-term development goals, and given a year and a half, had the time to adequately develop, test, and validate each subsystem, each fully autonomous robot, as well as our entire fleet in numerous search and rescue missions. Together, these two powerful changes allowed a small, lean team, to produce a highly functional autonomy solution that could perform under pressure.

2.11 Conclusions

In this chapter, we showcase a flexible autonomy solution for exploring unknown subterranean environments. The highly performant autonomy solution directly led to a third-place finish at the DARPA SubT Final Event which was focused on search and rescue. Moreover, the specific innovations presented in graph planning, flexible communications, and mission management are directly applicable to other multi-robot teaming applications under limited human supervision.

Specifically, the combination of legged and wheeled robots allowed for heterogeneous teaming enabling both rapid exploration and a robust communication network. The deployed mesh network enables flexible configuration and prioritization of data sent both to other robots, and a human supervisor for review. A powerful graph planning framework paired with semantically encoded Octomaps enabled safe, rapid exploration. The final system was able to explore a variety of underground environments, including gold mines and subway stations, with minimal human input.

Human input was reserved for specific situations where higher-level reasoning had the potential to improve the mission. The flexible mission management system described enables safe transitions between human input and the underlying autonomy system. One of the most important lessons learned from the developed system is that focusing on autonomy is core for human-robot teaming. Robots need to be able to make decisions on their own, enabling the limited human resources to only act in critical situations. Subsequent chapters will explore the integration of higher-level semantic reasoning, leveraging natural language-based encodings, into the navigation stack established here. This aims to minimize the need for human decision-making.

Chapter 3

Communication Systems

3.1 Introduction

Practical field robotics exploration experiments often require a wireless communications backhaul in austere conditions, where there is no existing infrastructure. Such communication backbones are also necessary to utilize large language models (LLMs) [35] which typically run in the cloud. The framework described here is utilized in Chapters 5 and 6 for semantic-driven search and exploration.

In underground scenarios, a mesh network, self-deployed as part of the exploration task, becomes an ideal solution for extending comm network coverage deep into unknown environments. The proposed solution is based on three DARPA events which were held in a research coal mine, an abandoned nuclear power plant, and a former limestone mine; the use of environments dominated by solid rock and/or concrete presented formidable difficulties for RF communication. This chapter presents the development of our network solution, encompassing physical, logical, and transport layers, validated in testing and by our third-place finish at the final event; we emphasize our novel contribution, a transport layer called *udp_mesh*. Additionally, our modular pipeline enables the networking solution to run on a variety of commercially available hardware, in compact form factors making it an ideal choice for UGV and UAV operations alike. Our system has application in any field robotics context (for example, search and rescue or exploration) using mesh infrastructure, with a wide range of implementation options.

We first present the system requirements as either specified by DARPA or determined through

design, then introduce several examples of existing systems that partially fulfill those requirements to motivate the development of our own system. We then describe our complete infrastructure, including several lessons learned through design iterations over the course of the Challenge. Finally, we present a detailed analysis of the performance of our prize-winning system at the final event, with potential directions for future work.

3.2 Problem Requirements

Our system design had to fulfill a multitude of requirements, some of which were specified to our team from DARPA as part of the competition, while others were practical implications dictated by our team composition and infrastructure.

As part of the competition setup, DARPA required any RF system to operate within FCC limitations; while the use of licensed bands was not forbidden, we quickly made the determination that the use of ISM bands reduced the administrative burdens needed to test our system in the field. Further, all data products produced by our exploration system had to be funneled through a single wired Ethernet port co-located with a human supervisor monitoring the robots at the base station located outside of the exploration area. Finally, DARPA did not guarantee any kind of spectrum conditions or interference metrics.

From our team’s perspective, the required elements of our solution included SWaP-C (size, weight, power, and cost) constraints on hardware, as well as software requirements for integration. Our network infrastructure had to be available in sufficient form factor options to permit deployment on aerial robots and various sizes of ground robots. Further, the cost per node was limited to approximately \$1k, with an emphasis on cheaper hardware. From the software perspective, the communication system had to support a ROS1 (simplified as ROS) [98] environment running on Linux; we needed to support ROS natively to reduce needed integration efforts and enable testing of higher-level functions in simulation.

From a systems design perspective, we required our solution to have minimal runtime configuration requirements, node discovery, online/offline detection, guaranteed transmission, and traffic

prioritization. We desired fast reconnect times but were willing to trade off with decreased throughput if needed.

3.3 Background

Previous work has developed several solutions to common problems encountered with deploying mesh networks. As details about vendor-specific implementations are often unavailable, we detail several open-source or readily available commercial options that were considered as our system design proceeded.

3.3.1 Physical Layer

Physically, utilizing a standard 802.11 waveform and PHY has significant advantages in that there are many chipset vendors to draw from. We desired a chipset with well-understood Linux support, potentially yielding on the latest features such as high-density MIMO connectivity. Using a standard 802.11 waveform on an unlicensed frequency enables the unregulated use of this system worldwide, potentially also supporting integration with non-mesh networks using layers such as *mac80211* on Linux. As 802.11 is ubiquitous, our system could be used in a wide range of applications and mobile vehicles, including search and rescue, agriculture, and self-deploying mesh networks.

3.3.2 Logical Layer

Meshing layers lay between the physical transmission of frames over the medium and a higher-level protocol such as IP. For applications involving rapid changes to mesh topology, a responsive mesh layer that minimizes lost link time is a major requirement. Further, to reduce integration effort, a mesh layer that operates at layer 2 of an OSI stack¹ is desirable to allow transparent use of higher-level protocols such as ARP and IP. Typically, meshing algorithms such as OLSR [22] and AODV [95] select a single best path for routing between nodes which hinders algorithmic

¹ https://en.wikipedia.org/wiki/OSI_model

performance in dynamic environments. A more recent example of a single-path logical meshing layer is Better Approach to Mobile Ad-hoc Networking-Advanced (*batman-adv*) [112], an open source implementation of a layer 2 mesh. In contrast to *batman-adv*, *meshmerize* [92] provides multiple paths between nodes to ensure a reliable connection while still operating at layer 2; these multiple paths allow for a dramatic decrease in reconnect times when mesh topology changes. Other options for meshing layers such as the original implementation of *batman* operate at layer 3 as packet routers; a downside of this class of mesh layers is a more complicated IP addressing scheme needed over the entire mesh.

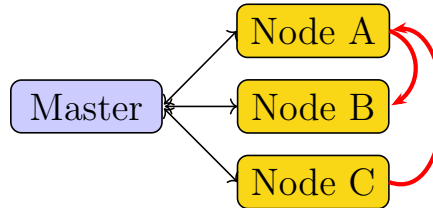
3.3.3 Transport Layer

Many robotic systems utilize middleware layers based on the publisher and subscriber model to coordinate activities of different parts of the system; these middlewares are designed to enable both inter- and intra-compute node connectivity. In general, a middleware layer needs to provide discovery, data encapsulation, and transport services. In many applications, the concept of *prioritization* can also be modeled to reserve access to a limited communication channel and ensure high-priority data (such as an emergency stop request) is given preferred treatment. Contemporary examples include ROS[98], OROCOS[16], and YARP[91]. While we use ROS as a case study of a publish/subscribe architecture to motivate the requirements of a transport layer, any architecture that utilizes a centralized directory-like service is likely to face similar limitations.

In a traditional networked ROS architecture, a single computer runs a special node known as the *rosmaster* that coordinates the publish-subscribe mechanisms. When a node wishes to exchange data with another node via named *topics*, the master is consulted to determine the computer to connect to, as in Figure 3.1. A single *rosmaster* serves as a central directory of nodes and topics; when a subscription to a topic is requested, a list of publisher nodes is returned so that point-to-point TCP connections can be made directly between publisher and subscriber. When computers are connected via always-on, high-speed links such as wired Ethernet, this system works quite well. However, when computers are connected by an unreliable mesh network, this single-master,

TCP-dependent model breaks down; if the master suddenly becomes unavailable, dependent nodes become unreachable.

Figure 3.1: A basic single-master ROS network node graph. Red lines indicate data transfer, whereas black lines indicate directory management.



To mitigate the risks of a single-master architecture, we used a multi-master architecture in which each robot (and human interface) ran a single master instance (as in Figure 3.2). Within the local network of the robot, the master was always available through a high-speed wired link, resulting in stable local operations. However, additional mechanisms are needed to enable messages to be passed between masters.

One reference implementation is *multimaster_fkie* [126], which enables nodes connected to different ROS masters to communicate with one another. *multimaster_fkie* uses broadcasts to discover other instances on the local Ethernet segment; *multimaster_fkie*² links independent masters together by cross-publishing their topic and node directories so that TCP connections can be made across *rosmaster* boundaries. Functionally, when a local node wishes to subscribe to a remote master’s topic, the local master now contains the remote connectivity information; the node then establishes a standard TCP link between nodes and messages can flow.

While *multimaster_fkie* solves the discovery and advertisement problems, it does nothing to establish prioritization of data flow. With the standard TCP transport provided by ROS, there is no centralized means of monitoring inter-node connections to arbitrate data priorities. We are aware of at least two alternatives that provide this critical quality-of-service prioritization within a ROS environment by channelling all inter-robot traffic through a single monitoring point to enforce data priorities.

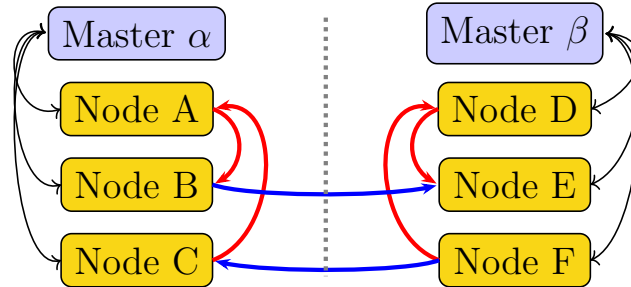
² http://wiki.ros.org/multimaster_fkie

The first, *Pound*³ [123], is specifically designed for use in unreliable mesh networks and implements many of the desired requirements. However, *Pound* relies on hardcoded topic names and fixed addressing information, which were critical requirements for our system for ease of testing and integration.

Alternatively, *nimbro_network* [111] implements a similar set of functions with regards to transport over wireless networks, but omits prioritization. Crucially, *nimbro_network* still utilizes TCP for reliable inter-robot communication, preventing adaptation of core TCP behavior (particularly retransmits) to unreliable mesh networks; UDP links are only used for non-guaranteed data delivery.

We sought to design a transport layer for use in a ROS environment that allowed for runtime reconfiguration, implemented prioritization, and re-implemented reliable communication over UDP to have better control over retransmits and fragmentation.

Figure 3.2: A multi-master ROS network node graph. Red lines indicate data transfer, where black lines indicate directory management. Blue lines are data paths that cross network segments.



3.4 Methods

3.4.1 Physical

We utilized *ath9k*-supported commercially available 802.11 hardware based on Atheros (and subsequently Qualcomm) chipsets capable of 2x2 multiple-input, multiple-output (MIMO) transmission rates across our entire robot fleet, base stations, and comm beacons. Due to the widespread

³ <https://github.com/dantard/unizar-pound-ros-pkg>

industry availability of *ath9k* supported hardware using OpenWRT⁴, we had the freedom to use high-power 2W radios in our deployable comm beacons, 1W radios in our robots, and very low power radios for testing and validation exercises without having to change any underlying code or settings. Newer chipsets supporting 3x3 MIMO were evaluated, but found to not have the firmware stability in the ‘ad-hoc’ mode used by higher mesh networking layers. Each of our Husky ground robots utilized a pair of radios, each with a 2x2 sector antenna pointing fore and aft, while our Spot ground robots utilized a pair of omnidirectional antennas on a single radio.

3.4.2 Logical

We partnered with Meshmerize GmbH⁵ to improve upon the *batman-adv* layer 2 open-source mesh networking solution, as well as provide subject-matter expertise. In our preliminary evaluations, we determined that on average *batman-adv* was able to re-establish connection within 10s while *meshmerize* was able to do so in 1s by prioritizing connectivity over optimal routing. This fundamentally different architecture enables a data ferrying paradigm; robots need only have brief windows of connectivity to have a functional exploration and reporting strategy. However, *meshmerize*’s throughput is limited to approx 10Mbit/s due to CPU performance limitations on radio system-on-chip processors. *meshmerize* is capable of running on all *ath9k*-supported hardware using an OpenWRT integration. With *meshmerize* functioning at layer 2 bridged to each robot’s internal network, all IP protocols were enabled; being able to transparently use SSH without any additional software was a significant aid in debugging and troubleshooting.

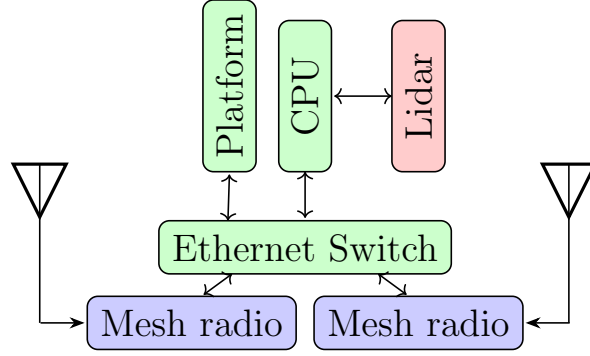
3.4.3 Transport

The main innovation in our system is our transport layer, *udp_mesh*. We designed *udp_mesh* to replace *fkie_multimaster* to address the need to be able to prioritize message transmissions and provide for better performance over intermittently connected networks. As implemented, our ROS interconnect layer now provides the following services: discovery, address resolution,

⁴ <https://openwrt.org/>

⁵ <https://www.meshmerize.net/>

Figure 3.3: On-robot network architecture for the Husky platform. Dual mesh radios are bridged to provide mesh access for onboard compute and motion control. Spot and UAV platforms similar.



ROS message encapsulation, point-to-point transport, point-to-multipoint transport, and quality-of-service prioritization.

Each of these services is utilized to support the higher-level communications functions needed for mission success. Fundamentally, the *udp_mesh* layer uses only unicast and broadcast UDP datagrams to implement higher-level services without requiring multicast support. In principle, multicasting would offer a performance benefit by reducing broadcast traffic. However, in a wireless mesh environment, these potential gains are offset by multicast group membership management overhead.

3.4.4 Discovery and Address Resolution

Discovery in our layer is the process of identifying nodes that are available for communication. We implement discovery through the use of a periodic heartbeat broadcast that advertises the node's availability and provides name resolution information. In concept, this service is similar to the *mcast_dns* service in Linux, where peers advertise their naming information to be able to address nodes by hostname instead of layer 2 MAC or layer 3 IP address. Nodes identified through discovery are added to the list of available nodes for communication as well as status reporting. This discovery heartbeat is also used as a lost-communications detector to prevent higher-level messages from queueing for unreachable nodes.

3.4.5 ROS Message Encapsulation

In the ROS ecosystem, messages are translated from a message definition language specification into internal representations appropriate to the implementing language⁶. This same language specification is used to serialize and deserialize messages; that is, to transform a ROS message into a buffer of bytes suitable for transmission over an arbitrary channel. *udp_mesh* implements a generic message passing system such that the message to be transmitted is never deserialized, saving a significant amount of processing time in the case of complex, large message types such as images. Instead, a generic subscriber is used to acquire the serialized bytes for direct use to be transmitted to other nodes. On the receiver side, the transmitted byte stream is deserialized to instantiate the message in a format that other ROS ecosystem nodes can readily consume. These two functions abstract the transport of arbitrary messages over the *udp_mesh* layer and remove any requirement to define a list of acceptable message types.

3.4.6 Point to Point Transport

In the *udp_mesh* system, point-to-point transport is implemented via UDP datagrams. This envelope contains provisions for sequence tracking, fragmentation, and message reconstruction. As part of preparing a message for transmission, the byte buffer provided by the ROS encapsulation service is split into chunks that fit within the underlying medium's maximum transmit unit (MTU). For the standard 802.11 framing that is used in our system, this MTU is 1500 bytes, out of which 100 bytes are reserved for overhead, leaving 1400 bytes for payload out of every datagram.

In the implementation of our system, a configurable number of message fragments are permitted to be 'in flight' at any given time, similar to TCP congestion window control. In order for the next fragment to be transmitted, the receiver must send an acknowledgment. During unit testing to determine an appropriate value for the number of in-flight fragments permitted, an initial increase yields improved throughput. However, after a certain point, throughput decreases as multiple packets are queued for transmission on the medium and start to destructively interfere. As

⁶ <http://wiki.ros.org/msg>

a compromise, three packets are permitted to be in-flight between any two nodes at a time. With this configuration, our transport-layer throughput is approximately 20 Mbit/s of payload data, measured using raw images as representative high-density traffic over a wired gigabit Ethernet link.

Retransmits are automatically queued until either an acknowledgment is received or the host is marked offline due to non-reception of any heartbeat or acknowledgment messages. Once a host is marked offline, any attempts to send messages are discarded. Hosts may become online once again after receipt of a discovery message. On the receiver side, the message is kept in a temporary state while the fragments arrive. Should message fragments stop arriving, the partial message is purged after a timeout and the host is once again marked offline which indicates to higher levels that reliable transport is unavailable.

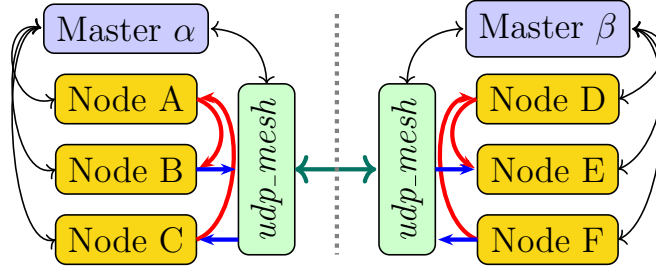
3.4.7 Quality of Service

Quality of Service (QoS) is the notion that some traffic should be prioritized over other traffic for use of a limited communications channel. As observed in prior competition events, sending large chunks of data such as full maps prevented other, more high priority data from being transmitted such as artifact reports and telemetry. Fundamentally, TCPROS (the default transport used in ROS v1) is not capable of implementing a QoS scheme where a limited channel is shared between different topics (Figure 3.4), as every node subscribing to a topic uses an individual TCP point-to-point link with no information about other links. This need to prioritize traffic was the driving rationale behind the development of the *udp_mesh* layer. As part of the configuration of the layer, each topic to be transported includes a priority number; internally, this priority number is used as a sorting key to order message fragments for transmission.

3.4.8 Point-to-Multipoint Transport

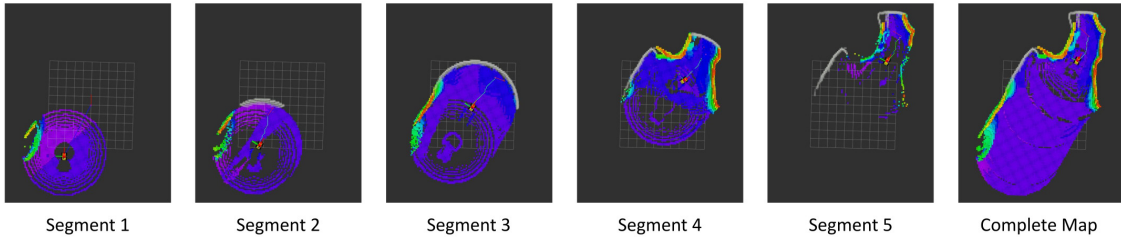
Although *udp_mesh* is based around point-to-point message transfer, mission requirements sometimes necessitate system-wide messaging. To facilitate these type of messages, a broadcast

Figure 3.4: In contrast to basic multi-master ROS (Figure 3.2), *udp_mesh* creates a single virtual channel between nodes, shown in green, to implement data prioritization.



mechanism is provided by the transport layer. For messages that fit within a single MTU, a single, unacknowledged UDP broadcast is used to distribute the message. For larger messages, individual links to each node are used to send the broadcast as a series of unicast fragments using the same accounting and acknowledgments as the point-to-point mechanism. In both cases, the receiver is unaware that the message was sent as a broadcast versus a single, directed message.

Figure 3.5: Differential map transmission. Instead of a complete map message, incremental map portions are shared between robots, reducing the required network bandwidth.



3.5 Functional Test

Our architecture was validated in a testing campaign leading up to our final competition performance. We present an analysis of our prize-winning run to describe several objective measurements, as well as present a useful case-study balancing network loading versus mission objectives.

For our final run, we deployed four robots (each acting as a dynamic mesh node) to explore the environment. Two of the robots, the Husky ground vehicles, were equipped with six comm beacons and a dispenser, where each beacon contained a 2W radio running the *meshmerize* stack.

Operationally, the Spot quadrupeds were deployed first, while the Huskies followed to establish a comm link back to the base station. Nodes were deployed autonomously by higher-level software. Figure 3.7 overlays the generated map (shown in white) with each robot trajectory; the green-blue colors show where each robot was within communication, while the red-magenta colors show a disconnected state. This figure shows that our network was able to maintain communication through a majority of the course. Our base station operator was able to observe live telemetry from each robot, including position and mapping information, while reserving the ability to take command as needed. Over the entire one-hour run, our robots transmitted 125.2MB of data, which included maps, telemetry, and discovered object reports. Our robots also shared data with each other to implement multi-agent coordination, which this total figure does not include.

Figure 3.6 plots a PDF of inter message arrival times for our main management message, as sent from a single robot (D01) to the base station over the duration of our final run. These messages publish at 1 Hz; an ideal system would observe all intermessage arrival times to be 1 second in duration. Due to network latency and usage, we observe an approximately normal ($\mathcal{N}(1.00, 0.04)$) distribution of arrival times. Since messages may experience delays, the immediately following message may exhibit an intermessage time of less than one second. Our key observation of this plot is that the bulk of messages arrive within five percent of their expected times across a distance of hundreds of meters and multiple mesh hops.

3.5.1 Case Study: Differential Maps

One particular effort from our team to reduce bandwidth requirements was the ‘differential maps’ mechanism used to transmit map portions amongst robots and the base station. Shown in Figure 3.5, each of the five component segments are combined to form the complete map. During a robot’s exploration phase, the telemetry data is used to trigger the creation of a new segment to keep individual ‘diff’ elements sufficiently compact. Within our prioritization framework, these diff elements are at a lower priority than artifacts, telemetry, and human supervisor commands. In practice, this innovation helped us to make sure that our limited bandwidth was used for the most

Figure 3.6: PDF of inter message arrival times for D01 with a notional publishing rate of 1 Hz, overlaid with $\mathcal{N}(1.00, 0.04)$. The highlighted red region represents the first σ value which contains 68% of the message times.

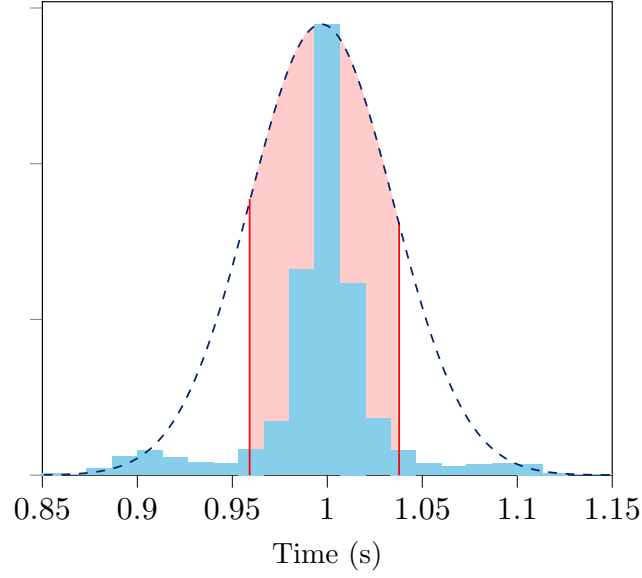


Figure 3.7: Map from the final run of the DARPA SubT challenge depicting the connection status of each of the four robots as they traversed through the course. Blue-green indicates connected, while red-magenta indicates disconnected. The red star indicates the location of the human operator, base station, and root communication node.



important information. As a robot proceeds along its mission, the human supervisor would often have a live view of telemetry and instant command response, while mapping data would gradually

fill in as communication bandwidth became available.

3.5.2 Case Study: Adding Point-Of-View Imagery

During our testing campaign leading to the final event, our improved communication stack based on *meshmerize* and *udp_mesh* had proven to be quite reliable and robust, particularly with respect to link saturation and data prioritization. This confidence in our system allowed our team to be more aggressive in our communications risk posture to utilize our bandwidth to improve scoring. In testing, we enabled point-of-view (POV) video using the front camera from each robot to send highly compressed imagery back to the human supervisor for review. This modification required a single line configuration change, to instruct the *udp_mesh* layer to transmit the FPV video topic at a priority level below that of all other traffic (telemetry, artifacts, and map data). Sent at 1 Hz and JPEG quality level 10 (out of a maximum of 100), these grainy images with visible compression artifacts (Figure 3.8) were key to our human supervisor being able to navigate through fog and identify artifacts that our autonomous artifact detection system missed. As a result of our pilot run, this change was rolled out to all robots in the fleet and significantly contributed to our success. In post-run analysis, this live imagery consisted of 51.6MB of the 125.2MB total data received by the base station over our mesh network.

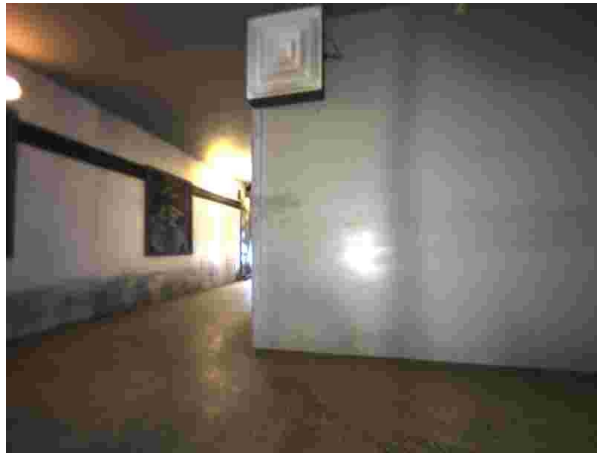
3.6 Discussion

In the development of our system, several key ideas emerged that warrant discussion and analysis.

3.6.1 Porting to arbitrary publish / subscribe architectures

While we have used ROS as an integration middleware, there is no fundamental tie to the ROS ecosystem that precludes adapting *udp_mesh* to an arbitrary publish/subscribe architecture. Internally, *udp_mesh* includes a pair of reader/writer adapters that marshal data to and from the underlying architecture. These adapters could be replaced for any other type of architecture, such

Figure 3.8: First-person video frame showing compression artifacts and a vent artifact



as OROCOS [16], which uses Common Object Request Broker Architecture, or Yet Another Robot Platform (YARP) [91], an architecture specifically designed for quality-of-service management.

3.6.2 Scaling to larger team sizes

Mesh network systems typically implement broadcast datagrams as a ‘flood’, which requires each node to repeat the message to ensure maximum distribution. Due to the difficulty in managing multicast group membership, broadcast is often utilized for multicast traffic as well. Because *udp_mesh* relies on broadcast messages to implement discovery, online detection, and name resolution, there is the potential for negative consequences as node count increases. As nodes are added to the network, these broadcast messages result in geometric growth of discovery / advertising traffic. One potential mitigation for this effect relies on eliminating redundant broadcast traffic when a stable link exists between two nodes and messages are being exchanged reliably. Alternatively, *udp_mesh* nodes could be configured with fixed neighbors to eliminate the broadcast mode.

3.6.3 Commodity Hardware

By relying only on commodity hardware, ranging in price from approximately \$30 to \$1000 each, our team had the freedom to select the hardware realization that best fit our particular

needs for each robot class and our beacons without being locked into a particular manufacturer. *ath9k* support has been well-established within the open-source community, with many other users around the world relying on the same hardware. Even though our radios were manufactured by three separate companies, we experienced zero problems related to vendor interoperability. Our development team was physically separated; being able to develop mesh capabilities using budget hardware allowed each team member to have a set of development nodes.

3.6.4 Interoperability with Existing Systems

Building off of the concept of the well-known OSI network model, we specifically engineered our system so that each layer need not be aware of how other levels are implemented. This modularity is present in the *meshmerize* software being able to support arbitrary IP packets, as well as our *udp-mesh* being completely ROS message agnostic. As a result, we were able to swap out components on a layer-wise basis with minimal reconfiguration of our systems. Further, by utilizing the ROS ecosystem, our communications system was completely compatible with every ROS node and tool that our team had already written, requiring no code modifications. However, this agnostic behavior meant that there was no awareness of network conditions at higher levels within our autonomy stack. For example, if there were to be a tighter coupling between mesh conditions and autonomy, a robot might adjust its data products to the underlying network conditions (such as improving first-person video quality, etc). Our initial evaluations led to a very conservative estimate of mesh performance; the limited data rate observed at the final event suggests that additional bandwidth was available, but unused.

3.6.5 Fast Reconnect Times

As expected, in early testing underground our radios were not able to penetrate rock formations and concrete which limited us to line-of-sight connectivity or unreliable multipath reflections around corners. Under such environmental limitations, being able to leverage a momentary alignment of robots could be key to passing information, particularly when coupled with our ‘smart’

mission management system [52]. As a result of our partnership with Meshmerize GmbH, we were able to exploit these brief connectivity moments without pausing our mission to allow the mesh network to re-establish itself.

3.6.6 Prioritization Controls

In testing prior to the final event, we passed full maps (before implementing the differential maps of Section 3.5.1) from each robot to the human supervisor. As a rough estimate, transmitting full maps at 1 Hz would have required 5.4GB of mesh data. With that setup, we were quickly saturating available channel bandwidth, leading to delays in passing high-priority data such as artifact reports and telemetry. The ability to establish prioritization controls on our message traffic was a key component of our communication system’s effectiveness, particularly once first-person view was added. In our system, the prioritization was only established on a per-robot basis; there was no coordination between robots about the use of the shared medium. As a result, one robot sending FPV data could be interfering with another robot’s telemetry; since our system is completely decoupled, there is no way to coordinate these types of traffic. In general, depending on the system architecture, such detailed coordination may or may not be necessary, desirable, or even feasible. However, our limited implementation in the final event has amply demonstrated that data prioritization must be a consideration in such communication-restricted environments.

3.7 Conclusions

In this chapter, we present a novel mesh networking system composed of open-source, commercial, and custom components that powered our competition team to a prize-winning victory. Our novel contribution, the *udp-mesh* transport layer, serves to link robots together by providing several communication services, including discovery, reliable transport, and prioritization, implemented over UDP. The framework is the backbone for the language-driven semantic reasoning described in Chapters 5 and 6.

Chapter 4

Background On Language In Robotics

As seen in previous chapters, advancements in perception, path planning, mapping, and autonomy have enabled teams of robots and humans to operate in complex subterranean environments at the DARPA Subterranean Challenge [28] with varying degrees of success. Even the top-performing teams at these events were only able to perform at a 50% success rate. In many situations, robots needed help from a human supervisor who gave inputs using a cumbersome waypoint-based interface. A chief limitation of most of these state-of-the-art systems was the ability of humans to send high bandwidth instructions in the form of natural language commands to the robots. The lack of this ability both prevents efficient supervision of the system and the ability for non-robotic experts to interact with the system. This survey will examine how other robotic systems have been connected to natural language with an emphasis on robotic navigation tasks.

Additionally, humans reason over landmarks using cognitive maps [79] rather than the traditional metric-based methods used in robotics. As such, translation mechanisms or reformulations are needed to transform traditional metric-based localization systems used by robots [43, 84, 119]. Natural language provides compelling mechanisms to help shift robots away from requiring metric-based localization in order to understand their environments.

Using natural language to communicate with robots can largely be divided into two facets [124]. The first is the challenge of language understanding which involves extracting meaning from language as well as the meaning's relation to the physical world. This is usually known as grounding where a language utterance is interpreted in terms of both the robot's physical state and its sur-

rounding environment. The second facet involves language generation where the robot interprets its surroundings and is able to communicate information about its state and its environment using natural language. While both of these facets share similar challenges, this review will focus on the language understanding problem within the context of robot navigation.

4.1 Language grounding:

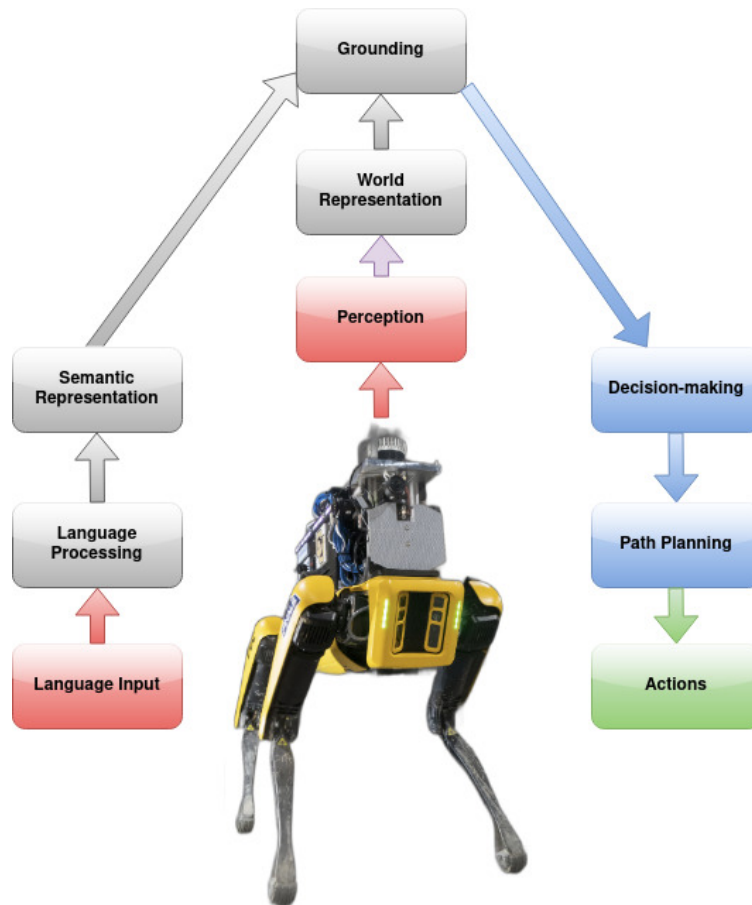
or situated language refers to the challenge of interpreting language within the context of the physical world [41]. Regarding robotics, groundings can represent objects in the physical world, motor commands, and sequences of actions [83]. For example, a sentence such as “Go to the backpack on your right” would require a robot to use a camera (or similar) to determine that a backpack exists. The robot would then need a method to determine where the backpack is which it could do with a range sensor. Finally, the “Go to” part of the sentence would tell the robot to start walking or turning its wheels to reach the destination. Figure 4.1 shows the components that are needed for language understanding within the context of mobile robots.

4.2 How Can Language Be Represented

The linguistics and cognitive science communities have spent extensive time studying the structure of natural language [62]. Natural language has a compositional syntax that follows a hierarchical structure which enables the creation of novel utterances. In certain cases this syntax can be exploited to create parse trees such as Context-Free Grammars (CFGs) [44]. CFGs divide sentences into determiners (DT), nouns (NN), noun phrases (NP), verbs (VB), and verb phrases (VP). From there, a parse tree can be created using the verb or verb phrase as the root. An example parse tree for the sentence “Grab an apple.”, is shown in

Parse trees are incredibly useful because they can be formally verified which enables the creation of logical programs. For example, λ -calculus can be used to express a parse tree in terms of function arguments. Taking the same sentence the λ -calculus expression $\exists x(\text{APPLE} \wedge \text{GRAB}(X))$ can be made. This expression states that there exists an apple (x) that is being grabbed. Formal

Figure 4.1: A system diagram showing the necessary components for visual language navigation in robotics.



structures like these are important to robotics because navigation requires a degree of certainty so that the robot both drives to the correct location and does so in a safe manner.

CFGs have also successfully been converted into temporal logic. Temporal logic is a series of logical statements that are evaluated in sequence for an overall boolean true or false value for the entire statement [33, 63]. Taking a look at the sentence “Drive to the backpack”, we can express it as $\Diamond \text{AtBackpack}$. AtBackpack is now a boolean proposition that can only be true when the robot arrives at the backpack. These types of formalization both ensure the robot will do what is expected of it and reduce the ambiguity naturally presented in language.

While these methods haven’t been directly deployed within the context of robotic navigation, they highlight some of the early work toward goals researchers are trying to achieve with more

modern methods. These early language parsing techniques are largely incompatible with robotics because they don't capture more complex language such as spatial relations and they also do not address the symbol grounding problem.

More modern language parsing techniques tend to rely on global vector representations [61] and contextual embeddings [73]. Of note, is the shift towards global embeddings such as Global vectors for word representations (GLOVE) [94] and Bidirectional Encoder Representations from Transformers (BERT) [30] which are used by many of the top language parsing systems.

In the rest of this review, we will examine three styles of approaches that are commonly used to connect natural language to robotics. First, we will look at logic-based methods and then we will look at probabilistic methods. Finally, we will take a look at large language model-based approaches.

4.3 Probabilistic Models

In recent years, the challenge of grounding language to the physical world has been addressed by inferring over probabilistic graphs. [125, 46]. Early works in this domain collected a corpus of natural language directions using a user study where humans walked through the MIT campus and provided directions between rooms [59]. All of the subjects were familiar with the area before they were instructed to create instructions. In this work, the corpus was parsed into a sequence of Spatial Description Clauses (SDC). An SDC is a graphical model that consists of a *figure* or the subject of the sentence, a *verb* which is the action, and a spatial relation between the environment and the figure.

The SDC planning system described in [59] then builds a topological map using inputs from a lidar and a camera. Inference is performed on the language by:

$$\arg \max_p p(P, S|O) = P(S|P, O) \times p(P|O) \quad (4.1)$$

where P is the path, S is a sequence of SDCs and O represents the observed landmarks. The

authors factor Equation 4.1 into four parts which correspond to each field of the SDC. The authors solve the action grounding problem by limiting the robot to a series of control sets that can go left, right, or straight. The method can either operate in a global mode if an *a priori* map is known or it can greedily explore if the map is unknown. The overall success rate of this method in finding the “best path” which the authors describe as the path that ended up closest to the true destination was 59.3% percent.

Overall, this early work sets a baseline for grounding graph-based methods but it is limited to sequentially structured language and is unable to handle nested clauses. In other words, when given a sentence such as: “Put the pen into the backpack” this method looks at the entire sentence in a flat way and is unable to separate the two arguments for “put”. Both the backpack and pen are necessary to comprehend the true meaning of the sentence. “Put” requires two arguments and this method would produce two independent SDCs rather than a joint one.

An extension to the SDC approach for grounding language is presented in [125, 60] where a Generalized Grounding Graph (G^3) is introduced. The authors define a *grounding graph* as a dynamically instantiated probabilistic graph that takes the hierarchical structure of natural language command.

In order to bound, the search space the authors limit the action sequences to events, objects, places, and paths. Events are in the form of an action sequence and objects are things in the world. Paths represent words that define a way to traverse through the world such as “towards”. The inspiration for these classifications comes from early work on semantics and cognition presented in [53].

In the grounding graph, two types of nodes are present: random variables and factors. Each random variable is binary and can either be true or false. Groundings are represented as γ and the random variable ϕ_i is true if it corresponds to the grounding γ_i . There are random variables λ_i^f , λ_i^r , λ_i^l which correspond to figures, relations, and landmarks.

Generalized graphs can be automatically generated using conditional random fields. The system was able to learn the meaning of words such as “put” by pairing them with the corresponding

robot action. Despite promising early results G^3 graphs still face many limitations. For instance, can't represent anaphora and negations. The search space also grows exponentially as the language present in the domain increases.

Distributed Correspondence Graphs (DCG) [46] are an extension to G^3 graphs that discretize the space of groundings into regions and motion constraints. These graphs also take advantage of planning in the space of language constraints and then passing off those constraints to a trajectory planner. This decoupling of the language understanding and planning problem enables efficient trajectory generation.

Like generalized grounding graphs, the DCG model can support infinite recursion of noun phrases. Additionally, the method is agnostic to the planning method because it is only used to form constraints for a planner. The method only works when all constraints intersect each other and is therefore incompatible with complex spatial language. Additionally, the method requires training data that aligns with the domain and the data needs to be hand annotated.

4.4 Large Language Models

Large language models (LLM) have recently entered the spotlight with the release of ChatGPT [35]. LLMs refer to language models that contain tens to hundreds of billions of parameters [146]. Traditionally, these models have been limited to text-based inputs and outputs but recently multimodal versions have emerged that are able to take in images [89], embodied sensor data [31], and even generate arbitrary programs [122]. LLMs have shown a phenomenal ability to generalize across different contexts which greatly reduces the need to hand-label training data. In this section, we will explore how LLMs are being used in robotics and their future potential for them.

Visual-language models (VLMs) are extremely capable of generalizing pre-trained data available from the Internet such as pictures with captions to other domains [99, 67]. For instance, LSeg [67] is a transformer-based encoder that performs pixel-level semantic segmentation on images even when the caption is unknown.

In stark contrast to traditional Convolutional Neural Network (CNN) based visual detection

methods such [137, 74, 101] these methods do not require manually annotating data. For robotics, this is a potentially game-changing capability as robots will frequent unknown environments to them when they are navigating through the world. Recently, visual language maps for robot navigation have been created by exploiting the transformer-based architectures of LLMs [48, 117, 36].

LM-Nav [117] is a method of doing robotic navigation that is trained on unannotated large datasets of trajectories. LM-Nav combines a vision model (CLIP) [99], a language model (GPT-3) [35], and a robot navigation model ViNG [116]. ViNG is a learning-based navigation system that is capable of reaching visually indicated goals. ViNG is capable of building a plan by learning a topological graph from ego-centric images. The topological graphs generated by the method are only associated with image frames and do not form a bird’s eye view. ViNG is capable of successfully navigating using only offline experience but it can also adapt to new environments including season changes.

LM-Nav extends ViNG by leveraging LLMs to navigate in novel environments. LM-Nav uses GPT-3 to parse instructions into a list of landmarks. The landmarks are then passed into CLIP to ground the landmarks to image space. ViNG is then used to predict the capability of the robot to navigate between each pair of nodes in the generated graph. This is done using temporal distances and the actions are then generated on a real robot.

Lm-Nav was implemented on a Clearpath Jackal UGV. A key limitation of the system is that the GPT and CLIP queries need to be pre-computed on a remote computer. This is due to computing constraints on board the robot. LM-Nav also heavily relies on landmarks and doesn’t account for other verbs in a sentence. As such the method doesn’t know how to “drive the robot slowly past the dog”. Slow isn’t inferable from an image and therefore is unknown to this architecture.

An alternative navigation approach is presented in [48] which emphasizes the generation of spatial maps. VLMaps [48] are generated directly from videos and are indexed by natural language. Specifically, they are created by fusing pre-trained visual-language features with reconstructions of the environment. The system uses a Lseg visual encoder to create a segmented map. Landmarks are indexed using a text encoder.

Experiments are run on 91 different task sequences. Starting positions are randomly specified and there are four subgoals that the robot needs to navigate towards.+ Competing methods such as LM-Nav are only able to go to 2 subgoals. VLMMap outperforms LM-Nav by a significant margin (62 vs 5 successful tasks) when considering subgoals.

Additional methods have leveraged CLIP [36] to perform zero-shot robotic navigation. For instance, CLIP on Wheels builds a saliency map which it uses to generate a segmentation mask. Specifically, Grad-CAM [113], a gradient-based interpretability localization map generator is leveraged to build the mask. That mask is then planned over using a frontier-based [140] exploration method. Experiments were performed both in simulation and on an HSR mobile robot.

Perhaps the most intriguing example of LLMs being used in the context of robotics comes from the PaLM-E model [31]. PaLM-E is a 562 billion parameter model that is trained on robotic tasks but can also perform Visual Question and Answering. Palm-E’s inputs consist of text and continuous observations which are interleaved within the text. The model interleaves observations that happen between two image frames in the encoded input. Palm-E’s output is text that conditions low-level commands.

Palm-E is also able to learn across different domains. The authors observe that the model performs over 50% better when using text and visual observations than using either domain alone.

Palm-E experiments are run in a Task and Motion Planning Environment (TAMP). Example tasks include opening drawers and moving blocks around. Palm-E is able to perform zero-shot planning. Palm-E is also capable of chaining together multiple sequences of instructions.

ViperGPT [122] is a model that is able to perform visual inference and reasoning by executing Python programs. ViperGPT combines a series of models and generates Python code using GPT-3 Codex [18]. This allows for the programmatic composition of specialized vision, language, math, and logic functions. Importantly any module can be substituted because ViperGPT is only generating the specification for the program rather than the implementation. For robotics, this is important because it enables traceability and explainability. When end-to-end methods fail it is difficult to understand why. However, by combining other approaches with ViperGPT it may be possible to

see what triggers failures and also provide a level of assurance that a robot will actually perform the task that was requested of it.

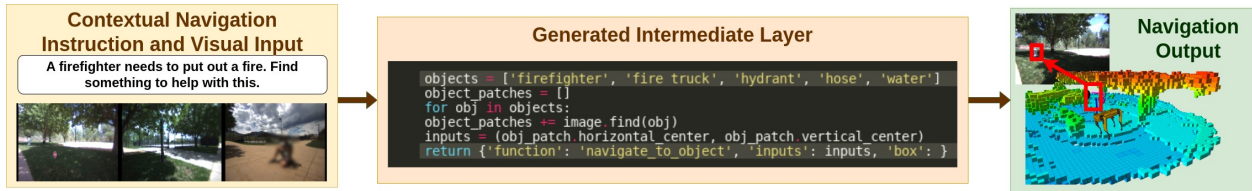
Chapter 5

Code Generation For Language Grounding

5.1 Introduction

During the DARPA challenge, we saw that human intervention was critical to the overall success of the search-and-rescue mission. Humans have the remarkable ability to navigate through unfamiliar environments, e.g., in a town or through a building, by relying solely on our priors and descriptions of the environment [79]. Motivated by the difficulties in directing robots in collaborative teams with humans such as those used in search and rescue [28, 1, 14, 130] operations, we aim to develop a framework that allows humans to provide high-bandwidth instructions to robots in the form of natural language.

Figure 5.1: Contextual navigation example for a firefighting situation. We see our compositional framework generates code for the robot to both find and then plan to a fire extinguisher.



To achieve this, it is generally required that a robot associate natural language utterances to the physical world using sensing modalities onboard the robot, a process known as grounding. However, unlike humans, robots are not currently capable of integrating prior experiences into a vast wealth of priors to aid this association.

Recently, Large Language Models (LLMs) [146] have been able to reason over billions of parameters and utilize the results in domains such as dialogue-based responses [39, 131, 72], code generation [18, 70, 139], and multimodal visual question and answering (VQA) tasks [7]. This type of unsupervised learning shows incredible potential for the generalization of prior knowledge but these models lack real-world experience. A lack of physical experience is perhaps one reason why transitioning techniques developed through research in VQA and view-based navigation into embodied agents has not kept pace with the research in these fields.

Additionally, with the current generation of LLMs, it’s inherently challenging to interpret the rationale behind the outputs that are generated by the model. In this work, we address both the grounding and transparency issues using a composable framework designed for robot navigation. Our pipeline extends the reasoning framework for visual inference presented in [122] to an embodied robotic agent. With our composable framework, we leverage LLM priors (GPT-3.5), state-of-the-art object detectors [69], and classical robotic planning algorithms [66, 26, 3] to perform zero-shot natural language based navigation in four unique environments.

Specifically, contributions in this chapter are as follows:

- We extend the concept of modular neural networks and define a new framework for composable robot navigation which we call NavCon. Our framework requires a minimal uplink for the robot since all of our planning, mapping, and localization is performed onboard.
- We evaluate different 2D input representations to determine an effective way to extract spatial and conceptual knowledge from LLMs.
- We perform extensive real-world experiments in a variety of environments and show that our framework is able to navigate to landmarks based on natural language. Furthermore, the framework is able to deduce appropriate navigational goals from the context of a sentence.

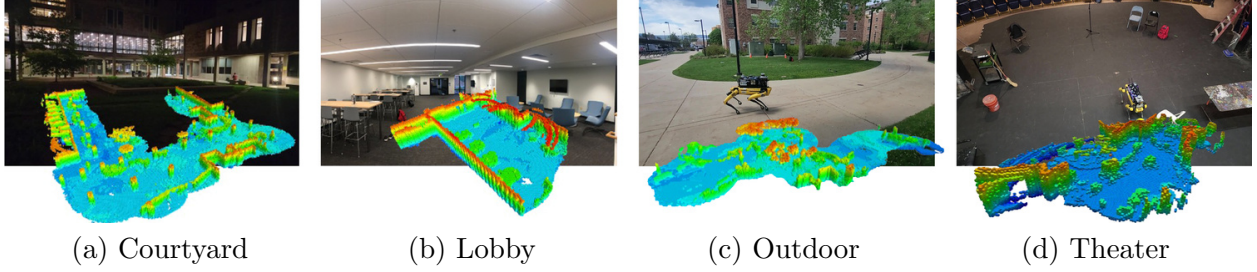
5.2 Related Works

Natural-language-based navigation with an embodied agent requires grounding utterances to the physical world in which the robot operates. Various approaches have been used to associate language with the physical domain ranging from probabilistic graph-based structures [125, 46, 59] to end-to-end learning-based methods [31]. Graph-based approaches have shown promising results but their generalizability is limited to a fixed training corpus. On the other hand, LLMs have proven to be adept at reasoning over large amounts of unsupervised training data [35, 127] using transformer-based backends [133]. These models also known as foundation models due to their comprehensive knowledge have recently been leveraged for task and motion planning using reinforcement learning on a set of skills [15], and in an end-to-end fashion [31] with object scene representation transformers [106]. While these approaches have made remarkable strides they still suffer from the explainability problem and further they don't share any of the path planning guarantees that traditional planning methods share [86].

5.2.1 Modular Neural Network Frameworks and Code Synthesis

Modular Neural Networks [55, 47, 141] are remarkably adept at answering questions about images in a task commonly known as VQA [9]. Until recently many of these methods were limited in their ability to generalize to other domains due to the difficulty of generating interfaces between modules. In [122] this was addressed by using recent code generation techniques [76] from LLMs to create these interface layers. Specifically, these modular frameworks enable the generation of code that dictates the interaction between robust object detectors [69, 68, 42, 101, 37], depth estimators [100, 145, 12]. What's more remarkable is that these LLMs can also utilize existing functions inside of the Python language such as sorting and conditionals without any additional training [132, 85, 139, 127]. Based on these exceptional findings we aim to extend the use of these modular concept-learning style frameworks to the robotic navigation domain.

Figure 5.2: Full volumetric maps and representative images from the four testing environments.



5.2.2 Path Planning for Robotic Navigation

Path planning for robotic navigation and exploration is a long-studied research area [75, 144] with solutions ranging from sampling-based methods [66, 56] to graph-based lattice structures [96]. Variations of these probabilistic approaches are being used for self-driving cars [65], exploration [3, 14, 1, 130] in complex environments. These approaches benefit from probabilistic guarantees on convergence and optimality [66] and adaptability to new environments making them natural choices to utilize in language grounding framework.

5.2.3 Embodied Language-Based Navigation

Recently a few other works have leveraged foundation models for embodied navigation. Clip on Wheels (CoW) is proposed in [36] which combines text and image caption models [99] with frontier-based exploration algorithms [140] to perform object exploration in simulation. In [117] landmark-based navigation is performed using pre-trained visual [36], language, and navigation modules [116]. Visual Language Navigation Maps (VLMaps) [48] learn a spatial language representation by combining RGB video feeds, code generation, and foundation models. SayCan [6], and Palm-E [31] utilize the PALM [21] language model to perform skill-based navigation and end-to-end navigation respectively. In general, these methods are not tested on contextual examples, and some require the preprocessing of maps; in contrast, we test on situated examples with an experimental platform and require no situated priors.

5.3 Methodology

We develop a framework for Navigation with Context (NavCon) that leverages the rich contextual priors of LLMs and creates an executable code layer that interfaces with planning algorithms running on an embodied agent. Drawing on paradigms from modular neural networks [8, 40, 122], and concept learning techniques [77], we define a modular system that takes inputs at various stages to fully define a final navigational output for our system. This enables the use of arbitrary intermediate and substitutable layers which can also be executed on a distributed computing infrastructure. Formally, we have a collection of inputs ψ composed of a visual input $v \in V$ an RGB image or multi-perspective collection of images, $m \in M$ a volumetric map of the space to be navigated, and $c \in C$ a natural language navigation command. Our system takes initially as input a command c to generate code γ_c . This code consumes v in order to resolve the grounding problem, i.e., to determine which object is being referred to in c for navigational instructions, as well as m in order to emplace the object into the world around the robot. This outputs $i = \theta(v, m | \gamma_c) \in \mathbb{R}^3$, a 3D waypoint to which navigation will commence in order to output $\mathcal{P}$ a robot trajectory which is a continuous function in $\mathbb{R}^3$. A graphical overview of the framework can be found in Figure 5.3.

5.3.1 Input Representations

Our visual input v is in the form of an RGB image, either a semi-panoramic view or three separately labeled spatial images (left, front, right). Typical inference methods over RGB images [101, 42, 69] reason in 2D over the image. For embodied navigation, we need to reason in 3D and associate different camera viewpoints with their associated 3D spatial relations. Embodied navigation introduces spatial relations that are difficult to reason over in 2D such as “behind”, “in front of”, or “on your right”. To determine an appropriate input representation we evaluate the difference between sending in a concatenated image of all viewpoints (semi-panoramic) from the agent or sending in each frame separately with a spatial definition, i.e., right, front, or left. We find that spatial reasoning is best performed on a concatenated image and we present the details

in Section 5.4.

5.3.2 Intermediate Layer

We generate θ using recent advances in code generation models [18, 85]. We provide a functional set of navigation instructions in the form of a Python API for the code generation model. The full prompt containing the API can be found in the Appendix. These instructions include specifications for how to perform visual inference on the input v using a similar paradigm found in [122]. These API instructions include directions for finding an object, checking if an object has a certain property, etc. Additionally, we provide an API specification for interfacing with our geometric planner, resulting in code generated based on the natural language prompt γ_c .

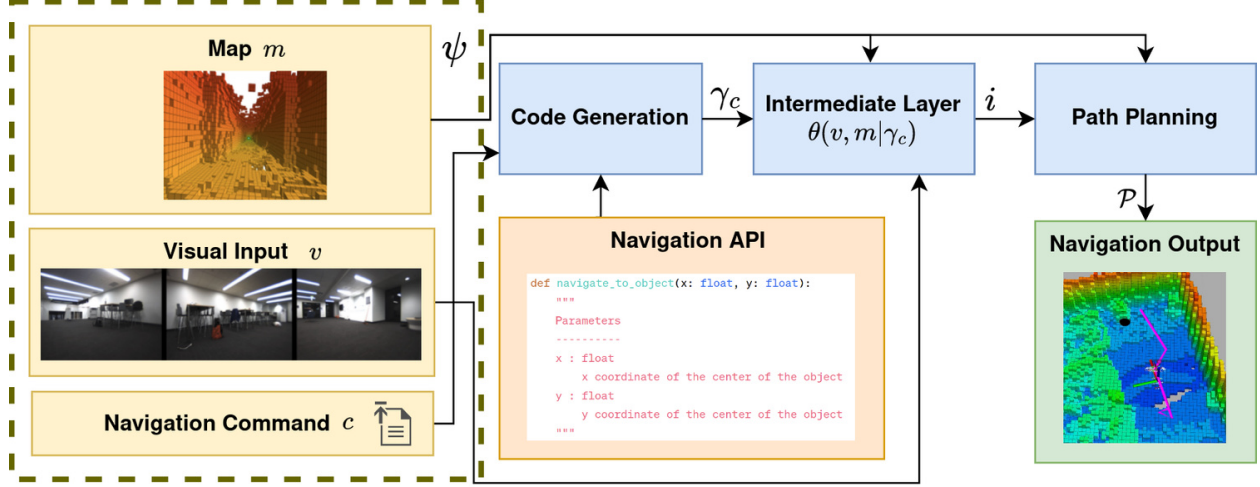
5.3.3 Planning

We use a graph-based planner first presented in [3, 14] where each sample is a robot position parameterized by the robot’s width and length. During sampling, the collision checks are performed by projecting the samples along the path to the ground on the map m below the sample. The elevation change at the footprint of the sample is calculated and if these differences exceed a given threshold or there are not enough sampled points (indicating a hole in the ground) the path is marked invalid and not added to the graph. We create a navigational API that takes in the center coordinate p of an object in image space and projects it as a landmark l on a 3D map. Using this strategy, a graph of plans $\mathcal{G}$ is constructed by sampling points parameterized by the robot’s width and length [66]. Additional semantic labels are added to the map for staircases as described in [81] using normal estimation on point clouds. When a 3D waypoint is input to the planner we select the best path $\mathcal{P}$ from $\mathcal{G}$, for the robot to follow. If no path exists, we plan to the boundary of the graph and resample until the goal is reached.

Waypoints are passed to the planner using the output i from the intermediate layer. Specifically, image coordinates are translated into waypoints by executing θ and associating the result onto a 3D map m [45] using ray casting. The map m is generated online using [119] and the

translation layer θ creates the necessary code to translate between the inputs c , v , m as and the output i . We then plan to the waypoint by selecting the best path from $\mathcal{P}$.

Figure 5.3: A graphical overview of the full system framework.



5.4 Experimental Results

We run two sets of experiments on a Boston Dynamics Spot equipped with a custom sensing suite consisting of a 3D 64 beam Ouster lidar, an IMU, and 3 RGB cameras providing a semi-panoramic view of the environment. Our first experiment is designed to determine the best input representation for the visual layer and the second set tests the ability of our system to perform navigation in a wide variety of real-world environments.

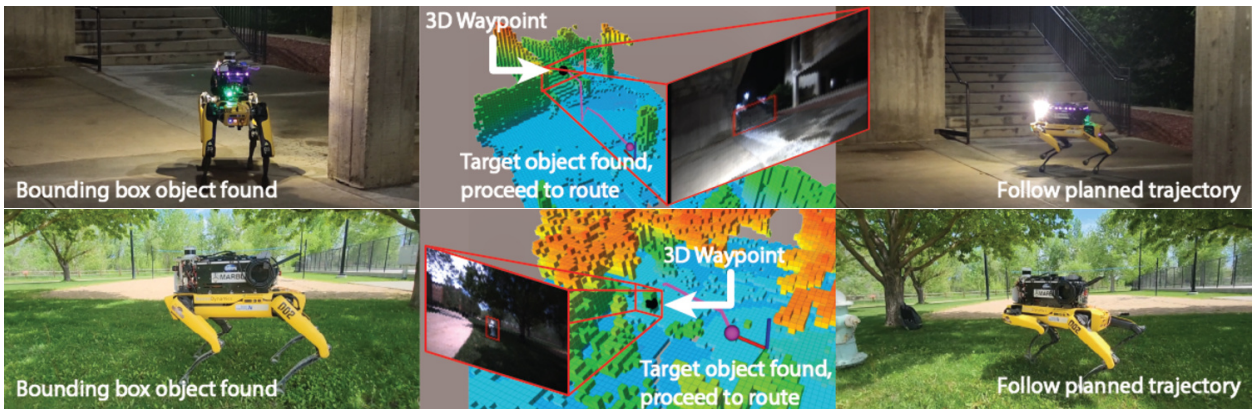
For all of our experiments, we leverage ideas from human concept learning as in [77] to categorize our sentences into four categories: Generic, Specific, Relational, and Contextual. Generic sentences are simply sentences that imply “Go to something”, e.g. “Walk to the backpack.”. Specific sentences include a distinguishing piece of language such as a color attribute that directs the robot to one specific object in the scene. For example, if there are two backpacks, one red, and one black, then a specific sentence would be “Drive to the black backpack.” Relational sentences are any sentences that describe spatial relationships between objects in the scene such as, to the right

of, on top of, etc. An example in this category is “Move to the backpack on top of the chair.” Contextual examples, require the robot to interpret the navigational goal based on background information. For instance, the sentence “Find something that can carry water.” requires the robot to know that a cup or a bucket can hold water.

We generated sentences by eliciting verbs for navigational policies from an LLM (GPT 3.5-0301). Then objects in the scene were inventoried, and sentences were constructed based on verb-object combinations as well as their properties and relationships to other objects in the environment. These sentences were then diversified into contextual commands by providing reference to the objects’ attributes or affordance and our full list of sentences can be found in the Appendix.

We evaluate the success of the method based on the following criteria. The code generation step is considered successful if the generated intermediate layer γ_C is both syntactically and logically correct. In other words, if every component worked successfully then the generated code would solve the navigation command. Object Detection (OD) is considered successful if the correct object is extracted from the visual input in the intermediate layer as visualized by the bounding box. Similarly for waypoint projection (WP) if the 3D projection is in the correct location on the map m this step is considered a success. The planning and execution step is considered successful if the robot plans and then traverses to within 0.2m of the target object. For all tables percentages are calculated based on the total number of evaluated sentences.

Figure 5.4: Example sentences, visual detections, maps, and planned paths.



Encoding spatial relationships into a model without a concept of the physical world presents significant challenges. We determine an effective scheme for inputting three different viewpoints into the framework through empirical evaluations. Take the sentence, “go to the chair on the right” implies only the right half of the total FOV should be investigated but ”Go to the backpack that is to the right of the chair.” could be in any image. To solve these limitations we use two different sets of input representations, the first is all three images stitched together with padding in between frames (**A**). In this case, we instruct the LLM on the order of the images (left, front, right) and let the model handle the spatial reasoning. In the second case, we process the frames individually (**B**) and let the model decide which frames to look at. We explicitly prompt the model with example code snippets that solve the difference between “to your right” and “right of” styles of language. at configuration. In this set of experiments, we evaluate the correctness of the generated code as well as if the code produced the correct object in the map m . This scene was created in a classroom setting with good lighting conditions as the emphasis was to evaluate the effect of the input representation. Results for successful code generation and object detections for the two input schemas are summarized in Table 5.1.

We also compare our results to OFA [136], a sequence-to-sequence multi-modal model fine-tuned on RefCOCO [142] expressions for the object detection stage. An additional baseline is performed where we utilize an LLM (GPT-3.5-0301) to convert a navigational command to a referring expression such as the ones in RefCOCO. Our full prompt for this pipeline is provided in the Appendix.

Table 5.1: Results showing the difference between the concatenated input representation **A** and the sequential representation **B**.

Category	Count	A (%)	B (%)	OFA (%)	OFA+LLM (%)
Generic	12	100	100	25.0	50.0
Specific	12	91.67	66.67	25.0	50.0
Relational	15	86.67	53.33	6.67	13.33
Contextual	11	81.82	45.45	0	36.37
Total	50	90	66	14	36

From Table 5.1 we find that concatenating images significantly outperforms sending in the three frames separately. As expected, in the generic navigation case we are able to achieve 100% success in generating the intermediate layer and for object identification. We see that the individual frame configuration (**B**) begins to break down when reasoning over specific objects. Specifically, the model fails to reason over spatial relations and object ordering when objects appear in more than one frame. For example, taking the sentence “Go to the middle outlet” only works when the three outlets are present in the same camera frame. If the middle outlet is in the front frame but the right outlet is in the right camera frame, this method fails. We explicitly see this in the generated code:

```
outlet_patches.sort(key=lambda x: x.horizontal_center)
middle_outlet = outlet_patches[len(outlet_patches) // 2]
```

where each detected outlet is ordered based on its horizontal coordinate in the image. Since all outlets are merged into the same list the images have overlapping coordinates causing these failures. Configuration **B** struggles with relational sentences between objects for similar reasons.

All the prompts used in the classroom environment for testing the two different input representations A & B are presented in Table 5.2.

Table 5.2: Scene-05: Classroom Full Sentence List

Category	Sentence	A	B	OFA	OFA + LLM Expression	OFA+ LLM
Generic	Go to the backpack	Pass	Pass	Fail	Backpack	Fail
Generic	Move towards the backpack	Pass	Pass	Fail	Backpack	Fail
Generic	Drive to the backpack	Pass	Pass	Fail	Backpack	Fail
Generic	Run towards the backpack	Pass	Pass	Fail	Backpack	Fail
Generic	Go to the cone	Pass	Pass	Fail	Cone	Fail

Generic	Go to the conical traffic delineator	Pass	Pass	Fail	Conical traffic delineator	Pass
Generic	Go to the trash can	Pass	Pass	Fail	Trash can	Pass
Generic	Go to the whiteboard	Pass	Pass	Pass	Whiteboard	Pass
Generic	Proceed to the broom	Pass	Pass	Pass	Broom	Pass
Generic	Trek towards the wagon	Pass	Pass	Pass	Wagon	Pass
Generic	Find paper towels	Pass	Pass	Fail	Paper towels	Fail
Generic	Go to the outlet	Pass	Pass	Fail	Power outlet	Pass
Specific	Go to the red backpack	Pass	Pass	Fail	Red backpack	Fail
Specific	Go to the black backpack	Pass	Pass	Pass	Black backpack	Pass
Specific	Navigate to the backpack on the left	Pass	Pass	Fail	Backpack on the left	Fail
Specific	Drive to the backpack on the right	Pass	Pass	Fail	Backpack on the right	Fail
Specific	Go to the whiteboard in front of you	Pass	Pass	Pass	Whiteboard in front of you	Pass
Specific	Move to the whiteboard on your right	Pass	Pass	Fail	Whiteboard on your right	Fail
Specific	Move to the whiteboard on the right	Pass	Fail	Fail	Whiteboard on the right	Fail
Specific	Go to the backpack on your right	Pass	Pass	Fail	Backpack on your right	Pass
Specific	Walk to the backpack on the left	Pass	Fail	Fail	Backpack on the left	Pass
Specific	Go to the tv on the right	Pass	Fail	Pass	TV on the right	Pass

Specific	Go to the orange cone on your right	Fail	Pass	Fail	Orange cone on your right	Pass
Specific	Go to middle outlet	Pass	Fail	Fail	Middle outlet	Fail
Relational	Go to the backpack that is to the right of the red backpack	Pass	Pass	Fail	Backpack to the right of the red backpack	Fail
Relational	Drive to the backpack that is to the left of the black backpack	Pass	Fail	Fail	Backpack to the left of the black backpack	Fail
Relational	Walk to the bag that is next to the black bag	Fail	Fail	Fail	The bag next to the black bag	Fail
Relational	Move towards the backpack under the whiteboard	Pass	Pass	Fail	Backpack under whiteboard	Pass
Relational	Walk to the backpack on the chair	Pass	Pass	Fail	Backpack on the chair	Fail
Relational	Go to the chair with the backpack	Pass	Fail	Fail	Chair with the backpack	Fail
Relational	Walk to the backpack on top of the chair	Pass	Fail	Fail	Backpack on top of the chair	Fail
Relational	Run to the rightmost backpack	Pass	Pass	Fail	Rightmost backpack	Fail
Relational	Walk to the leftmost backpack	Pass	Pass	Fail	Leftmost backpack	Fail
Relational	Go to the middle chair	Pass	Fail	Fail	Middle chair	Fail

Relational	Go to the leftmost backpack on your right	Fail	Pass	Pass	Leftmost backpack on your right	Pass
Relational	Go to the middle chair in the row of chairs	Pass	Pass	Fail	Middle chair row of chairs	Fail
Relational	Go to the backpack on the left of the cone	Pass	Fail	Fail	Backpack on the left of the cone	Fail
Relational	Go to the cone to the left of the backpack	Pass	Pass	Fail	Cone to the left of the backpack	Fail
Relational	Go to the second chair from the left	Pass	Fail	Fail	Second chair from the left	Fail
Contextual	Go to somewhere I can sit down	Pass	Pass	Fail	Chair	Pass
Contextual	Find a place for me to rest	Pass	Fail	Fail	Couch	Fail
Contextual	Go to somewhere I can speak from	Pass	Fail	Fail	Podium	Pass
Contextual	Find a place to store cleaning supplies	Pass	Fail	Fail	Storage area	Fail
Contextual	Find me something to write on	Pass	Pass	Fail	Piece of paper	Pass
Contextual	My friend has a question. Go to somewhere you can explain the answer to him	Fail	Fail	Fail	My friend	Fail

Contextual	I spilled a lot of sand. Go find me something to pick up my mess	Pass	Pass	Fail	Something to pick up my mess → dustpan	Fail
Contextual	Walk to something where I can put my laptop in	Fail	Fail	Fail	Table to put laptop in	Fail
Contextual	I spilled water. Find me something to clean this up	Pass	Fail	Fail	Towel	Fail
Contextual	Go to somewhere I can google something	Pass	Pass	Fail	Computer on the left	Fail
Contextual	Go to somewhere I can charge my phone	Pass	Pass	Fail	Charging outlet	Pass

5.4.1 Real-World Environments

In our second set of experiments, we evaluate the ability of our framework to navigate in real-world environments. The four environments shown in Figure 5.2 range from an indoor lobby setting to a dark outdoor courtyard. We select these environments to highlight our framework’s ability to generalize in multiple scenarios. In each environment, we test multiple sentences across the four categories (see the Appendix for the full list of sentences). Sentences are primarily based on landmarks already present in the environment but in some cases, we add additional artifacts to enable more extensive testing.

Further details about each of the environments are listed below:

- **Theater:** an indoor environment simulating a stage-like environment during set construction.
- **Lobby:** an indoor environment, mostly occupied with chairs and tables, posing more challenges in distinguishing specific objects and planning

- **Outdoor:** a larger environment with objects such as trees, cars, bikes, and sporting areas.
- **Courtyard:** an outdoor environment tested at night to challenge the framework with low-light images.

Along with the general objects present in the scene, we placed some additional items like backpacks, shoes, and cones into the scene. All the prompts tested in these 4 different environments are presented in Tables 5.5, 5.6, 5.7, and 5.8. For each of the prompts, we also highlighted whether each step succeeded or not.

Table 5.3: Summary of category-wise success rate for each of the four command types.

Category	Count	Code(%)	OD(%)	WP(%)	Path&Exec(%)	OFA (%)	OFA+LLM (%)
Generic	22	100	81.82	68.18	68.18	13.64	22.73
Specific	19	89.47	89.47	78.95	73.68	0	15.79
Relational	44	72.73	59.10	59.10	59.10	18.19	36.37
Contextual	29	65.52	41.38	41.38	41.38	6.90	24.14
Total	114	78.95	64.04	59.65	58.77	11.40	27.19

We find that we are able to successfully generate code for navigational plans using a variety of verbs e.g., walk, go, drive, run, etc by leveraging the rich vocabulary knowledge present in foundation models. In fact, we can even say phrases like “sashay to the stop sign” which will be interpreted as a “go to” style navigation command.

From Tables 5.3 and 5.4, we see that for generic objects our framework has a 100% success rate for code generation. In this case, code is manually evaluated and deemed successful if the generated code would produce the correct results assuming all the rest of the composed modules were executed successfully. In both tables **Code** represents the success of the code generation step, **OD** represents the success of the object detector (GLIP), **WP** is the success of the 3D waypoint projection and **Path&Exec** is the success of the graph-based planning and navigation module. Our failures occurred in the object detection module where either the wrong object was detected or the object was not detected at all. However, since our framework is modular, the object detector

Table 5.4: Summary of scene-wise success rate for each of the four command types.

Scenes	Count	Code(%)	OD(%)	WP(%)	Path& Exec(%)	OFA (%)	OFA+ LLM (%)
Theater	30	90	70	66.67	63.33	3.33	13.33
Lobby	29	65.52	48.28	44.83	44.83	13.79	24.14
Outdoor	24	87.5	79.17	70.83	70.83	12.5	33.33
Courtyard	31	74.19	61.29	58.06	58.06	16.13	38.71
Total	114	78.95	64.04	59.65	58.77	11.40	27.19

can be substituted and as state-of-the-art vision detectors improve our framework will improve in turn. Additional failures occur in the waypoint projection step. These occurred for two reasons. In the first case, the vision system detected further out than the map resulting in projections at the edge of the map rather than the actual object. For the second case, projections missed the correct voxel on the map. This was either due to a ray “clipping” a closer object in the projection process. For longer projections (10–12m), the camera-to-lidar calibration caused enough error to make the projection inaccurate, which could be resolved through mean point clustering. We also note that we observed a 100% success rate in following all planned paths.

Full results for the real-world experiments are shown in Tables 5.5, 5.6, 5.7, 5.8. Sentences were specifically chosen to contain different action verbs which highlights the many-to one action mappings LLMs are capable of generating.

5.4.2 Qualitative Examples:

Leveraging the vast knowledge base encoded into LLMs enables a degree of spatial reasoning and the ability to infer navigation instructions based on the context of the sentence. For example, take the sentence “Find me something to help a firefighter” as shown in Figure 5.1 requires the robot to identify a list of objects that could help a firefighter and then look for them. From the snippet of the generated code, we see that the robot successfully looks for fire extinguishers and fire hydrants, both of which would aid a firefighter. Additionally, we are able to ask the robot to “Find something to clean up a mess” and it will find a mop or a broom.

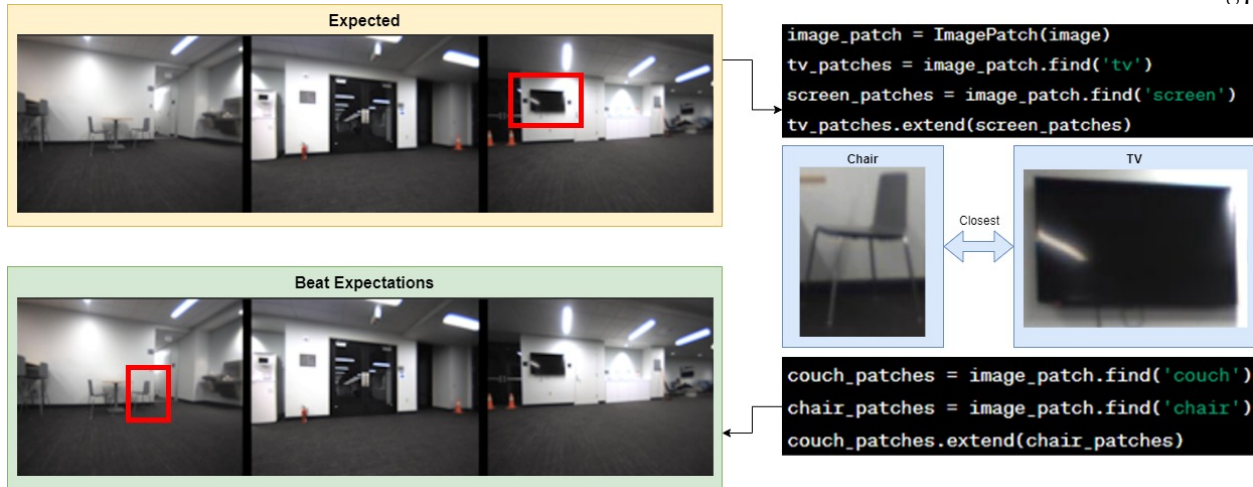


Figure 5.5: Results for the sentence “Find me somewhere to watch a movie.”

What’s more remarkable than simply interpreting correct navigation commands from context is our framework’s ability to leverage foundation models to find more refined solutions to a task in some instances. Take for example the sentence “Find me somewhere to watch a movie,” we would expect the robot to go and find a TV or some kind of screen. Of course, the first object the robot looks for is the screen but more remarkably the robot also looks for a chair which is closest to the screen. We can see this in Figure 5.5 where the generated layer θ explicitly looks for “TVs, screens” and then looks for “chairs” and “couches.”

As powerful as the contextual engine (GPT-3.5) is in our framework, it still has limitations when interpreting the physical world. In Figure 5.6 we tell the robot to “Run upstairs.” and to no surprise, it will successfully generate code and plan up the stairs. However, when given the utterance “Go to the second floor.” the following code is generated:

```

floor_patches = ImagePatch(image).find('floor')
floor_patches.sort(key=lambda x: x.vertical_center)
if len(floor_patches) < 2:
    return {'function': 'None', 'error': 'Image does not contain at least two
        floors.'}
second_floor_patch = floor_patches[1]

```

Table 5.5: Scene-01: Theater

Category	Scene	Code	OD	WP	Path& Exec	OFA	OFA Expression	OFA+ LLM
Generic	Drive to the helmet	Pass	Pass	Pass	Pass	Fail	Helmet	Fail
Generic	Navigate to the mop	Pass	Pass	Pass	Pass	Fail	Mop	Fail
Generic	Go to the vacuum	Pass	Fail	Fail	Fail	Fail	Vaccum	Fail
Generic	Walk to the table	Pass	Pass	Pass	Pass	Fail	Table	Pass
Generic	Go to the fire extinguisher	Pass	Pass	Pass	Pass	Fail	Fire extin- guisher	Fail
Generic	Walk to the saw	Pass	Fail	Fail	Fail	Fail	Saw	Fail
Specific	Go to the red backpack	Pass	Pass	Pass	Pass	Fail	Red backpack	Fail
Specific	Move to the orange bucket	Pass	Pass	Pass	Pass	Fail	Orange bucket	Fail
Specific	Drive to the chair on the right	Pass	Pass	Pass	Fail	Fail	Chair on the right	Fail
Specific	Walk to the white fan	Pass	Pass	Pass	Pass	Fail	White fan	Fail
Specific	Navigate to the black fan	Pass	Pass	Pass	Pass	Fail	Black fan	Fail
Specific	Navigate to the black speaker	Pass	Pass	Pass	Pass	Fail	Black speaker	Pass
Specific	Run to the red bag	Pass	Pass	Fail	Fail	Fail	Red bag	Fail
Specific	Move to the black chair in front of you	Pass	Pass	Pass	Pass	Fail	Black chair in- front of you	Fail
Specific	Walk to the blue chair	Pass	Pass	Pass	Pass	Fail	Blue chair	Fail
Relational	Go to the chair with the black backpack on it	Pass	Pass	Pass	Pass	Pass	Chair with black backpack	Pass
Relational	Go to the chair with the helmet on it	Pass	Pass	Pass	Pass	Fail	Helmet Chair	Fail
Relational	Go to the backpack next to the chair	Pass	Fail	Fail	Fail	Fail	Backpack next to the chair	Fail
Relational	Go to the backpack in front of the ladder	Pass	Pass	Pass	Pass	Fail	Backpack in front of ladder	Fail
Relational	Run to the person on the ladder	Pass	Pass	Pass	Pass	Fail	Person on the ladder	Fail
Relational	Drive to the man sitting on the table	Pass	Pass	Pass	Pass	Fail	Man sitting on table	Fail
Relational	Go to the third chair from the left	Pass	Pass	Pass	Pass	Fail	Third chair from left	Fail
Contextual	Go to something that can carry water	Pass	Pass	Pass	Pass	Fail	Water con- tainer	Fail
Contextual	Navigate to something that can wash the floor	Fail	Fail	Fail	Fail	Fail	Mop	Fail
Contextual	My floor is dirty. Go to something that can fix this	Fail	Fail	Fail	Fail	Fail	Mop	Fail
Contextual	Go somewhere that I can throw something away in	Pass	Pass	Pass	Pass	Fail	Trash can	Pass
Contextual	Walk to something that put out a fire	Pass	Fail	Fail	Fail	Fail	Fire extin- guisher	Fail
Contextual	Go to something that can cool me down	Pass	Fail	Fail	Fail	Fail	Fan	Fail
Contextual	Go to something white that will cool me down	Fail	Fail	Fail	Fail	Fail	Fan	Fail
Contextual	I am in the mood to listen to music. Go to something that can do that	Pass	Fail	Fail	Fail	Fail	Speaker with a music player	Fail

Table 5.6: Scene-02: Lobby

Category	Scene	Code	OD	WP	Path& Exec	OFA	OFA + LLM Expression	OFA+ LLM
Generic	Run to the trash bag	Pass	Pass	Pass	Pass	Pass	Trash bag	Pass
Generic	Navigate to the water fountain	Pass	Fail	Fail	Fail	Fail	Water fountain	Fail
Generic	Drive to the broom	Pass	Pass	Pass	Pass	Fail	Broom	Fail
Generic	Walk to the monitor	Pass	Pass	Fail	Fail	Fail	Monitor	Fail
Specific	Walk to the tiny monitor	Fail	Fail	Fail	Fail	Fail	Tiny monitor	Fail
Specific	Walk to the smallest monitor	Pass	Pass	Pass	Pass	Fail	Smallest monitor	Fail
Relational	Walk to the right most cone	Pass	Pass	Pass	Pass	Fail	Rightmost cone	Fail
Relational	Walk to the table with a can on it	Pass	Fail	Fail	Fail	Fail	Can on the table	Pass
Relational	Walk to the table next to the red backpack	Pass	Pass	Pass	Pass	Fail	Table next to red backpack	Pass
Relational	Go to the blue chair with the backpack on it	Pass	Pass	Pass	Pass	Pass	Blue chair with backpack	Pass
Relational	Walk to the leftmost table	Pass	Fail	Fail	Fail	Fail	Leftmost table	Fail
Relational	Go to the backpack closest to the shoes	Pass	Pass	Pass	Pass	Pass	Backpack closest to shoes	Pass
Relational	Walk to the shoe next to the red backpack	Pass	Pass	Pass	Pass	Fail	Shoes next to red backpack	Fail
Relational	Remove the trash bag from the floor	Fail	Fail	Fail	Fail	Fail	Trash bag on the floor	Pass
Relational	Walk to the table with the monitor on it	Fail	Fail	Fail	Fail	Pass	Table with monitor	Pass
Relational	Drive to the closest monitor to the table	Pass	Pass	Pass	Pass	Fail	Closest monitor to the table	Fail
Relational	Go to the table with the bottle on it	Fail	Fail	Fail	Fail	Fail	Table with the bottle	Fail
Relational	Go to the bottle on top of the table	Pass	Pass	Pass	Pass	Fail	Bottle on top of the table	Fail
Relational	Go to the tv closest to the person	Pass	Pass	Pass	Pass	Fail	TV closet to person	Fail
Relational	Go to the person with the blue shirt	Pass	Fail	Fail	Fail	Fail	Person with the blue shirt	Fail
Relational	Run to the chair with the blue coat on it	Pass	Fail	Fail	Fail	Fail	Run to the chair with the blue coat	Fail
Relational	Move to the backpack on the chair	Fail	Fail	Fail	Fail	Fail	Backpack on chair	Fail
Relational	Move to the black backpack on the chair	Fail	Fail	Fail	Fail	Fail	Black backpack on the chair	Fail
Relational	Drive to the chair with the backpack on it that is not red	Fail	Fail	Fail	Fail	Fail	Backpack on the chair	Fail
Contextual	Find something that can help a firefighter	Pass	Pass	Pass	Pass	Fail	Fire extinguisher	Fail
Contextual	Go to something that can clean a dirty floor	Fail	Fail	Fail	Fail	Fail	Mop	Fail
Contextual	Go get a drink of water	Fail	Fail	Fail	Fail	Fail	Water bottle	Fail
Contextual	Go to a place where I can watch a movie	Pass	Pass	Pass	Pass	Fail	Movie theater	Fail
Contextual	Drive to a place where I can watch a video	Fail	Fail	Fail	Fail	Fail	TV or screen	Fail

Table 5.7: Scene-03: Outdoor

Category	Scene	Code	OD	WP	Path& Exec	OFA	OFA + LLM Expression	OFA+ LLM
Generic	Wander to the fire hydrant	Pass	Pass	Pass	Pass	Fail	Fire hydrant	Fail
Generic	Step towards the grill	Pass	Fail	Fail	Fail	Pass	Grill	Pass
Generic	Walk to the skateboard	Pass	Pass	Pass	Pass	Fail	Skateboard	Fail
Generic	Walk to the bike	Pass	Pass	Fail	Fail	Fail	Bike	Fail
Generic	Walk to the bike	Pass	Pass	Pass	Pass	Fail	Bike	Fail
Generic	Go to the bike rack	Pass	Pass	Pass	Pass	Fail	Bike rack	Fail
Generic	Go to the sign	Pass	Pass	Fail	Fail	Fail	The sign	Fail
Generic	Go to the bench	Pass	Pass	Pass	Pass	Fail	Bench	Fail
Specific	Sashay to the stop sign	Pass	Pass	Pass	Pass	Fail	Stop sign	Fail
Specific	Go to the basketball hoop	Pass	Pass	Pass	Pass	Fail	Basketball hoop	Pass
Specific	Roam towards the blue car	Pass	Pass	Pass	Pass	Fail	Blue car	Pass
Specific	Trot towards the red bag	Pass	Pass	Pass	Pass	Fail	Red bag	Fail
Specific	Go to the red object	Fail	Fail	Fail	Fail	Fail	Red object	Fail
Relational	Proceed to the middle cone	Pass	Pass	Pass	Pass	Pass	Middle cone	Pass
Relational	Journey to the tree next to the backpack	Fail	Fail	Fail	Fail	Fail	Tree next to the backpack	Pass
Relational	Trek to the backpack by the tree	Pass	Pass	Pass	Pass	Fail	Backpack by the tree	Fail
Contextual	A firefighter needs water. Walk to a source of water	Pass	Fail	Fail	Fail	Fail	Source of water	Fail
Contextual	Head towards something that can help firefighters	Pass	Pass	Pass	Pass	Fail	Fire hydrant	Pass
Contextual	You are a dog that needs to mark its territory. Go find a place to do this	Pass	Pass	Pass	Pass	Fail	Fire hydrant	Fail
Contextual	You are carrying trash, Find somewhere to dump it	Pass	Pass	Pass	Pass	Fail	Dumpster	Pass
Contextual	Find me something to do a kickflip on	Pass	Pass	Pass	Pass	Fail	Skateboard	Fail
Contextual	I want to shoot some hoops. Take me there	Fail	Fail	Fail	Fail	Fail	Basketball court	Fail
Contextual	Move towards a faster mode of transportation	Pass	Pass	Pass	Pass	Fail	Sorry I can't generate a referring expression	Fail
Contextual	Head to the fastest mode of transportation	Pass	Pass	Pass	Pass	Pass	Fastest mode of transportation	Pass

This code is nonsensical to the actual task because it's a literal interpretation of the sentence. Counting floors has nothing to do with going to the second story of a building. Examples like these highlight the need for additional methods of encoding 3D spatial reasoning into natural language-based embodied navigation frameworks.

Limited testing showed that GPT-4 can improve these types of scenarios but generated sequences were insufficient to perform the task. This was primarily due to a lack of understanding where a staircase but these types of examples highlight that more work needs to be done in encoding spatial priors into language-based representations.



Figure 5.6: Scene used to tell the robot to “Run upstairs” and “Go to the second floor”. The second sentence fails due to the code trying to “count floors”

5.5 Conclusion and Limitations

Our composable framework shows remarkable accuracy and performance in generating code that fits the navigation task. In controlled experiments, we achieve 90% accuracy across 50 different sentences. When taken to the real world the, model still performs exceptionally well with zero additional input. This is highlighted by a 70% execution accuracy in large outdoor environments.

While this work makes significant strides toward embodied navigation, we observe many opportunities for future effort. Namely, operating in only 2D space creates challenges for spatial relationships such as “behind” the robot or “to the right of.” Additionally, we are limited by the visual range of the robot: therefore 3D projections need to be exponentially more accurate for both the smaller and further away an object is in the given input image. Large multimodal foundation models are promising, but promise uncertain runtime performance given their lack of availability and scalability. Finally, the current implementation of the framework is limited to landmark-based navigation. Chapter 6 will focus on an alternative framework that is not limited to landmark-based navigation. In this Chapter, we have shown that we can effectively ground language through code to perform landmark-based navigation. Moreover, the generated code sequences can resolve geometric constraints between objects in the environment, and the robot.

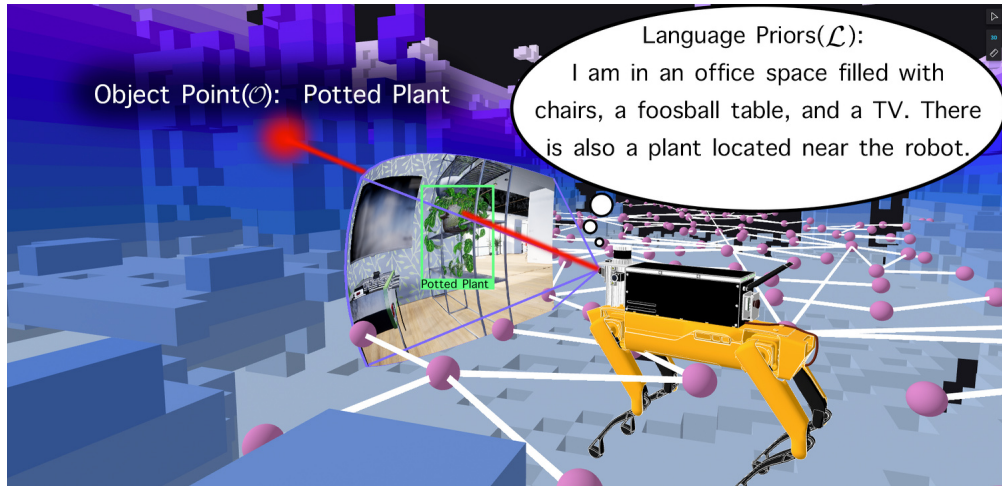
Table 5.8: Scene-04: Courtyard

Category	Scene	Code	OD	WP	Path& Exec	OFA	OFA + LLM Expression	OFA+ LLM
Generic	Run to the door	Pass	Pass	Pass	Pass	Fail	Door	Fail
Generic	Run towards the backpack	Pass	Pass	Pass	Pass	Fail	Backpack	Fail
Generic	Drive to the wagon	Pass	Pass	Pass	Pass	Fail	Wagon	Pass
Generic	Navigate to the stairs	Pass	Pass	Pass	Pass	Pass	Stairs	Pass
Specific	Proceed towards the garbage can on the right	Pass	Pass	Pass	Pass	Fail	Garbage can on the right	Fail
Specific	Stroll to the recycle bin on the left	Pass	Pass	Fail	Fail	Fail	Recycle bin on the left	Fail
Specific	Sprint to the picnic table	Pass	Pass	Pass	Pass	Fail	Picnic table	Fail
Relational	Go to the bench with water container on it	Pass	Pass	Pass	Pass	Pass	Bench with water container	Pass
Relational	Walk to the bench with nothing on it	Pass	Pass	Pass	Pass	Fail	Bench with nothing on it	Pass
Relational	Proceed to the bench with most objects	Fail	Fail	Fail	Fail	Fail	Bench with the most objects	Pass
Relational	Move towards the backpack farthest from a bench	Pass	Fail	Fail	Fail	Fail	Backpack farthest away from bench	Fail
Relational	Head towards the middle cone	Fail	Fail	Fail	Fail	Fail	Middle cone in row of cones	Fail
Relational	Head towards the middle cone in the row of cones	Pass	Pass	Pass	Pass	Fail	Middle cone in row of cones	Fail
Relational	Go to the table with only one chair	Fail	Fail	Fail	Fail	Pass	Table with only one chair	Pass
Relational	Go to the table with only one chair. There are multiple groups of chairs around multiple tables	Pass	Pass	Pass	Pass	Pass	Table with only one chair	Pass
Relational	Step towards the column closest to the cart	Pass	Pass	Pass	Pass	Fail	Column closest to the cart	Pass
Relational	Move to the largest group of benches	Pass	Pass	Pass	Pass	Fail	Largest group of benches	Fail
Relational	Walk towards the table with the umbrella	Pass	Pass	Pass	Pass	Fail	Umbrella on the table	Pass
Relational	Walk towards the table with the yellow umbrella	Pass	Pass	Pass	Pass	Fail	Table with yellow umbrella	Fail
Relational	Drive to a table without any chairs	Pass	Pass	Pass	Pass	Fail	Table without chairs	Fail
Relational	Walk to black table with six chairs	Fail	Fail	Fail	Fail	Fail	Black table with six chairs	Fail
Relational	Walk to the table with the most chairs	Pass	Pass	Pass	Pass	Fail	Table with the most chairs	Fail
Relational	Navigate to the table with the backpack	Fail	Fail	Fail	Fail	Fail	Backpack on the table	Fail
Contextual	Go to the nearest entrance to the building	Fail	Fail	Fail	Fail	Fail	Nearest entrance to the building	Fail
Contextual	Go to something that you would hold open for someone elderly	Pass	Fail	Fail	Fail	Fail	Door	Fail
Contextual	Go to something that will make it easier to carry heavy luggage	Pass	Fail	Fail	Fail	Pass	Cart	Pass
Contextual	Go to somewhere I can eat my lunch	Pass	Fail	Fail	Fail	Fail	Lunch table	Pass
Contextual	Go up to the second floor	Fail	Fail	Fail	Fail	Fail	Second floor	Fail
Contextual	Go find something to climb	Fail	Fail	Fail	Fail	Fail	Rock wall	Fail
Contextual	Run upstairs	Pass	Pass	Pass	Pass	Fail	Stairs	Fail
Contextual	Find me somewhere to park my bike	Pass	Pass	Pass	Pass	Fail	Bike rack	Pass

Chapter 6

CogExplore: Planning Based On Contextual Cues with Temporal Reasoning

Figure 6.1: Spot characterizing its environment through its VQA model (Language Priors), searching for specific objects with its object detection model and creating projections (Object Points shown in red) surrounded by a set of navigable graph points (shown in purple).



Exploring unmapped environments is paramount to modern robotic applications, such as search-and-rescue and assistive robotics as seen in the DARPA Subt challenge [28]. In Chapter 2 and [14, 130, 1] we see that state-of-the-art methods have made incredible progress in exploring challenging search and rescue scenarios as seen in, they typically rely on hand-engineered heuristics based on the geometry of the environment, e.g. optimizing for volumetric gain. Such hand-engineered features fail to incorporate the rich semantics and contextual cues of an environment [13]. As seen in Chapter 5 code-generation from natural language can be an effective means of grounding navigation commands to landmarks within a robot's environment. However, a key limitation of

the method presented in [13] is the reliance on objects being within the robot’s field of view. In this Chapter, we develop a framework to address this limitation through natural-language-based encodings.

Given the problem of exploring a previously unknown environment and looking for an object of interest, we develop a framework, Contextual Exploration (Cog Explore), that leverages foundation models, or large language models (LLMs) with natural-language-based representations of the environment to perform navigation using both geometric and contextually rich features, as well as temporal states.

To effectively leverage natural language, we must ground language utterances to the physical world in which the robot operates. Various approaches have been used to associate language with the physical domain, ranging from probabilistic graph-based structures [125, 46, 59] to end-to-end learning-based methods [31]. We represent the grounding problem as a probabilistic set of planning points, object points, and language priors obtained from a set of vision models. Our framework then selects the next best location to explore given this set of priors and a goal. Specifically, we leverage foundation models [35] [54] which have shown the ability to exhibit remarkable contextualization and reasoning by utilizing efficient next token predication.

Navigation solutions powered by LLMs offer distinct advantages over metric-based methods such as frontier point finding [140]. For instance, a robot operating in a school instructed to “Go find a whiteboard” would spend equal time exploring a bathroom or a classroom when using a frontier-based planner. However, by integrating an LLM’s world knowledge into search strategies, the resulting planning method can leverage the fact that erasers are far more likely to be found in a classroom. Moreover, we can rely on other cues in the environment to influence the search behavior. For example, if a whiteboard is present, the robot should be more exploitative than explorative. This adaptive form of reasoning, aiming for the most plausible explanation—or abductive reasoning allows CogExplore to create a form of working memory.

LLMs have shown remarkable capabilities in various domains, yet they do not explicitly encode a time dimension, meaning their appreciation of time is implicit [29]. Specifically, LLMs

encode memories to be used in future reasoning implicitly as an autoregressive relation instead of an explicitly modeled recurrent relation [134]. We introduce a method to simulate working memory by compressing prior states and justifications into a concise linear log. By leveraging temporal and contextual reasoning, CogExplore guides robots to navigate diverse environments, while adapting the exploration strategy as new information is acquired.

Our method presents possibilities for advances in scenarios where environments are semantically rich, but of which very little is known a priori. Such environments included domestic interiors, where robots might be asked to help locate and retrieve household items (*Bring the groceries in from the garage*), identify conditions (*Check if the oven is on*), or interact with humans (*Open the front door and guide the people to my room*). We validate CogExplore’s exploration capabilities with 210 simulations of 45 minutes each operating in 3 different environments across 7 different tasks and show the method is capable of performing temporal, geometric, and semantic reasoning.

6.1 Related Works

Navigating with language-based representations requires grounding utterances in the physical environment. Various approaches have been used to associate language with the physical domain, ranging from probabilistic graph-based structures [125, 46, 59] to end-to-end learning-based methods [31]. Graph-based approaches have shown promising results but their generalizability is limited to a fixed training corpus. On the other hand, Large Language Models (LLMs) have proven to be adept at reasoning over large amounts of unsupervised training data [35, 127] using transformer-based backends [133]. These models, also known as foundation models due to their comprehensive knowledge, have recently been leveraged for task and motion planning using reinforcement learning on a set of skills [15], and in an end-to-end fashion [31] with object scene representation transformers [106]. While these approaches have made remarkable strides, they still suffer from the explainability problem and further, they don’t share any of the path planning guarantees that traditional planning methods offer [86].

More recent work in robotic navigation has seen significant advancements with the incorpo-

ration of LLMs [143] and semantic cues. The approach of utilizing LLMs was first introduced as a heuristic planner constructed for manipulation tasks [6]. In [117] a novel approach that utilizes semantic guesswork from foundations as heuristics for planning in unexplored environments is introduced. Unlike traditional methods that rely on geometric or direct instruction-based navigation, their methodology leverages the semantic understanding of LLMs to guide exploration indirectly, without requiring direct knowledge of the physical environment.

ORION (Open-world Interactive Personalized Navigation) is presented in [24], which integrates LLMs for personalized, interactive object navigation. ORION focuses on natural language interaction and personalization, proposing a task where robots navigate to user-specific objects in unseen environments through conversation. The framework comprises modules for perception, navigation, communication, and decision-making, and showcases the ability of LLMs to aid in robotic navigation through conversation.

Embodied evaluation of actions across objects with specific affordances in reference to natural language instruction has been explored in [108] using code generation. Language models contain sufficient world information to decompose basic tasks [49] but must be evaluated according to the metrics of executability and correctness. This work highlights the importance of assessing the practical applicability of LLM-generated instructions in real-world robotic scenarios.

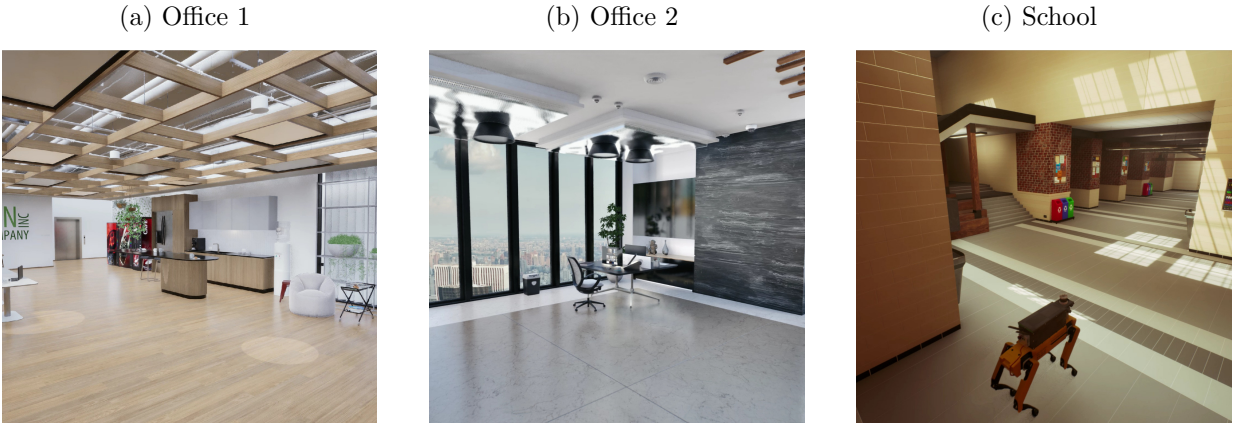
Alternative approaches include the Language Trajectory Transformer (LATTE) [17] and Cliport [121], which use end-to-end mechanisms to encode user intentions and translate directly to robot action. CLIPort encodes spatial and semantic attributes separately using individual U-Nets, combining them in the final layer to inform trajectory generation. Imitation learning, as seen in Hierarchical Universal Language Conditioned Policies (HULC) [80], uses language conditioning to construct controls for HULC using the semantic knowledge of VAPO, learning efficiently from unlabeled examples.

Prompt engineering plays a crucial role in tackling highly specific robotic navigation tasks. In [135], Vemprala et al. introduce PromptCraft, a mechanism for handling the complexity inherent in the myriad ways of describing any one task in natural language. By carefully constructing prompts

that elicit the desired behavior from LLMs, PromptCraft aims to bridge the gap between natural language instructions and executable robot actions.

In [115] an approach called Language Frontier Guide (LFG) is presented which leverages the semantic reasoning capabilities of LLMs to guide robotic exploration in novel environments. Rather than directly following the narratives generated by LLMs, which can be ungrounded and arbitrarily wrong, LFG incorporates LLM inferences as heuristic scores into a search-based planning framework. By polling the LLM multiple times with task-relevant prompts and chain-of-thought reasoning, LFG empirically estimates the likelihood of subgoals leading to the goal query. These semantic “guesswork” scores are then combined with traditional geometric heuristics to bias the search toward a promising frontier point. The authors demonstrate that this approach outperforms both uninformed exploration and other LLM-based methods on the challenging Habitat ObjectNav benchmark and in real-world environments.

Figure 6.2: Renderings from Unreal Engine environments. An example of a simulated spot can be seen in the rightmost image.



While LLMs excel at reasoning over unstructured data [35, 127, 124, 146] using transformer architectures [133], guiding them to desired outcomes and maintaining temporal coherence remains an open problem. Recent works integrating LLMs with object scene representation transformers for task and motion planning [15, 31, 108] still lack the interpretability and guarantees of traditional methods [86].

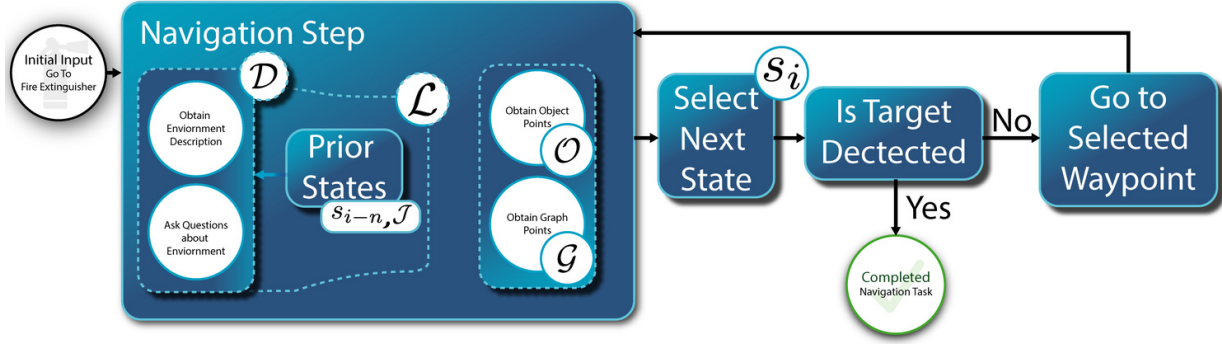
6.2 Methodology

The semantic reasoning abilities of LLMs have been leveraged to guide exploration and planning in novel environments [115, 117]. LLMs have also been utilized for semantic grounding in complex outdoor environments using contextually relevant instructions [117]. Our method introduces new capabilities to this line of work, allowing LLMs to selectively encode relevant states based on environment descriptions and then reason directly over these natural language-encoded states, along with temporal states derived from a log of past movements.

The proposed exploration framework, CogExplore, represents relevant features of the environment as natural language strings. Formally, we represent possible states for the robot as, $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ where s_i is a possible state for the robot. At each state, the robot has a set of planning graph points $\mathcal{G} = \{g_1, g_2, \dots, g_k\}$, a set of object points $\mathcal{O} = \{o_1, o_2, \dots, o_m\}$ and a set of language priors $\mathcal{L}$. Graph points represent traversable areas of the environment, and we represent them as a list in text from $\mathcal{G}_i = \{x, y, z\}$. Object points are represented similarly but with a corresponding label and probability $\mathcal{O}_i = \{x, y, z, c, p\}$ where c is the class and p is the confidence (represented as a probability), obtained from an open vocabulary object detector. Specifically, we utilize YoloWorld [20] and Segment Anything [58] to project open vocabulary detections into 3D points. Examples of $\mathcal{G}$, $\mathcal{L}$, and $\mathcal{O}$ can be seen in Figure 6.1 and full details of, the open vocabulary 3D detection system can be found in Chapter 2.5. Note, YOLO v3 [102] is replaced with YOLO World to support open vocabulary detection [20].

The language priors, $\mathcal{L}$, consist of three components; environmental descriptions, prior robot states, and the justification for choosing the next state. Environment descriptions $\mathcal{D}$ are obtained from a series of questions generated by a foundation model and answered using a multimodal visual question and answering model (VQA) called LLaVA -1.5 [71], as shown in Figure 6.3. Prior states (s_{i-n}) and the justification for choosing the state ($\mathcal{J}_{i-n}$) and the full set of language priors is $\mathcal{L} = \{\mathcal{J}_{i-n}, s_{i-n}, \mathcal{D}\}$ along with the model’s justification for selecting the state. To select the next

Figure 6.3: CogExplore System Diagram. All prior information is encoded into text strings for use with language-based foundation models



best state, we can model the task as a maximum likelihood estimate:

$$\arg \max_{s_1, s_2, \dots, s_t} P(s_g | s_t, \mathcal{S}, \mathcal{O}, \mathcal{G}, \mathcal{L}) \prod_{i=1}^t P(s_i | s_{i-1}, \mathcal{S}, \mathcal{O}, \mathcal{G}, \mathcal{L}) \quad (6.1)$$

At each iteration, the foundation model is asked to direct the robot to the state most likely to find the goal state, s_g given the robot's past states and any new observations. The process repeats until the robot arrives at s_g . Specifically, once the robot reaches a desired waypoint, the model is tasked to choose the next waypoint given all of the prior observations and a truncated list of past states

6.2.1 Exploration Planner

CogExplore's autonomous navigation uses a state-of-the-art graph-based exploration planner with frontier point finding [27] to help the language model select reliable states. This planner uses a similar approach to the one presented in Chapter 2 but comes from Team Cerberus [130], the number one performing team at the DARPA SubT challenge. This version [27] of the graph-based planner was chosen due to a more efficient implementation for waypoint-based navigation. Real-time mapping is performed using the voxel-based map [45].

At each planning iteration, a set of sparse global graph points and dense local points are generated. We employ a 2D Gaussian function to sparsity the distribution of these points, where

weights are assigned based on their proximity to the robot’s current location. The result is a distribution of low-density points in distant areas with a high concentration of points near the agent allowing CogExplore to plan with high granularity locally while still having the ability to explore regions of interest that are further away. The total sets of points are encoded into $\mathcal{G}$. In practice, we find using a list of 50 graph points works well for the framework.

6.2.2 Foundation Model Prompting Schemes

We rely on language foundation models to perform four tasks: Generating questions about the environment, generating object labels, compressing state information, and performing state selection. Foundation models are highly sensitive to the particular presentation of the underlying information [19]. To assist the model’s geometric reasoning, we label newly identified graph points ($\mathcal{G}$) as if they are in an unexplored region. A point is labeled as new if it is beyond a certain distance threshold from all existing points in the graph. This distance threshold is determined using a KD-Tree [11] structure to efficiently compute the nearest neighbors.

Specifically, we encode each graph points $\mathcal{G}$ into natural language. The relative position of each point to the robot is also provided. Relative positions are computed based on quadrants (front, left, right, behind) using the angle of the point from the robot. An example is shown below:

```
1: Graph new - x=-18.17, y=-10.23, z=0.02 (new) - Distance: 9.47 meters - Relative Position:
right (-65.07)
2: Graph new - x=-32.88, y=-1.42, z=0.02 (new) - Distance: 10.35 meters - Relative Position:
behind (174.99)
...
24: Graph - x=-18.02, y=-4.28, z=-0.06 - Distance: 5.16 meters - Relative Position: front
(-30.95)
25: Graph - x=-7.38, y=-1.56, z=0.04 - Distance: 15.17 meters - Relative Position: front
(-1.61)
```

Object points are encoded similarly except with the added label of the object. Note, the object point is an estimate of the object centroid obtained from the Segment Anything model [58] and projected into 3D space using the method described in Chapter 2. An example of encoded object points can be seen below:

51: Object TV - x=-9.075, y=-0.38, z=0.68 - Distance: 6.64 meters - Relative Position: front
(3.97)

52: Object chair - x=-8.475, y=-0.38, z=0.22 - Distance: 7.04 meters - Relative Position: front
(0.25)

...

All of our prompts are highly modular in construction allowing us to incorporate custom visual grounding information ($\mathcal{O}, \mathcal{L}$), prior states s_{i-n} , navigation points ($\mathcal{G}$), and specific instructions designed to handle the strengths and weaknesses of a particular model based on empirical observations. Once the prompts were tuned, the same prompts and additions were kept consistent for all experiments.

A specific behavior that was observed with GPT-3.5 was that the model would occasionally select waypoints near windows when a potential object was visible through the transparent glass. To mitigate this, we explicitly informed the model that windows might obscure routes to prospective targets, encouraging more practical waypoint choices. In the case of Claude Haiku, we found that by default the model was quite conservative in its exploration behavior especially before semantic cues were discovered. Explicit language was added to prioritize the exploration of new areas. We used the same prompts across all experiments, and the full prompt details can be found in Section A.2 of the Appendix.

For each iteration of the exploration cycle, the LLM is given a form to fill out, which creates a set of fields for the next waypoint output, as well as fields for a characterization of the environment and a justification field to insert reasoning for the selection of the particular waypoint it chose. All of our prompts follow this “mad lib” style form:

I need to select a waypoint from a numbered list of graph points, frontier points and object points. The point I have selected is point number: [insert the graph, frontier, or object point numeral number here]. I am selecting this point because I believe it makes strategic sense to get me closer to solving my navigation task “INSERT_QUERY_HERE”. My environment can be described as [describe the environment]. My reasoning is that this point [insert your reasoning here].

Once the generation is complete, the LLM is queried again to compress the prior output into a concise state of 50-100 words ($\mathcal{J}$) these compressed states with justifications enable CogExplore to reason over its past states, facilitating a contiguous exploration process. Each of these logs is appended to a memory window with some fixed length, in practice we discovered a length of 10 was appropriate for both GPT-3.5-Turbo and Claude Haiku.

An example compressed log can be seen below:

...

4. The sim was running for this much time: 93.20000267028809, and was at position: x=18.291, y=2.82, z=0.35 when the following occurred: I went to point x=13.76, y=2.3, z=0.06 in reference to the "Go to the coffee table" navigation task. I performed the action of selecting a new, unexplored waypoint that was relatively close to my current position and located in front of me. I did this because I believed it would help me by gaining a fresh perspective on the environment and potentially discovering new vantage points or areas that may contain the target object. The environment I was observing was a domestic setting with large windows, natural light, and some greenery like a palm tree and potted plants, but no visible furniture. This suggested that the coffee table I was searching for was likely in a different, unexplored region of the environment.

5. The sim was running for this much time: 116.06667137145996, and was at position: x=15.557, y=2.62, z=0.35 when the following occurred: I went to point x=7.52, y=-0.12, z=0.05 in reference to the "Go to the coffee table" navigation task. I performed the action of

selecting a new waypoint, as it was labeled as an unexplored area. I did this because I believed it would help me by gaining a fresh perspective on the scene and potentially discovering new vantage points or areas that may contain the target object. The environment I was observing can be characterized as a domestic setting with large windows and a long hallway, with a table visible in the scene. The distance to this point was relatively close to my current position at 8.5 meters, allowing me to efficiently explore a new region of the environment.

...

6.3 Experimental Setup

We evaluate the performance of CogExplore’s ability to explore and find a target object using a simulated Boston Dynamics Spot in three different environments using Unreal Engine [34]. We compare CogExplore against an A^* direct path metric and the Vision-Enabled Frontier Exploration Planner (VEFEP).

VEFEP calculates the exploration gain heuristic g for a given path σ_i by summing the volumetric gain **VG** weighted by a function for each vertex in the path [10]. Parameters for VEFEP are based on the ones used by the authors in [27]. This planner explores until it detects the target object using the same 3D open Vocabulary detector used by CogExplore and described in section 2.5. In the case of VEFEP, the object detection is only looking for the target object which is passed into the pipeline at the beginning of the task.

Two office scenes (Office 1, Office 2) and a more complex school environment (School) were designed, which provided the structure for a total of 7 object-retrieval scenarios to be tested, as shown in Table 6.1. In Office 1, there were three tasks involving locating a fire extinguisher in different locations, denoted as FE1, FE2, and FE3. Office 2 featured two tasks: finding a coffee table (CT) and an office chair (OC). In School, the tasks were to locate a whiteboard (WB) and a bookshelf (BS).

For each task, we simulate the navigation capabilities of a Boston Dynamics Spot and allow the robot to explore for 45 minutes. A run terminates once the target object is found. We conduct 15 trials per task on Azure “NC64as.T4.v3” instances with 4 Nvidia T4 GPUs. We utilize 3 GPUs

per simulation, one for the unreal simulator, one for the object detector, and one for the VQA model. We note that for Office 1, and the school scene, the simulator runs in real-time. While in Office 2 the simulator runs in half-time due to higher quality graphical assets. While CogExplore does run in real-time, with the assistance of cloud computing we find runtimes are not an accurate measurement of exploration performance due to uncontrollable factors such as latency and API call throttling from leading foundation model providers. Instead, we evaluate CogExplore’s performance based on path efficiency, e.g., the shortest path taken to discover the target object.

Table 6.1: Simulation Environments and Corresponding Tasks. Each run was given 45 minutes of simulation time to complete. None of the CogExplore runs timed out, and the VEFEP timeouts are shown here.

Office 1 (572 m^2): Figure 6.2a			Office 2 (1450 m^2): Figure 6.2b			School (1287 m^2): Figure 6.2c		
Task	Direct Path (m)	VEFEP # Timeout	Task	Direct Path (m)	VEFEP # Timeout	Task	Direct Path (m)	VEFEP # Timeout
Fire Extinguisher 1 (FE1)	25.2	0	Office Chair (OC)	33.7	3	Whiteboard Eraser (WB)	26.1	5
Fire Extinguisher 2 (FE2)	27.1	0	Coffee Table (CT)	26.2	6	Book-shelf (BS)	13	2
Fire Extinguisher 3 (FE3)	15.6	0						

We compare the performance of CogExplore running with Anthropic’s **claude-3-haiku-20240307** (CE-H) and OpenAI’s **gpt-3.5-turbo-0125** (CE-3.5) as the backend against VEFEP. Starting positions and goal states are kept consistent across all experiments.

6.4 Experimental Results

From Figure 6.5 we observe that in Office 1, the smallest of the three environments at 572 m^2 VEFEP, CE-H, and CE-3.5 all have similar performance. This environment was specifically kept small to evaluate when the need for more advanced semantic reasoning such as the one utilized by CogExplore is required. In the task “Go to the fire extinguisher” with configuration 1 we see in Table 6.2 that CE-H had the lowest mean distance of 62.3m for the task. CE-3.5 and VEFEP had similar performance for this task with path length means of 112.5 m and 132.3 m respectively. A birds-eye view for this configuration is provided in Figure 6.7.

Figure 6.4: Example Runs demonstrating geometric reasoning (left), semantic reasoning (center), and temporal reasoning (right).

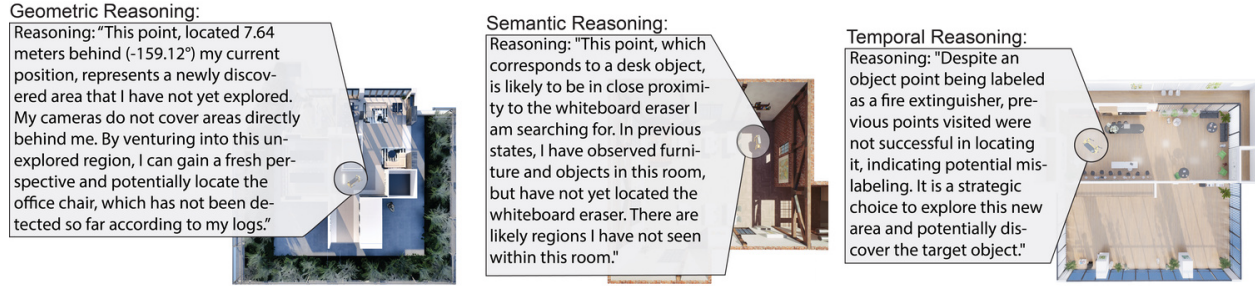
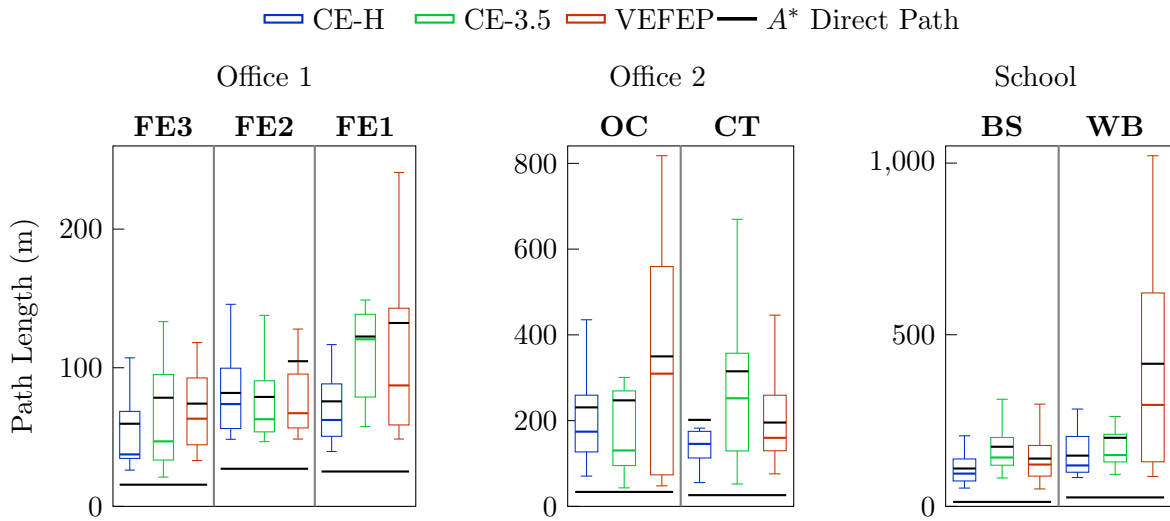


Figure 6.5: Path length comparisons for each method (CE-3.5, CE-H, VEFEP) on completing each of the seven tasks. Office Environment 1 is on the left, Office Environment 2 is in the middle and the School Environment is on the right. The short black line for each whisker plot is the mean while the long black line at the bottom represents a direct path assuming a known map.



For configuration 2 (Figure 6.8), CE-H and CE-3.5 had nearly identical performance with path length means of 78.9 *m* and 81.8 *m*, as seen in Table 6.3. VEFEP performed slightly worse on average with a mean of 104.7. However, all methods were able to find efficient routes with minimum path lengths of 46.7 for CE-3.5, 48.4 for CE-H, and 48.6 for VEFEP.

A similar trend to Configuration 1 is seen in Configuration 3 (Figure 6.9), where CE-H has the lowest mean path length of 26.2 *m*. CE-3.5 and VEFEP had similar performance with means of 78.4 *m* and 74.1 *m* respectively as seen in Table 6.4. While it may look like CE-H had the overall

best performance in Office 1, it's important to note that due to the small size of the environment, a small deviation in waypoint selection can lead to an observation of the target object, resulting in shortened paths. As such, we can conclude that all methods performed with similar levels of performance in a smaller environment which was the expected outcome.

Table 6.2: Path lengths for FE1 in Office 1. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	25.2	57.6	369.6	120.5	122.5
CE-H	25.2	39.6	194.1	62.3	75.8
VEFEP	25.2	48.6	491.7	87.3	132.3

Table 6.3: Path lengths for FE2 in Office 1. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	27.1	46.7	155.2	62.9	78.9
CE-H	27.1	48.4	145.7	73.8	81.8
VEFEP	27.1	48.6	507.8	67.2	104.7

As the environment size increases in Office 2, making the exploration tasks more complex, we observe that on the office chair task, both CE-H and CE-3.5 outperform VEFEP (Table 6.6). We also note that VEFEP, timed out in 3 out of 15 of the runs as noted in Table 6.1. VEFEP had the lowest median run length of 195.5 m in the coffee table task. However, it also failed to find the table in 6 out of 15 experiments, as shown in Table 6.1. Meanwhile, CE-H only had a slightly larger mean path length of 201.7 m but had a smaller minimum distance of 55.5 m compared to VEFEP's minimum distance of 75.9 m . CE-3.5 had the widest variance but found the shortest path length of 52.0 m . The Office Chair configuration can be seen in Figure 6.12 and the Coffee Table overhead view can be found in Figure 6.11.

In the School, one of the larger environments at 1287 m^2 we observe a similar result to Office 2 with the bookshelf task where CE-H outperforms the other methods with a mean path length of 110.3 m . VEFEP outperforms CE-3.5 on average with a mean of 139.2 m versus 173.7 m on path length (Table 6.8). However, both CE methods complete the task 100 percent of the time and VEFEP timed out in 2 out of 15 tasks. An overhead of the bookshelf task is shown in Figure 6.15.

Table 6.4: Path lengths for FE3 in Office 1. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	15.6	21.1	255.0	46.9	78.4
CE-H	15.6	26.2	172.5	37.5	59.6
VEFEP	15.6	33.1	185.6	63.3	74.1

In the whiteboard eraser task (Figure 6.14), both CE models significantly outperform VEFEP, which also had 5 out of 15 failures or a 33.3% failure rate. CE-H had a path length mean of 147.9 m and CE-3.5 had a mean length of 199.8 m . In contrast, VEFEP had a mean of 415.4 m and a maximum path length of 1021.4 m , nearly double that of CE-3.5 at 598.3 m . Of note, is that **none** of the CE methods failed at the exploration task in any of the environments.

6.5 Discussion

From the 210 experiments run across 7 different environments, we have strong evidence that illustrates CogExplore can more efficiently locate objects than a conventional graph-based exploration planner with frontier point finding. In smaller environments (Office 1), CogExplore has a similar level of performance to VEFEP which is expected due to the small size. As the environment size increases and semantics become more relevant to the task CogExplore outperforms VEFEP on average, highlighting the benefits of the framework’s ability to semantically reason. We provide anecdotal examples of the model’s reasoning in Figure 6.4, where we see examples of the model performing geometric reasoning (left), semantic reasoning (center), and temporal reasoning (right). We define geometric reasoning as the ability of the model to reason over spatial constraints in 3D Euclidean space. Semantic reasoning is the ability of the model to form logical connections between objects and temporal reasoning is the ability of the model to reason over its past states to influence future states. At each iteration, we can probe why the model made a decision that highlights the explainability of CogExplore. In the following section, we will dive into specific examples that support CogExplore’s ability to reason geometrically, temporally, and semantically.

6.5.1 Geometric Reasoning

When taking a look at Office 1, which has little furniture to obstruct the robot’s view and only two rooms to explore, VEFEP and all variants of CogExplore perform well for the FE2 and FE3 tasks. We observe that all models have fairly direct path lengths in Figures 6.8 and 6.9. In the FE1 task (Figure 6.7, VEFEP performs significantly worse than CE-H and CE-3.5. In this case, VEFEP heads to the hallway in the top left corner but fails to observe the fire extinguisher around the corner. Since the planner is based on volumetric gain, the narrow hallway does not provide a large enough frontier cluster to prioritize continued exploration. Instead, the planner goes to other parts of the environment in an effort to expand the corners of its map. In some instances, VEFEP does go far enough around the corner to observe the Fire Extinguisher and the paths are shorter but this depends on the initial set of points and frontiers sampled for the graph. Meanwhile, CogExplore will look at all the sampled points and explore new areas based on its past state log and newly observed points. This results in reasoning that enables the robot to round the corner:

My reasoning for selecting this point is based on the fact that it is a new graph point, meaning it represents an area that I have recently discovered and have not explored yet. By choosing this point, I aim to expand my exploration into uncharted territory, potentially gaining new vantages for my cameras and lidar. Exploring new areas is crucial in uncovering new information and potentially locating the fire extinguisher.

The above example is taken from a CE-3.5 run where the robot’s lidar mapped areas around the corner, and local graph points were labeled as new. Since no other areas were new, and the log showed the robot had been to other general areas in the sampled points the model chose to direct the robot around the corner where it observed the fire extinguisher. This can be seen in Figure 6.6 where CE-3.5 (GPT) rounds the corner and immediately finds the fire extinguisher.

Further evidence of geometric reasoning is provided in the case of CE-H in the same environment. In Figure 6.7 we observe that Cluade’s path saw the extinguisher, got a clipped projection, and went to the wall at the corner of the hallway. At that point the next waypoint was selected

Figure 6.6: Overhead path examples for Office Environment 1.

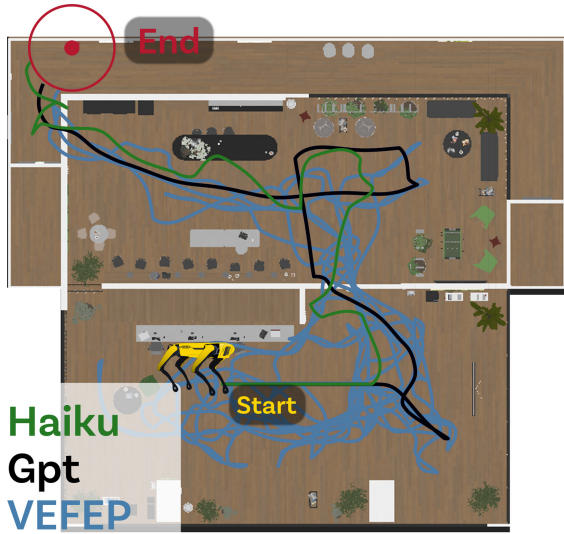


Figure 6.7: Birds-eye view of Office Environment 1 (OE1) for task Fire Extinguisher 1 (FE1).

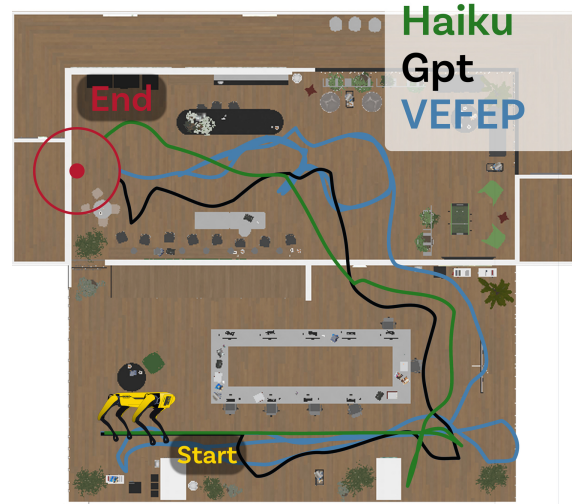


Figure 6.8: Birds-eye view of Office Environment 1 (OE1) for task Fire Extinguisher 2 (FE2).

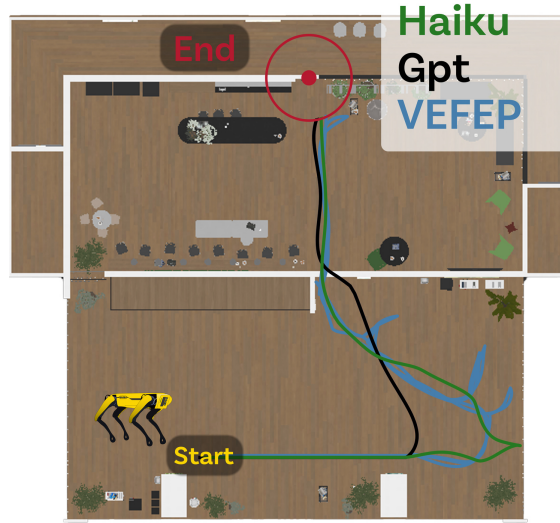


Figure 6.9: Birds-eye view of Office Environment 1 (OE1) for task Fire Extinguisher 3 (FE3).

with the following rationale:

The point I have selected is point number 42. I am selecting this point because I believe it makes strategic sense to get me closer to solving my navigation task “Go to the fire extinguisher”. My environment can be described as a domestic indoor setting with large windows, wooden floors, and white walls, creating an open and well-lit atmosphere. My reasoning is that this point corresponds to the location of a fire extinguisher object, which is the target I have been instructed to navigate to. While I have not yet directly observed the fire extinguisher, this point represents the closest known location of the target object, so by navigating to this point, I can increase my chances of successfully completing the navigation task. Additionally, this point is located behind my current position, which is a direction I have not thoroughly investigated yet.

The above reasoning block is a sample of model’s justification for its navigational choices. Notably, the model exhibits a degree of geometric, and temporal reasoning. For geometric reasoning, model understands that the directed point is behind the robot where it likely hasn’t observed the environment as well. This is based on the past states and observations which suggest the framework can make informed decisions based on past actions, or temporal reasoning. Once the decision is made the robot sees the extinguisher, gets an accurate projection of the location, and proceeds directly there. Moreover this highlights, CogExplore’s robustness to the noise created by imperfect vision solutions.

Another example of geometric reasoning can be seen on the right side of Figure 6.4 where the model explicitly chooses a point behind the robot. The prompt makes the model aware that the robot can not detect objects, directly behind it, since there is no rear camera. Note, the lack of a rear camera is inferred by the model which is only told that “the robot has a left, right and front camera.” From this example, it’s clear that this type of intuitive geometric reasoning contributes to the shorter exploration path lengths demonstrated by CogExplore.

Figure 6.10: Overhead path examples for Office Environment 2.

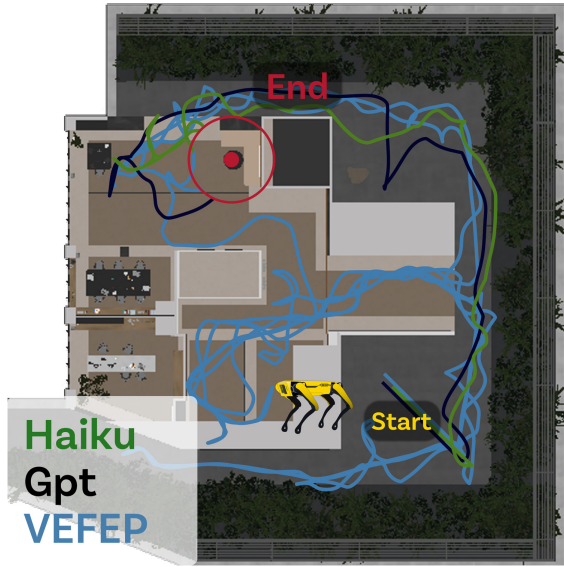


Figure 6.11: Birds-eye view of Office 2 for task Coffee Table.

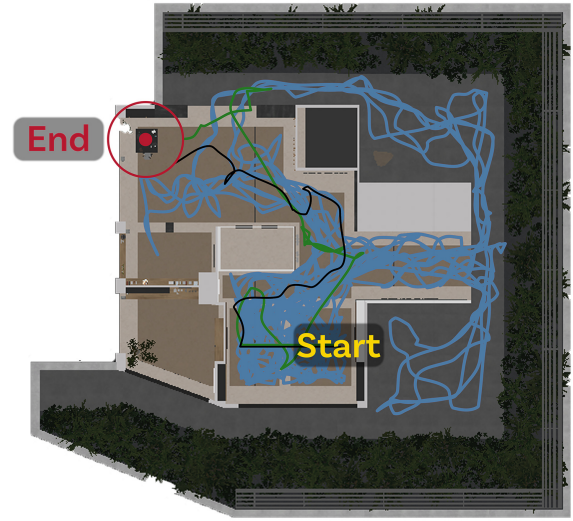


Figure 6.12: Birds-eye view of Office 2 for task Office Chair.

6.5.2 Semantic Reasoning

Taking a look at the results of two larger environments, Office 2 (OE2) and the School, we can see that CE-H has the most consistent performance across all of the methods. Moreover, in the more challenging exploration tasks such as the “Go to the whiteboard eraser” task in the School environment all versions of CogExplore significantly outperform VEFEP. A likely explanation for this is CogExplore’s ability to reason over semantic observations. For example in Figure 6.14, we observe that CE-H and CE-3.5 only enter the room with the eraser once whereas VEFEP enters, exits, and proceeds to loop around the environment. In this case, CogExplore can use past semantic observations to influence rapidly exploring the environment versus performing a local search in an area with a higher likelihood of finding the object. This is in contrast to VEFEP which can only explore based on the perceived amount of new volumetric gain. It’s worth noting that VEFEP can be tuned to achieve better performance for individual environments but to maintain consistency, we kept the tuning the same across all methods. Figure 6.4 provides an example of semantic reasoning

where we see CE-3.5 deciding to explore the room more thoroughly due to the presence of desks and other furniture which it reasons “could be ... close proximity to the eraser”.

In the Go to the Office Chair task CE-H and CE-3.5, immediately go to the chair and take more efficient paths to get there (Figure 6.12). In contrast, VEFEP heads outside and explores the areas significantly before returning indoors to find the chair. The coffee table example in Figure 6.11 shows similar results. An example of CE-3.5’s reasoning during one of the coffee table runs can be seen below:

I selected this point for exploration as it presented an uncharted area within the domestic setting.

While it’s arguable that a domestic setting isn’t the same as an office setting, the two are semantically similar. In both settings, chairs, coffee tables, desks, and other common pieces of furniture can be found. As such, it makes sense for CogExplore to explore this area rather than the outdoor patio area of the Office 2 Scene. The combination of shorter path lengths in the whiteboard eraser and office chair tasks and the semantic examples clearly show that CogExplore benefits from reasoning over contextual cues in the environment which can directly lead to performance improvements over a hand-engineered heuristic based on an environment’s geometry alone.

6.5.3 Temporal Reasoning

Beyond reasoning over the current state, efficient path exploration requires an agent to reason over its past states. In CogExplore’s case, we note the robot venturing to a new area despite having a point labeled as a fire extinguisher (the target exploration object). The robot already explored the nearby area in detail in a prior state, and the target object was never reached. The model correctly concludes the detection was in error. CogExplore justifies this change based on the understanding that object detections can be fallacious, which also highlights the framework’s ability to perform fault-tolerant reasoning and remain robust to noisy sensor observations. In contrast, it was observed that the VEFEP planner would frequently stick around objects as the location of the 3D object

projections were refined (as new observations were obtained), despite this not being an effective strategy after a few iterations.

We can also see from Figure 2.14 that VEFEP suffered from exploring near the office chair but not actually reaching it. This is attributed to obstructed projections in the object detection system. Since the system performs a ray cast, obstructed views can cause object projections to fall short of the actual location. If no new observations update this position, then VEFEP will veer away from the area, whereas CogExplore will usually probe the area or come back to it if other cues indicate it’s a good area to keep exploring.

Cog Explore’s ability to leverage a variety of reasoning mechanisms through encoding environmental features into natural language strings can enable sharp performance gains from heuristic-based methods like VEFEP. Frontier-based exploration methods like VEFEP rely on hand tuning of the volumetric gain weights to influence the exploration behavior, whereas CogExplore adapts based on observations. This is highlighted in the example where our framework tells the robot to choose a further state because no clues were found indicating the fire extinguisher is nearby:

“The point is far enough to explore a different vantage point within the domestic setting as no indications of the fire extinguisher are present”

Table 6.5: Path lengths for CT in Office 2. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	26.2	52.0	1135.8	252.1	315.0
CE-H	26.2	55.5	759.9	145.7	201.7
VEFEP	26.2	75.9	446.0	159.8	195.5

Table 6.6: Path lengths for OC in Office 2. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	33.7	43.2	709.2	130.4	247.3
CE-H	33.7	70.6	705.7	174.3	230.9
VEFEP	33.7	47.8	817.9	309.5	349.8

Table 6.7: Path lengths for WB in the School. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	26.1	93.2	598.3	149.5	199.8
CE-H	26.1	84.0	283.5	119.3	147.9
VEFEP	26.1	87.1	1021.4	295.6	415.4

Figure 6.13: Overhead path examples for School Environment.

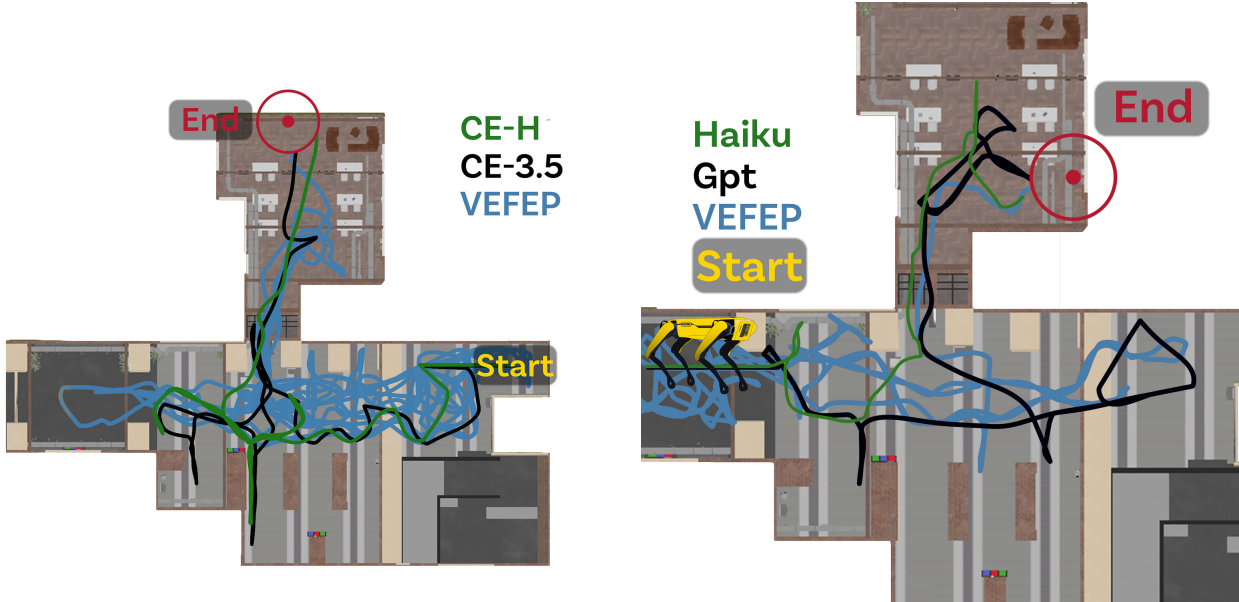


Figure 6.14: Overhead path examples for the whiteboard eraser task.

Figure 6.15: Birds-eye view of School for the Bookshelf task Bookshelf

6.5.4 Limitations of CogExplore

From all of the anecdotal evidence presented we have seen that CogExplore has the ability to perform geometric, semantic, and temporal reasoning. Moreover, holistic evidence in the form of efficient path lengths from 210 45-minute experiments has shown that CogExplore outperforms a heuristic-driven explore and search method in larger environments, especially where contextual cues can help locate an object. However, it's important to note that not all the justifications that come out of CogExplore are helpful. For example, there are issues where the justification provided by the model is not relevant to the task at hand:

Table 6.8: Path lengths for BS in the School. All distances are in m

Model	Direct	Min	Max	Median	Mean
CE-3.5	13.0	82.8	388.0	142.4	173.7
CE-H	13.0	53.3	205.6	95.5	110.3
VEFEP	13.0	50.8	297.9	121.9	139.2

I aimed to expand the search into uncharted territories within the domestic environment and increase the likelihood of discovering the target object. The environment observed was a domestic setting with large windows, trees, plants, and an open patio, providing a tranquil atmosphere conducive to the search mission.

While exploring new areas is the goal, it’s clear that the tranquil atmosphere isn’t helpful in the above example where CE-3.5 was trying to go to the coffee table task. In addition to poor reasoning, there are also instances where both CE-H and CE-3.5 will simply provide the justification of “I need to explore a new area” or another phrasing along those lines. In some cases, this is to be expected if there are no contextual cues but in other cases, the model will ignore the semantics.

It’s also important to note that not every visited place on an overhead path can be explained with this method. Sometimes this is due to the wrong justification and other times the model chose a random new point. This is highlighted in Figure 6.11 where it looks like the black path of CE-3.5 (GPT) went to explore an area in the top right corner of the environment. The action looks deliberate in the plot but in fact the model simply chose one of the newly sampled points to explore. Despite these limitations, CogExplore still represents a significant step forward in using natural-language-based representations for exploration and search tasks.

6.6 Conclusions

Our results highlight the robustness of CogExplore’s exploration abilities with a **100% success rate** at finding the target object, across all environments with both GPT and Haiku variants, in contrast to VEFEP’s 15.2% overall failure and 40% failure rate at the coffee table task. Moreover, CogExplore’s use of natural language justifications for each state selection allow us to directly

probe why the framework is acting in a certain manner and understand the rationale behind the more efficient exploration paths. From these justifications, we can see that the model is capable of performing geometric reasoning, semantic reasoning, temporal reasoning, and fault-tolerant reasoning. The resulting paths can be more efficient than the underlying heuristic method CogExplore is based off of, especially in larger more semantically rich environments.

Chapter 7

Future Work and Conclusions

7.1 Future Work

While this work has made significant contributions to search and exploration algorithms that leverage classical techniques with foundation models, open challenges remain. All of the methods presented here have no formally verified outputs or safety guarantees. Consequently, this can cause problems if the natural-language command is parsed into a series of plans that are not safe for the robot to perform. Generally, the underlying planning methods ensure the robot won't collide with objects but they don't ensure the robot is following the expected task. Algorithms that verify the proposed solution for a navigation command matches the expected outcome are still needed.

Additionally, challenges in perception still remain. While object detectors such as the YOLO family [102] report confidence levels. These levels aren't sufficient to know if an object has been correctly classified. These object detectors can report with 99% certainty that a red object is a fire extinguisher even when the object is in fact a red backpack. Detecting such anomalies is critical to improve the reliability of the methods presented here.

Finally, CogExplore has demonstrated to be robust in a photo-realistic simulator. While this provides strong evidence the method will work on a physical platform given the underlying planning framework does, open challenges remain in tuning the algorithm to account for noisy maps, localization drift, and dynamic objects in order to deploy this method on a physical robot. Future work includes testing the algorithm on a physical platform and determining solutions for the above limitations.

7.2 Conclusions

Throughout the course of this work, we have developed methods that are capable of encoding the state of a robot’s environment into natural language representations. In Chapter 2 we introduce a flexible autonomy system for exploring subterranean environments. Next in Chapter 3 we detail the communication backbone developed to enable multi-robot communication and human interaction with the system. The communications system serves as the backbone for natural-language-based systems that were developed from the observed performance limitations of the system.

A notable limitation of the system developed for the DARPA Subterranean challenge in Chapter 2 was its inability to reason over contextual cues without human supervision. This reliance on hand-engineered heuristics led to suboptimal performance when communication was poor. Additionally, human supervision does not scale well to large fleets of robots due to fanout. This work addresses these issues by providing two key frameworks for incorporating world knowledge from foundation models into classical planning frameworks. Chapter 5 presents a method based on code generation from natural language. The framework is capable of performing open-world landmark navigation by converting natural-language commands into a block of python code that represents specific constraints symbolically. The method is validated in 4 real-world environments on a Boston Dynamics spot and the results show that LLMs can be used to perform spatial reasoning from natural language.

Next, Chapter 6 introduces CogExplore, a framework that encodes environmental descriptions, object locations, and spatial points into natural language for navigation based on contextual cues. The framework is capable of exploring environments, and searching for specific objects while performing geometric, semantic, and temporal reasoning. These reasoning capabilities can improve the performance in terms of path efficiency when compared to heuristic-based navigation systems alone.

Ultimately, this work supports and validates the thesis statement:

Integrating natural language encodings of key semantic features in a robot's environment with classical planning algorithms and foundation models can enable more efficient search and exploration paths by leveraging semantic, geometric, and temporal reasoning.

Bibliography

- [1] Ali Agha, Kyohei Otsu, Benjamin Morrell, David D Fan, Rohan Thakker, Angel Santamaria-Navarro, Sung-Kyun Kim, Amanda Bouman, Xianmei Lei, Jeffrey Edlund, et al. Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge. [arXiv preprint arXiv:2103.11470](#), 2021.
- [2] Ali Agha, Kyohei Otsu, Benjamin Morrell, David D. Fan, Rohan Thakker, Angel Santamaria-Navarro, Sung-Kyun Kim, Amanda Bouman, Xianmei Lei, Jeffrey A. Edlund, Muhammad Fadhil Ginting, Kamak Ebadi, Matthew O. Anderson, Torkom Pailevanian, Edward Terry, Michael T. Wolf, Andrea Tagliabue, Tiago Stegun Vaquero, Matteo Palieri, Scott Tepsuporn, Yun Chang, Arash Kalantari, Fernando Chavez, Brett Thomas Lopez, Nobuhiro Funabiki, Gregory Miles, Thomas Touma, Alessandro Buscicchio, Jesus Tordesillas, Nikhilesh Alatur, Jeremy Nash, William Walsh, Sunggoo Jung, Hanseob Lee, Christoforos Kanelakis, John Mayo, Scott Harper, Marcel Kaufmann, Anushri Dixit, Gustavo Correa, Carlyn-Ann Lee, Jay L. Gao, Gene B. Merewether, Jairo Maldonado-Contreras, Gautam Salhotra, Maíra Saboia da Silva, Benjamin Ramtoula, Seyed Abolfazl Fakoorian, Alexander Hatte-land, Taeyeon Kim, Tara Bartlett, Alex Stephens, Leon Kim, Charles F. Bergh, Eric Heiden, Thomas Lew, Abhishek Cauligi, Tristan Heywood, Andrew Kramer, Henry A. Leopold, Chris Seoyun Choi, Shreyansh Daftry, Olivier Toupet, Inhwan Wee, Abhishek Thakur, Micah Feras, G. Beltrame, George Nikolakopoulos, David Hyunchul Shim, Luca Carlone, and Joel W. Burdick. Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge. [ArXiv](#), abs/2103.11470, 2021.
- [3] Shakeeb Ahmad and J Sean Humbert. Efficient sampling-based planning for subterranean exploration. In [2022 IEEE/RSJ International Conference on Intelligent Robots and Systems \(IROS\)](#). IEEE, 2022.
- [4] Shakeeb Ahmad, Andrew B Mills, Eugene R Rush, Eric W Frew, and J Sean Humbert. 3d reactive control and frontier-based exploration for unstructured environments. In [2021 IEEE/RSJ International Conference on Intelligent Robots and Systems \(IROS\)](#), pages 2289–2296. IEEE, 2021.
- [5] Shakeeb Ahmad, Zachary N. Sunberg, and J. Sean Humbert. Apf-pf: Probabilistic depth perception for 3d reactive obstacle avoidance. In [2021 American Control Conference \(ACC\)](#), pages 32–39, 2021.
- [6] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can,

- not as i say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691, 2022.
- [7] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. Advances in Neural Information Processing Systems, 35:23716–23736, 2022.
 - [8] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 39–48, 2016.
 - [9] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In Proceedings of the IEEE international conference on computer vision, pages 2425–2433, 2015.
 - [10] Abraham Galton Bachrach. Trajectory bundle estimation for perception-driven planning. PhD thesis, Massachusetts Institute of Technology, 2013.
 - [11] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509–517, 1975.
 - [12] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. Adabins: Depth estimation using adaptive bins. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4009–4018, 2021.
 - [13] Harel Biggie, Ajay Narasimha Mopidevi, Dusty Woods, and Christoffer Heckman. Tell Me Where to Go: A Composable Framework for Context-Aware Embodied Robot Navigation. In Conference on Robot Learning. PMLR, 11 2023.
 - [14] Harel Biggie, E. Rush, D. Riley, S. Ahmad, M. Ohradzansky, K. Harlow, M. Miles, D. Torres, S. McGuire, E. Frew, C. Heckman, and J. Humbert. Flexible Supervised Autonomy for Exploration in Subterranean Environments. Field Robotics, 3:125–189, 2023.
 - [15] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In Conference on Robot Learning, pages 287–318. PMLR, 2023.
 - [16] Herman Bruyninckx. Open robot control software: the OROCOS project. In Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164), volume 3, pages 2523–2528. IEEE, 2001.
 - [17] Anthony Buckner, Lucas Figueredo, Sami Haddadin, Ashish Kapoor, Shih-Yun Ma, Sai Vemprala, and Rogerio Bonatti. Latte: Language trajectory transformer, 2022.
 - [18] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

- [19] Yanda Chen, Ruiqi Zhong, Narutatsu Ri, Chen Zhao, He He, Jacob Steinhardt, Zhou Yu, and Kathleen McKeown. Do models explain themselves? counterfactual simulatability of natural language explanations, 2023.
- [20] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2024.
- [21] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311, 2022.
- [22] Thomas Clausen, Philippe Jacquet, Cédric Adjih, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized link state routing protocol (OLSR), 2003.
- [23] R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, The Robotics Institute, Carnegie Mellon University, 1992.
- [24] Yinpei Dai, Run Peng, Sikai Li, and Joyce Chai. Think, act, and ask: Open-world interactive personalized robot navigation. ICRA, 2024.
- [25] Tung Dang, Shehryar Khattak, Frank Mascarich, and Kostas Alexis. Explore locally, plan globally: A path planning framework for autonomous robotic exploration in subterranean environments. In 2019 19th International Conference on Advanced Robotics (ICAR), pages 9–16. IEEE, 2019.
- [26] Tung Dang, Marco Tranzatto, Shehryar Khattak, Frank Mascarich, Kostas Alexis, and Marco Hutter. Graph-based subterranean exploration path planning using aerial and legged robots. Journal of Field Robotics, 37(8):1363–1388, 2020.
- [27] Tung Dang, Marco Tranzatto, Shehryar Khattak, Frank Mascarich, Kostas Alexis, and Marco Hutter. Graph-based subterranean exploration path planning using aerial and legged robots. Journal of Field Robotics, 10 2020.
- [28] DARPA. DARPA Subterranean Challenge, 2022. <https://www.darpa.mil/program/darpa-subterranean-challenge>.
- [29] Fei Deng, Junyeong Park, and Sungjin Ahn. Facing off world model backbones: Rnns, transformers, and s4. Advances in Neural Information Processing Systems, 36, 2024.
- [30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
- [31] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. arXiv preprint arXiv:2303.03378, 2023.

- [32] Kamak Ebadi, Yun Chang, Matteo Palieri, Alex Stephens, Alex Hatteland, Eric Heiden, Abhishek Thakur, Nobuhiro Funabiki, Benjamin Morrell, Sally Wood, Luca Carlone, and Ali Akbar Agha-Mohammadi. LAMP: Large-Scale Autonomous Mapping and Positioning for Exploration of Perceptually-Degraded Subterranean Environments. IEEE International Conference on Robotics and Automation, pages 80–86, 2020.
- [33] E Emerson. Temporal and modal logic in handbook of theoretical computer science, chapter 16, leeuwen j., editor, 1990.
- [34] Epic Games. Unreal engine. <https://www.unrealengine.com>, 11 2023. Version 5.3.2.
- [35] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. Minds and Machines, 30:681–694, 2020.
- [36] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. arXiv preprint arXiv:2203.10421, 2022.
- [37] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YoloX: Exceeding yolo series in 2021. arXiv preprint arXiv:2107.08430, 2021.
- [38] Ross Girshick. Fast R-CNN. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015.
- [39] Amelia Glaese, Nat McAleese, Maja Trebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of dialogue agents via targeted human judgements. arXiv preprint arXiv:2209.14375, 2022.
- [40] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14953–14962, 2023.
- [41] Stevan Harnad. The symbol grounding problem. Physica D: Nonlinear Phenomena, 42(1-3):335–346, 1990.
- [42] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.
- [43] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D LIDAR SLAM. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1271–1278, 2016.
- [44] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. Acm Sigact News, 32(1):60–65, 2001.
- [45] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. Autonomous robots, 34(3):189–206, 2013.
- [46] Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 6652–6659. IEEE, 2014.

- [47] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In Proceedings of the IEEE international conference on computer vision, pages 804–813, 2017.
- [48] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. arXiv preprint arXiv:2210.05714, 2022.
- [49] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.
- [50] Yi-Wei Huang, Chen-Lung Lu, Kuan-Lin Chen, Po-Sheng Ser, Jui-Te Huang, Yu-Chia Shen, Pin-Wei Chen, Po-Kai Chang, Sheng-Cheng Lee, and Hsueh-Cheng Wang. Duckiefloat: a Collision-Tolerant Resource-Constrained Blimp for Long-Term Autonomy in Subterranean Environments, 2019.
- [51] Nicolas Hudson, Fletcher Talbot, Mark Cox, Jason Williams, Thomas Hines, Alex Pitt, Brett Wood, Dennis Frousheger, Katrina Lo Surdo, Thomas Molnar, Ryan Steindl, Matt Wildie, Inkyu Sa, Navinda Kottege, Kazys Stepanas, Emili Hernandez, Gavin Catt, William Docherty, Brendan Tidd, Benjamin Tam, Simon Murrell, Mitchell Bessell, Lauren Hanson, Lachlan Tychsen-Smith, Hajime Suzuki, Leslie Overs, Farid Kendoul, Glenn Wagner, Duncan Palmer, Peter Milani, Matthew O’Brien, Shu Jiang, Shengkang Chen, and Ronald C. Arkin. Heterogeneous ground and air platforms, homogeneous sensing: Team csiro data61’s approach to the darpa subterranean challenge, 2021.
- [52] Danny G Riley Li and Eric W Frew. Assessment of coordinated heterogeneous exploration of complex environments. In 2021 IEEE Conference on Control Technology and Applications (CCTA), pages 138–143. IEEE, 2021.
- [53] Ray S Jackendoff. Semantics and cognition, volume 8. MIT press, 1985.
- [54] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L  lio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Th  ophile Gervet, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mixtral of experts, 2024.
- [55] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In Proceedings of the IEEE international conference on computer vision, pages 2989–2998, 2017.
- [56] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In 2011 IEEE international conference on robotics and automation, pages 1478–1483. IEEE, 2011.
- [57] Shehryar Khattak, Huan Nguyen, Frank Mascarich, Tung Dang, and Kostas Alexis. Complementary Multi-Modal Sensor Fusion for Resilient Robot Pose Estimation in Subterranean Environments. In 2020 International Conference on Unmanned Aircraft Systems (ICUAS), pages 1024–1029, Athens, Greece, 2020. IEEE.

- [58] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 4015–4026, 2023.
- [59] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In 2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI), pages 259–266. IEEE, 2010.
- [60] Thomas Kollar, Stefanie Tellex, Matthew Walter, Albert Huang, Abraham Bachrach, Sachi Hemachandra, Emma Brunskill, Ashis Banerjee, Deb Roy, Seth Teller, et al. Generalized grounding graphs: A probabilistic framework for understanding grounded commands. arXiv preprint arXiv:1712.01097, 2017.
- [61] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. Information, 10(4):150, 2019.
- [62] Angelika Kratzer and HEIM Irene. Semantics in generative grammar, volume 1185. Blackwell Oxford, 1998.
- [63] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. Annual Review of Control, Robotics, and Autonomous Systems, 1:211–236, 2018.
- [64] Pierre Yves Lajoie, Benjamin Ramtoula, Yun Chang, Luca Carlone, and Giovanni Beltrame. DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams. IEEE Robotics and Automation Letters, 5(2):1656–1663, 2020.
- [65] Xiaodong Lan and Stefano Di Cairano. Continuous curvature path planning for semi-autonomous vehicle maneuvers using rrt. In 2015 European control conference (ECC), pages 2360–2365. IEEE, 2015.
- [66] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. Research Report 9811, 1998.
- [67] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and René Ranftl. Language-driven semantic segmentation. arXiv preprint arXiv:2201.03546, 2022.
- [68] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. arXiv preprint arXiv:2301.12597, 2023.
- [69] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10965–10975, 2022.
- [70] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. Science, 378(6624):1092–1097, 2022.

- [71] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024.
- [72] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9):1–35, 2023.
- [73] Qi Liu, Matt J Kusner, and Phil Blunsom. A survey on contextual embeddings. arXiv preprint arXiv:2003.07278, 2020.
- [74] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, pages 21–37. Springer, 2016.
- [75] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. Robotics and Autonomous Systems, 86:13–28, 2016.
- [76] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. arXiv preprint arXiv:2210.07128, 2022.
- [77] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. arXiv preprint arXiv:1904.12584, 2019.
- [78] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 922–928. IEEE, 2015.
- [79] Timothy P McNamara, James K Hardy, and Stephen C Hirtle. Subjective hierarchies in spatial memory. Journal of Experimental Psychology: Learning, Memory, and Cognition, 15(2):211, 1989.
- [80] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. IEEE Robotics and Automation Letters (RA-L), 7(4):11205–11212, 2022.
- [81] Michael J Miles, Harel Biggie, and Christoffer Heckman. Terrain-Aware Semantic Mapping for Cooperative Subterranean Exploration. Frontiers in Robotics and AI, 10, 2023.
- [82] Ian D. Miller, Avraham Cohen, Adarsh Kulkarni, James Laney, Camillo Jose Taylor, Vijay Kumar, Fernando Cladera, Anthony Cowley, Shreyas S. Shivakumar, Elijah S. Lee, Laura Jarín-Lipschitz, Akhilesh Bhat, Neil Rodrigues, and Alex Zhou. Mine Tunnel Exploration Using Multiple Quadrupedal Robots. IEEE Robotics and Automation Letters, 5(2):2840–2847, 2020.
- [83] Raymond J Mooney. Learning to connect language and perception. In AAAI, pages 1598–1601. Chicago, 2008.
- [84] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. IEEE transactions on robotics, 31(5):1147–1163, 2015.

- [85] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2203.13474, 2022.
- [86] Iram Noreen, Amna Khan, and Zulfiqar Habib. Optimal path planning using rrt* based approaches: a survey and future directions. International Journal of Advanced Computer Science and Applications, 7(11), 2016.
- [87] Michael T. Ohradzansky, Andrew B. Mills, Eugene R. Rush, Danny G. Riley, Eric W. Frew, and J. Sean Humbert. Reactive control and metric-topological planning for exploration. In 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 4073–4079, 2020.
- [88] Michael T. Ohradzansky, Eugene R. Rush, Danny G. Riley, Andrew B. Mills, Shakeeb Ahmad, Steve McGuire, Harel Biggie, Kyle Harlow, Michael J. Miles, Eric W. Frew, Christoffer Heckman, and J. Sean Humbert. Multi-agent autonomy: Advancements and challenges in subterranean exploration, 2021.
- [89] OpenAI. Gpt-4 technical report, 2023.
- [90] Kyohei Otsu, Scott Tepsuporn, Rohan Thakker, Tiago Stegun Vaquero, Jeffrey A. Edlund, William Walsh, Gregory Miles, Tristan Heywood, Michael T. Wolf, and Ali-Akbar Agha-Mohammadi. Supervised autonomy for communication-degraded subterranean exploration by a robot team. In 2020 IEEE Aerospace Conference, pages 1–9, 2020.
- [91] Ali Paikan, Daniele Domenichelli, and Lorenzo Natale. Communication channel prioritization in a publish-subscribe architecture. In 2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), pages 41–45. IEEE, 2015.
- [92] Sreekrishna Pandi, Frank Gabriel, Oleksandr Zhdanenko, Simon Wunderlich, and Frank HP Fitzek. MESHMERIZE: An interactive demo of resilient mesh networks in drones. In 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), pages 1–2. IEEE, 2019.
- [93] C. Papachristos, S. Khattak, F. Mascarich, T. Dang, and K. Alexis. Autonomous aerial robotic exploration of subterranean environments relying on morphology-aware path planning. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pages 299–305, 2019.
- [94] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.
- [95] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. RFC3561: Ad hoc on-demand distance vector (AODV) routing, 2003.
- [96] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics, 26(3):308–333, 2009.
- [97] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 652–660, 2017.

- [98] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: an open-source robot operating system. In ICRA workshop on open source software, volume 3, page 5. Kobe, Japan, 2009.
- [99] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In International conference on machine learning, pages 8748–8763. PMLR, 2021.
- [100] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. IEEE transactions on pattern analysis and machine intelligence, 44(3):1623–1637, 2020.
- [101] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 779–788, 2016.
- [102] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.
- [103] Danny G Riley and Eric W Frew. Assessment of Coordinated Heterogeneous Exploration of Complex Environments. In 2021 IEEE Conference on Control Technology and Applications (CCTA), San Diego, CA, 2021.
- [104] Danny G. Riley and Eric W. Frew. BOBCAT: Behaviors, Objectives and Binary states for Coordinated Autonomous Tasks. IEEE Robotics and Automation Letters, 2022. Under review.
- [105] Tomáš Rouček, Martin Pecka, Petr Čížek, Tomáš Petříček, Jan Bayer, Vojtěch Šalanský, Daniel Heřt, Matěj Petrлік, Tomáš Báča, Vojtěch Spurný, François Pomerleau, Vladimír Kubelka, Jan Faigl, Karel Zimmermann, Martin Saska, Tomáš Svoboda, and Tomáš Krajník. Darpa subterranean challenge: Multi-robotic exploration of underground environments. In Jan Mazal, Adriano Fagiolini, and Petr Vasik, editors, Modelling and Simulation for Autonomous Systems, pages 274–290, Cham, 2020. Springer International Publishing.
- [106] Mehdi SM Sajjadi, Daniel Duckworth, Aravindh Mahendran, Sjoerd van Steenkiste, Filip Pavetic, Mario Lucic, Leonidas J Guibas, Klaus Greff, and Thomas Kipf. Object scene representation transformer. Advances in Neural Information Processing Systems, 35:9512–9524, 2022.
- [107] Angel Santamaria-Navarro, Rohan Thakker, David D. Fan, Benjamin Morrell, and Ali-akbar Agha-mohammadi. Towards Resilient Autonomous Navigation of Drones. arXiv:2008.09679 [cs], 2020. arXiv: 2008.09679.
- [108] Ozan Saycan, Shivam Garg, Elliot Meyerson, Jonathan Tsai, Vicente Ordóñez, Roozbeh Mottaghi, and Ali Farhadi. Progprompt: Generating situated robot task plans using large language models, 2022.
- [109] A. Schang, J. Rogers, and A. Maio. Map analysis (version 1.0) [source code], 2021.

- [110] S. Scherer, V. Agrawal, G. Best, C. Cao, K. Cujic, R. Darnley, and et al. Resilient and modular subterranean exploration with a team of roving and flying robots. Field Robotics, 2021.
- [111] Max Schwarz, Marius Beul, David Droeschel, Sebastian Schüller, Arul Selvam Periyasamy, Christian Lenz, Michael Schreiber, and Sven Behnke. Supervised autonomy for exploration and mobile manipulation in rough terrain with a centaur-like robot. Frontiers in Robotics and AI, 3:57, 2016.
- [112] Daniel Seither, André König, and Matthias Hollick. Routing performance of wireless mesh networks: A practical evaluation of BATMAN advanced. In 2011 IEEE 36th Conference on Local Computer Networks, pages 897–904. IEEE, 2011.
- [113] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision, pages 618–626, 2017.
- [114] James Albert Sethian. Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science, volume 3. Cambridge University Press, 1999.
- [115] Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In Conference on Robot Learning, pages 2683–2699. PMLR, 2023.
- [116] Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Ving: Learning open-world navigation with visual goals. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 13215–13222. IEEE, 2021.
- [117] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In Conference on Robot Learning, pages 492–504. PMLR, 2023.
- [118] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping. arXiv:2007.00258 [cs], July 2020. arXiv: 2007.00258.
- [119] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In 2020 IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 5135–5142. IEEE, 2020.
- [120] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointcnn: 3d object proposal generation and detection from point cloud. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 770–779, 2019.
- [121] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In Proceedings of the 5th Conference on Robot Learning (CoRL), 2021.

- [122] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. arXiv preprint arXiv:2303.08128, 2023.
- [123] Danilo Tardioli, Ramviyas Parasuraman, and Petter Ögren. Pound : A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks. Robotics and Autonomous Systems, 111:73–87, 2019. QC 20181210.
- [124] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. Annual Review of Control, Robotics, and Autonomous Systems, 3:25–55, 2020.
- [125] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 25, pages 1507–1514, 2011.
- [126] Alexander Tiderko, Frank Hoeller, and Timo Röhling. The ROS multimaster extension for simplified deployment of multi-robot systems. In Robot operating system (ROS), pages 629–650. Springer, 2016.
- [127] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.
- [128] Marco Tranzatto, Frank Mascarich, Lukas Bernreiter, Carolina Godinho, Marco Camurri, Shehryar Khattak, Tung Dang, Victor Reijgwart, Johannes Loeje, David Wisth, Samuel Zimmermann, Huan Nguyen, Marius Fehr, Lukas Solanka, Russell Buchanan, Marko Bjelonic, Nikhil Khedekar, Mathieu Valceschini, Fabian Jenelten, Mihir Dharmadhikari, Timon Homberger, Paolo De Petris, Lorenz Wellhausen, Mihir Kulkarni, Takahiro Miki, Satchel Hirsch, Markus Montenegro, Christos Papachristos, Fabian Tresoldi, Jan Carius, Giorgio Valsecchi, Joonho Lee, Konrad Meyer, Xiangyu Wu, Juan Nieto, Andy Smith, Marco Hutter, Roland Siegwart, Mark Mueller, Maurice Fallon, and Kostas Alexis. Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge, 2022.
- [129] Marco Tranzatto, Takahiro Miki, Mihir Dharmadhikari, Lukas Bernreiter, Mihir Kulkarni, Frank Mascarich, Olov Andersson, Shehryar Khattak, Marco Hutter, Roland Siegwart, and Kostas Alexis. Cerberus in the darpa subterranean challenge. Science Robotics, 7(66):eabp9742, 2022.
- [130] Marco Tranzatto, Takahiro Miki, Mihir Dharmadhikari, Lukas Bernreiter, Mihir Kulkarni, Frank Mascarich, Olov Andersson, Shehryar Khattak, Marco Hutter, Roland Siegwart, et al. Cerberus in the darpa subterranean challenge. Science Robotics, 7(66):eabp9742, 2022.
- [131] Maria Tsimpoukelli, Jacob L Menick, Serkan Cabi, SM Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models. Advances in Neural Information Processing Systems, 34:200–212, 2021.
- [132] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In Chi conference on human factors in computing systems extended abstracts, pages 1–7, 2022.

- [133] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.
- [134] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017.
- [135] Sai Vemprala, Rogerio Bonatti, Arthur Buckner, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft, February 2023.
- [136] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In International Conference on Machine Learning, pages 23318–23340. PMLR, 2022.
- [137] Xiu-Shen Wei, Chen-Wei Xie, Jianxin Wu, and Chunhua Shen. Mask-cnn: Localizing parts and selecting descriptors for fine-grained bird species categorization. Pattern Recognition, 76:704–714, 2018.
- [138] Jason Williams, Shu Jiang, Matthew O’Brien, Glenn Wagner, Emili Hernandez, Mark Cox, Alex Pitt, Ron Arkin, and Nicolas Hudson. Online 3d frontier-based ugv and uav exploration using direct point cloud visibility. In 2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pages 263–270, 2020.
- [139] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, pages 1–10, 2022.
- [140] Brian Yamauchi. A frontier-based approach for autonomous exploration. In Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97. ‘Towards New Computational Principles for Robotics and Automation’, pages 146–151. IEEE, 1997.
- [141] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. Advances in neural information processing systems, 31, 2018.
- [142] Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context in referring expressions. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part II 14, pages 69–85. Springer, 2016.
- [143] Fanlong Zeng, Wensheng Gan, Yongheng Wang, Ning Liu, and Philip S. Yu. Large language models for robotics: A survey, 2023.
- [144] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. Path planning for the mobile robot: A review. Symmetry, 10(10):450, 2018.
- [145] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. Science China Technological Sciences, 63(9):1612–1627, 2020.

- [146] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.
- [147] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4490–4499, 2018.
- [148] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. arXiv preprint arXiv:1905.05055, 2019.

Appendix A

Full Language Model Prompts

A.1 NavCon: Code Generation Language Prompt

Navigation Prompt for the concatenated single image input in real-world environments. The prompt was based on the one presented in[122].

```
import math

class ImagePatch:

    """A Python class containing a crop of an image centered around a particular
        object, as well as relevant information.

    Attributes
    -----

    cropped_image : array_like
        An array-like of the cropped image taken from the original image.

    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of
        the crop's bounding box in the original image.

    frame: name of camera frame

    Methods
    -----
```

```

find(object_name: str)->List[ImagePatch]

    Returns a list of new ImagePatch objects containing crops of the image
    centered around any objects found in the
    image matching the object_name.

exists(object_name: str)->bool

    Returns True if the object specified by object_name is found in the image,
    and False otherwise.

verify_property(property: str)->bool

    Returns True if the property is met, and False otherwise.

best_text_match(option_list: List[str], prefix: str)->str

    Returns the string that best matches the image.

simple_query(question: str=None)->str

    Returns the answer to a basic question asked about the image. If no
    question is provided, returns the answer to "What is this?".

llm_query(question: str, long_answer: bool)->str

    References a large language model (e.g., GPT) to produce a response to the
    given question. Default is short-form answers, can be made long-form
    responses with the long_answer flag.

compute_depth()->float

    Returns the median depth of the image crop.

crop(left: int, lower: int, right: int, upper: int)->ImagePatch

    Returns a new ImagePatch object containing a crop of the image at the
    given coordinates.

"""

def __init__(self, image, left: int = None, lower: int = None, right: int =
    None, upper: int = None, frame = None):

```



```

"""Initializes an ImagePatch object by cropping the image at the given
    coordinates and stores the coordinates as
    attributes. If no coordinates are provided, the image is left unmodified,
    and the coordinates are set to the
    dimensions of the image.

Parameters
-----

image : array_like
    An array-like of the original image.

left, lower, right, upper : int
    An int describing the position of the (left/lower/right/upper) border
    of the crop's bounding box in the original image.
"""

if left is None and right is None and upper is None and lower is None:
    self.cropped_image = image
    self.left = 0
    self.lower = 0
    self.right = image.shape[2] # width
    self.upper = image.shape[1] # height
else:
    self.cropped_image = image[:, lower:upper, left:right]
    self.left = left
    self.upper = upper
    self.right = right
    self.lower = lower

self.width = self.cropped_image.shape[2]

```

```

self.height = self.cropped_image.shape[1]

self.horizontal_center = (self.left + self.right) / 2
self.vertical_center = (self.lower + self.upper) / 2

self.frame = frame

def find(self, object_name: str) -> List[ImagePatch]:
    """Returns a list of ImagePatch objects matching object_name contained in
        the crop if any are found.
    Otherwise, returns an empty list.
    Parameters
    -----
    object_name : str
        the name of the object to be found

    Returns
    -----
    List[ImagePatch]
        a list of ImagePatch objects matching object_name contained in the
        crop

    Examples
    -----
    >>> # return the foo

    >>> def execute_command(image) -> List[ImagePatch]:

    >>> image_patch = ImagePatch(image)

```

```

>>> foo_patches = image_patch.find("foo")

>>> return foo_patches

"""

return find_in_image(self.cropped_image, object_name)

def exists(self, object_name: str) -> bool:

    """Returns True if the object specified by object_name is found in the
        image, and False otherwise.

    Parameters
    -----

    object_name : str

        A string describing the name of the object to be found in the image.

    Examples
    -----

    >>> # Are there both foos and garply bars in the photo?

    >>> def execute_command(image)->str:

    >>> image_patch = ImagePatch(image)

    >>> is_foo = image_patch.exists("foo")

    >>> is_garply_bar = image_patch.exists("garply bar")

    >>> return bool_to_yesno(is_foo and is_garply_bar)

    """

    return len(self.find(object_name)) > 0

def verify_property(self, object_name: str, visual_property: str) -> bool:

    """Returns True if the object possesses the visual property, and False
        otherwise.

```

Differs from 'exists' in that it presupposes the existence of the object specified by object_name, instead checking whether the object possesses the property.

Parameters

object_name : str

A string describing the name of the object to be found in the image.

visual_property : str

A string describing the simple visual property (e.g., color, shape, material) to be checked.

Examples

```
>>> # Do the letters have blue color?
```

```
>>> def execute_command(image) -> str:
```

```
>>> image_patch = ImagePatch(image)
```

```
>>> letters_patches = image_patch.find("letters")
```

```
>>> # Question assumes only one letter patch
```

```
>>> return bool_to_yesno(letters_patches[0].verify_property("letters", "
    blue"))
```

```
"""
```

```
return verify_property(self.cropped_image, object_name, property)
```

```
def best_text_match(self, option_list: List[str]) -> str:
```

```
    """Returns the string that best matches the image.
```

Parameters

```
option_list : str
```

```
    A list with the names of the different options
```

```
prefix : str
```

```
    A string with the prefixes to append to the options
```

Examples

```
-----
```

```
>>> # Is the foo gold or white?
```

```
>>> def execute_command(image)->str:
```

```
>>> image_patch = ImagePatch(image)
```

```
>>> foo_patches = image_patch.find("foo")
```

```
>>> # Question assumes one foo patch
```

```
>>> return foo_patches[0].best_text_match(["gold", "white"])
```

```
"""
```

```
return best_text_match(self.cropped_image, option_list)
```

```
def simple_query(self, question: str = None) -> str:
```

```
    """Returns the answer to a basic question asked about the image. If no
```

```
    question is provided, returns the answer
```

```
    to "What is this?". The questions are about basic perception, and are not
```

```
    meant to be used for complex reasoning
```

```
    or external knowledge.
```

Parameters

```
-----
```

```
question : str
```

```
    A string describing the question to be asked.
```

Examples

```
>>> # Which kind of baz is not fredding?

>>> def execute_command(image) -> str:

>>> image_patch = ImagePatch(image)

>>> baz_patches = image_patch.find("baz")

>>> for baz_patch in baz_patches:

>>> if not baz_patch.verify_property("baz", "fredding"):

>>> return baz_patch.simple_query("What is this baz?")


>>> # What color is the foo?

>>> def execute_command(image) -> str:

>>> image_patch = ImagePatch(image)

>>> foo_patches = image_patch.find("foo")

>>> foo_patch = foo_patches[0]

>>> return foo_patch.simple_query("What is the color?")


>>> # Is the second bar from the left quuxy?

>>> def execute_command(image) -> str:

>>> image_patch = ImagePatch(image)

>>> bar_patches = image_patch.find("bar")

>>> bar_patches.sort(key=lambda x: x.horizontal_center)

>>> bar_patch = bar_patches[1]

>>> return bar_patch.simple_query("Is the bar quuxy?")

"""

return simple_query(self.cropped_image, question)
```

```

def compute_depth(self):

    """Returns the median depth of the image crop

    Parameters
    -----

    Returns
    -----

    float

        the median depth of the image crop

    Examples
    -----

    >>> # the bar furthest away

    >>> def execute_command(image)->ImagePatch:

    >>> image_patch = ImagePatch(image)

    >>> bar_patches = image_patch.find("bar")

    >>> bar_patches.sort(key=lambda bar: bar.compute_depth())

    >>> return bar_patches[-1]

    """

    depth_map = compute_depth(self.cropped_image)

    return depth_map.median()


def crop(self, left: int, lower: int, right: int, upper: int) -> ImagePatch:

    """Returns a new ImagePatch cropped from the current ImagePatch.

    Parameters
    -----

    left, lower, right, upper : int

```

```

        The (left/lower/right/upper)most pixel of the cropped image.

-----

"""

return ImagePatch(self.cropped_image, left, lower, right, upper, self.
    frame)

def overlaps_with(self, left, lower, right, upper):
    """Returns True if a crop with the given coordinates overlaps with this
        one,
    else False.

    Parameters
    -----

    left, lower, right, upper : int
        the (left/lower/right/upper) border of the crop to be checked

    Returns
    -----

    bool
        True if a crop with the given coordinates overlaps with this one, else
        False

    Examples
    -----

    >>> # black foo on top of the qux

    >>> def execute_command(image) -> ImagePatch:

    >>> image_patch = ImagePatch(image)

    >>> qux_patches = image_patch.find("qux")

```



```

>>> qux_patch = qux_patches[0]

>>> foo_patches = image_patch.find("black foo")

>>> for foo in foo_patches:

>>> if foo.vertical_center > qux_patch.vertical_center

>>> return foo

"""

return self.left <= right and self.right >= left and self.lower <= upper

    and self.upper >= lower

def best_image_match(list_patches: List[ImagePatch], content: List[str],
return_index=False) -> Union[ImagePatch, int]:

    """Returns the patch most likely to contain the content.

    Parameters
    -----

    list_patches : List[ImagePatch]

    content : List[str]

        the object of interest

    return_index : bool

        if True, returns the index of the patch most likely to contain the object

    Returns
    -----

    int

        Patch most likely to contain the object

    """

    return best_image_match(list_patches, content, return_index)

```

```

def distance(patch_a: ImagePatch, patch_b: ImagePatch) -> float:
    """
    Returns the distance between the edges of two ImagePatches. If the patches
        overlap, it returns a negative distance
    corresponding to the negative intersection over union.

    Parameters
    -----
    patch_a : ImagePatch
    patch_b : ImagePatch

    Examples
    -----
    # Return the qux that is closest to the foo

    >>> def execute_command(image):
    >>> image_patch = ImagePatch(image)
    >>> qux_patches = image_patch.find('qux')
    >>> foo_patches = image_patch.find('foo')
    >>> foo_patch = foo_patches[0]
    >>> qux_patches.sort(key=lambda x: distance(x, foo_patch))
    >>> return qux_patches[0]

    """
    return distance(patch_a, patch_b)

def bool_to_yesno(bool_answer: bool) -> str:

```

```

    return "yes" if bool_answer else "no"

def coerce_to_numeric(string):
    """
    This function takes a string as input and returns a float after removing any
    non-numeric characters.

    If the input string contains a range (e.g. "10-15"), it returns the first
    value in the range.

    """
    return coerce_to_numeric(string)

### Nav Client
"""
Navigates an agent to an object in an image given its center coordinates

Parameters
    -----
    x : x coordinate of the center of the object
    y : y coordinate of the center of the object
"""

Examples
    -----
    >>> # Go to the blue foo.

    >>> def execute_command(image)

    >>> image_patch = ImagePatch(image)

    >>> foo_patches = image_patch.find("foo")

    >>> # Verify visual property

```

```

>>> blue_color_patches = []

>>> for foo_patch in foo_patches:

>>> if verify_property(blue, "color")

>>> blue_color_patches.append(foo_patch)

>>> inputs = (blue_color_patches[0].horizontal_center, blue_color_patches
              [0].vertical_center)

>>> return {'function': 'nav_function', 'inputs': inputs, 'box': [
              blue_color_patches[0].left, blue_color_patches[0].lower,
              blue_color_patches[0].right, blue_color_patches.upper]}
"""

def navigate_to_object(double x, double y):

"""

```

Write a function using Python **and/or** the ImagePatch **class** (above) that could be executed to provide an answer to the query by calling one of the functions **in** the Nav Client (navigate_to_object).

Note using the ImagePatch **is not** required **for all** queries. If the query **is not** relevant to navigation, **return** None **for** the function **and** a string describing the problem.

Consider the following guidelines:

- Use base Python (comparison, sorting) **for** basic logical operations, left/right/up/down, math, etc.
- Use the llm_query function to **access** external information **and** answer

```

    informational questions not concerning the image.
- All properties such as color should be verified using the verify_property
  function. "go to the blue foo" implies "go to foo if foo is blue"
- The output of this function should be a dict {function: 'function in the nav
  client', inputs: 'inputs to the chosen nav client function', box: [left, lower
  , right, upper]}, or for an error {function: 'None', error: 'message
  describing the problem'}
- If more than one object is found pick the best match

Query: INSERT_QUERY_HERE

```

A.2 Cog Explore Language Model Prompts:

The full foundation model prompts for the foundation model are provided in the following Subsections.

A.2.1 VQA Prompting

Your task is to create a list of 3 to 5 questions for a VQA (visual question answering model) that generate a clear and concise description of the environment that can later be used by the robotic agent to navigate to the object or area described by this command: ***INSERT_QUERY_HERE***

The description you elicit by providing questions to the VQA model is going to be passed to an operator who will use its content to make guesses about what objects in the environment to try to detect to assist with navigation.

VQA models are great at providing specific feedback to specific questions.

The robot you are guiding is the spot robot from Boston Dynamics.

The object or area we are after may be in plain view or may require exploration to discover. Do not presume the existence of any objects that you think may be in the scene.

For example, DO NOT ask any questions about the target area or object described in the command above. For example it is correct to ask if the target object is visible in the scene, but it is incorrect to ask for details about the target object since it may or may not be in the image.

Ask yes or no questions about whether an object exists or ask general questions about the kind of environment depicted within the image. DO NOT ASK ANY OTHER KIND OF QUESTION.

Your work will be added after two starter questions (do not include them in your response):

["What is the general setting of the environment: domestic, industrial, natural?", "What are the most prominent features visible in this environment?"]

Expand on these questions with 3 to 5 of your own. Do not replicate the work done by the above generic questions. Write your response formatted as a python list. That is a list denoted by brackets with each question separated by a comma. Only write this list. Do not include anything else in your response.

For example, a list with the two above questions included would look like: ["What is the general setting of the environment: domestic, industrial, natural?", "

What are the most prominent features visible in this environment?"]

Note, your response must be a perfectly formatted list. You cannot include any other comments or characters in your response that are not precisely formatted in the same manner as the list example above. Make sure to include commas in between entries within the list. Make sure to include quotes around each entry within the list. These are strings within python, they require quotes.

Do not include any of the starter questions in your response. Just include the questions you have thought up.

The VQA model does not know anything about the position of the robot or that it is examining images from a robot. The questions must be straightforward and directly about the image itself.

A.2.2 Object Detection Label Prompting

INSERT_SCENE_DESCRIPTION

Given you have this description of an environment we want to identify a set of objects that help us navigate a robotic agent to fulfill the goal: *
INSERT_QUERY_HERE*

Given this information, generate a comprehensive list of all objects including the objects referenced in the prompt that may help the robotic agent fulfill the navigation request. This list will be used to guide an object detection machine learning model. So words in this list should be general and likely to

be spotted within the environment as we have characterized it in the scene description details above. You should look for specific individual items, such as tables, and plants that can be easily distinguished from the background. DO NOT look for general entities that could take up the entire frame such as wall, floor, path, road, or park. Labels like this should not be in your list.

Do not look for very large objects.

Always include the object or area you are searching for in this list. Remember this object detector is running in realtime on a moving robot.

This list should be between 1 and 5 objects in length. It should be formatted as a python list. For example, a list of the following objects would be precisely formatted as: ["chair", "table", "helmet", "tree"]

Note the first entry must be the goal object!

This list is the only thing that should be included in your response. The python list format must be followed precisely. Be sure to include quotes around the elements of the list.

Do not return anything besides this list.

A.2.3 Explore Iteration Prompting

You are guiding a robotic agent to solve the following robotic navigation task

`*INSERT_QUERY_HERE*`.

You are assisted by an exploration planner that produces potential new areas to explore in the form of graph and frontier points. Frontier points tend to be in areas you have explored less or have just discovered.

There is also an onboard object detector that produces 3D projections of points in space that correspond to objects from a given object list. Remember object points represent the centroid for an object and your planner will get you as close as it can to them.

Object points are often not perfectly labeled. Frontier points are calculated based on exploration potential. Object points will come from areas you have already observed.

Your objective is to select a point from among these within the scene, and determine what will serve as the robot's next waypoint.

You will only have waypoints in areas your robot can see with its lidar and cameras. So, it is important to probe areas that potentially offer advantages for views you have not yet seen. When you see new areas, you should explore them.

If you have been near a target for more than one call to this point selection logic, then it is not the correct target. The computer vision your robot has is not perfect but is tuned towards producing more false positives. Be adventurous in your exploration, travel to areas far away from you.

Remember your robot is continuously detecting objects and has a left, right and front cameras. When you select a point the planner will plan a direct route for your robot to take. If the target object is detected along your path, the robot will go to it and add an entry to the summary below.

Use the available information from the robot below along with any contextual information you have available to select the next point to navigate to.

The time since start of sim is:

INSERT_CURRENT_TIME seconds

The spot robot is now at:

INSERT_CURRENT_POSITION

Here is a description of the area spot is currently in:

INSERT_SCENE_DESCRIPTION

The robot has been instructed to search for these objects:

INSERT_OBJECT_LIST

You have the following list of points you may choose from (graph points are not labeled, frontier points and object points are). If a graph point is labeled as "new" then it is from a region you have discovered it in the last step and are now able to plan to.

You should go to new points! They represent unknown areas and new views for your cameras and lidar. You may select either a graph point or an object point.

INSERT_FRONTIER_OBJECT_POINT_NUMBERED_LIST

You have been asked to help in the past *INSERT_TOTAL_CALLS* times to inform this navigation task. You should never go to the same point multiple times. The simulation ends when you reach the target.

Here is a summary of the actions you performed in the last *INSERT_MEMORY_LENGTH* calls:

INSERT_PRIOR_STATES

INSERT_INTERRUPT_DESCRIPTION

Now that you know where you've been, try to explore areas you have not yet been to. Use the description of the environment in past states to go where the object is most likely. If you have no prior states, be aggressive in your exploration.

If you have been by a point for more than one call, you should move, even if it is the target object. The robot's vision makes mistakes. Do not go to the same point more than once! Check the coordinates (x,y,z) of the point, if you've gone near there before in any of these prior states, don't go there again.

Change points every time, even if the point is an object point which appears to be the target. Cycling back and forth between two areas will not give new observations. The environment is static if you have already been to an area your cameras should have found the object.

As you explore the scene, new points will become available for you to plan to. Try to select points on the edge of those available to you. Consider areas that will allow you to discover new points. If a point is far from the other points in your list and in your prior states, it is a good candidate.

Reason about the area of space defined by your points. Go to areas and rooms outside of spaces you have explored.

Avoid considering points near your current location if you do not believe they are near the target.

Do not worry about collisions. All points are safe to travel to. Explore the space aggressively. Seek out the object and trust the robot's vision to find it.

If you have a point or points which are labeled as a new point, you should go to one of these points, your task is to explore with contextual awareness. These new points are newly discovered and represent a high potential for discovering your target.

Recall, you should consider x, y, and z when deciding where to go for both the object locations and the potential frontier points you want to go to. Do not just look at one coordinate! Consider both x and y equally! Euclidian distance should be used, we want to be close to the target object across all dimensions.

The distance field in your point list is the Euclidian distance from the robot to the object.

NEVER GO TO THE SAME AREA TWICE IN A ROW!!!

INSERT_SPECIFIC_MODEL_INSTRUCTIONS

Based on the information above fill out the form below (if you select an object, make sure to explain why that object is related to the navigation task. Reference past states in your reasoning. You should always navigate in

reference to where you've been. Reason about the space you are in. Reference the objects you see and categorize your environment.):

I need to select a waypoint from a numbered list of graph points, frontier points and object points. The point I have selected is point number: [insert the graph, frontier or object point numeral number here]. I am selecting this point because I believe it makes strategic sense to get me closer to solving my navigation task "**INSERT_QUERY_HERE**". My environment can be described as [describe the environment]. My reasoning is that this point [insert your reasoning here].

A.2.4 GPT-3.5 Turbo Specific Prompt Additions

If you see the object go directly to it.

Avoid going to regions that you have been to before. Consider how close the point you are selecting is with those listed in your prior states. If it is nearby, don't return to it.

Use your vision to go to areas that are rich with objects like the one you are after.

Newly discovered areas that are far away are highly advantageous if they are far from where your robot has traveled before.

Use the points to gain a sense of space. Travel to regions where you have not yet been.

Don't be tricked by windows, if you can't get to interesting objects, they might be behind a window.

GO TO AREAS THAT ARE FAR AWAY! DON'T STAY IN ONE REGION!!

A.2.5 Claude Haiku Specific Prompt Additions

Avoid looping between areas you have already been to. Try to find new areas so you will generate graph points for them. If there is an area you have not been to yet, go to it. Never double back, keep exploring nearby unknown areas.

Consider each prior state, do not return to anywhere near these locations.

Newly discovered areas that are far away are highly advantageous if they are far from where your robot has traveled before.

Use the points to gain a sense of space. Travel to regions where you have not yet been.

Always prioritize new points when they are available.

GO TO AREAS THAT ARE FAR AWAY! DON'T STAY IN ONE REGION!!