# UC Santa Cruz

## UC Santa Cruz Electronic Theses and Dissertations

### Title

Integrating Vision Language Navigation with Autonomous Driving in Unmapped, Dynamic, Off-Road Environments

### Permalink

https://escholarship.org/uc/item/2fs9w3mq

### Author

Stefani, Eliana

### Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

# INTEGRATING VISION LANGUAGE NAVIGATION WITH AUTONOMOUS DRIVING IN UNMAPPED, DYNAMIC, OFF-ROAD ENVIRONMENTS

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE AND ENGINEERING

by

**Eliana Stefani**

December 2024

This Dissertation of Eliana Stefani is approved:

_____
Professor Gabriel Elkaim, PhD; chair

_____
Professor Leilani Gilpin, PhD

_____
Andrew Coats, PhD

_____
Peter Biehl
Vice Provost and Dean of Graduate Studies

Copyright © by

Eliana Stefani

2024

# Table of Contents

# List of Figures

# List of Tables

# Abstract

Integrating Vision Language Navigation with Autonomous Driving
in Unmapped, Dynamic, Off-Road Environments

Eliana Stefani

Autonomous robots capable of navigating complex, unstructured environments based on natural language commands represent a significant advancement in useability and capability, yet this functionality is currently underdeveloped in the AI and Robotics world. Traditional autonomous robots are typically limited to navigating pre-mapped areas with relatively simple environmental topologies (eg: vehicle roadways). This dissertation addresses the challenge of integrating Vision Language Navigation (VLN) with autonomous driving technologies, enhancing robots' ability to comprehend and execute natural language instructions in diverse, unstructured environments, while simultaneously performing mapping, navigation, and obstacle avoidance.

While traditional autonomous vehicles excel at on-road waypoint navigation in structured environments, they struggle to interpret and act on natural language commands. Additionally, research on autonomous driving in unmapped, dynamic environments remains limited, creating a gap in the utility of autonomous systems for tasks requiring complex, multi-step instructions or navigation in novel settings.

This research proposes a novel software stack that integrates VLN with autonomous driving technologies, including: SLAM (Simultaneous Localization and

Mapping) for mapping and localization; dynamic obstacle detection and avoidance; and path planning and execution. The VLN agent interprets natural language commands and generates waypoints for the autonomous driving system, which then performs real-time navigation, mapping, and obstacle avoidance. The approach is validated in 2D and 3D environments, using simulated and real-world scenarios.

The proposed system demonstrates successful navigation and task execution across multiple settings. In both real and simulated settings, the autonomous driving system reached its goals in 100% of test cases despite dynamic obstacles. In uneven terrain, the agent effectively mapped and navigated through complex, obstacle-ridden environments, accurately tracking 6-DOF location and orientation. The integration of the VLN agent with the autonomous driving stack enabled real-time navigation and task completion based on natural language instructions.

# Dedication

The seeds of this dissertation, which developed into my pursuit of robotics, autonomous driving, and Artificial Intelligence, were planted in my childhood imagination by my playful mechanical constructs “Garlin” and “Underwater Helicopter.” Fruits of those seeds are collected in this work, which is dedicated to the following people who helped bring it to fruition.

Thank you to the professors at Cal Poly, especially in Mechatronics, whose “Learn by Doing” curriculum taught me how to breathe life into my robots. Thank you to my UCSC professors and committee members who guided me through this PhD, teaching me how to build both the body and the brains – the autonomous robot and the AI.

To my family: Joe “Dr. Greywacke” Stefani, for introducing me to the beauty of science and answering an unending stream of questions; Meike Stefani, for setting an example of dogged tenacity in executing projects every day; and Jovanni Stefani, for his help and insight with complex math, quaternions, and manifolds.

To my partner: Dominic Pasquali, my adventure-buddy, who was there as I built and flew my first quadcopter in undergrad, to seeing my autonomous ground vehicles during my PhD. A very special thank you for his support in all aspects of this journey, including review of this dissertation, all the outings and excursions along the way, and the many more to come!

To my friends: Aurora Myers, my hype girl, for her unwavering support by means of “treats & retreats.” Matt Hudson, for his intriguing discussions, and in-depth review of this dissertation.

Cheers to those who “forge ahead and figure out,” that’s what engineering is all about!

# 1 Introduction

## 1.1 Overview

Recent advances in autonomous robotics have led to significant progress, with autonomous vehicles, for example, nearing the point where they can transport passengers to a set destination with little to no human oversight. Meanwhile, natural language processing (NLP) has advanced considerably, enabling systems to interpret increasingly complex language and allowing embodied agents to follow instructions. However, while autonomous systems such as self-driving cars excel in structured environments with predefined maps, the ability to interpret natural language commands for navigation in dynamic and unstructured environments remains a major challenge, particularly in unmapped and dynamic settings.

This dissertation bridges this gap by developing a software stack that combines autonomous driving capabilities: Simultaneous Localization and Mapping (SLAM), path planning, and dynamic obstacle avoidance; with a VLN (Vision Language Navigation) agent that converts natural language commands into actionable waypoints. This integration allows the robot to autonomously navigate dynamic and unstructured environments while following natural language instructions.

The key contributions of this research include: (1) The deployment of a real-world VLN agent capable of processing natural language commands and navigating dynamic environments. (2) The decoupling of waypoint prediction from obstacle

avoidance, thereby improving navigation efficacy. (3) The development of a path-planning system that operates without pre-existing maps, enabling navigation in unknown terrain. (4) Advancements in off-road autonomous driving, including the integration of real-time 6-DOF localization and mapping in complex 3D environments.

This novel approach decouples high-level waypoint prediction from real-time obstacle avoidance, enabling more efficient navigation by relying on dedicated autonomous driving sub-modules for obstacle handling. Additionally, it supports map-less navigation, enabling the robot to navigate and execute tasks in unmapped and dynamically changing environments.

Validated in both simulated and real-world tests, the system demonstrated successful waypoint navigation based on natural language commands, thereby bridging the gap between autonomous driving and natural language processing.

This research paves the way for future robots, with applications in disaster response, indoor tasking, and extraterrestrial exploration. Additionally, the findings contribute to the broader field of Human-Machine Teaming, facilitating more intuitive communication and more effective collaboration between humans and robots.

## 1.2 Understanding the Challenge

Imagine giving directions: “Drive past the two hills, then head west to where the forest meets the lake and drop off the bag at the medical tent.” An average driver or even a teenager navigating an ATV could likely execute that command with ease. But could an SAE Level 5 autonomous vehicle interpret and carry out such a natural language instruction? Not quite.

Now consider a different scenario in a burning building: “Drive down the hallway, take the second corridor on the right, and rescue any survivors.” While straightforward for a firefighter in a controlled environment, this task remains beyond the reach of current indoor robots, especially in complex or emergency conditions.

Why is this so challenging? Why is it that, for a human, simple instructions like “get the TV remote from the couch in the living room” are trivial, yet no autonomous vehicle or robot can perform even this simple task reliably?

The answer lies in the gap between autonomous vehicles’ ability to follow pre-defined waypoints and their inability to interpret and act upon natural language commands in dynamic, unstructured, unmapped environments. This dissertation addresses this gap by developing a novel system that integrates Vision Language Navigation (VLN) with autonomous driving technologies. By enabling robots to interpret natural language and navigate using autonomous driving capabilities, this research pushes the boundaries of autonomous systems – transforming how they can interact with and respond to complex, real-world tasks.

## 1.3 Building Blocks

This research develops an autonomous driving software stack that enables robots to navigate terrain based on natural language commands. A Vision Language Navigation agent interprets the natural language command and issues the next incremental waypoint to which the robot autonomously drives to, thus completing the inferred task. The autonomous driving agent navigates to the waypoint by performing mapping, localization, obstacle detection, path planning, and path execution.

The integration of VLN and autonomous driving combines the ability to navigate complex environments based on natural language instructions with the technological and safety advancements inherent in autonomous driving technology.

Advances in conversational NLP, which enhance intent comprehension, are paralleled by improvements in computer vision's ability to interpret images. The intersection of these two fields gives rise to Vision Language Navigation [1]: a type of cross-modal reinforcement learning (RL).

An overview of Autonomous Driving and Vision Language Navigation is provided in Figure 1.1. Here the VLN agent [2] is given an instruction in natural language: "Walk to the right around the fireplace, through the doorway, and turn right. Wait in the hallway by the pocket door." The agent observes its environment using RGB-D camera (Red, Green, Blue, and Depth channels), map, and pose (location and heading), and determines the next intermediate step for a collision-free route to the goal.

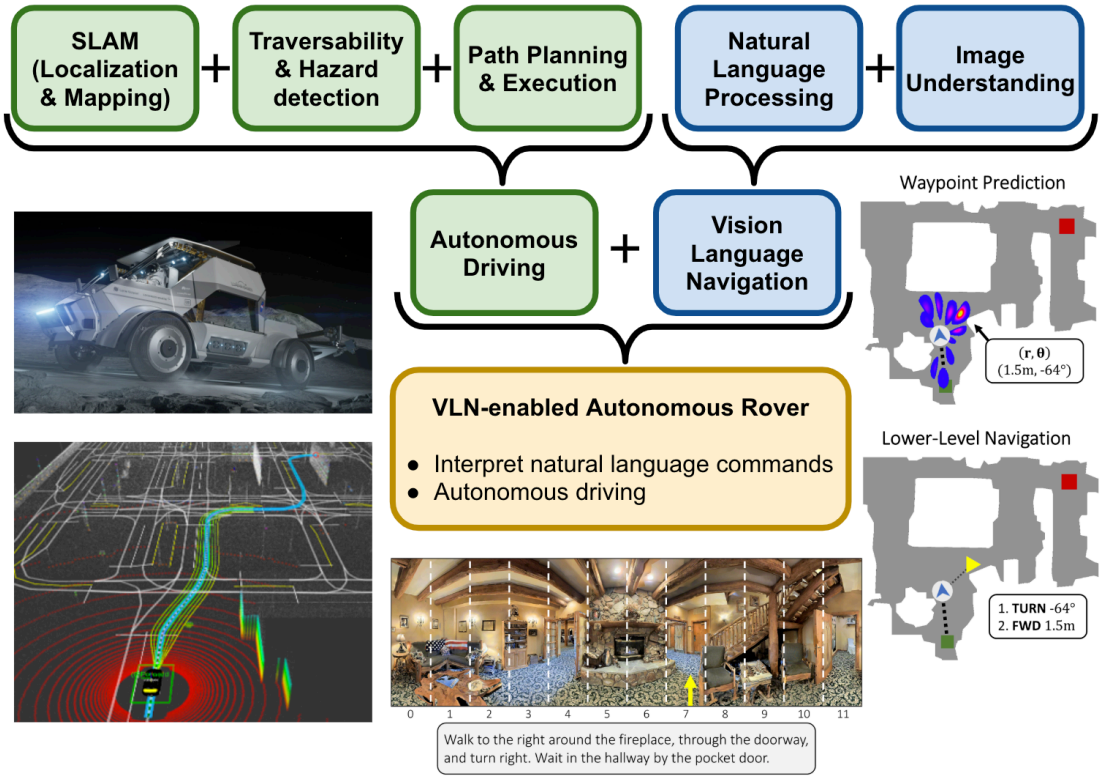



Figure 1.1 Building blocks of a VLN enabled autonomous driving robot

As explored in this research, the VLN agent need not explicitly determine a collision-free path; it can utilize dedicated subsystems for hazard detection and local planning. For instance, in scenarios where the agent has a clear line of sight to its target beyond an obstacle, it would be more efficient for proven autonomous driving technologies to handle path planning, rather than dedicating additional training resources for the VLN agent to learn and execute obstacle avoidance in addition to its instruction-following capabilities.

Autonomous vehicles perform the following to navigate through their environment (detailed in Section 5.2: Autonomous Driving Sub-Modules).

- SLAM: builds a representation of the environment while tracking position and orientation
- Hazard Detection & Traversability: determines viable driving areas
- Path Planning & Execution: plans a viable path through the environment, and executes said path while avoiding moving objects

While self-driving vehicles are progressing towards SAE Level 5 autonomy, the integration of these two well-established research domains remains underexplored in deployed AI; this research aims to addresses that gap.

## 1.4 Motivation

Deployed in the real world, these autonomously driving VLN agents can assist people, businesses, governments, and emergency response, addressing aspects of modern technological and societal needs.

Advancing the field of Human-Machine-Teaming, VLN allows a more seamless interaction between humans and robots by allowing users to communicate more intuitively in natural language. An autonomous agent operating in lockstep with a VLN agent can more efficiently navigate complex real-world environments (eg: disaster response, unmapped terrain, large-scale indoor environments). Autonomously driving VLN agents can provide disabled or limited-mobility individuals opportunities to operate independently, thereby allowing them to navigate public spaces independently, and live without an at-home assistant.

### 1.4.1 Applications

As at-home assistants, autonomously driving VLN agents can help fetch medicine, food, or a fallen TV remote control: tasks that require both eyesight, autonomous navigation, and the ability to interpret natural language commands. Several companies are developing robots in this area; some that focus on just the robot manipulation, others that focus on the NLP (Natural Language Processing) side of embodied agents, some that perform both. Amazon's Astro [3] can monitor the house (navigating to specific rooms to check on ingredients in the pantry or reporting a fire alarm); checking in on elderly relatives, and integrating with Amazon Alexa, but is limited to indoor environments and cannot operate independently from other equipment in new environments. Honda's ASIMO [4] robot was developed to move in spaces shared with humans, perform tasks using its hands, and interact with people by understanding natural spoken language. Boston Dynamic's Atlas [5], a humanoid robot, fluidly runs and jumps through complex environments, can manipulate objects with its hands, perceives the world in real-time, and understands its own dynamics to predict its motion over time.

In outdoor environments, construction sites, and assembly lines, these embodied VLN agents and autonomous driving robots can perform tasks that are too dangerous for people to perform. These autonomous agents could also deliver aid to impacted regions, provide emergency response, or perform Search & Rescue more rapidly than a human-only crew could perform.

Figure 1.2 Robot performing pick and place

This work also has applications that are “out of this world” – extraterrestrial exploration depends on autonomous driving in “off-road” terrain. This autonomy is critical to allow humans to explore unknown landscapes without the typical constraints of needing a driver onboard, or a continuous data link (as would be required for remote piloting).

Figure 1.3 Concept design of NASA’s Lunar Terrain Vehicle [6]

# 2 Background Literature

## 2.1 What is VLN: Vision Language Navigation?

### 2.1.1 An Introduction to VLN

Vision Language Navigation (VLN) is the task of navigating an embodied agent to carry out natural language instructions in real life 3D environments. It is the intersection of computer vision, Natural Language Processing (NLP, though sometimes known as NLU: Natural Language Understanding), and robotics. The goal of VLN is to enable autonomous robots (real and simulated) to navigate and interact with complex environments based on natural language instructions, and its perception of the environment.

There are several key components of Vision Language Navigation:

**Natural Language Processing:** VLN systems interpret natural language instructions from human users. These instructions can range from simple commands ("Go straight and turn left at the door") to more complex spatial descriptions ("walk past the red chair and go up the stairs to the second floor").

**Computer Vision:** Visual perception is crucial for VLN systems to understand and navigate through their environments effectively and efficiently. This may involve object detection (a classification problem), semantic segmentation (labelling every

pixel of an image with a category or label), depth estimation, and scene understanding.

**Spatial Reasoning:** VLN embodied agents must reason about spatial relationships within a given environment. This includes understanding distances, orientations, landmarks, and the layout of the environment based on visual and language cues.

**Path Planning and Navigation:** Once the natural language instructions and visual inputs are processed, the VLN agent will plan a path and navigate through the environment while avoiding obstacles and adhering to the commanded mission.

Given the complexity of this cross-modal task, VLN faces several key challenges:

**Ambiguity and uncertainty**: Natural language instructions can be ambiguous and context-dependent, requiring robust interpretation mechanisms.

**Cross-modal vision-language reasoning:** integrating visual perception with natural language understanding is non-trivial, as it involves performing inference on different modalities of data.

**Generalization:** VLN systems need to generalize across different environments and user instructions that may vary in complexity, detail, and users.

**Real-time performance:** For robotic applications, VLN agents must operate in real-time, which requires effective and efficient decision-making and navigation, under computational constraints.

**Non-sequential instructions:** VLN agents may have to understand and execute instructions that are not presented in chronological order

### 2.1.2 VLN Focus Area

This research focuses on the type of VLN agent that that receives a command in the form of 1-2 sentences (natural language text) which are instructions on how to proceed through an environment to get to a goal. Upon receiving the instruction, the agent continually observes its environment. The following sensors and data representations may be used:

- panoramic RGB-D (360° Red, Green, Blue, and Depth imagery)
- Map (occupancy grid)
- Location (offset with respect to starting location)
- Orientation (angle with respect to starting orientation)

The agent interprets the mission (described in natural language) with context to what it sees in its environment to determine the next step to take. The VLN agent does this by breaking down the natural language instructions into intermediate waypoints that the autonomous robot drives towards.

Section 4: Research: The VLN Agent describes how the VLN agent is designed as an embodied agent that interprets the world around it. Section 5: Research: Autonomous Driving describes how the autonomous driving agent is designed to execute the commands of the VLN agent by performing mapping in real-time, tracking location and heading, and executing waypoint navigation.

## 2.2 Previous Works in VLN

### 2.2.1 Navigations Graphs

Initial work in VLN traversed Navigation Graphs using the Room-to-Room (R2R) dataset. [7] This Navigation Graph in Figure 2.1 can be seen below in blue, superimposed on a top-down view of a building.

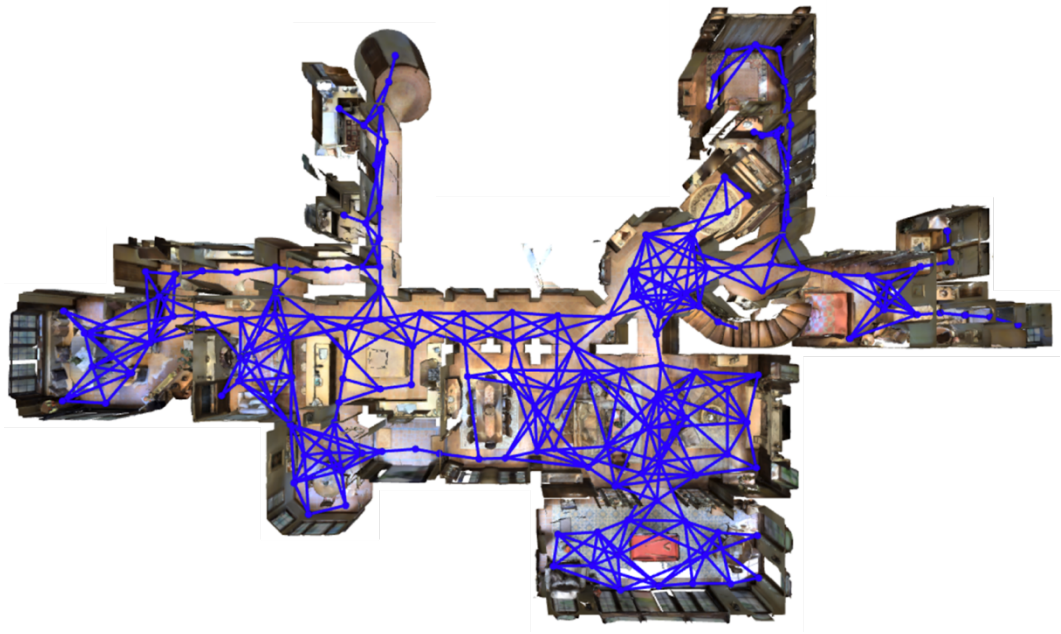


Figure 2.1 Navigation Graph in blue; panoramic images at vertices; agent traverses edges

The Navigation Graph is a graph of RGB-D panoramas at each vertex (node on graph), with collision-free edges connecting adjacent vertices. The robot traverses the simulated building by travelling along the connected vertices, node-to-node, across the Navigation Graph.

At a given node, a VLN agent in simulation observes the environment (eg: Figure 2.2), and determines which adjacent node (denoted in blue circles) to travel to next.

**Instruction:** Head upstairs and walk past the piano through an archway directly in front. Turn right when the hallway ends at pictures and table. Wait by the moose antlers hanging on the wall.

Figure 2.2 A scene in the R2R dataset; blue circles represent adjacent nodes

The first model architecture trained on this Navigation Graph was a Sequence-2-Sequence LSTM model (a Long Short-Term Memory recurrent neural network). [7] The first half encodes the natural language instruction into embedded space. The second half (decoder) receives the instruction embeddings from the encoder and receives sequential camera images of the environment embedded with ResNet-154, and infers action output.

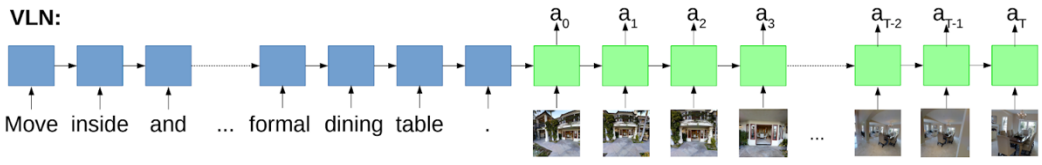



Figure 2.3 Sequence-2-Sequence LSTM

While this model did not execute anywhere near human performance, it still outperformed random actions, as shown in Figure 2.4. There is a loss of performance in new environments, which is further illustrated in Section 2.2.2: Sim-to-Real Transfer.

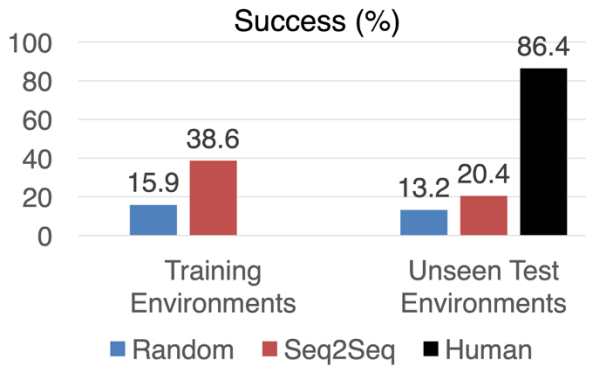



Figure 2.4 Results of Sequence-2-Sequence VLN agent on Navigation Graph

## 2.2.2 Sim-to-Real Transfer

The authors of the initial VLN paper [7] then tried deploying their agent on a real robot. [8] (Note that up to this point, all other VLN research had been strictly in simulation.) They reused the same Sequence-2-Sequence VLN agent trained in simulation over discrete action spaces, where it needs to be given "oracle knowledge" (knowledge assumed to be available, and exact) on pose (location and orientation).

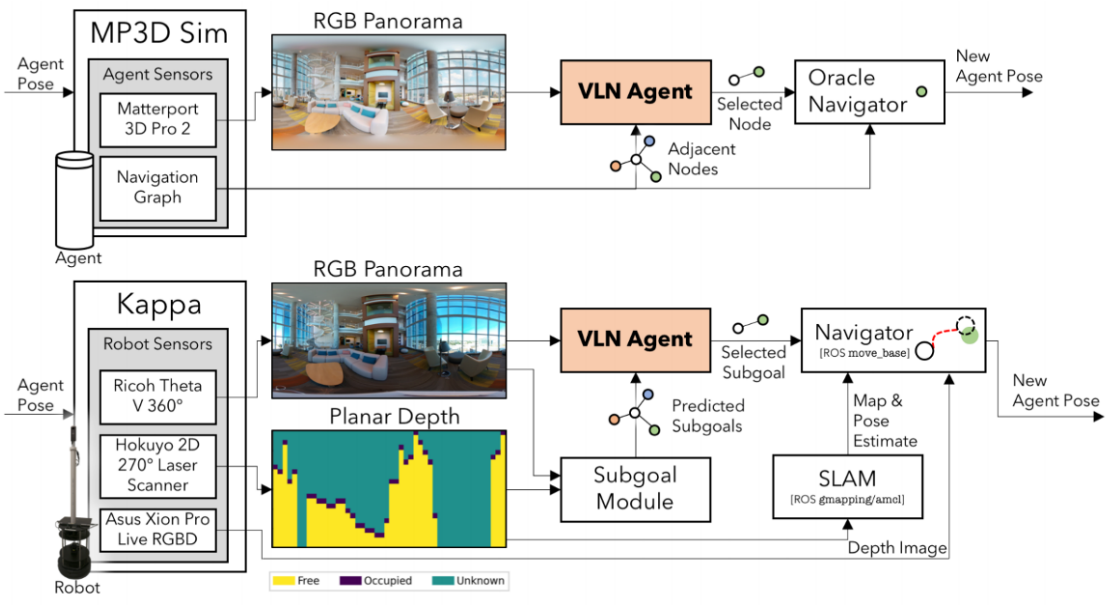



Figure 2.5 Software diagram showing the new "subgoal module" that gives the VLN agent candidate nodes

To give the VLN agent the same "choice of candidate nodes," a new subgoal module is trained to look at the RGB-D data and predict candidate nodes to select from. From here the agent turns to the goal and drives forward the specified distance.

The above technique resulted in a 22% success rate with no prior map collected, and a 47% success rate when the occupancy map and Navigation Graph were collected in advance. This research demonstrates that knowing the occupancy map in advance is helpful.

Section 5.5.2: Iterative Waypoint Navigation shows that the autonomous driving agent can create the map (and occupancy map) in real-time: a feature that is required for autonomous navigation, and is shown to help the VLN agent find better candidate nodes.

### 2.2.3 Dense Spatiotemporal Grounding

The researchers at Google Brain introduced a new RxR dataset with an associated model and results, which leverages Dense Spatiotemporal Grounding [9], where "each word in an instruction is time-aligned to the virtual poses of instruction creators and validators."

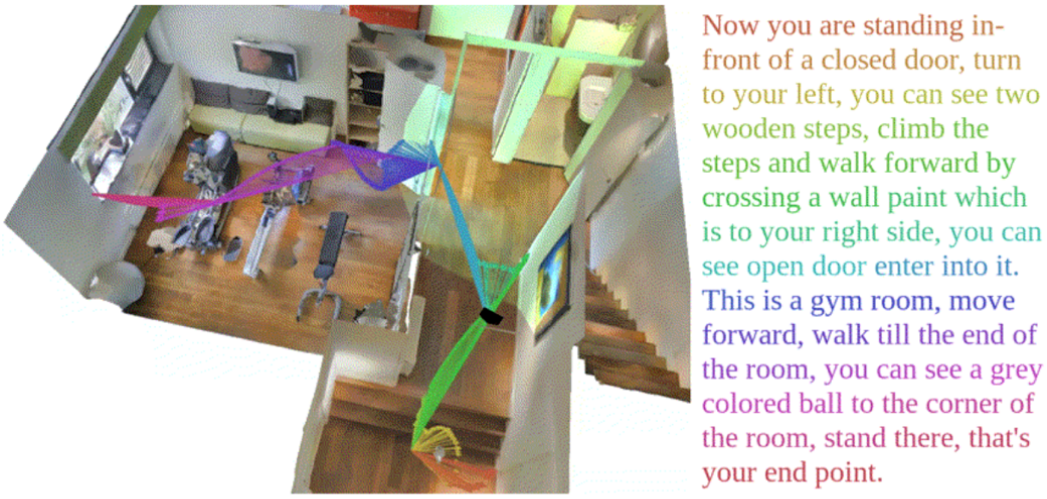



Figure 2.6 Instructions time-aligned to virtual poses

In the image above, we see the color-coded instructions matching with the different colored viewpoints in the pose trace (path of poses the agent took at each timestep). This alignment between pose and instruction fragment gives intuition to:

- How to relate words in the sentence to objects in the scene
- Gives continuous feedback to an agent as to how well it is following the target path
- Helps track path completion

## 2.2.4 Beyond the Navigation Graph

In the paper [10], the agent was forced to use only 90° FOV (Field of View) cameras, and had to decide on low-level actions {Left, Right, Forward, Stop}. To illustrate how much more complex this lower-level action space is, in one example it took 32 lower-level actions to complete a path that in the Nav Graph only took 3 hops (traveled along 3 edges). This is more complex because ill-picked lower-level actions may cause the agent to run into an obstacle or dead-end. Additionally, lower-level actions are less amenable to agents that tend to go straight, since if given a correct initial heading, agents have an advantage on the Navigation Graph, whereas an actual straight path in a continuous environment may result in a collision.


Figure 2.7 Scenes where agent chooses from Left/Right/Forward/Stop actions

### 2.2.5 Data Augmentations

Rather than just training on the correct paths, Data Augmentations [10] better utilize the limited data given. To improve the agent's training, three data augmentations were examined:

- "Aug.": (Synthetic Data Augmentation)
- "Dagger": (Dataset Aggregation)
- "PM": (Progress Monitor)

**Synthetic Data Augmentation (Aug.)** generates "new" labeled training data by learning the inverse-speaker model: it learns to describe its own trajectories, thus creating more instruction + trajectory pairs. During inference the agent can compare the verbalization of a potential path to the commanded instruction, allowing it to re-rank possible trajectories. Furthermore, synthetic training data can be generated by having the agent drive arbitrary paths and describe said paths, creating more instruction-path pairs. This is important because there is a very limited amount of labeled data to train from.

**Dataset Aggregation (DAgger)** [11] trains on both the expert trajectories and its own trajectories. This allows it to add to the dataset its own deviations from the goal path, and thus learning what not to do as it learns from its own mistakes. It does this by training on the aggregated set of trajectories from all iterations $1 \rightarrow n$ . Thus, the resulting policy after iteration $n$ is optimized over all past experiences.

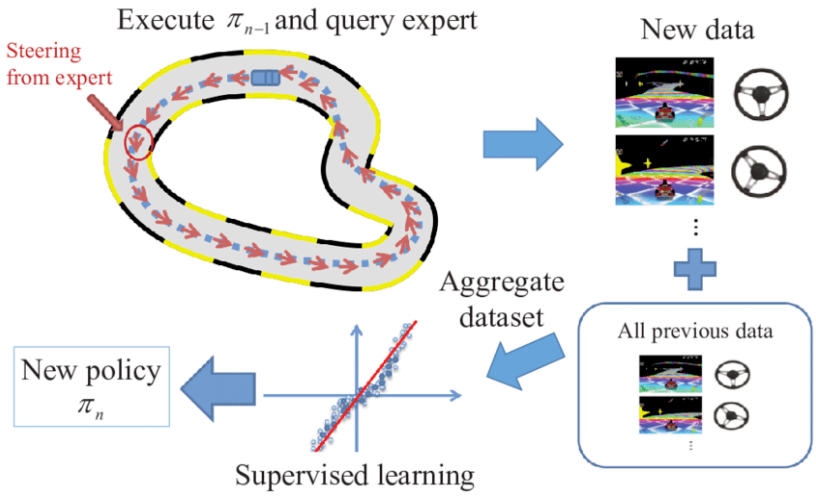



Figure 2.8 Example of DAgger (Dataset Aggregation) where new trajectory data is aggregated into the dataset which is then trained on

A **Progress Monitor (PM)** is a model that learns percent-completion of a trajectory, by receiving oracle distance information at every timestep. This helps the robot know when to terminate its path rather than overshooting its goal.

Table 2.1 Improvements using different Data Augmentations in the new model

| # | Model | PM [17] | DA [24] | Aug. [26] | Val-Seen | | | | | | Val-Unseen | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | TL ↓ | NE ↓ | nDTW ↑ | OS ↑ | SR ↑ | SPL ↑ | TL ↓ | NE ↓ | nDTW ↑ | OS ↑ | SR ↑ | SPL ↑ |
| 1 | Seq2Seq Baseline | - | - | - | 8.40 | 8.54 | 0.45 | 0.35 | 0.25 | 0.24 | 7.67 | 8.94 | 0.43 | 0.25 | 0.20 | 0.18 |
| 2 | | ✓ | - | - | 8.34 | 8.48 | 0.47 | 0.32 | 0.22 | 0.21 | 8.93 | 9.28 | 0.40 | 0.28 | 0.17 | 0.15 |
| 3 | | - | ✓ | - | 9.32 | 7.09 | 0.53 | 0.44 | 0.34 | 0.32 | 8.46 | 7.92 | 0.48 | 0.35 | 0.26 | 0.23 |
| 4 | | - | - | ✓ | 8.23 | 7.76 | 0.51 | 0.34 | 0.26 | 0.25 | 7.22 | 8.70 | 0.44 | 0.26 | 0.19 | 0.17 |
| 5 | | ✓ | ✓* | ✓ | 9.37 | 7.02 | 0.54 | 0.46 | 0.33 | 0.31 | 9.32 | 7.77 | 0.47 | 0.37 | 0.25 | 0.22 |
| 6 | Cross-Modal Attention | - | - | - | 8.26 | 7.81 | 0.49 | 0.38 | 0.27 | 0.25 | 7.71 | 8.14 | 0.47 | 0.31 | 0.23 | 0.22 |
| 7 | | ✓ | - | - | 8.51 | 8.17 | 0.47 | 0.35 | 0.28 | 0.26 | 7.87 | 8.72 | 0.44 | 0.28 | 0.21 | 0.19 |
| 8 | | - | ✓ | - | 8.90 | 7.40 | 0.52 | 0.42 | 0.33 | 0.31 | 8.12 | 8.00 | 0.48 | 0.33 | 0.27 | 0.25 |
| 9 | | - | - | ✓ | 8.50 | 8.05 | 0.49 | 0.36 | 0.26 | 0.24 | 7.58 | 8.65 | 0.45 | 0.28 | 0.21 | 0.19 |
| 10 | | ✓ | ✓* | ✓ | 9.26 | 7.12 | 0.54 | 0.46 | **0.37** | **0.35** | 8.64 | **7.37** | **0.51** | **0.40** | **0.32** | **0.30** |
| 11 | | ✓ | - | ✓ | 8.49 | 8.29 | 0.47 | 0.36 | 0.27 | 0.25 | 7.68 | 8.42 | 0.46 | 0.30 | 0.24 | 0.22 |
| 12 | | - | ✓* | ✓ | 9.32 | **6.76** | **0.55** | **0.47** | **0.37** | 0.33 | 8.27 | 7.76 | 0.50 | 0.37 | 0.29 | 0.26 |

In the unseen validation data, the use of the new model alone improves Success weighted by Path Length (SPL) by 22%. With the data augmentations, the new model improves SPL by 36%, a significant improvement over previous works.

## 2.3 What is Autonomous Driving?

An autonomous vehicle operates independent of human intervention by sensing and navigating its environment through a sophisticated integration of perception, path planning, and control systems. It utilizes an array of sensors, which may include lidar, radar, and cameras, to gather comprehensive data about its surroundings, such as road conditions, obstacles, and other vehicles. This sensory data is then processed to construct an accurate and detailed representation of the environment. [12] Once the vehicle's environment and its own position are understood, an onboard planning system devises a trajectory for the vehicle to reach its destination. Subsequently, the control system performs low-level actions to execute the planned trajectory.

## 2.4 Past Works in Autonomous Driving

There are a variety of autonomous vehicles, both terrestrial and extraterrestrial; both on-road and off-road. This work focuses predominantly on off-road rovers for terrestrial and extraterrestrial environments and building interiors.

### 2.4.1 Mars Rovers

The Mars rovers Spirit and Opportunity (operating 2003 – 2018) only had local obstacle avoidance with stereo vision cameras, but no global mapping, global path planning, or global localization. The rovers had to stop every 0.5 meters to process imagery of the environment to determine the next move. [13] [14]

The Perseverance Mars rover (beginning operation 2021) had faster cameras, and a computer dedicated to image processing, enabling it to perform navigation tasks while in motion. [15] [16]

## 2.4.2 DARPA Challenges

The DARPA Grand Challenge [17] was a race through the Mojave Desert from 2004-2005. Stanford's race vehicle “Stanley” pioneered Reinforcement Learning and won the Grand Challenge. It received a pre-defined course where it would stay on dirt roads, and not have to perform global localization or planning since it just moved forward on dirt roads (road finding). Stanley utilized lasers for terrain mapping and labelling, while using computer vision (color cameras) for terrain analysis. Stanley implemented obstacle detection; recorded 2D maps showing drivable, occupied, or unknown terrain; and used the same 2D map for its navigation engine [18]. The work shown in Section 5: Research: Autonomous Driving similarly creates a 2D costmap with three possibilities cell types (traversable, lethal, or unknown), which is ingested by the path planner. This concept is used for both global and local planners.


Figure 2.9 Stanley: Stanford’s race car and winner of DARPA Grand Challenge

The DARPA Urban Challenge [19] featured a city-based alternative in the year 2007. While Stanford competed again with Junior [20], Carnegie Mellon's "Boss" won first place. As shown in Figure 2.10, the system architecture of Boss consists of five broad areas: Missions Planning, Motion Planning, Behavior Generation, Perception and World Modeling, and Mechatronics. [21]

In Section 5.1: Autonomous Driving System Architecture, a similar system architecture is implemented with slightly different responsibilities to overcome the lack of a priori maps: Global Planning, Local (Motion) Planning, SLAM, and hardware interface (sensors and motor controllers).

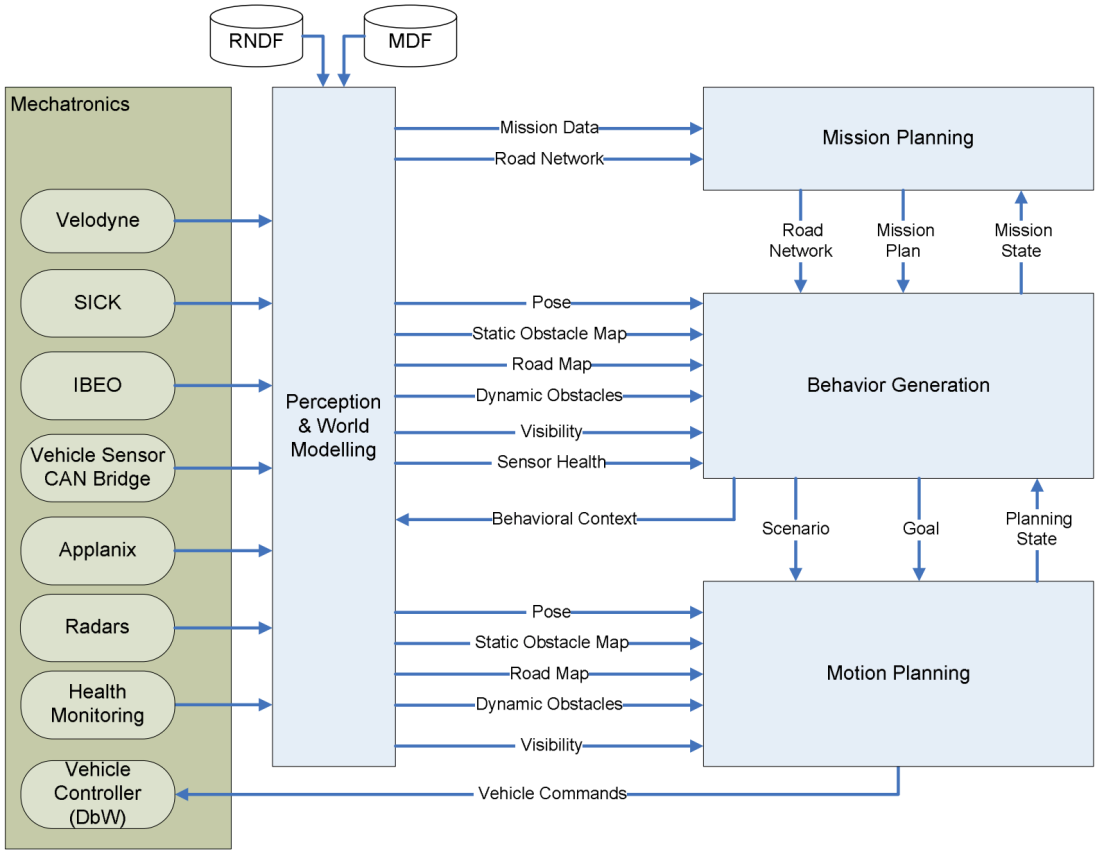



Figure 2.10 System architecture of Carnegie Mellon's Tartan Racing vehicle "Boss" [21]

The DARPA Subterranean Challenge [22] [23] hosted multiple underground navigation competitions in unstructured environments between the years 2017-2024. The winner [24] of this challenge was Cerberus [25], a team of 4-legged dog robots and aerial robots, shown in Figure 2.11. The robot performed SLAM, object detection, path planning, and multi-agent reasoning autonomy. It implemented a stop and go behavior as it was path planning, looking at patches of surface norm and height difference to determine traversability. [26]

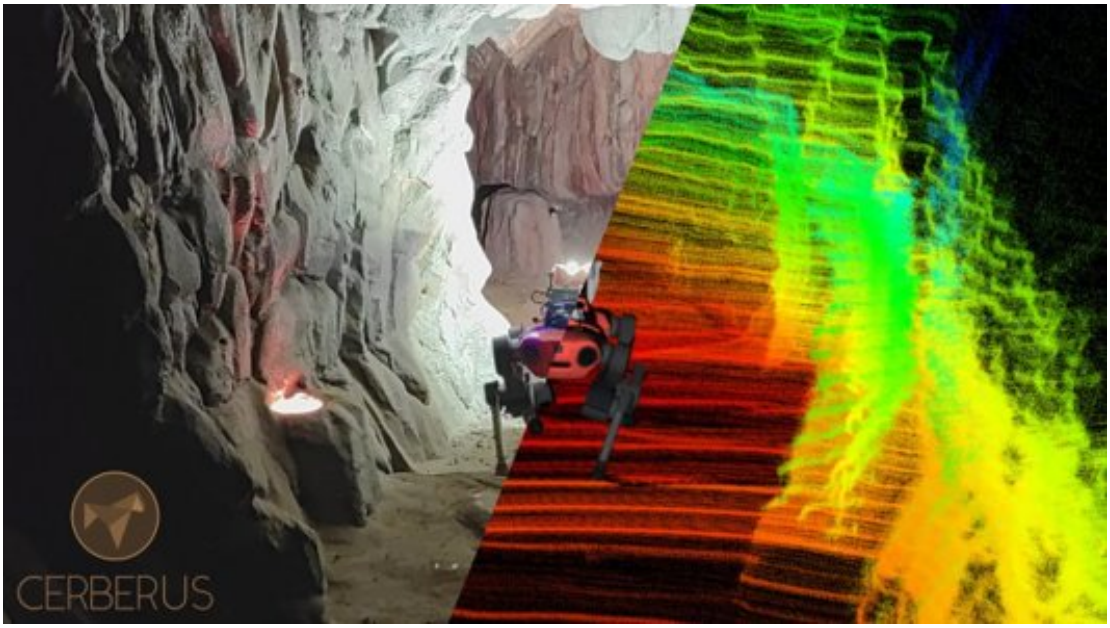



Figure 2.11 Winning robot CERBERUS for DARPA Subterranean Challenge [27]

### 2.4.3 Differences in On-Road vs Off-Road Autonomy

There are several on-road commercial autonomous vehicles, eg: Waymo, and several others that are seen out driving the streets of Silicon Valley. Even Jeep is now developing off-road autonomy [28]. While significant progress has been made for on-road autonomy, there are significant aspects that differentiate it from off-road driving, offering both additional challenges and conveniences.

On-road vehicles need to understand other aspects of its environment, eg: traffic signals, right of way, lane change and signaling, attention to pedestrians and cyclists, and “aggressions understanding.”

Though here are several places where on-road autonomy enjoys convenience not found in most off-road applications, such as:

- Assuming surface is flat and drivable
- Speed limit defines upper limit of allowable speed
- Area is pre-mapped
- Area has GNSS (eg: GPS, differential GPS), and magnetometer
- Area has lane-lines to help localize
- Onboard processing power unconstrained by SWaP (Size, Weight, and Power)

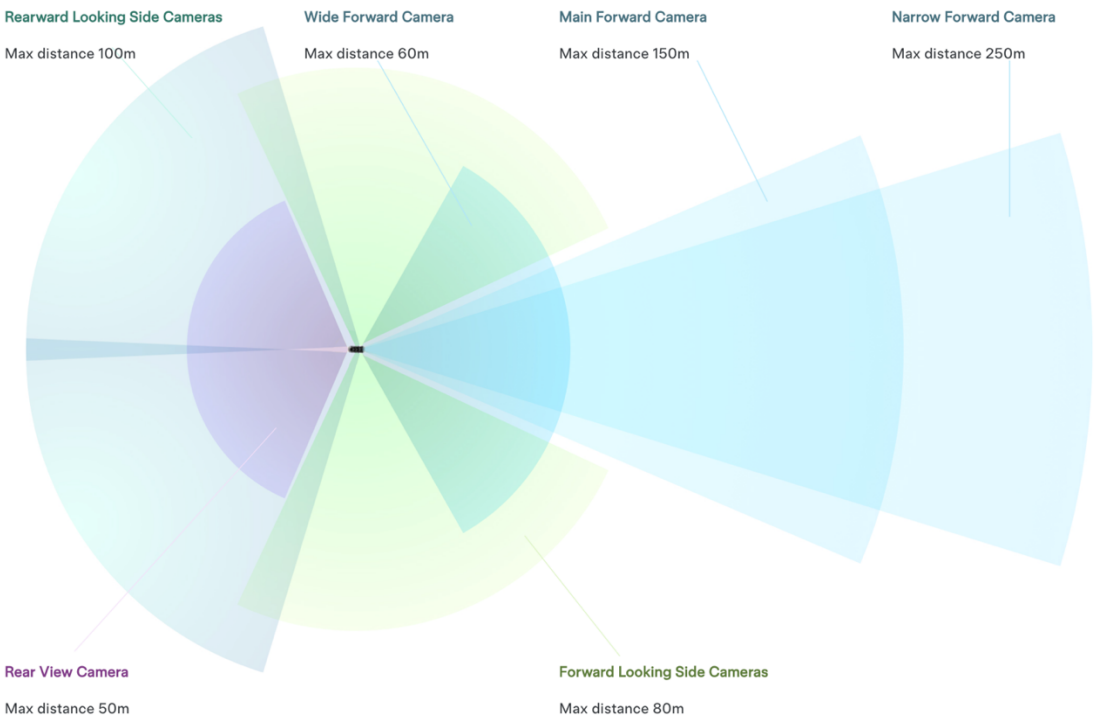



Figure 2.12 Example sensor configuration of on-road autonomous vehicles [29]

# 3 Contributions

## 3.1 Deployed VLN: Overcoming Unrealistic Assumptions

To facilitate the shift from simulated VLN tasks to practical real-world applications, a VLN agent and autonomous rover system was deployed on real hardware, operating on a real-time embedded platform. The VLN-enabled autonomous robot was constructed utilizing foundational elements from both autonomous driving and VLN embodied agents, as illustrated Figure 1.1.

First, (as discussed in Section 4:Research: The VLN Agent) a VLN agent was trained to interpret natural language commands, perceive the environment, and generate waypoints that progressively align with the provided natural language commands. To facilitate the transition from a simulated to a real-world environment, the VLN agent avoids reliance on unrealistic, simulation-specific assumptions prevalent in literature such as: a pre-defined obstacle-free navigation graph with a known topology, oracle navigation (“teleporting” node-to-node along each edge in one time-step), perfect localization (ground truth provided by simulation), or using a panoramic RGB-D sensor [10].

Subsequently, (as discussed in Section 5: Research: Autonomous Driving) an autonomous driving software stack was developed to support the VLN agent by

driving to the commanded waypoints. The autonomous driving stack generates an in situ map and continually tracks its position, overcoming the reliance of "known topology" – knowing the topology in advance, and "perfect localization" – receiving perfect location information during an episode. Furthermore, the autonomous driving stack also performs path planning, execution, and obstacle avoidance, providing the agent a way to navigate through a real environment without collision into other obstacles.

## 3.2 Decoupled Obstacle Avoidance

Decoupling waypoint prediction from obstacle avoidance enables a VLN agent to concentrate solely on high level waypoint estimation.

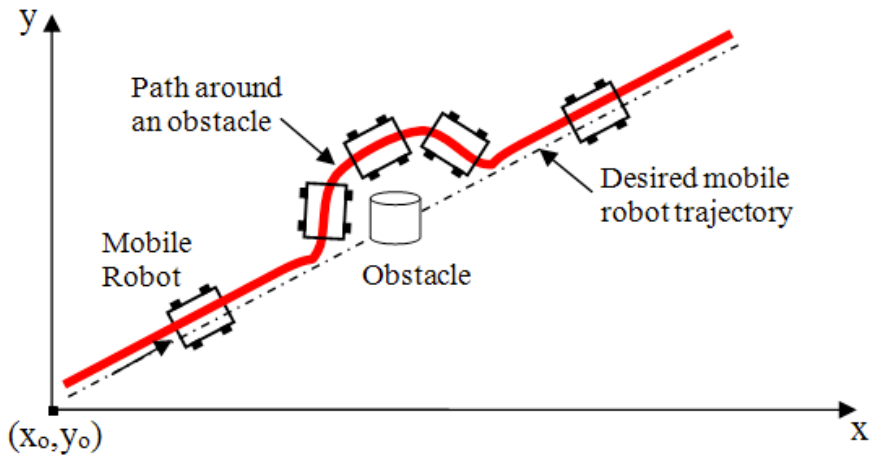



Figure 3.1 Deterministic planners and low-compute controllers aid obstacle avoidance [30]

In existing research, VLN agents are tasked with simultaneously planning for proximate waypoints and avoiding obstacles. This dual responsibility complicates the training process since the VLN agent must learn to navigate at a granular level, generating a path through immediate waypoints while ensuring collision-free travel. This approach requires an understanding of both autonomous driving, and cross-modal reasoning between natural language instructions and sensor images.

In contrast, allowing the VLN agent to focus on higher-level waypoint selection is akin to being dictated directions in a parking lot where one is told to walk through the parking lot to the front door of a building. In this analogy, the detailed navigation through intermediate obstacles, such as cars and trees, is managed independently by the individual, with the preference being for the directions to address the broader waypoint goal.

To implement this, waypoint prediction is decoupled from obstacle avoidance by employing deterministic, low-computation planners and controllers that operate in real-time without the need for extensive training. Specifically, the A* algorithm is implemented for the global path planner as outlined in Section 5.2.10: Global Planning. An Elastic Band planner is used for the motion/local planner outlined in Section 5.2.11: Local Planning. For the velocity controller, a straightforward PID (Proportional-Integral-Derivative) controller is implemented, as outlined in Section 5.2.3: Drive Control & Odometry Feedback.

## 3.3 Path Planning Without An A Priori Map

The autonomous driving stack is extended to navigate to waypoints in novel environments without a pre-existing map. Rather than localizing relative to an a priori map, the agent employes SLAM to both track its location and construct a map relative to its initial reference frame or any predetermined coordinate system. This approach allows waypoints to be issued in either a relative or global coordinate system, even if the map is incomplete or has not yet been created for that specific waypoint, thereby enabling path planning into previously unmapped areas. This capability is showcased in Section 5.5.1: 3D SLAM: Implementation & Results, and Section 5.5.2: Iterative Waypoint Navigation in Unmapped Environments.

To do this, the planners make opposite and complementary assumptions: the global planner is optimistic in assuming an unmapped area is traversable, whereas the local planner is cautious and assumes an unmapped area is not traversable.

The global planner operates under the assumption that unmapped areas are traversable and free of obstacles. This optimistic approach enables the planner to navigate towards waypoints even in regions where the map is incomplete. As the vehicle progresses, SLAM incrementally fills in the map. If the global planner were to assume unknown areas were hazardous, it would never venture into unmapped regions altogether, limiting exploration.

Conversely, the local planner adopts a cautious stance, assuming that unmapped areas are potentially hazardous and not traversable due to the risk of encountering

obstacles, either confirmed or undetermined. The local planner's cautious strategy (assuming unmapped areas are hazardous) requires real-time confirmation of traversability within its sensor field of view. As it follows the global path, the local planner ensures that no hazards are present and that the ground plane is present before proceeding.

This real-time validation aligns with the human intuition of an explorer who, while moving towards an out-of-sight destination, verifies the safety of each step along the way.

## 3.4 Initial Work in Off-Road Autonomy

This section outlines progress made in advancing off-road autonomous driving technologies for mapping and navigating complex, unstructured 3D terrains. This work encompasses several key areas:

Firstly, real-time 3D 6-DOF SLAM (Simultaneous Localization and Mapping) system is integrated with navigation capabilities, enabling the autonomous vehicle to continuously map and localize itself within dynamic environments. The SLAM is considered "3D" since it maps in a 3-dimensional point cloud of its environment, and is considered "6-DOF" because tracks pose within six degrees of freedom: three spatial dimensions (X,Y,Z, or Latitude, Longitude, Altitude), and three orientation dimensions (roll, pitch yaw, though this is represented using a unit quaternion). This

is demonstrated in Section 5.5.1.1: 3D Mapping and 6-DOF Localization in Cave Environment.

Additionally, groundwork for future work is explored by initiating research into 6-DOF localization in GPS-denied environments. This effort aims to develop techniques for precisely estimating position and orientation in a global reference frame, which is crucial for effective navigation through challenging terrains. This is discussed in Section 5.2.6.1: Methods for Global Localization in GPS denied environments.

These advancements represent a significant improvement over previous methodologies: unlike earlier approaches, the system operates continuously in real-time, eliminating time consuming stop-and-go behavior. It is developed in a physics-based simulation, accounting for vehicle rollover, traction, and inertia, which enhances both realism in simulation, and reliability when transferring to a real environment. It uses realistic sensors models that factor in sensor noise and field of view constraints, offering a more accurate representation of environmental conditions, and easing the transition to real-world deployment and testing. Notably, it functions effectively in new, unmapped environments without relying on pre-existing maps, and operates in GPS-denied settings, overcoming limitations of past approaches that depend on a priori map information. Potential applications of this system include search and rescue operations in disaster areas, subterranean environments, or extraterrestrial exploration.

# 4 Research: The VLN Agent

This chapter discusses the development and deployment of the VLN agent. The agent was initially trained using RL in a simulated environment with simulated sensor streams. The agent then transitioned to utilizing real hardware sensors and an Autonomous Driving stack for real-time path execution, location and orientation estimation, and map creation. The Autonomous Driving stack is later detailed in Section 5: Research: Autonomous Driving. The transition validates the agent's capability, and highlights the challenges of real-world navigation, including issues of sensor alignment, localization accuracy, and environmental variability.

The subsequent sections will detail the architecture of the VLN agent, the methodologies employed in its training, and the integration of its components within the autonomous driving framework. The successes and limitations encountered during deployment provide insights into the ongoing development of embodied, autonomous VLN agents that bridge the gap between simulation and real-world applications.

## 4.1 VLN Problem Statement

In a VLN task, the agent follows natural language instructions $X = \{x_1, x_2, \dots, x_l\}$ to follow a trajectory through its environment, stopping at a target location. A total of length $l$ word tokens $x_i$ make up each instruction. Initial simulated environments were represented by undirected graphs consisting of panoramic RGB-D images at the

nodes $v \in \mathbb{V}$, and edges $(v, u) \in \mathbb{E}$ that the agent travels along in one timestep. [31] In this work however, the agent is relieved of the navigation graph and moves in a continuous environment. Actions are low level velocity commands, movement is not deterministic, and location estimation is approximate but not perfect. The agent's state at time $t$ is $s_t = (v_t, P_t, M_t)$ where $v_t$ is the current state's view, $P_t$ is the current pose (location and heading), and $M_t$ is the map. Given the instructions $X$, the agent determines a waypoint $W = (d, \theta)$ defined by relative polar coordinates: $d$ is the distance from the rover, $\theta$ is the change in heading from current position. As it traverses the environment, the agent produces a sequence of state-action pairs $\langle (s_1, a_1), (s_2, a_2), \ldots, (s_n, a_n) \rangle$, where the final action is a stop command implying that the agent has reached its goal.

# 4.2 Reinforcement Learning

## 4.2.1 Reinforcement Learning Fundamentals

Reinforcement Learning (RL) is a type of machine learning where the agent learns to make decisions by interacting with an environment to maximize the cumulative reward. [32] This learning framework is characterized by several critical components which play a role in shaping the agent's behavior in the learning process. These components include:

- Environment: Where the agent operates
- State: Where the agent is in within environment
- Reward: Feedback from environment
- Policy: Maps the agent's states to actions
- Value: Future reward the agent would receive by taking action

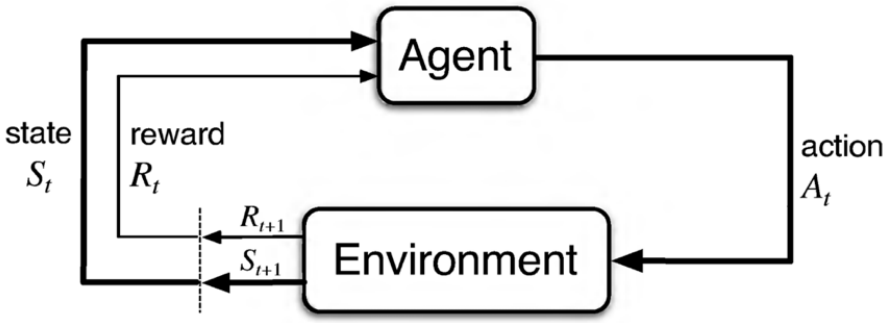



Figure 4.1 Reinforcement Learning can be represented with the above loop, showing the interaction between the agent and the environment [33]

The environment in an RL system represents the external system in which the agent interacts. The environment can be acted upon by the agent, and may be dynamic or static, and stochastic or deterministic. The environment allows the agent to interact with it through a series of states and rewards based on the agent's actions, allowing the agent to learn optimal strategies.

The state represents the current configuration or status of the agent in the environment, representing all relevant information necessary to make an informed decision, or all information available to a real, deployed instance of the agent. The state space is the set of all possible states which can be finite or continuous. When

programmed, the state is a tuple that represents all the variables to describe the environment and agent's status at a given time.

The reward function quantifies the benefit or cost associated with the agent's action in a particular state; it is a scalar value that provides feedback to the agent about the advantageousness of the action selected for a given state. The reward structure guides the agent's learning thereby influencing the optimization of the policy. In RL, the objective is to maximize the cumulative reward over time by maximizing the expected return.

The policy maps states to actions: for a given state, the agent must determine the next action. The goal of an agent is to learn the optimal policy to dictate its behavior. An agent may learn a policy as a function of a state, or it can have a policy that determines an action based on a probability distribution. There are several policy optimization techniques, including value-based methods, policy-gradient methods, actor-critic, and DD-PPO [34] which is used in this work.

The value estimates the expected cumulative reward that may be obtained from an action taken from a given state. The state value function $V(s)$ represents the expected return starting from state $s$ and acting based on its policy. In contrast, the action value function $Q(s, a)$ represents the expected return if we start from a state and take an arbitrary action, and then act according to the policy.

Illustrated in Figure 4.2, the relationship between the agent and the environment can be modeled with a Markov Decision Process (MDP): a mathematical framework for modelling decision making, which is a 4-tuple of $(S, A, P_a, R_a)$ where:

- S is a set of states called the state space
- A is a set of actions called the action space available from state S
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action $a$ in state $s$ at time $t$ will lead to state $s'$ at time $t + 1$
- $R_a(s, s')$ is the immediate (or expected) reward received after transitioning from state $s$ to state $s'$ after action $a$

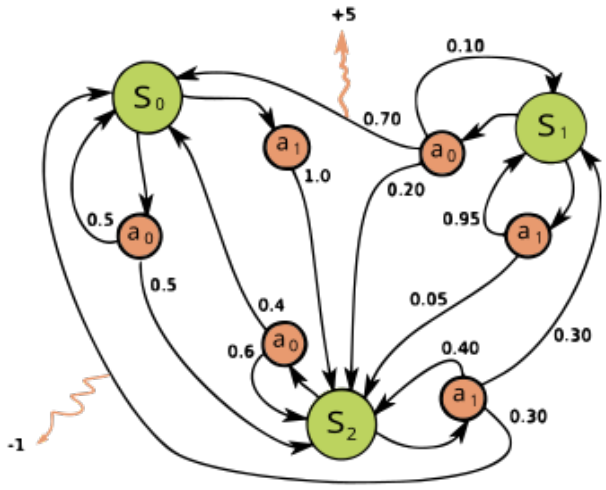



Figure 4.2 Example of an MDP with states (green circles), actions (orange circles), rewards (orange arrows), and transition probabilities (numbers listed)

### 4.2.2 Evaluation Criteria

There is now a need to evaluate how well the agent performed. Even though the objective is challenging: receiving a string of text, and incrementally observing and navigating until the goal is reached; the evaluation is relatively easy: how closely did the robot follow the instructed path? To properly deliver a reward for the RL agent, we look at evaluation criteria, of which there are two broad categories: goal-oriented metrics and path-fidelity metrics. VLN agents are not only be evaluated on whether they reach their goal, but also how well they follow the instructed path.

Goal-Oriented metrics mainly consider the agent's proximity to the goal. Examples of this measure include success rate and path length. However, goal-oriented metrics do not judge how well the described path was followed. Also, there is no incremental reward given during an episode, only at the end of each episode, making it harder to train. While Success rate weighted by Path Length is a common metric for embodied agent evaluation [35], it is a bad measure if the instruction does not describe the shortest path to the goal. Examples of goal-oriented metrics include:

- Path Length: length of agent's path
- Goal Progress: measures reduction in remaining distance to goal
- Navigation Error: distance between agent's final position and goal
- Success Rate: frequency agent's final position is within threshold of goal
- Oracle Success: measures whether the path is within a threshold from the target location.

- Success rate weighted by (normalized inverse) Path Length (SPL): scales the success rate with respect to the Path Length – Values both shortest path and success rate while penalizing circuitous routing.
- Success weighted by Edit Distance (SED): Compares expert vs agent actions/trajectory, balancing SR and PL.

In contrast, path fidelity metrics evaluate how closely an agent follows a reference path. This is important for tasks that require the agent to not only find the goal location, but also to follow a specific path. Normalized Dynamic Time Warping (nDTW) [36] better evaluates path fidelity; it judges the agent's ability to follow the specified path (which may not be the fastest path) rather than its ability to simply arrive at the end goal. NDTW is a "similarity function for time-series" which softly penalizes deviations from reference path to calculate match between two paths. Normalized Dynamic Time Warping also feeds back incremental rewards during an episode which speeds up training.

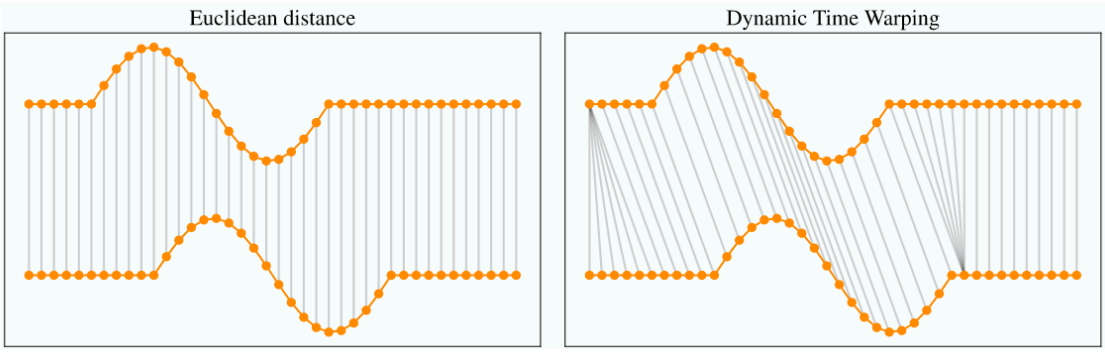



Figure 4.3 Comparison of two paths related by Euclidean distance (left) and via Dynamic Time Warping (right)

## 4.3 Simulation Setup

The VLN agent trained in Habitat Sim [37] [38]: an environment which simulated reconstructed buildings as shown in Figure 4.4. The training data included paths through buildings, and ~5 human-annotated descriptions of each path.

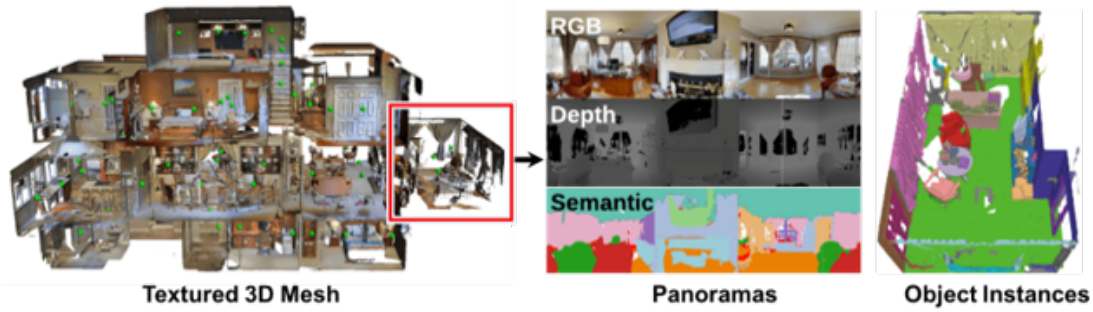



Figure 4.4 Simulation environment rendering a reconstructed building, with data streams showing RGB and depth images

When the VLN agent is in the observation step, it perceives a depth image, an RGB image, a map, and a heading. During the action step, it follows the policy it has learned to determine the next location to travel to (either the next subgoal, or it determines the goal has been reached and it stops).

## 4.4 VLN Agent Model & Training

The VLN agent is built on the waypoint models paper [2]; whose architecture is shown in Figure 4.5. Here, the RGB-D inputs are encoded with pretrained ResNet [39] models: the RGB images are encoded with ResNet-18 pretrained on ImageNet

[40], and the depth images are encoded with ResNet-50 pretrained on a VLN task: PointGoal Navigation [34].

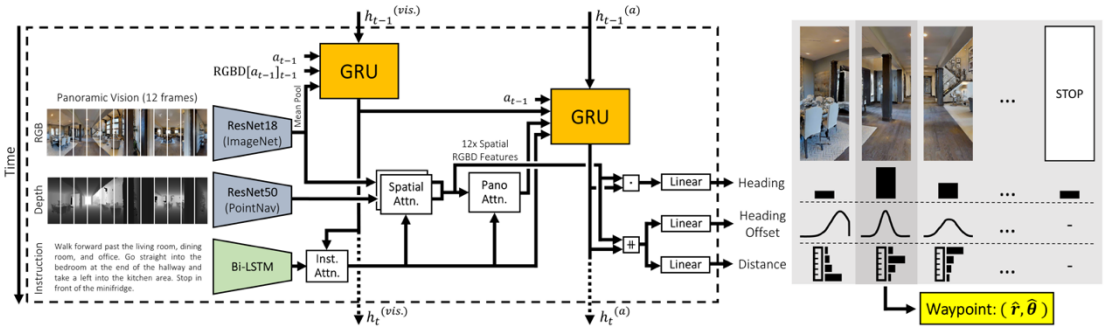



Figure 4.5 "Waypoint Models" [2] VLN Architecture

The VLN model operating with the autonomous driving stack leveraged the pretrained weights of "Waypoint Models" [2] model #2: the waypoint navigation model with discrete distance and continuous offset. This model gives the VLN agent more flexibility to choose waypoints up to 4 meters away, decreasing the estimated execution time, and allowing the autonomous driving agent more opportunities to navigate around obstacles, saving the VLN agent from potential collisions which may have caused other agents using a similar model to fail. The continuous offset was required because in a 90° FOV, no offset would have resulted in only three possible rough headings.

While this model was originally built for a panoramic RGB-D sensor, it was adapted to work with a 90° FOV camera, thus only the center frames four frames of the 12 total frames contained data, the remainder were zero-padded. The model was then fine-tuned using DDPPO (Decentralized Distributed Proximal Policy Optimization) [41]. Performance dropped to a 22% success rate, which may be

related to the fact that with just a 90°FOV, the agent cannot look directly left or right in any given frame. The Gated Recurrent Unit dedicated to tracking visual history may partially relieve, but not fully alleviate the agent's peripheral blindness.

## 4.5 Porting to Real Sensors

Outside of simulation, the VLN agent and autonomous driving stack ran on a NVIDIA Jetson Xavier NX [42] – an embedded computer. An iRobot Create 2 [43] was used for mobility: executing wheel commands and reporting wheel encoder values via serial. An Intel RealSense D455 [44] camera streamed both IMU (Inertial Measurement Unit) and RGB-D (Red Green Blue Depth) which informed both the autonomous driving stack, and fed directly to the VLN agent. While use of a Ricoh Theta V [45] was explored, the VLN agent performed better in simulation on 90°FOV RGB-D data than 360° RGB data that lacked the depth signal.

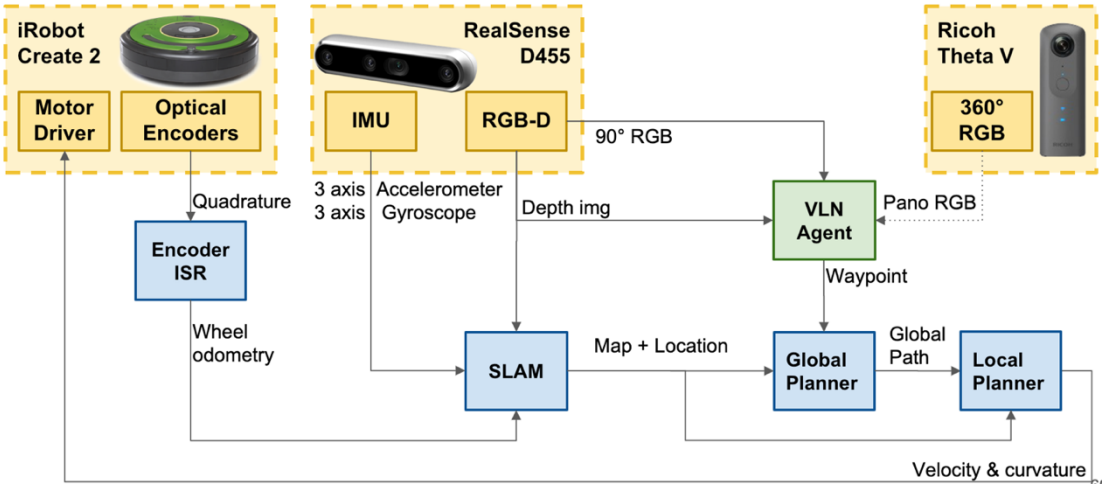



Figure 4.6 System diagram of driving architecture with VLN agent

Figure 4.6 represents how the Autonomous Driving Agent (developed in Section 5 Research: Autonomous Driving) ingests the waypoint from the VLN agent, and how they interact with the hardware. The autonomous driving agent performs mapping, localization, hazard detection, path planning, and path execution to make it to the waypoint in a collision free manner.

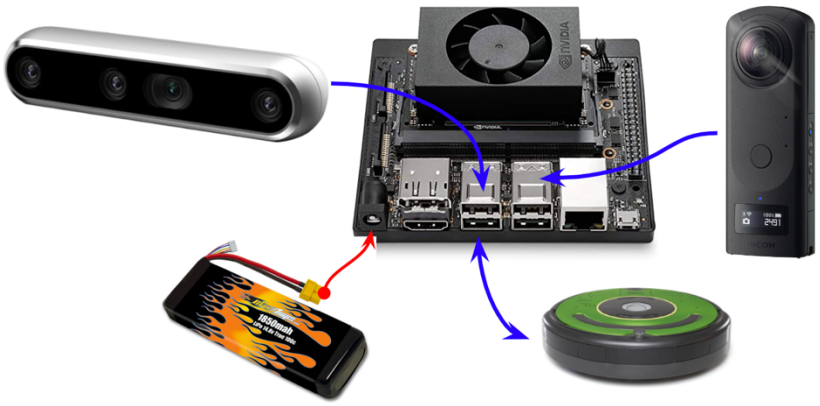


Figure 4.7 Hardware components of VLN agent

As shown in Figure 4.7, the Jetson connects via USB to the two camera sensors (Intel RealSense D455 and Ricoh Theta V [46]), and the robot chassis I/O for wheel commands and odometry feedback. The sensors and Jetson are powered by either a 3S 11.1V or 4S 14.8V LiPo battery. ("LiPo" is a common abbreviation for a Lithium Polymer battery. The "3S" and "4S" notation refers to the number of 3.7[V] cells in series in the battery.)

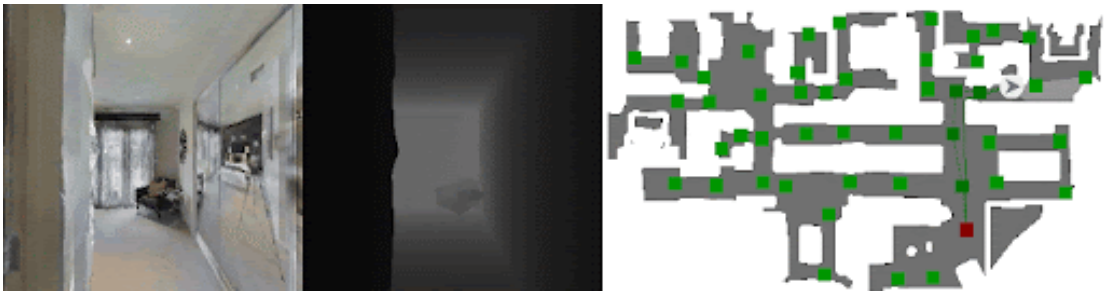
# 4.6 VLN Results

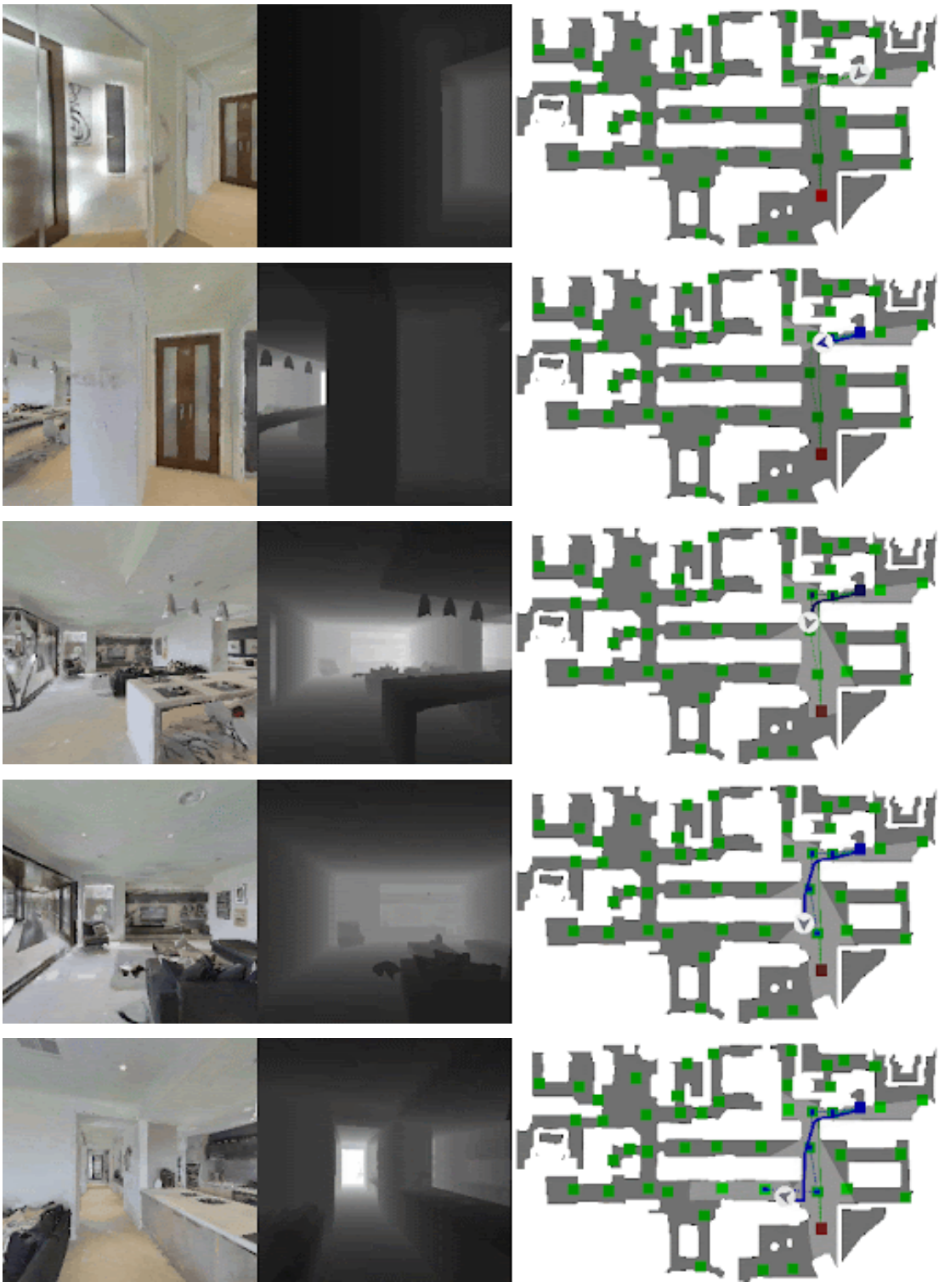
The images in Figure 4.8 show the trained VLN agent tested on unseen validation data: navigating an environment it has never seen before, following an instruction it has never received before. The following results are shown with a series of video frames. On the left half of each image is the RGB image and depth image in the 90° x 90° FOV that the agent observes at each timestep.

On the right half of each image is a top-down view of the map where: dark gray shows traversable area; light gray: area observed within sensor range; blue box: start location; red box: goal location described in instruction; blue line: traversed path; white arrow: agent's current location; green boxes: panorama points from the original navigation graph.

## 4.6.1 VLN Example Run in Simulation

In this example, shown by the series of images in Figure 4.8, the instruction given is: "Exit the bedroom and turn left. Walk straight passing the gray couch and stop near the rug."





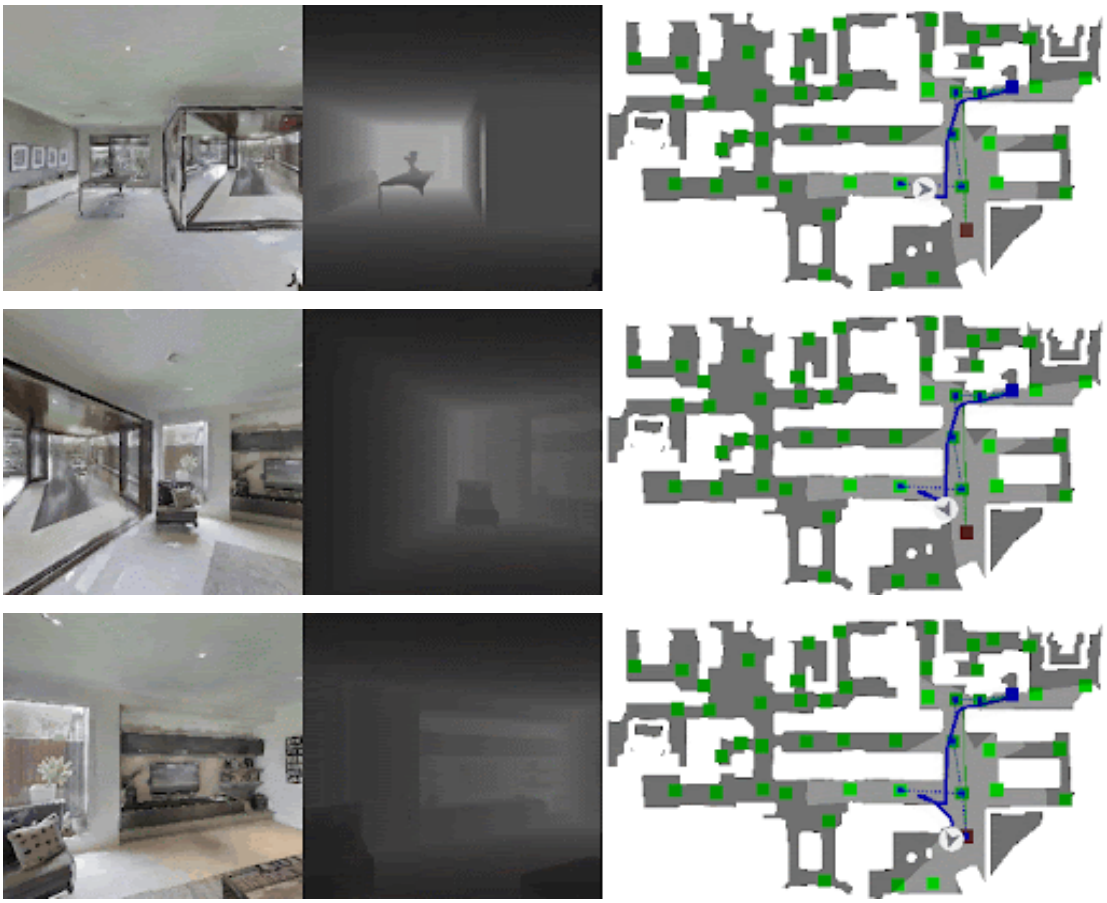


Figure 4.8 Resulting ground track of VLN example run

### 4.6.2 VLN Results in Real-World Environments

The performance of the VLN agent in the real-world environment was notably lower than in simulation, with a success rate of 22%. Several factors contributed to this reduced performance:

Discrepancy in Sensor Alignment: A disparity in sensor FOV and resolution between the depth and color cameras may have led to an imperfect mapping between depth pixels and color pixels. This discrepancy may affect the accuracy of the sensor data interpretation.

Field of View Limitations: The RGB-D camera used in the real-world setting had a significantly smaller horizontal FOV (approximately 90°) compared to the panoramic (360°) FOV utilized in the "Waypoint Models" [2] benchmark. This reduction in effective sensor coverage contributed to diminished performance.

Localization Accuracy: The agent's location was determined using SLAM, with the depth channel of the Intel RealSense D455 camera. Although the location estimates generally had an error margin of no more than 5cm, it was still less precise than the ground truth values available in simulation. This reduction in localization accuracy likely impacted overall performance.

Non-Deterministic Motion: Unlike the deterministic movement in the simulation, the real-world environment introduced inherent variability in motion. The autonomous drive agent drove the vehicle to within ±5[cm] and 10° of its target goal. Motion was also non-deterministic due to it traveling from hardwood flooring over

lips to carpeted flooring in an industrial environment. This contrasts with the "teleportation" abilities available in the simulated training environments.

Real-world obstacles and environmental challenges: the real-world environment presented additional challenges not encountered in simulation. For example, there was an increase in ground-based obstacles (eg: chairs, desks, boxes, backpacks, books), which did not show up as frequently in simulation. In simulation, the floors were typically free of objects laying on the ground, and it's likely that the VLN agent "lost focus" on the goal with all the other objects to look at on the ground. In contrast, glass walls presented challenges due to their transparency. The depth camera struggled to detect glass surfaces and boundaries because the infrared projection passed through the glass, rendering the obstacles nearly invisible to the sensor. This limitation reduced the VLN agent's ability to recognize glass walls as "walls." The autonomous drive agent, while not seeing a glass wall as an obstacle from far away, would identify it as an obstacle once it was within closer range (less than 1 meter).

These environmental challenges, both seen and unseen, demonstrate the complexity of real-world navigation compared to simulation. Despite this, the autonomous system demonstrated its capability to avoid collisions, ensuring safe navigation throughout the testing phase. This performance underscores the robustness of the autonomous driving stack in mitigating collision risks, even when the VLN agent faced performance limitations as deployed to a real environment.

## 4.7 VLN Conclusions & Commentary

### 4.7.1 Improvements

#### 4.7.1.1 Decouples Obstacle Avoidance

Previous work did not incorporate obstacle avoidance, and the robot relied entirely on the VLN agent to predict a clear path. The work outlined in this paper maps in real-time, creating a costmap, upon which it performs obstacle avoidance. Relieving the VLN agent of obstacle avoidance allows the agent to focus on the broader goal rather than the incremental steps to get there.

#### 4.7.1.2 Drops unrealistic assumptions

The VLN agent does not rely on unrealistic assumptions like known topology, perfect localization, teleportation, and use of impossible sensors. SLAM is used to map and localize, iterative waypoint navigation to drive, and sensor models that have realistic FOVs and noise models.

#### 4.7.1.3 Low Cost and Deployed

This research demonstrates the practical deployment of a VLN agent on a real robot platform with onboard computing, achieving a total hardware cost of under $900 at the time of purchasing, as outlined in Table 4.1. In contrast, previous studies [8] that deployed a VLN agent in real environments relied on high-cost hardware solutions exceeding $5000, including expensive laser scanners systems, prefabricated

robot chassis, and offboard computing. This substantial cost reduction highlights the advancements presented in this study.

Table 4.1 Breakdown of hardware costs

| **Hardware used** | | **Cost** |
|---|---|---|
| Computer: | NVIDIA Jetson Xavier NX | $399 |
| Sensor: | Intel RealSense D455 | $256 |
| Chassis: | iRobot Create 2 | $199 |
| Power supply: | LiPo battery | ~$35 |
| **Total** | | **$889** |

This reduction in deployment cost enhances accessibility and promotes wider adoption of VLN technology in real-world applications. By utilizing affordable yet effective components, this study establishes a viable model for deploying advanced VLN agents, paving the way for future advancements in autonomous navigation systems. The results highlight the potential for practical integration of VLN agents into robotic systems at a fraction of the previous investment.

### 4.7.2 VLN Lessons Learned

Certain failures of the VLN agent can be attributed to what might be termed as "ambiguity related errors," where the agent failed according to the ground truth data, yet these failures were not due to any error on the agent's part. For instance, the agent struggled with vague instructions that lacked sufficient contextual cues – an issue that might similarly challenge a reasonable human. Additionally, some instructions presented multiple valid but noticeably different paths, despite only one path being "correct" in the training data. For example, the instruction "continue down the hallway" leaves ambiguity regarding the initial direction to take if oriented arbitrarily. While these reasonable faults contributed to lower accuracy, they are not considered a shortcoming of the VLN agent itself since it still followed a reasonable interpretation of the instruction as a human might.

In contrast, some faults were deemed "agent performance errors," where the agent's performance deviated from expected behavior in ways a human could clearly identify; these failures stemmed from shortcomings on the agent's part. For example, the agent struggled with out-of-order instructions, such as "fetch the cup on the table next to the couch in the living room; it is down the hallway." It additionally encountered difficulties in processing longer sentences effectively; both issues are likely due to the limitations of the bi-directional LSTM architecture used at the time. Lastly, grammatical and syntactical errors in the path descriptions of the training data may have negatively impacted overall training quality.

### 4.7.3 VLN Future Work

This section explores promising directions for advancing VLN agents. Integrating transformer-based instruction embeddings, incorporating a multi-camera setup for panoramic RGB-D data, and leveraging 3D scene graphs with Logic Tensor Networks (LTNs) can enhance VLN agents' capabilities. Additionally, focusing on adaptation to outdoor environments and improving training data quality will further contribute to the robustness and effectiveness of VLN systems.

#### 4.7.3.1 Integrating LLMs for Instruction Embeddings

Future research directions may involve applying LLM embeddings as instruction encoding for VLN agents navigating in continuous environments.

Transformer-based LLMs such as BERT [47] [48] and GPT [49] are trained on extensive corpora beyond the VLN training data, facilitating an understanding of the English language and comprehension. This relieves the agent from having to learn the English language and specific distinctions – such as between "couch" and "sofa" during training. This head start on language and context understanding allows the agent to spend its training time learning how to navigate, rather than learning word relations.

Transformer-based LLMs can "pay attention" [50] [51]: they can effectively manage longer sentences and maintain context. This characteristic can lead to improved performance by mitigating challenges associated with processing lengthy or out-of-order instructions [52].

Additionally, exploring embedding models that are pretrained on both visual and textual data could further enhance the agent's contextual understanding and navigational capabilities.

#### 4.7.3.2 Multi-cam for panoramic RGB-D

While a panoramic RGB-D camera would be ideal deployed on a VLN agent, commercial options capable of real-time operation while in motion are currently unavailable; while some existing solutions rely on a scan to create a panoramic image, they must be stationary to scan the panoramic image. Although some panoramic color-only cameras such as the Ricoh Theta [53] exist, they lack depth measurement. An effective alternative may be to use an array of multiple 90°FOV RGB-D cameras to cover the full 360°FOV, as illustrated in Figure 4.9. This setup not only aligns more closely with the FOV used in several VLN agents trained in simulation, but also provides the VLN agent and autonomous driving agent more peripheral information, allowing it to look directly (90°) to its left or right, or even behind itself. This is helpful if it is told to "go into the office with the chandelier" as it may not see the chandelier unless it was looking directly to its left when driving past the door.


Figure 4.9 Multi camera panoramic array on Waymo self-driving car [54]

#### 4.7.3.3 Improve Training Data

A VLN agent's performance would improve with the improvement of the training data in multiple manners. Primarily, there are several language typos, syntax errors, and spelling errors which could be fixed. Secondly, clarifying instructions that may refer to multiple possible and valid paths either by having two valid target paths per instruction, or having each instruction only describing one possible path. Thirdly, add new training data that contains out-of-order instructions so that the agent does not rely on chronological instructions.

#### 4.7.3.4 3D Semantic Graphs

Future work should integrate two key concepts [55]: the development of a spatial ontology and the construction of 3D scene graphs using Logic Tensor Networks (LTN).

A spatial ontology (Figure 4.10) describes hierarchies of concepts in complex environments can provide a VLN agent richer contextual frameworks, enhancing its ability to interpret navigation instructions that reference spatial relationships and environmental features. This integration would enable VLN agents to navigate diverse settings, improving waypoint prediction and reasoning.

Incorporating logical rules (axioms) such as "a forest contains trees" into the 3D scene graph using LTNs allows the VLN agent to perform deductive reasoning based on inferred knowledge, reducing reliance on labeled data. The ability to predict unseen concepts during training suggests that VLN agents could adapt more flexibly

to novel environments or unexpected situations, which is crucial for real-world applications where agents may encounter unfamiliar settings. [55]

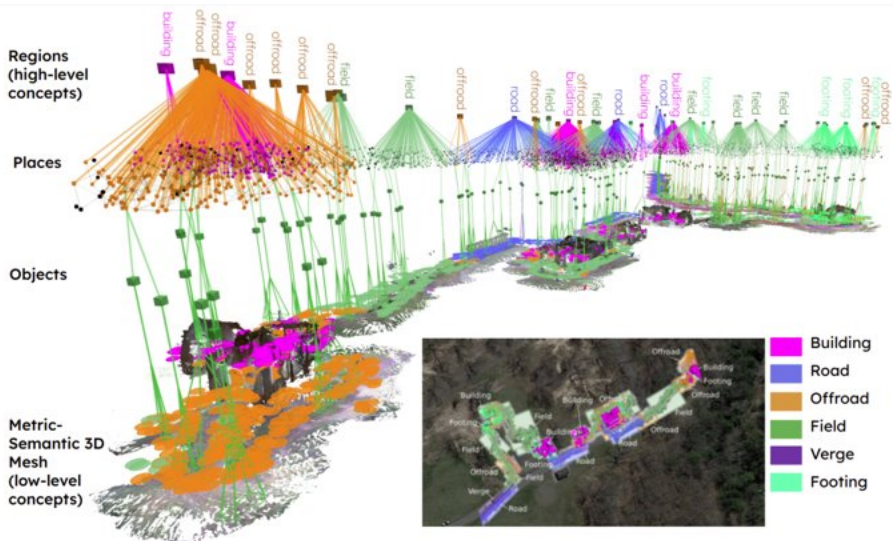



Figure 4.10 Outdoor scene graph generation [55]

### 4.7.3.5 Adaptation to Outdoor Environments

Adapting this research to perform in outdoor environments would be valuable to search and rescue missions. Scene graphs [55] trained in outdoor environments may aid in navigating large environments. However, while little research has been done in "natural" environments away from human civilization and construction, some works [56] [57] are exploring outdoor urban environments. VLN-Video [58] creates VLN style training data from videos, which provides groundwork for training VLN agents in outdoor environments.

# 5 Research: Autonomous Driving

This chapter discusses the design and build of the autonomous driving system. The autonomy stack uses a combination of SLAM, path planning, and obstacle avoidance to drive in unmapped environments to user-defined or VLN-specified waypoints. The subsequent sections outline system diagram of the autonomous driving software architecture, the detail the sub-modules in the autonomy stack, review prior architectures, and illustrate results.

## 5.1 Autonomous Driving System Architecture

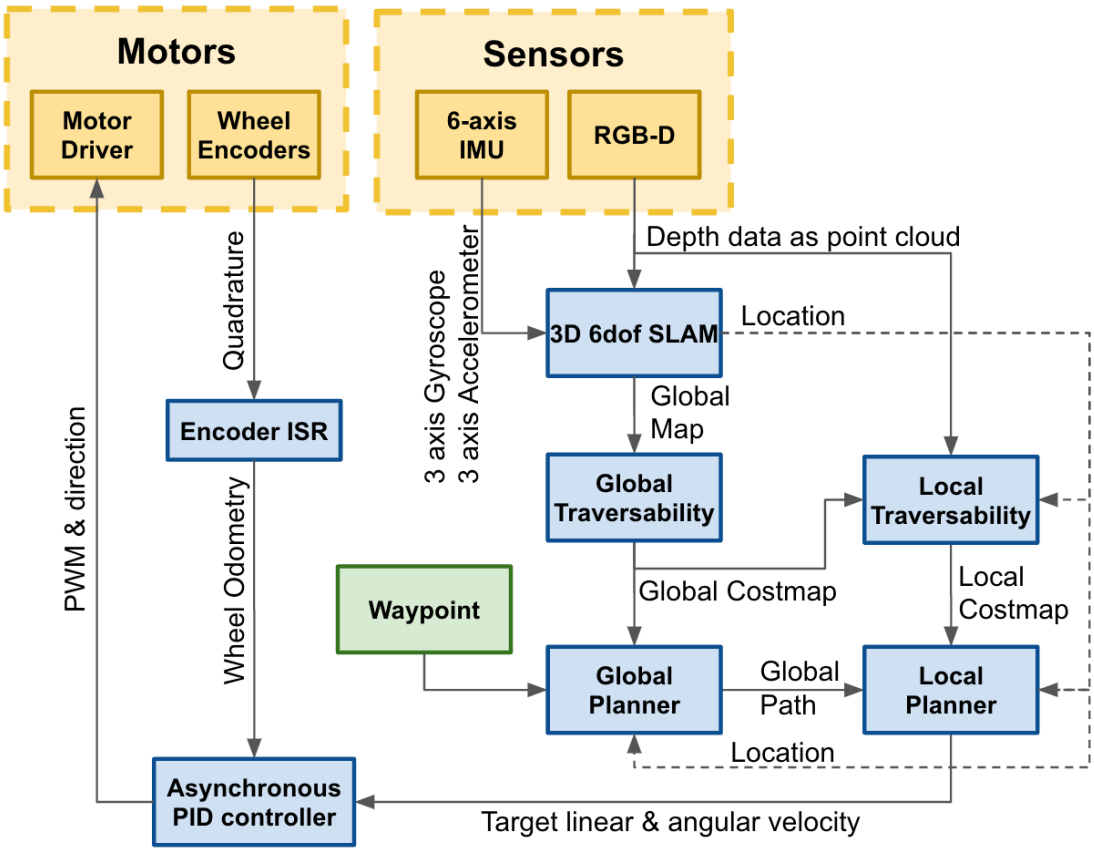



Figure 5.1 System diagram of autonomous driving software architecture

Figure 5.1 above shows the system architecture for the autonomous driving software stack. Subsystems are reviewed below, detailed in later sections, and are colored as follows:

- Yellow boxes: hardware (or simulated) interface to sensors and motor drivers.
- Blue boxes: nodes within the autonomy stack that perform independent tasks.
- Green boxes: external inputs, in this case, user-defined waypoints or VLN-agent determined waypoints.

**SLAM node:** The SLAM (Simultaneous Localization and Mapping) node processes data from the IMU, depth sensors, and optionally GPS/GNSS, to build a real-time map of the robot's environment while simultaneously tracking its position. This enables the system to create and update a local map as it moves through the environment, allowing for accurate localization even in unmapped areas.

**Global Traversability:** The Global Traversability node evaluates the entire global map to assess the traversability of each region. It outputs a global costmap, where each cell is assigned a cost representing the feasibility of traversing that area, factoring in obstacles, terrain difficulty, and other relevant considerations.

**Global Planner:** The Global Planner utilizes the global costmap to plan an optimal path from the robot's current location to the target waypoint. It determines the global path while accounting for static and dynamic obstacles within the global map, generating the most efficient path while avoiding non-traversable regions.

**Local Traversability Node:** The Local Traversability Node processes the global costmap, but focuses on a subset of it centered around the robot's current position. It dynamically updates the local costmap by incorporating new obstacle data from onboard sensors (eg: LiDAR, camera, etc.). This local costmap reflects the robot's immediate surroundings, enabling the system to react to obstacles in real-time and plan accordingly.

**Local Planner:** The Local Planner is responsible for navigating the robot along the global path within the local costmap. Based on the robot's current position and the updated local costmap, it computes the best path through the local environment. The planner generates target wheel velocities to move the robot along the path while avoiding obstacles and ensuring safe navigation.

**Wheel Velocity Control:** Wheel velocity control is achieved through feedback from wheel odometry, where quadrature encoders track the rotation of each wheel. The current wheel speeds are compared to the target velocities generated by the local planner, and a PID control loop is used to minimize the error and ensure precise wheel speed control, thus maintaining the robot's desired movement trajectory.

## 5.2 Autonomous Driving Sub-Modules

The autonomous driving system operates through a range of capabilities categorized into perception and planning roles, as shown in Table 5.1 with analogies to put them in context.

Table 5.1 Nodes in an autonomous driving software stack.

| | |
|---|---|
| Sensing | **Receive** raw data streams from sensors |
| Mapping | **What** is the world around me? |
| Localization<br>• Global Localization<br>• Local Localization | **Where** in the world am I now?<br>• **Roughly**, where am I on the map?<br>• **Precisely**, where am I with respect to obstacles near my path? |
| Traversability | **Where** can I go? |
| Path (Global) Planning | **Determines the high-level route** like Google Maps |
| Motion (Local) Planning | **Follows the route** like a taxi driver, detouring as necessary |
| Controllers | Use gas/brake pedals and steering wheel to **control** vehicle speed and heading. |

### 5.2.1 Simulation

Because complex robots may be expensive (eg: expensive to test, expensive to run, expensive to fix when broken, and time consuming), these robots are usually tested in simulation first to increase run frequency, decrease cost, and more easily "pause time" to debug and look at environment variables. Furthermore, simulations help quickly and repeatably set up and test cases, edge cases, and corner cases, in a

variety of different environments. Thus, building a good simulation representative of the robot environment is nearly as important as the autonomy software itself.

The simulation environments, rendered in Gazebo [59], consist of several indoor and outdoor worlds, with complex features and obstacles. Gravity was set equal to Earth gravity (9.8[m/$s^2$]). Collision calculations were enabled for the rover and all objects that the rover may run into near ground height. To reduce resource utilization for the simulation, the geometry of some objects' collision boundaries was simplified. For example, while a car's visual representation was rendered, collisions were calculated for a box of a similar footprint and height. Visual rendering was enabled for all objects, obstacles, and textures. The wheels, rendered with treads, were simplified to cylinders of a similar radius and depth for the collision detection and friction calculation. These simplifications still represented the real system and example environments with sufficient fidelity, as shown by negligible performance degradation from simulation to real.

### 5.2.2 Robot Model

The autonomous drive agent performs SLAM and path planning in simulation which subscribes to (reads data from) sensor inputs and publishes (writes data to) motor controllers as seen in Table 5.2 and Table 5.3 respectively.

Table 5.2 Robot model's subscriptions (from simulation and hardware)

| **Sensor name:** | **Sensor type:** |
|---|---|
| Intel RealSense D455<br><br>With integrated Bosch BMI055 | Depth sensor<br>RGB camera<br>6-axis IMU |
| Quadrature wheel encoders | Wheel odometry |

Table 5.3 Robot model's publications (to simulation and hardware)

| **Output type** | **Data format** |
|---|---|
| Body velocity command<br>(to simulation) | Target linear velocity<br>Target angular velocity |
| Wheel velocity command<br>(to hardware) | Target angular velocity for left wheel(s)<br>Target angular velocity for right wheel(s) |

To do this, a custom URDF (Unified Robot Description Format) robot model was developed, defining the robot's body (visuals, inertia, and collision boundaries), joints (type, and limits), sensors, and Gazebo simulation plugins. The Gazebo plugins define how the sensor data is simulated and how the wheels are controlled while the robot is running in Gazebo. When the robot is deployed in a real environment, neither Gazebo nor these plugins are utilized, since the hardware motor controllers are used instead. Figure 5.2 shows an iRobot Create robot with a red cylinder representing a lidar (an interim depth sensor), and silver bar representing the RealSense. The Gazebo

plugin for the lidar streams point cloud data, as would a real lidar or RealSense camera. The blue area shows a horizontal slice of the depth range and horizontal FOV.

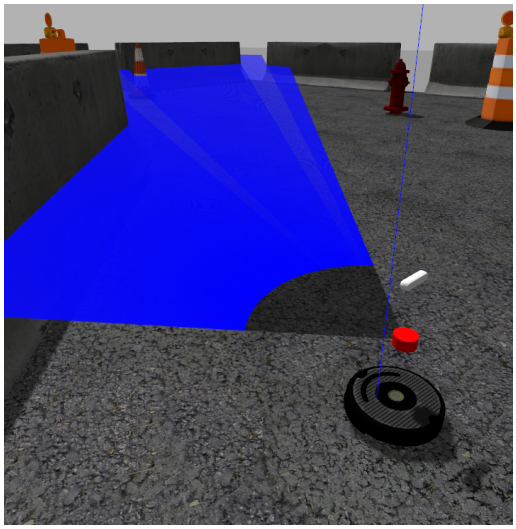


Figure 5.2 Gazebo Sim rendering gray robot chassis, and red cylinder representing depth range and horizontal FOV of RealSense D455

Simulated sensor models mimic real life sensors in terms of FOV, range, accuracy, resolution, and noise. The sensor model is the gray Intel RealSense D455 [60] camera which streams the RGB image and point cloud from both the camera hardware and the Gazebo simulation. The simulated sensor model uses a similar resolution, FOV, depth, and noise as the RealSense D455 hardware. To reduce computational resources required in simulation, the sensor hardware hovers over the robot chassis by a fixed transform; and the structural support and wiring is not modeled or rendered because it has no impact on SLAM or path planning, and has negligible impact on vehicle kinematics and dynamics.

PID velocity control models on the wheels of the simulated robot model mimicked how the chassis would accelerate in response to commands from the local planner. Simulating the physics dynamics response from the PID controller was important because both too slow of a PID and too fast of a local planner update will cause the robot to follow a sinusoidal "zig-zagging" trajectory path caused by the local planner overcompensating for the delay from the controller.

The physics-based simulation also identified scenarios where the robot may accelerate too quickly on steep or low friction surfaces, resulting in "burnouts" or loss of traction. The physics-based simulation identified where sharp turns or high acceleration would cause "rollover" or loss of stability. These findings were used to implement acceleration and velocity limits. Thus, ceiling limits were set for both the simulated PID controller and the hardware motor controllers for safety during hardware testing, and to avoid excessive current through the motors. These include: max linear velocity: $2[m/s]$, max acceleration: $2[m/s^2]$, max rotational velocity $2[rad/s]$.

### 5.2.3 Drive Control & Odometry Feedback

The controllers in the simulation execute PID velocity control. The drive control node receives linear velocity [m/s] and angular velocity [rad/s] commands, calculates individual left and right target wheel rotational velocities, and uses theose target wheel rotational velocities for each PID controller. The code for drive control used in simulation is the same as that used in the real robot; this allowed for lessons to be learned in simulation before testing in real-world environments. Some early fixes included PID tuning to stop overshoot caused by integral wind-up, zeroing out the PID's integral-component upon receiving a stop-command, and setting acceleration limits to prevent vehicle from flipping over if braking too quickly.

In the simulation the wheel velocity is published at 50Hz, and on the real robot the encoders for the left and right wheels are read as 16-bit values at 50 Hz. The same software interfaces with the wheel encoders and simulation alike, integrating and calculating the incremental angular and position offsets every loop, and calculates the following odometry:

- $x$ distance from starting location, along X-axis (forward facing)
- $y$ distance from starting location, along Y-axis (left of robot)
- $\theta$ angular offset from x-axis, around z-axis (up)
- $v$ instantaneous linear forward velocity
- $\omega$ instantaneous rotational velocity around z-axis

The PID velocity control was single-threaded, and performed wheel control asynchronously using ISRs (Interrupt Service Routines) to track quadrature channels. The motor controllers used an H-bridge to control each motor. While Lock Anti-Phase drive [61] was explored, it resulted in a loud humming when at low speeds or a stop. This humming was caused by the drive mechanism where the current supplied to the motors is near constant, and motor speed is controlled by the PWM (Pulse Width Modulation) signal determining what percent of the time to drive the motor in a particular direction, causing the motor to "quiver" (hum) in place at a stop due to the current alternating direction rather than turning off.

Sign-Magnitude drive [62] was chosen for its power efficiency, where the PWM pin controlled the voltage duty cycle (average voltage, and ultimately power), and the direction pin controlled the wheel direction (forward driving versus reverse). There is negligible computational difference between the two drive control methods. Lock Anti-Phase only requires the PWM signal, while Sign-Magnitude requires both PWM and direction signals. Sign-Magnitude is more power efficient, but less linear at low speeds.

A brake override function checks whether the target velocity is zero. If so, the function would zero out the "I-term" (integrated error) in each wheel's PID controller to remove any integral windup, turn off a flag that allowed velocity control, and apply the brake signal to the motor controllers. The PID I-term was zeroed so that if the rover was lagging behind the target velocity, it wouldn't continue driving to catch up

before slowing down, nor would it try “making up lost distance” when PID velocity control resumed.

The Sign-Magnitude braking command sets the PWM’s magnitude output to a 0% duty cycle (PWM’s direction channel does not matter because there is no current going to the motors), which on an H-Bridge effectively short circuits the two motor terminals to either the supply voltage or ground, resulting in dynamic braking. This short circuit between the motor terminals may have current recirculating through the motor and H-Bridge, but no current will flow through the H-Bridge from the supply voltage to ground. Mechanical friction in motor gearboxes further aids in deceleration.

While regenerative braking is possible with Sign Magnitude control, there are several considerations. Torque generated during regenerative braking decreases as velocity decreases, resulting in non-linear braking power and weak braking at low speeds. Furthermore, there is a tradeoff between the efficiency of energy regeneration and overall braking power. Lastly, the back-converted energy needs to be stored safely, avoiding the risk of over-charging a battery, and preventing attempts to “back charge” an AC to DC power supply if applicable. [62] Because of these concerns, regenerative braking was not implemented.

### 5.2.4 Perception

Autonomous robots and vehicles must perceive their environment and their state by turning sensor data into useful representations of the environment. This most significantly includes mapping, localization, traversability, sensor interface, and coordinate transforms.

Mapping is detailed in Section 5.2.5: Mapping.

Localization is detailed in Section 5.2.6: Localization.

Traversability is detailed in Section 5.2.8: Traversability.

Several sensors interfaced with the NVIDIA Jetson Nano, and later a Jetson Xavier NX. To use the full capabilities of the RealSense D455, UVC (USB Video Class) was chosen despite requiring more technical expertise to modify the kernel since native Linux kernel drivers are recommended, UVC results in higher performance, and allows for advanced hardware control. The alternative RSUSB (a user-space implementation of UVC), while cross-platform compatible and easier to implement, results in some performance and functional limitations such as limited hardware control and slower data speeds. To stream the RealSense D455 depth camera using UVC on the Jetson ARM64 architecture, modifications to the Linux kernel and modules were required, followed by a re-build and re-install. [63] The Intel RealSense SDK librealsense [64] was then built from source and successfully streamed the images and point clouds from the RealSense D455 sensor.

A series of transformations defined in the robot model relate locations of sensors on the robot to the base frame of the robot. Thus, a static extrinsic matrix is used to transform the point cloud from the sensor frame to the body frame. The transformation between the map frame and the body frame may be calculated in a variety of ways:

- The localization component of the SLAM module (see Section 5.2.6: Localization)
- Absolute localization updates (eg: GPS / GNSS) may help either correct the localization estimate, or update the SLAM pose graph. Non GNSS methods are explored in Section 5.2.6.1 Methods for Global Localization in GPS denied environments
- Dead reckoning from either IMU or wheel odometry (see Section 5.2.3 Drive Control & Odometry Feedback)

### 5.2.5 Mapping

Mapping, in different forms, is important for autonomous vehicles because it represents the world around the robot. Maps allow the robot to perform other tasks as outlined in Table 5.4.

Table 5.4 Different ways nodes use map data

| Used for: | Implemented by: |
|---|---|
| Global Localization in GPS-denied environments | ICP (Iterative Closest Point) comparing an in situ map to a priori maps |
| Hazard detection and traversability | Identify slopes, elevation differences, and breakover angle |
| Path Planning | Perform search over traversable area towards goal |
| Information gathering / data collection | Aside from driving purposes, the map data is valuable for remote inspection in remote areas, science missions, future missions, determining where to place roads and utility lines |

There are two important map representation types that the rover uses: a 3D point clouds show voxels (volumetric pixels) which represent the shape of the environment, while 2D maps use pixels to represent properties including occupancy (presence of obstacles), cost to traverse, or elevation.

The rover uses a 3D point cloud from the depth sensor for SLAM, which results in a map of the environment in the form of a 3D point cloud. That point cloud is analyzed to detect obstacles, creating a 2D occupancy map, and then a 2D costmap, as explained in Section 5.2.8.1. This abstraction to a 2D map increases path planning efficiency since the area searched reduces from a 3D space to a 2D manifold, which consequently lowers the computational complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. An example of this is a human driver looking at Google Maps while they drive: the driver can be agnostic to the elevation and slope of the paths, so long as the 2D map accurately predicts the shortest allowable path. However, the cost of traversing an area may be increased for many reasons, including mountainous terrain if energy efficiency is a concern, or predicted traffic if time is a concern.

Other features that may be represented on a map may include human-annotated keep-out-zones and landmark labels. Autonomous on-road vehicles may choose to represent sections of map that have toll-roads, allowing re-weighting costmap by features such as traversal time, fuel expenditure, and toll costs. On-road vehicles would have an advantage from mapping traffic signs, signals, speed limits, and crosswalks for future reference, as a redundancy to the computer vision (if, for example, a tree or snow obscures a stop sign one day). Vehicles operating in more austere environments may track areas with better solar coverage if they use photovoltaic panels to charge when en route; eg: routing them in an open area rather than through a forest. Similar vehicles may opt to drive slightly out of the way to pass

by a mobile data tower if needed to uplink or downlink data to a base command center.

Mapping is performed in lockstep with Localization in the SLAM module (as explained in Section 5.2.7).

### 5.2.6 Localization

Localization estimates the robot's global location in the world, and local location with proximity to obstacles. Each is calculated differently, used for different planners, requires different fidelities, is updated at different rates, and resides in different coordinate systems. Table 5.5 illustrates the differences between Global Localization and Local Localization.

Tracking global and local location at different accuracies enables the rover to be computationally more efficient and faster on constrained hardware, as further detailed in (Section 5.2.6.2: Differing Localization Accuracy Requirements).

Localization is performed in lockstep with Mapping in the SLAM module (Section 5.2.7). Location can be tracked in different degrees of freedom. 3-DOF (Degree of Freedom) estimation is sufficient for flat environments, but 6-DOF is required for any terrain that is not level:

- 3-DOF localization tracks: x, y, yaw (heading)
- 6-DOF localization tracks: x, y, z, roll, pitch, yaw (orientation represented by quaternions)

Table 5.5 Comparison of Local Localization versus Global Localization

| **Aspect** | **Global Localization** | **Local Localization** |
|---|---|---|
| Purpose | Where am I in a global reference frame? | How far did I just move in a local reference frame? |
| Description | Provides current LLA (latitude, longitude, altitude), and orientation | Provides movement details, such as rolling 0.1 [m] forward and rotating 0.02 radians in the last second, while currently moving at 0.12[m/s] |
| Sensors | Absolute reference sensors:<br>• GNSS [65] (eg: GPS [66], GLONASS, Galileo)<br>• Magnetometer *<br>• Star-Tracker *<br>*May be used for local localization | Relative referencing sensors:<br>• IMU<br>• Odometry<br>• Lidar<br>• RGB-D Camera |
| Uses | • Determining global position on a map<br>• Calculating path to destination | • Assessing obstacle proximity<br>• Informing motion planner about path progress |
| Calculation | Any absolute sensors can be used. Can compare an in situ map to an a priori map | SLAM (May be done in lockstep with mapping) |

#### 5.2.6.1 Methods for Global Localization in GPS denied environments

Global localization involves estimating a robot's position within a global reference frame using absolute measurements. Unlike relative measurements, which are relative to the robot's prior position, absolute measurements are independent of previous locations. Absolute measurements provide direct positioning information relative to a global frame, helping correct any drift in relative positioning (local localization), improving the robot's location estimate on the map.

GNSS (Global Navigation Satellite Systems) such as GPS (Global Positioning System, owned and operated by United States Space Force) is one of the easiest, most readily available absolute referencing sensors since it is free and available all over the surface of the earth. With the right equipment, one can also leverage other countries' GNSS systems such as the European Union's Galileo, and Russia's GLONASS. There are a few drawbacks, however: GNSS only works between the surface of the Earth and the satellites (ground vehicles and aircraft), and is unavailable for underwater, underground, or extraterrestrial missions. Furthermore, GNSS can easily be jammed [67]. Thus, it is valuable to be able to track absolute location in GNSS-denied environments.

Tracking a pose (that doesn't require knowledge of previous poses) requires knowing orientation and location. On Earth, where there is a magnetosphere, a 3-axis magnetometer may be used (commonly found in 9-axis IMUs). When the stars are observable (night sky, or locations with no atmosphere such as the Moon), Star Trackers may be used to determine ephemeris attitude (orientation).

A sensitive 6-axis IMU can perform 3-axis orientation determination (also known TRIAD [68]) calculating the robot's geographic latitude based on the rotation of the Earth (or other celestial body). The gyroscope triads measure the Earth's rate vector, while the accelerometer triads measure the local gravity acceleration vector. [69] [70]

In the daytime, a sun tracker or solar compass may additionally be used to determine latitude [71]. Combined with a chronometer or clock, longitude can be determined by comparing the time of local high noon to GMT noon.

When the sun isn't visible, the angle of the North Star above the horizon can determine latitude [72]. Star Trackers can determine latitude, longitude, and the North direction [73], resulting in accuracies are better than 200[m], but still insufficient for path planning.

Point cloud registration may also be used to localize. [74]

#### 5.2.6.2 Differing Localization Accuracy Requirements

Different types of localization run at different frequencies and accuracies. For example, the vehicle's global location certainty can be flexible – it does not need to know its global location to centimeter level accuracy with update rates nearing 20Hz. For global routes (even hundreds of kilometers long), the vehicle may only need to know its absolute location to within 10 meters, with update rates closer to once per minute, depending on the accuracy of the (relative) local localization estimate, and size of detour possible by the local planner.

An intuitive human example of this is if one were to be driving along a freeway to work. You may not know exactly how many meters (or kilometers) away your next exit is, which is fine so long as you know is coming up in the next few minutes. For humans, even without our phones dictating driving directions, a nearly 2000-meter uncertainty is perfectly acceptable for our commute if we know our next maneuver (freeway exit) is significantly further than the bounds of our location estimate's uncertainty, since when we recognize a landmark (eg: overpass or exit sign saying the exit is 0.5[km] away), we can update our location certainty sufficiently to take the correct exit in time. As applied to autonomous vehicles, this flexible localization certainty on the global level allows the vehicle to run faster without requiring more power and compute, while not sacrificing performance.

It should be noted that the local planner still requires local localization to be at the centimeter level in the local frame. This is critical in estimating motion for SLAM and estimating distance from obstacles so that the rover can pass through narrow passageways with 5cm of clearance. Similarly, a human driver may not need to know exactly how many meters it is to the next exit, but they know where they are between the lane-lines and surrounding cars to within less than a meter.

## 5.2.7 SLAM

SLAM (Simultaneous Localization and Mapping) is any algorithm that builds a map and localizes a vehicle in that map at the same time, allowing the vehicle to map new environments. SLAM works by processing sensor signal on the front-end, and optimizing a pose-graph on the back end as illustrated in Figure 5.3.

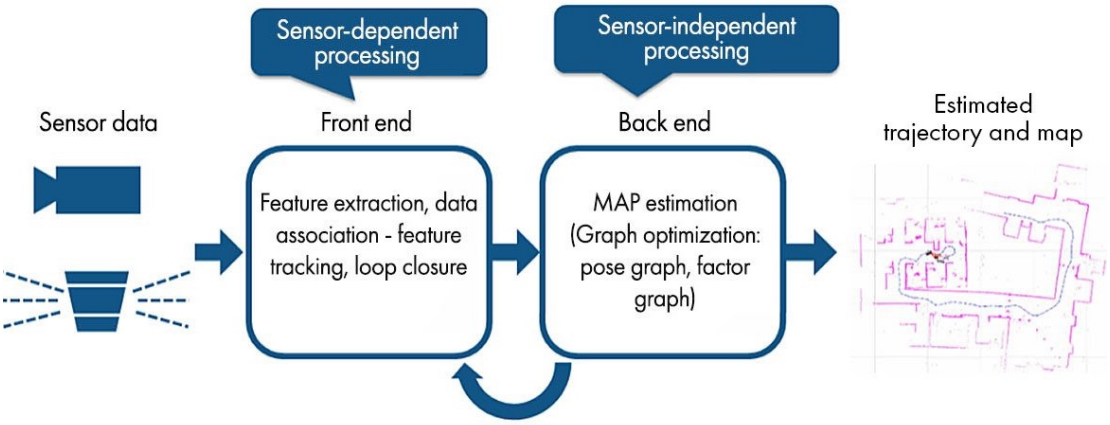



Figure 5.3 SLAM Processing Flow [75]

Visual SLAM uses imaging sensors (eg: RGB cameras, stereo vision cameras, wide angle cameras), which are frequently lower cost. Depth-based SLAM uses ranging sensors such as depth camera, lidar, and radar. Depth data provides better depth accuracy and is more efficient for map construction in SLAM algorithms. The autonomous driving software stack constructed in this paper explored different types of depth-based SLAM.

#### 5.2.7.1 SLAM Using Particle Filters

Initially, a particle filter SLAM approach was used: Grid-Based SLAM with Rao-Blackwellized Particle Filters [76] [77]. The SLAM module used depth readings for the mapping step, and odometry and depth readings for the localization step. The particle filter estimates potential trajectories $x_{1:t}$ of the robot by calculating its posterior $p(x_{1:t} \mid z_{1:t}, u_{0:t})$ based on depth observations $z_{1:t}$ and odometry measurements $u_{0:t}$ . The joint posterior about the map and trajectory is:

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t}) = p(m \mid x_{1:t}, z_{1:t}) \cdot p(x_{1:t} \mid z_{1:t}, u_{0:t})$$

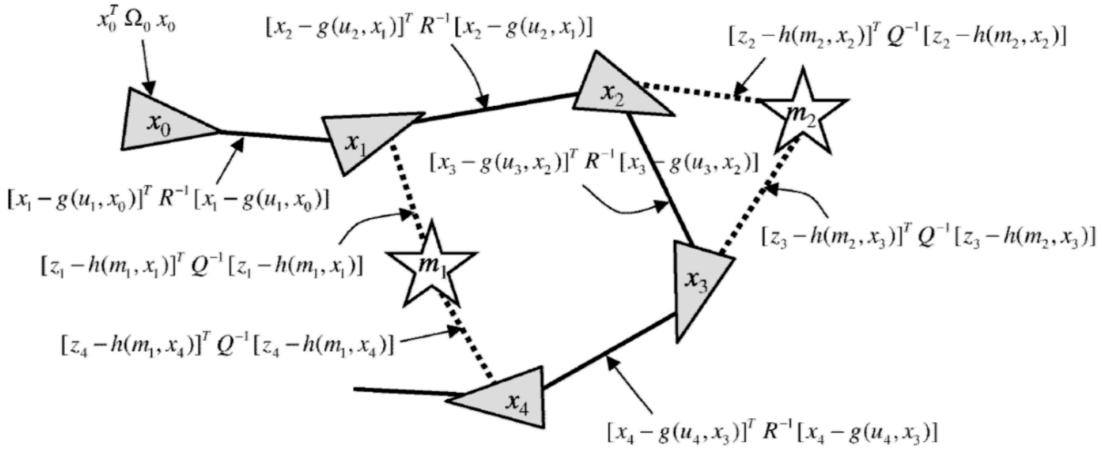
This estimates the trajectory of the robot, and then computes the map based on the trajectory. The posterior $p(x_{1:t} \mid z_{1:t}, u_{0:t})$ is calculated using SIR (Sampling Importance Resampling) [78], and updates the map by sampling the next generation of particles, assigning an importance weight to each particle based on how well they align with new observations, particles are resampled based on their weight, and for each pose sample the map estimate is computed.

In simulated environments, 80 particles were used to generate a topologically correct map when using a simulated 16-channel lidar, though closer to 100 particles were required when using the more noisy, narrower FOV Intel RealSense D455. As applied to real-world environments where the vehicle was tested on ~50[m] trajectories, 120 particles were needed to accurately localize and map using the Intel RealSense D455.

### 5.2.7.2 SLAM Using Factor Graphs

Factor graph SLAM approaches proved better adaptable in larger, more complex environments while using less memory allocation, and still running real-time. Sebastian Thrun's factor graph SLAM [79] was explored and is illustrated in Figure 5.4.

GT-SAM (Georgia Tech Smoothing And Mapping) [80] was later used. To enhance frame alignment and computational speed, the RANSAC (Random Sample Consensus) was employed for ground-plane detection.




Sum of all constraints:

$$J_{\text{GraphSLAM}} = x_0^T \Omega_0 x_0 + \sum_t [x_t - g(u_t, x_{t-1})]^T R^{-1} [x_t - g(u_t, x_{t-1})] + \sum_t [z_t - h(m_{c_t}, x_t)]^T Q^{-1} [z_t - h(m_{c_t}, x_t)]$$

Figure 5.4 GraphSLAM illustration [79] showing four poses and two features. Solid edges link consecutive poses, and dashed edges link each pose to feature(s) observed from that pose.

### 5.2.8 Traversability

When processing both the local and global maps, traversable areas and obstacles are represented on a costmap. Pixels with a lethal cost are placed in the costmap when an untraversable obstacle is detected. For instance, an object is classified as a lethal obstacle if it is within the robot's driving height range – specifically, when it is located more than half the radius of the wheel above the ground, and below the robot's maximum height.

#### 5.2.8.1 Costmaps

Lethal obstacles are projected onto a 2D costmap to both reduce the spatial complexity (memory allocation) of storing the obstacles, but also to reduce the time complexity (algorithm runtime) in the path planning step. A 2D traversability costmap allows path planners to quickly perform a search over a 2D map for the lowest cost path. This is far more efficient than executing searches over a 3D environment where one would have to use, for example RRT (Rapidly exploring Random Trees). If path planning were to be performed once at the beginning and only once ever, there may be an argument to perform path planning in the 3D space (searching through a 3D point cloud) instead of making an intermediary 2D costmap.

While the local costmap continuously updates to account for new obstacles or changes in the vehicle's environment, there are situations where an obstacle is too large to be avoided within the local costmap's range. In such cases, recalculating the global path on the global 2D costmap is more efficient than recalculating the path on

the global map represented by a 3D point cloud. The smaller, simplified 2D representation allows for faster re-planning, whereas path planning through the full 3D point cloud map is computationally more demanding and slower to process.

#### 5.2.8.2 Types of Obstacles and Costs

The costmap assigns different costs to different areas depending on whether it is used for local or global path planning, and whether an area has been observed or not.

Table 5.6 Different costs in different areas of map

| | |
|---|---|
| Known area (mapped) | Low cost: traversable area that is easy to traverse |
| | Medium/high cost: traversable area that is more difficult to traverse |
| | Max cost: lethal obstacles which will result in a collision if driven into |
| Unknown area (unmapped) | Global Costmap: assigns little to no cost to unmapped areas, allowing the global planner to explore beyond the mapped areas |
| | Local Costmap: assigns an extremely high cost (or lethal cost) to unmapped areas since the local crop of the map should not typically contain unknown (unmapped) areas, and an unknown area may be due to a lack of signal return caused by sensor obstruction, sensor fault, or the vehicle is near a cliff and there is nothing for the sensor to measure |

#### 5.2.8.3 Inflation Layers

Once lethal obstacles are identified, inflation layers are added around each lethal obstacle; these layers are also considered lethal. This first lethal inflation layer increases the width of all obstacles by roughly half the width of the robot. This gives the path planner the advantage of only needing to search for an "infinitely narrow" path, rather than one that would fit the width of the rover. In an algorithm such as A*, one may represent a lethal obstacle either as unavailable nodes, or nodes of infinite cost. In this work, locations that are lethal are represented with the max cell cost of 254 as shown in Figure 5.5.

The second inflation layer – the safety layer – is defined by a log-barrier decay function, the red line in Figure 5.5, which transitions from a lethal obstacle to no-cost. This safety layer encourages the path to detour slightly further from the obstacle given the higher cost of traversing this non-lethal but non-ideal area. However, since the safety inflation layer is non-lethal, the path planner may still choose to pass through areas marked by this safety inflation layer (eg: to pass between narrow obstacles) if doing so results in a lower cost (shorter/faster) path compared to a longer path through lower-cost areas.

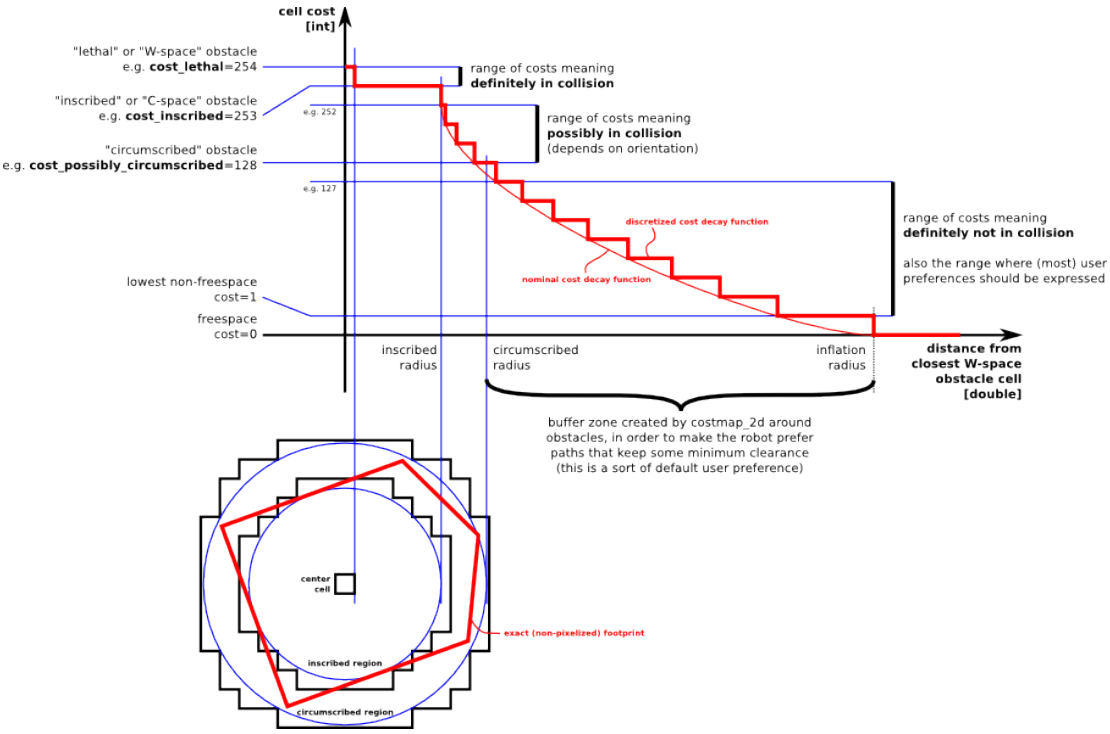



Figure 5.5 Graph showing lethal cost scaling to low cost as represented in inflation layers on a costmap [81]

In this work, instead of planning a path the width of the robot, the obstacles on the costmap are inflated to just over 50% of the robot's width. This simplifies path planning to search for the shortest path where path clearance does not need to be verified at every point, since the inflated obstacles (on the lethal inflation layer) account for the required vehicle clearance. The width of the lethal layer is further increased, if necessary, to just over the minimum depth sensor range; this ensures the obstacles stay within the sensors' detection range, even when passing through the safety layer. The safety layer is associated with a higher traversal cost, encouraging the rover to stay further away from obstacles.

#### 5.2.8.4 Live Local Updates versus Intermittent Global Updates

Two costmaps are used: one for global planning, and one for local planning. The global costmap inherits obstacles identified from the global map, which is being mapped in situ. The global path planner performs search on the global costmap.

The local costmap, in the robot coordinate frame, is a subsection of the global costmap, and inherits obstacles from the global costmap and the live sensor streams. This allows the local costmap to include real-time data of moving obstacles, which is important in scenarios where fast-moving obstacles are impeding the path faster than SLAM can update the global map.

#### 5.2.8.5 Obstacle Clearing

Stale obstacles are objects marked on a map which have since moved or been removed from the environment. Stale obstacles can be as simple as a mark indicating an object which is no longer present, or as complex as a moving object leaving a trail of obstacle markings behind it, as a train would. Stale obstacles are removed by raytracing through their previous location. For lidars and cameras, ray traces begin at the sensor's origin and pass through the marked obstacle. An obstacle is cleared when no obstruction is detected along this ray. To prevent false clearing due to missing depth data, a depth return ray from behind the old obstacle location is required to fully clear the stale obstacle marker.

### 5.2.9 Mapping & Search Combinations

There are several map and search combinations considered and discussed below; costmaps and A* searches (as shown in Figure 5.6) were ultimately implemented for their performance in dynamic environments and guaranteed shortest path.

Costmaps and heuristic informed search can search a large area in a computationally efficient and memory efficient manner. In the worst-case scenario, the entire map must be searched, but this is unlikely in non-constrained outdoor environments since the heuristic guides the expansion of the search in the direction of the goal. A*, an informed search algorithm, guarantees an optimal solution, and is used in the global planner as described in 5.2.10 Global Planning.

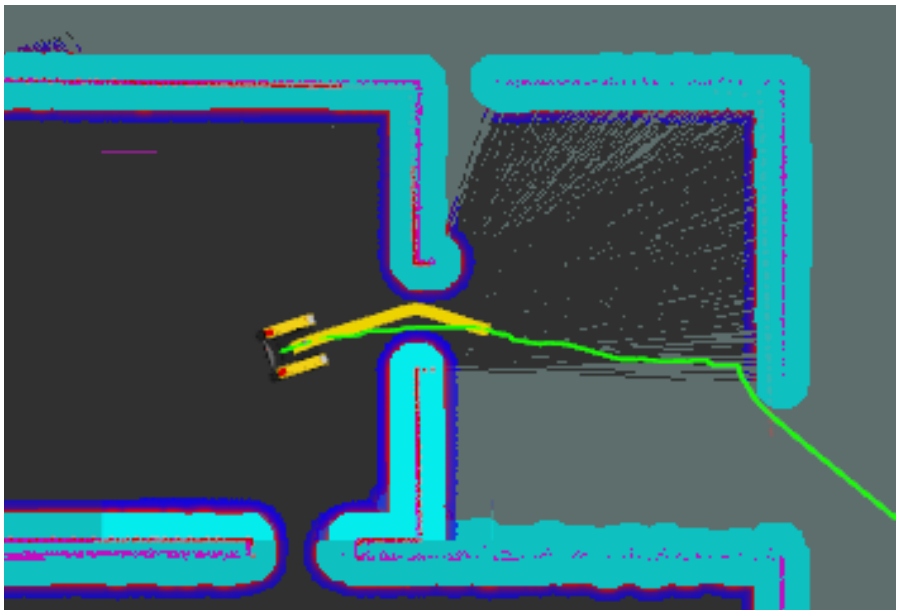


Figure 5.6 This costmap shows lethal obstacles (red) with inflation layers around them (cyan and blue). Black and blue cells are traversable, grey cells are unknown. The yellow local planner tracks the green global path while deflecting around the obstacles.

Potential Fields with gradient following is shown in Figure 5.7. A potential field consists of a low (attractive) potential at the goal, drawing the robot toward it, and a high (repulsive) potential around obstacles, pushing the robot away. A robot moves toward a lower energy state by following the negative gradient of the potential field toward the goal, and away from obstacles. However, there are some drawbacks: the robot may become trapped in local minima such as narrow corridors, and performance can degrade in dynamic environments.

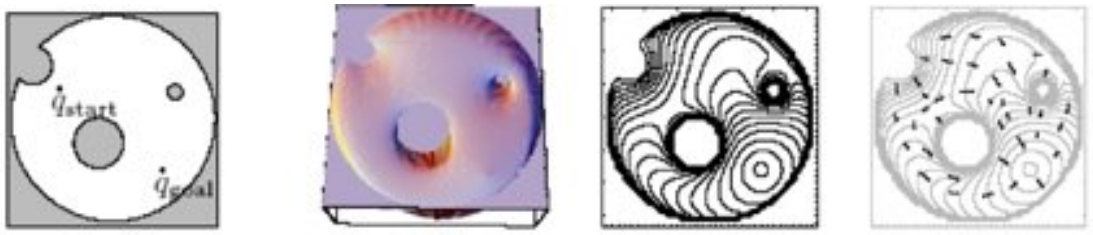



Figure 5.7 Potential fields and gradient descent [82]

Dynamic programming, shown in Figure 5.8, decomposes a search problem into smaller sub-problems, solving each and storing the intermediate solutions for future reference. While it guarantees an optimal solution, dynamic programming requires substantial memory to store these intermediate solutions, and increases computational cost by searching the entire state space.

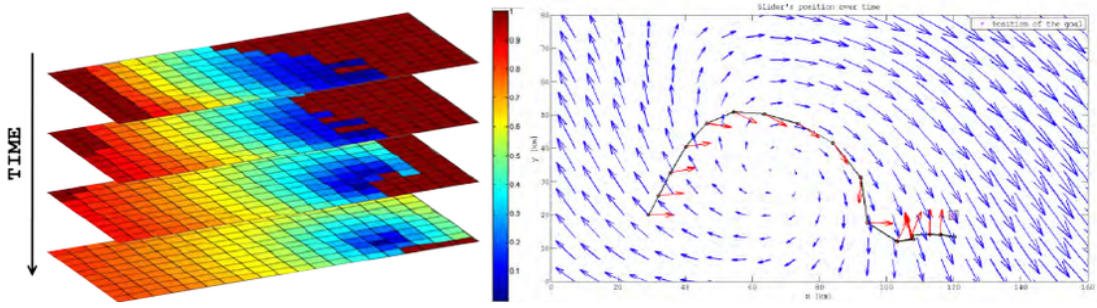



Figure 5.8 Dynamic Programming for path through static flow field [83]

Rapidly exploring Random Trees (RRTs), as shown in Figure 5.9, explore rapidly in high-dimensional space, but do not guarantee the optimal solution; sometimes paths between two points can be very circuitous. RRTs do not adapt well to dynamic environments.

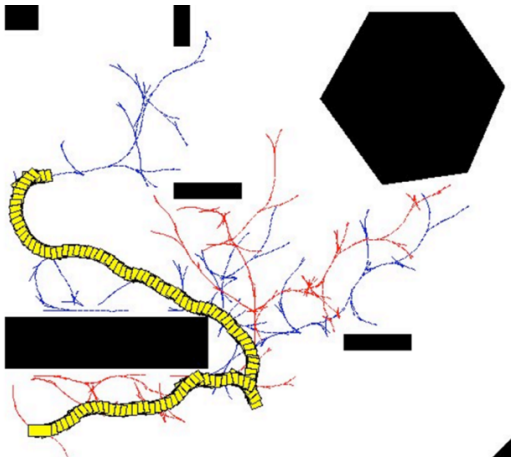


Figure 5.9 RRT for a 3-DOF car; obstacles (black); chosen path and vehicle pose (yellow); possible paths (red/blue) [84]

A Voronoi diagram turns a map into a simpler, memory efficient graph shown in Figure 5.10, but does not adapt well to new terrain, and does not always result in the shortest path.

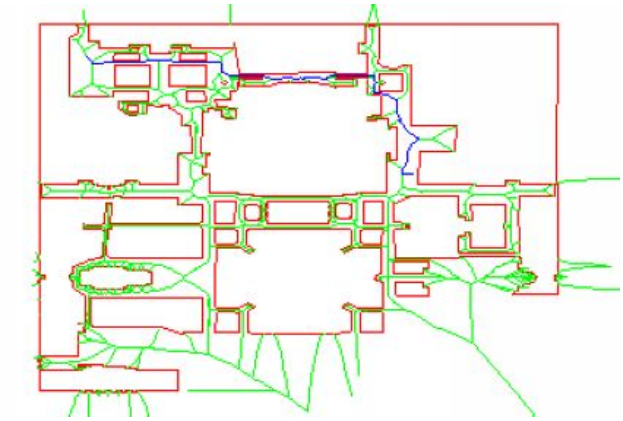


Figure 5.10 Voronoi diagram (green), obstacles (red), and sample path (blue) [85]

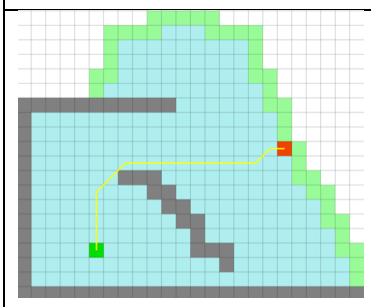
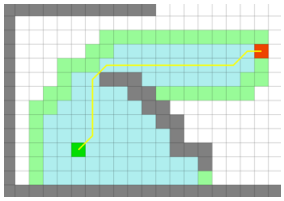
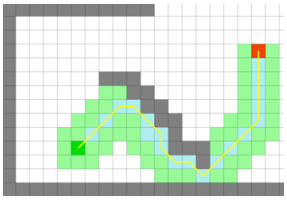
### 5.2.10 Global Planning

The global planner finds a path from the robot's current location to the waypoint goal. While the global costmap may not immediately reflect all obstacles when driving in new (unmapped) environments, it updates dynamically as the sensor horizon progresses. If the current path becomes unviable (eg: when the global path intersects an obstacle in global costmap), the global planner recalculates a new path based on the updated global costmap.

The global planner operates under the assumption that unmapped areas are traversable and free of obstacles. This optimistic approach enables the planner to navigate towards waypoints even in regions where the map is incomplete. As the vehicle progresses, SLAM incrementally fills in the map. If the global planner were to assume unknown areas were hazardous, it would never venture into unmapped regions altogether, limiting exploration.

Table 5.7 examines search algorithms: Dijkstra's, A*, and BFS (best first search). The illustrations in the table are color coded with: grey obstacles, cyan (closed / explored) nodes, green (open / edge) nodes, white (unexplored) nodes, dark green starting location, red ending location, and the yellow line representing the path calculated by each respective search algorithm.

Table 5.7 Table comparing search algorithms

| **Dijkstra's** | **A*** | **Best-First Search** |
|---|---|---|
|  |  |  |
| Guaranteed optimal path: 18.24 steps | Guaranteed optimal path: 18.24 steps | Sub-optimal path: 23.46 steps |
| Computationally slow: 573 operations | Computationally fast: 186 operations | Computationally fastest: 92 operations |
| $f(n) = g(n)$ | $f(n) = g(n) + h(n)$ | $f(n) = h(n)$ |
| Only considers **g(n)**: distance traversed. Not efficient. | Considers both **g(n)**: distance traversed; and heuristic **h(n)**: estimated remaining distance. Efficient, focused search. | Only considers heuristic **h(n)**: estimated remaining distance. Fast, but too greedy. |

A* was selected for global path planning due to its low computational complexity and ability to guarantee the shortest path. A* focuses on the most promising paths, leading to faster solutions, especially in larger or more complex environments. When combined with costmaps, A* provides a balanced approach that optimizes both performance and efficiency, particularly in dynamic environments. While alternatives like Potential Fields, Dynamic Programming, Voronoi Diagrams, and RRTs have specific applications, they often suffer from high computational costs, lack adaptability, or fail to ensure optimal paths. In contrast, the combination of costmaps and A* provides a robust, efficient approach to dynamic path planning.

### 5.2.11 Local Planning

The local planner attempts to follow the crop of the global path that is within the local costmap. The local costmap inherits the previously mapped obstacles from the global costmap, and superimposes new, higher-resolution obstacles from live sensor readings. The local planner uses the real-time local costmap, which shows new and dynamic obstacles, and plans a detour path around said obstacles, roughly following the global path, deviating where necessary.

The local planner adopts a cautious stance, assuming that unmapped areas are potentially hazardous and not traversable due to the risk of encountering obstacles: either confirmed or undetermined. The local planner's cautious strategy (assuming unmapped areas are hazardous) requires real-time confirmation of traversability within its sensor field of view. As it follows the global path, the local planner ensures that no hazards are present and that the ground plane is present before proceeding.

The simplest way to implement a local planner is by sampling curved trajectories; either Trajectory Rollout (TR) or Dynamic Window Approach (DWA). While DWA and TR run very fast, they cannot detour around dynamic or new obstacles, as would be present in un-explored environments, resulting in the global path updating very frequently, which is inefficient. Thus more complex motion planners, specifically elastic band planners, were required, as discussed later in this section.

The diagram in Figure 5.11 below shows obstacles in red, sampled trajectories with dotted lines, robot pose as blue squares, and a potential collision of the robot with an obstacle on the forward simulated trajectory with a black splotch.

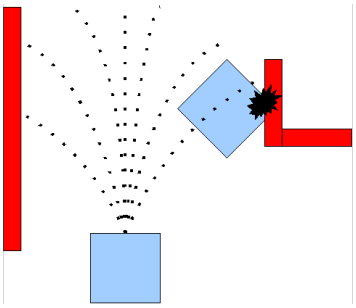


Figure 5.11 Sampled trajectories used by Dynamic Window Approach and Trajectory Rollout

Table 5.8 Comparison of DWA to TR

| | **Dynamic Window Approach (DWA)** | **Trajectory Rollout (TR)** |
|---|---|---|
| Step 1: | Sample Velocities in one timestep | Sample form all achievable velocities |
| Step 2:<br>Step 3:<br>Step 4: | Forward Simulate<br>Reject illegal trajectories that cause collisions with obstacles<br>Rank and pick best path | |
| Pros: | Lowest compute | Low compute, more optimal than DWA |
| Cons: | Cannot detour around dynamic obstacles<br>May be outperformed by TR | Cannot detour around dynamic obstacles<br>Slightly slower than DWA |

Elastically deforming path planners [86] can deform their path within the local costmap (or sensor FOV), so a new obstacle will not cause the global planner to re-search. The Elastic Band attempts to follow a local snippet of the global path, deforming based on obstacles around it. The initial path is the original crop of the global path within the local costmap (eg: the next 10 meters of path). It is then subjected to internal forces which "contract" the path, and external forces repel the path from obstacles. These deformations result in shorter, smoother paths around mapped and new obstacles.

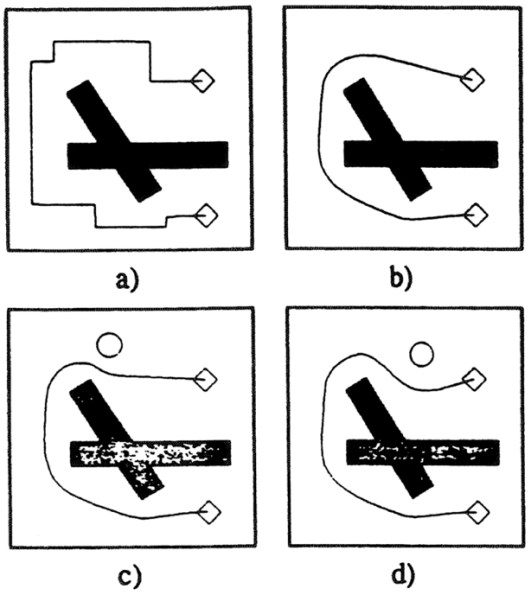



Figure 5.12 a) snippet of global path; b) contraction and repulsion forces applied; c) and d): forces applied in presence of another obstacle [86]

The contraction and repulsion forces mimic those in a rubber band. The contraction force represents the tension in a rubber band stretched around an obstacle, removing any slack in the path. The repulsive force comes from the obstacles deforming the elastic path to guarantee clearance from an obstacle.

If the local planner finds itself in a dead-end or against an obstacle too big to detour around, that obstacle will be added to the global costmap, and the global planner searches for a new global path. While slower than DWA or TR, the Elastic Band planner runs real-time.

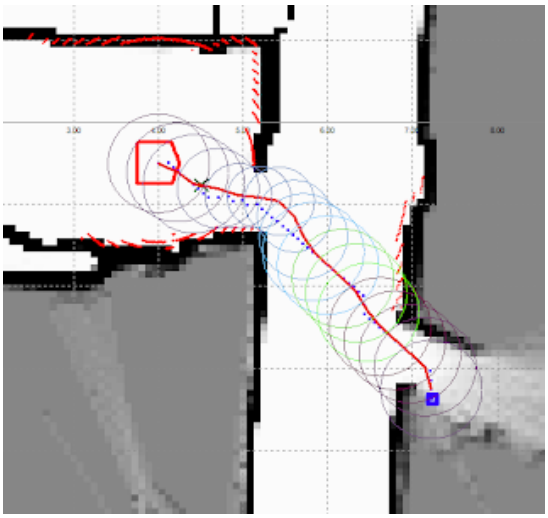


Figure 5.13 Illustration of obstacles deforming Elastic Band local path (red) as it follows global path (blue) [87] [88]

A Timed Elastic Band [89] local planner was considered: it is the time-optimal version of Elastic Band, convenient for modulating velocity through turns as one would around a racetrack. It provides the fastest path rather than the shortest path. However, with the computational load of the other autonomy nodes running simultaneously, Timed Elastic Band did not run real-time on the NVIDIA Jetson Xavier NX which it was tested on.

Since time-trial laps on a raceway are not within scope of this project, and seeking the shortest path was more important than the fastest path, the Elastic Band is used as the Local Planner for its runtime efficiency and safe detouring capability.

## 5.3 Prior Architectures: Incremental Improvements

Each prior architecture faced specific challenges that led to inefficiencies or inaccuracies in autonomous navigation. By addressing these issues through the introduction of dynamic costmaps, improved obstacle handling, and better local planning strategies, the current system architecture offers a more responsive and effective approach to autonomous driving.

### 5.3.1 Introduction of a Local Planner

To solve a slow path planning response, a local planner was introduced.

The initial architecture only included a global costmap and planner, and not a local costmap and planner. Any deviation from the target path due to stochastic movement or new obstacles necessitated a re-assessment of the entire global costmap, requiring a full re-plan of the global path. This resulted in slow path finding in dynamic environments.

A local costmap and planner were added to handle local deviations. This approach allowed the system to recalculate collisions and paths exclusively in the local costmap, only updating the global costmap and re-planning the global path when no feasible detour was available. This reduced the frequency of global re-planning, and improved responsiveness to dynamic obstacles.

### 5.3.2 Complementary Assumptions for Exploration in Unmapped Environments

To enable autonomous navigation into previously unmapped areas, the global and local planners were designed to make opposite assumptions about unknown terrain, balancing exploration with safety.

Initially, both the local and global planners were configured to be too cautious, assigning lethal costs to unmapped areas in the costmap. This assumption that all unmapped regions were unsafe prevented the global planner from venturing into areas where the map was incomplete. This cautious stance restricted the rover to only navigating within mapped regions, effectively preventing exploration. Later, when the cost to traverse unmapped areas was reduced (making both planners more optimistic), the rover faced an increased risk of the local planner leading the vehicle into an obstacle, as the local planner had not yet verified the safety of the unmapped area.

To overcome this limitation, the planners were modified to make complementary assumptions. The global planner was made optimistic by reducing the cost assigned to unmapped areas, assuming that unmapped areas were traversable and free of obstacles. This optimism enabled the system to navigate toward waypoints in regions that had not yet been mapped, thus facilitating exploration. In contrast, the local planner was designed to be cautious by assigning lethal costs to unmapped areas, assuming that unmapped areas were potentially dangerous and requiring real-time validation of their traversability. The local planner would only proceed once the area

had been confirmed safe upon being mapped by the onboard sensors, ensuring that the rover did not drive into obstacles or hazardous terrain.

By making complementary assumptions: optimism in the global planner and caution in the local planner, the system was able to strike a balance between exploration and safety. The global planner encouraged the rover to venture into new terrain, while the local planner ensured that each new step (segment of the route) was validated, allowing the rover to explore unknown environments, thus greatly reducing the risk of encountering unpredicted obstacles.

### 5.3.3 Dynamic Global Costmap

To prevent the robot from getting stuck in a dead end, the static global costmap became dynamic, allowing new obstacles to be added.

Using a static global map caused issues when the robot encountered dead ends not represented in the pre-mapped global map. This caused a “cul-de-sac” problem where the robot would find itself in a dead end that was not shown on the global map, try exiting the dead end based on the live sensor stream in the local map, and then re-attempt the dead end. The global planner would then re-route straight through the dead end, because the new obstacles causing the dead end were not being copied onto the global costmap.

The static global costmap was made dynamic to incorporate new obstacles detected. This resulted in better adaptation to new obstacles, and improved path planning out of dead ends that were previously unmapped.

### 5.3.4 Ray Tracing for Obstacle Removal in Dynamic Environments

Blockades caused by stale obstacles were cleared with raytracing for obstacle removal.

The newly introduced dynamic global costmap incorporated new obstacles, allowing the robot to get out of dead ends. However, it did not clear stale obstacles that were no longer present. When new obstacles were moving, the costmap added lethal costs where the obstacle was observed (but never removing them), effectively creating a barricade trailing behind the obstacle. For example, if a person were walking past the robot, the global map would mark each location that the person was in as lethal, leaving a “barricade” across the costmap.

Ray tracing for obstacle removal was implemented to clear obstacles from the costmap. By checking if points on the costmap could be ray-traced through, obstacles no longer present were removed, preventing the creation of barricades and improving map fidelity. For the global costmap to be dynamic, it needs to not only add obstacles, but also clear them as well.

### 5.3.5 Conditional Obstacle Removal

False obstacle clearance caused by raytracing out previous obstacles necessitated conditional obstacle removal, where clearing of obstacles requires both a ray-trace through the previous obstacle and an intersect of the ray with a new object beyond the location of the original obstacle.

Obstacles near the robot or obscured by debris on the sensor sometimes did not produce depth returns, nor did cliffs or drop-offs. While the lack of a depth measurement initially cleared obstacles, this also led to incorrect assumptions that obstacles moved or were not present, causing over-optimism in path traversability. Thus, no depth reading could mean any of the following:

- Path is clear: There is no depth return since there is no obstacle.
- Unseen obstacle: There is no depth return for a “negative obstacle”, eg: a cliff or drop-off, which is not traversable.
- Unseen obstacle: There is no depth return since the lethal obstacle is too close to the sensor for it to reconstruct the point cloud of the obstacle.
- Unseen obstacle: There is no depth return since there is debris on the sensor preventing point cloud reconstruction of the obstacle.

In an updated approach, obstacles were only removed from the costmap if there was a depth return ray traced through a marked obstacle’s location. This accounted for the sensors’ minimum range limitations, sensor debris, and cliffs (negative obstacles), improving obstacle clearing accuracy. While this conservative approach

occasionally caused lingering artifacts of overhead obstacles (since overhead obstacles only have open sky behind them), this may be improved with other sensors such as sonar, radar, or a multi-camera array performing obstacle learning.

### 5.3.6 Relaxed Path Adherence and Expanded Detour Area

To increase the efficiency of the local planner when making a detour, the allowable detour area was expanded, and the adherence to following the global path was reduced.

When detouring around obstacles, the local planner prioritized immediately returning to the next traversable section of the global path (once past the obstacles), frequently resulting in inefficient backtracking and longer paths. This caused the local planner to curve around the back end of obstacles and backtrack down to an earlier point in the global path, rather than taking a "short cut" to rendezvous with the global path at a later point.

To decrease total path length, the allowable detour area was increased from 1[m]x1[m] to 10[m]x10[m], increasing the local planner's capability to detour around larger obstacles without necessitating a replan of the global path. Additionally, the local planner prioritized progress along the global path (re-joining the path at a later point) rather than fidelity to the global path (returning to earliest segment of global path as quick as possible). This allowed the rover to "cut corners" and "take shortcuts" where safe to do so, resulting in more efficient paths.

### 5.3.7 Relaxed Global Localization Tolerance

To reduce exceedingly high computational demands, the tolerance on the global location estimate was relaxed.

Both the global and local planners initially required 1[cm] precision at 20Hz, which was demanding on processing power and sometimes led to failures in feature-sparse environments where the location uncertainty was out of tolerance, or computational lag if the map was too large to process at that update rate.

Relaxing the global location tolerance from 0.01[m] at 20Hz to 0.50[m] at 1Hz reduced the computational burden without negatively impacting the global planner or any other autonomous driving capabilities. While this resulted in slightly delayed global location updates, this did not impede the global planner since:

- Obstacles were observed from tens of meters away, so all obstacles in the robot's proximity were already mapped into the global costmap, for which the global planner had already determined a path.
- The local planner maintained high precision 20Hz, allowing for the detection and avoidance of new dynamic obstacles.

This balance between location accuracy and update frequency versus computational load improved system performance without compromising path execution. Further detail in 5.2.6.2: Differing Localization Accuracy Requirements.

## 5.4 Initial Development and Testing

For the VLN agent to execute waypoint navigation, the autonomous driving stack must map new environments, track its location, plan an obstacle-free path the goal, execute the path, and detour around new obstacles

To do this, the autonomous driving stack was first tested in simulation, in simplified environments, followed by HIL (Hardware in the Loop) testing. This included testing SLAM separately from path planning and dynamic obstacle avoidance. During these iterations in simulation, several edge and corner cases caused the robot to fail in reaching its goal, as detailed in Section: 5.3: Prior Architectures: Incremental Improvements.

### 5.4.1 SLAM in Simulation

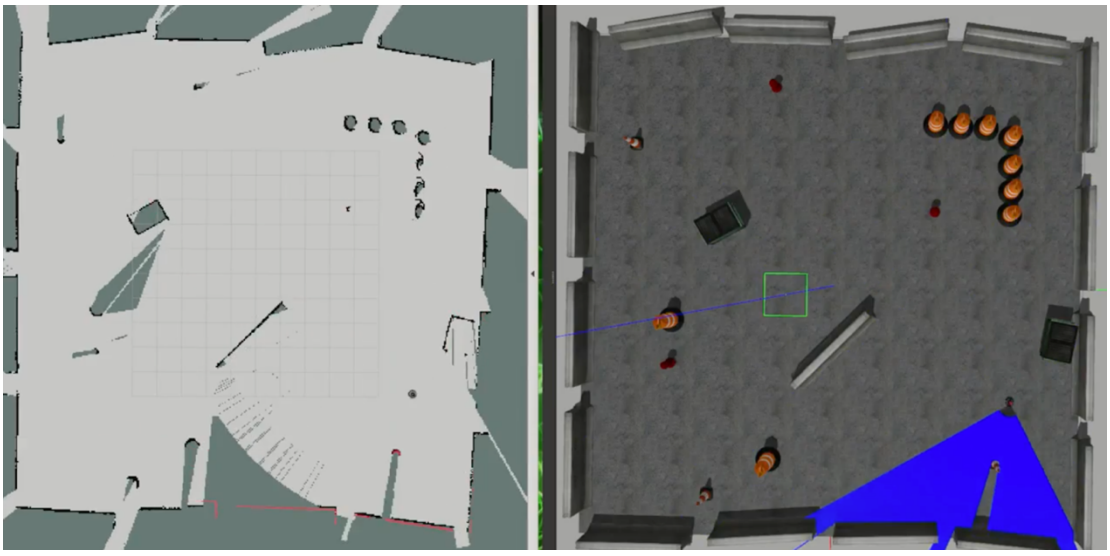


Figure 5.14 2D SLAM in simulation

In Figure 5.14, the robot is performing SLAM using the 90° sweep of the lidar sensor. On the right is the “ground truth” Gazebo simulation running, showing the environment, robot, and visualized lidar FOV and range in blue. On the left is a visual representation of the robot’s resulting map and location estimate, with the following color coding: {red line: collapsed point cloud from lidar sensor; black line: mapped obstacles; light grey area: mapped clear area; dark grey area: unmapped area}.

### 5.4.2 SLAM with Hardware in the Loop

To verify that SLAM works on the RealSense D455 camera hardware, the same SLAM node was tested using the RealSense D455 point cloud (instead of the simulated point cloud). The RealSense D455 was rotated by hand at 0.2 [rad/s] while a spoofed odometry signal reported the same odometry. The SLAM model properly mapped new areas with inputs from the D455 camera. This is shown in Figure 5.15.

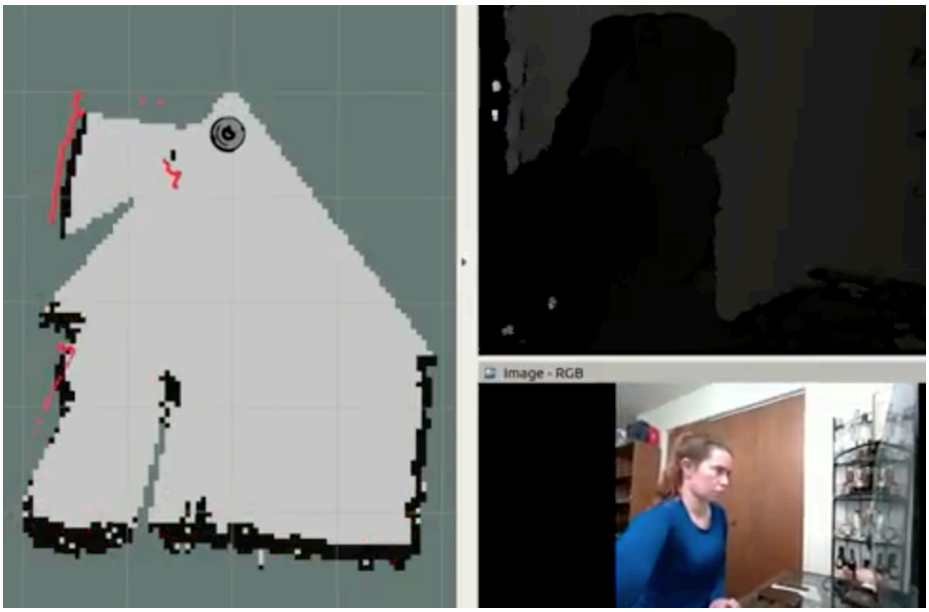



Figure 5.15 Hardware in the loop test of SLAM. Left: SLAM map; Top right: depth; Bottom Right: RGB

### 5.4.3 Iterative Waypoint Navigation in Unmapped Environments

The autonomous driving stack performs incremental waypoint navigation over unmapped terrain, allowing the robot to drive to new waypoints, while mapping and localizing in new unexplored environments. This is utilized by the VLN agent which iteratively issues waypoint goals.

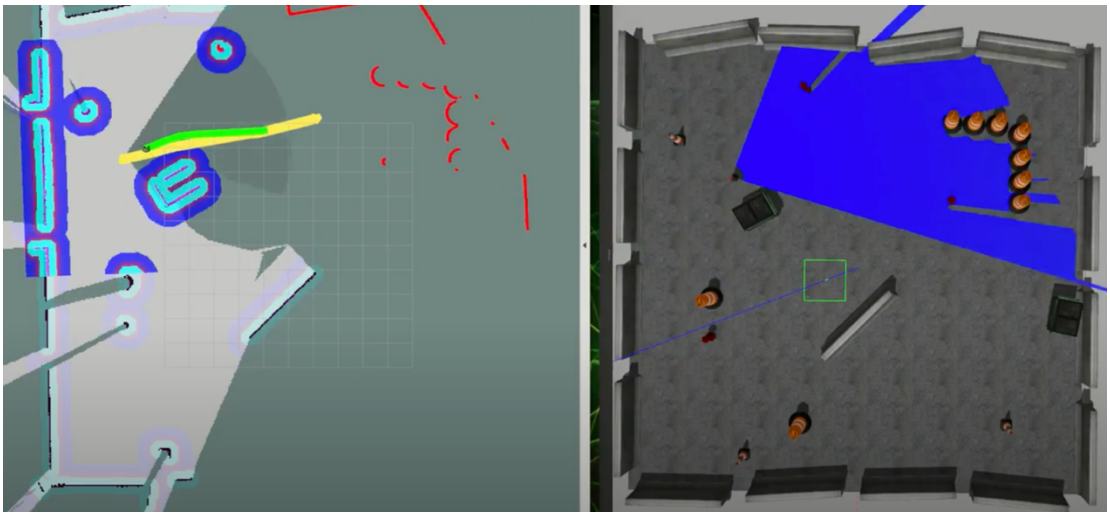


Figure 5.16 Iterative waypoint navigation in simulation. Left: sensory data, in situ map, and planned paths; Right: simulation (blue: sensor FOV and range)

Once the simulation is started, the robot is spawned with sensor models and transmission models. Figure 5.16 shows initial testing and implementation of: SLAM (creates map in grey, and performs localization); Global Costmap (adds opaque blue inflation layer over obstacles on map); Local Costmap (copies local crop of global costmap, adds additional solid blue inflation layer to obstacles currently visible by sensors); Global path (in yellow, determined by A* searching on Global Costmap); Local path (in green, determined by Elastic Band local planner).

## 5.5 Results

The implementation of autonomous driving demonstrated effective SLAM in both simulated and real environments, validating the system's ability to map and localize accurately. Figure 5.17 and Figure 5.18 highlight the mapping and localization capabilities in underground and GNSS-denied settings.

Path planning and dynamic obstacle avoidance were tested with various local planners, with the Elastic Band Planner showing robust performance in handling dynamic obstacles in cluttered environments, as shown in Figure 5.19. The real-world testing confirmed the system's capability to adapt to dynamic changes and follow a global path efficiently.

### 5.5.1 3D SLAM: Implementation & Results

The robot was adapted to for use in unstructured "off-road" environments which are more complex, involving changes in elevation and terrain structure. The robot performed SLAM in a simulated outdoor (and underground environment). Figure 5.17 shows the robot on a bridge in the simulation, and the associated 3D elevation-color-coded point cloud map.

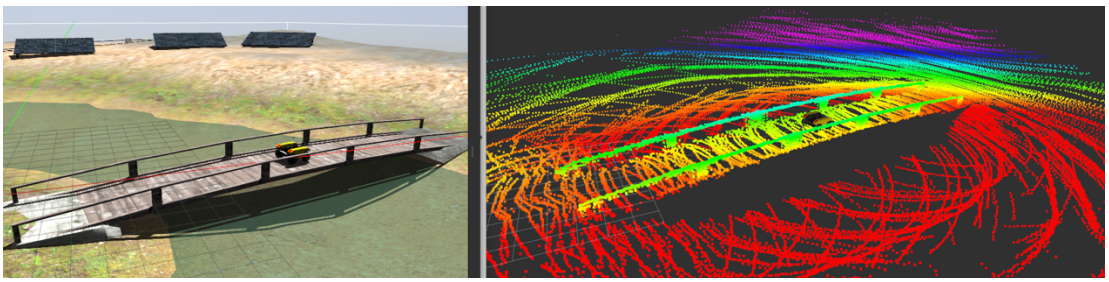
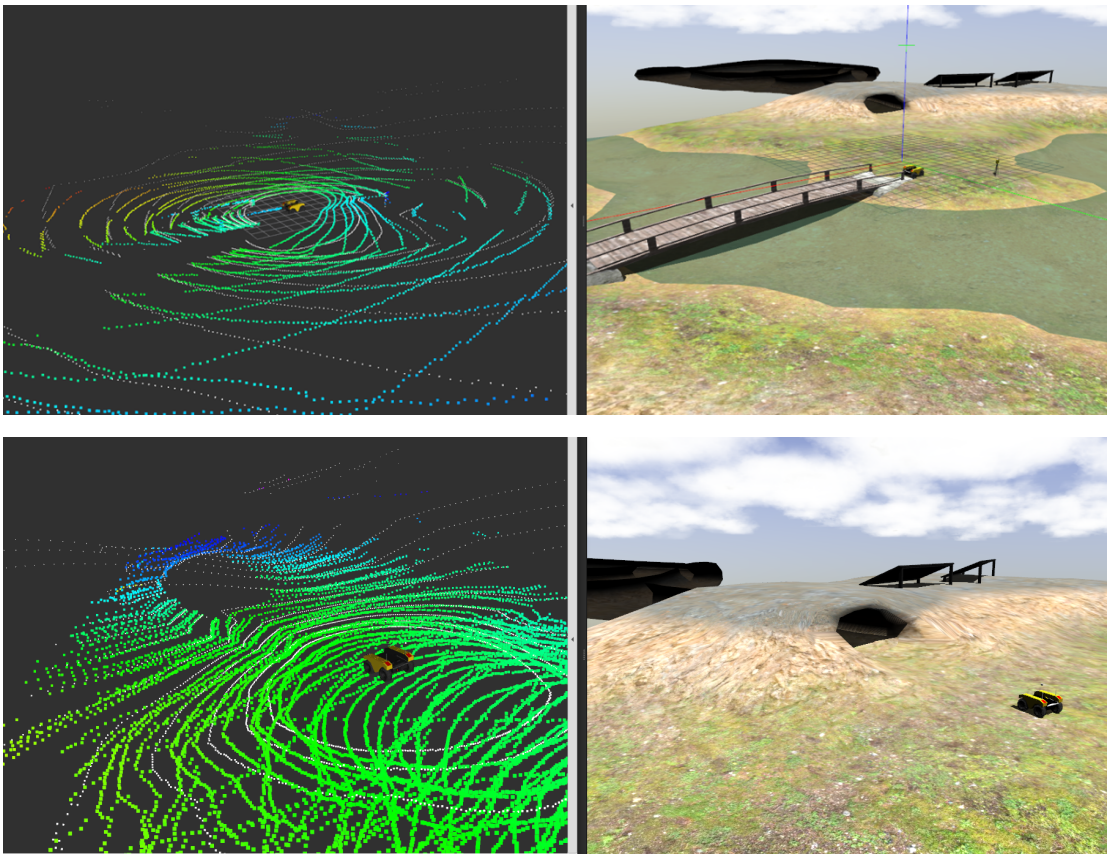


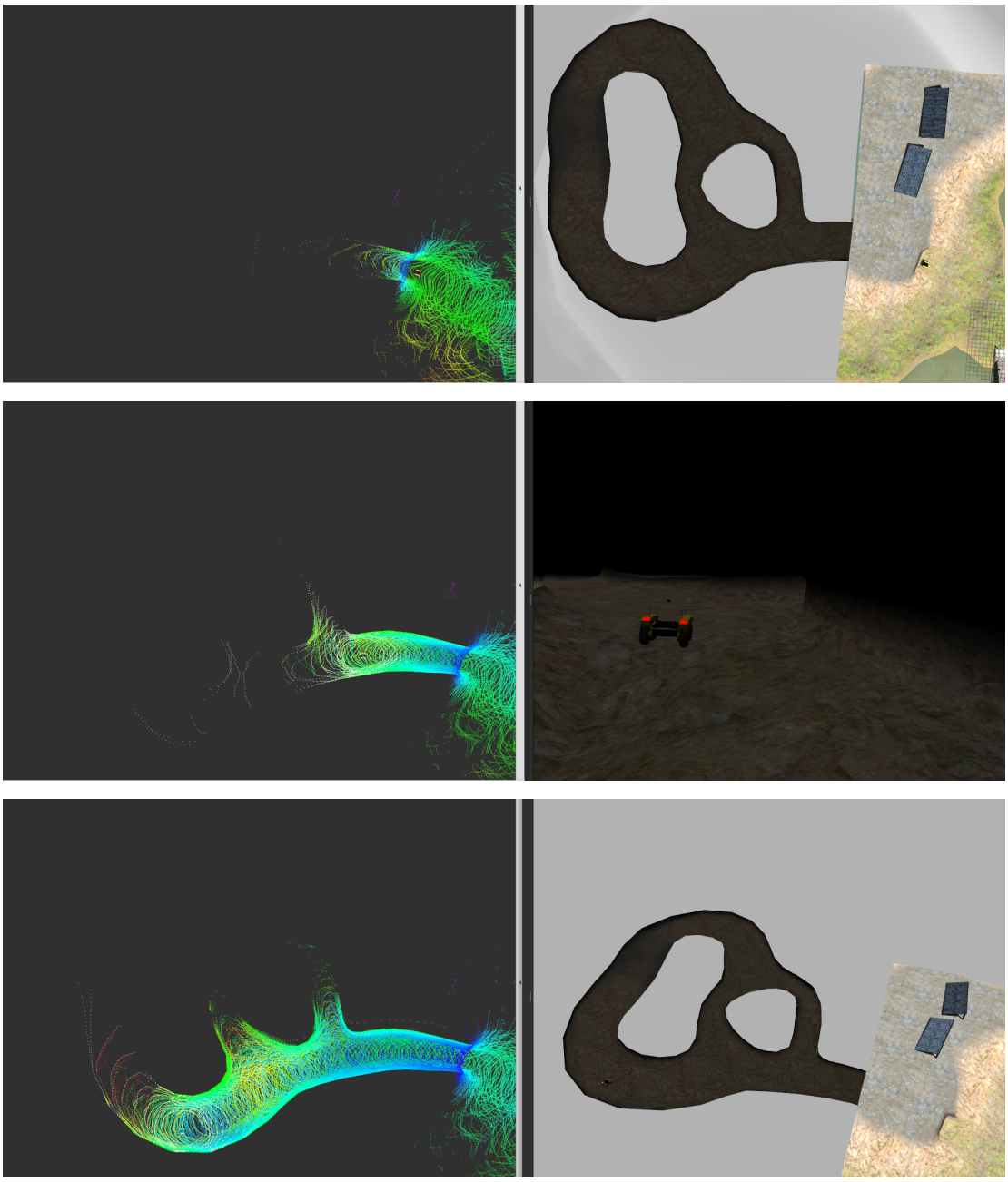
Figure 5.17 Robot mapping bridge. Left: simulation; Right: in situ SLAM map

#### 5.5.1.1 3D Mapping and 6-DOF Localization in Cave Environment

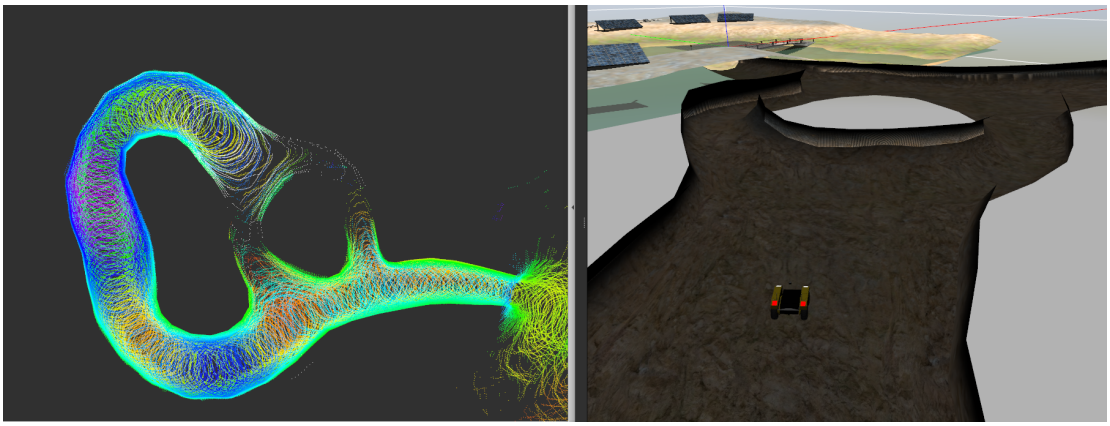
In a series of pictures, Figure 5.18 shows the robot crossing a bridge, entering a cave, and mapping the subterranean environment. On the left side of the images: white voxels show live sensor (point cloud) data, and the colored voxels show the color-coded elevation map. The right side of the below images show the live streamed images as given by the simulated RGB camera on the rover.



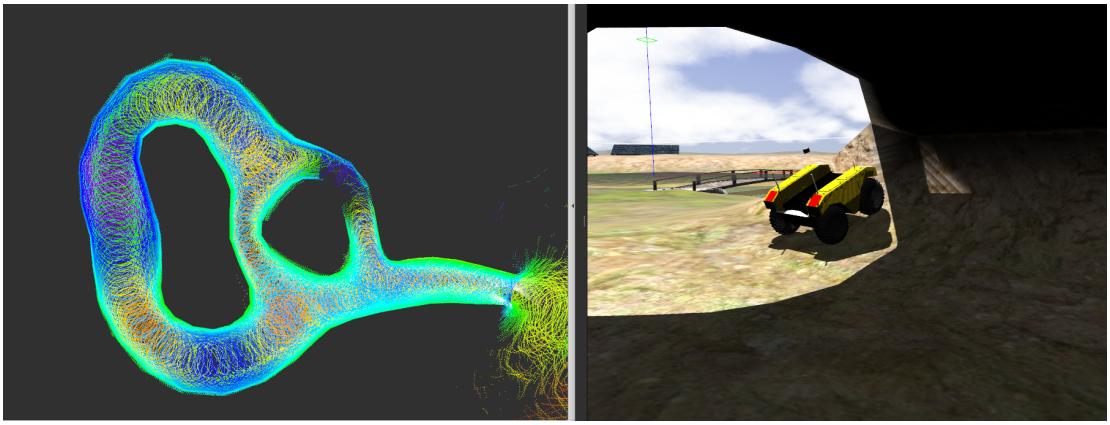
In situ map shows cave entrance.



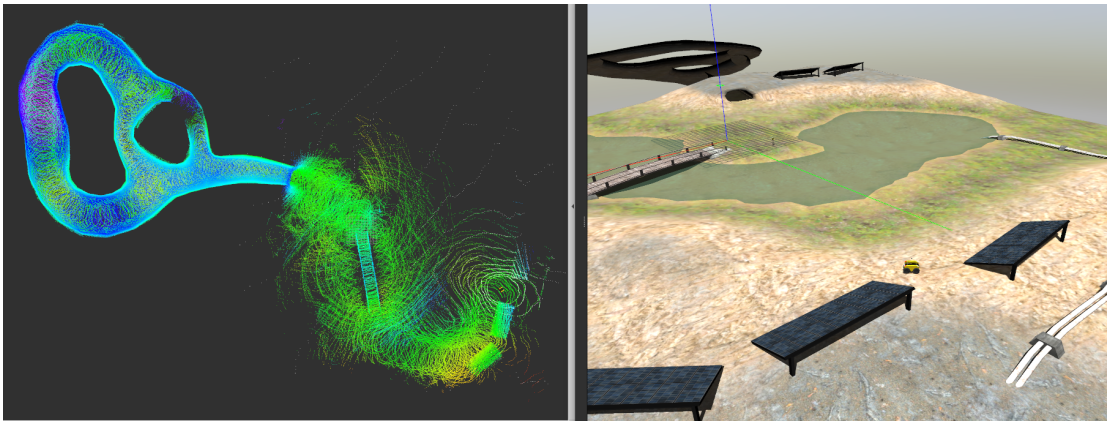
In situ map (left) matches outline of cave (right).



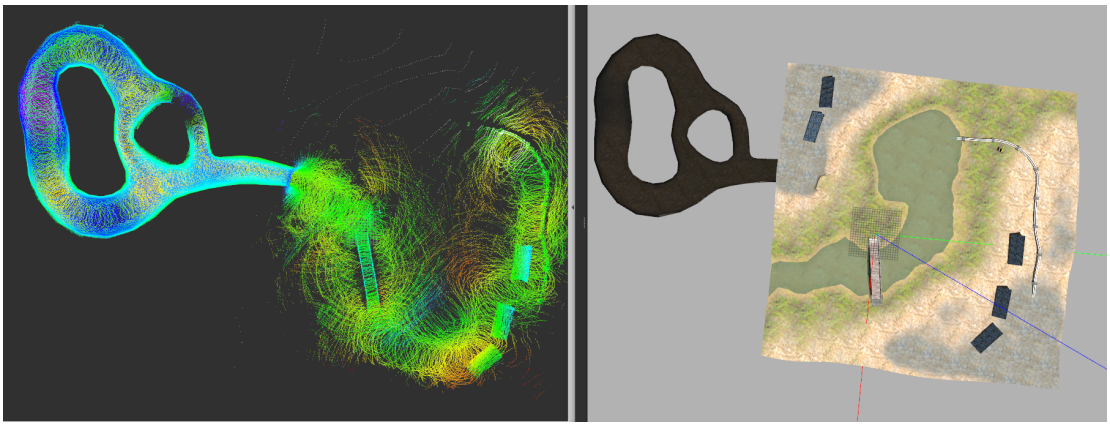
Candidate nodes available for loop closure when returning to previously mapped area.



Loop closure corrects odometry drift. Blue areas signify higher elevation in the cave. The robot finishes mapping the cave, and begins driving back to the bridge



Map depicts full cave system, bridge, and solar panels.



Completion of in situ map (left) compared to top-down view of simulation (right).

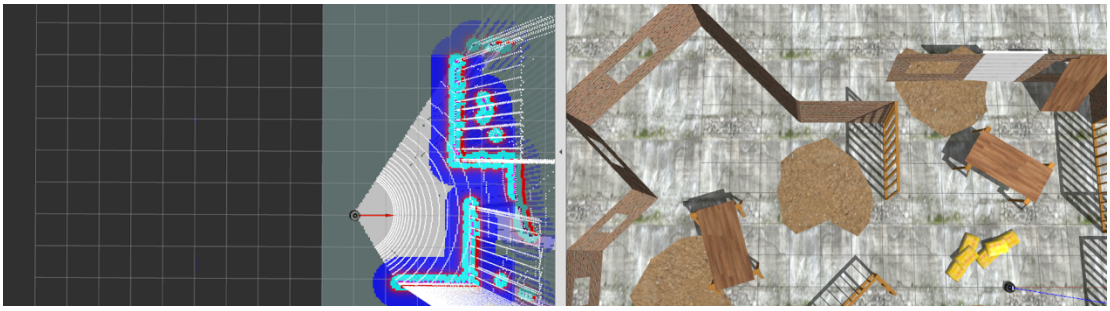
Figure 5.18 Robot mapping a subterranean cave environment

### 5.5.2 Iterative Waypoint Navigation in Unmapped Environments

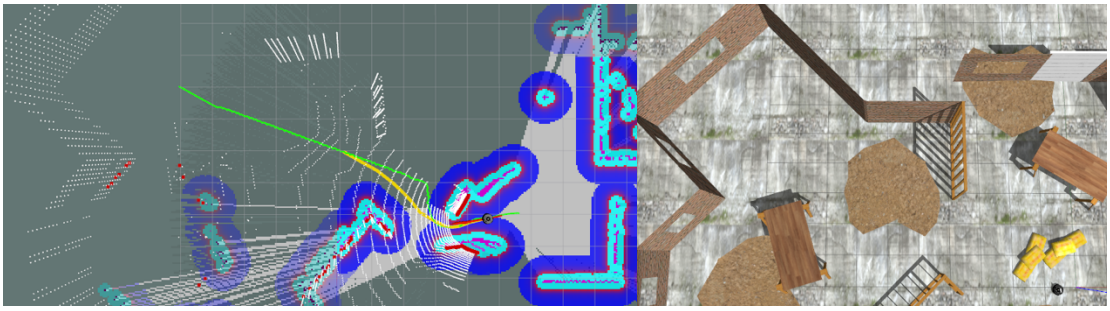
The images in Figure 5.19 show the rover performing iterative waypoint navigation in unobserved (and unmapped) environments. It does this by performing 3D 6-DOF SLAM (mapping a 3D environment while tracking 6-DOF pose), and navigating to waypoints in an unmapped environment, allowing it to perform frontier exploration.

On the global costmap, unmapped areas have a 3 times higher cost to traverse than mapped traversable areas. This encourages the global planner to stay in mapped terrain, while still allowing it to venture into new territory as necessary to find a viable path.

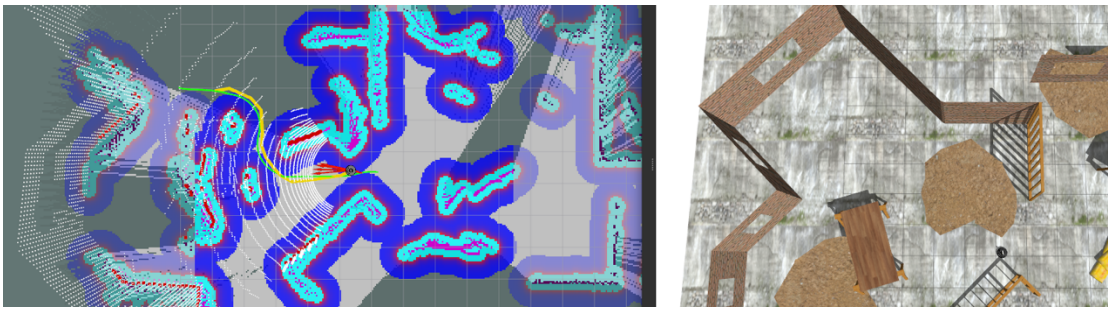
As a precaution, the local costmap assigns lethal costs to unmapped areas, forcing it to verify whether the terrain is traversable as it approaches, before driving over it. In rare cases, as the robot approaches an unmapped area, the robot may be unable to fill in the map, either due to the presence of a cliff (resulting in no point cloud return), or debris on the sensor (similarly resulting in no point cloud return). These two dangerous scenarios result in the local costmap conservatively assuming the worst-case scenario and representing the area as non-traversable.

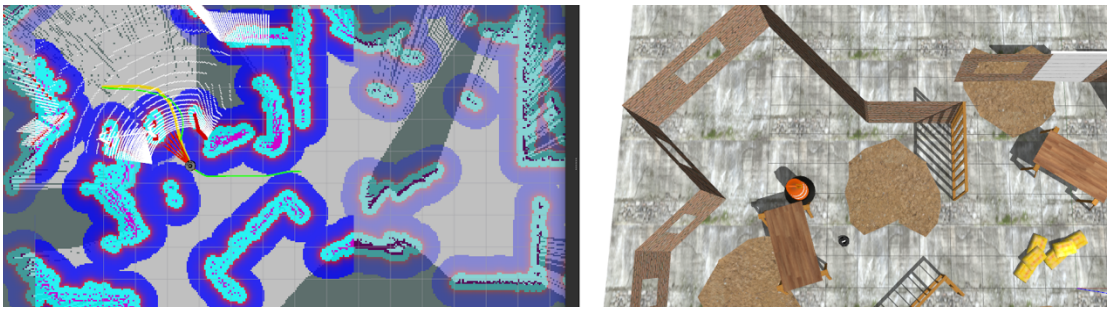


Robot starts with no previous map; initializes map with first reading from depth sensor.

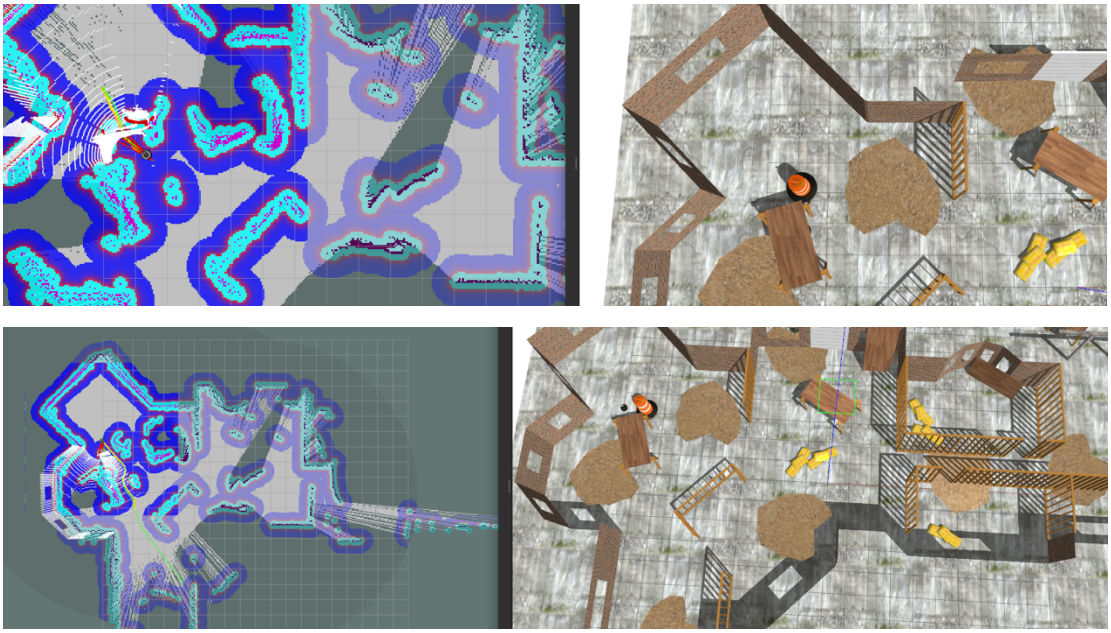


Waypoint is specified in the top-left corner of the environment, green global path is created, and yellow local path contracts the path around corners where possible.

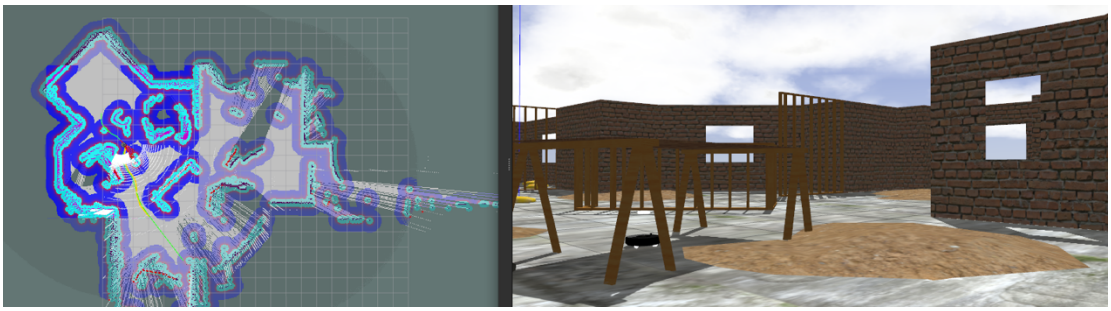


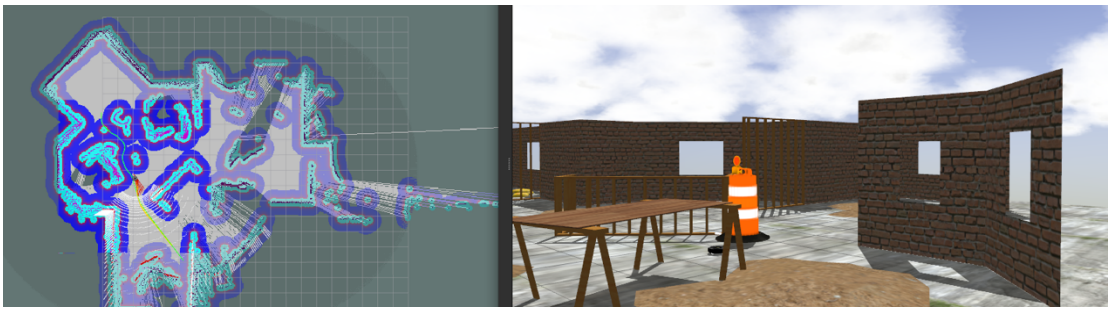


A traffic cone is placed in the robot's path, forcing it to detour.

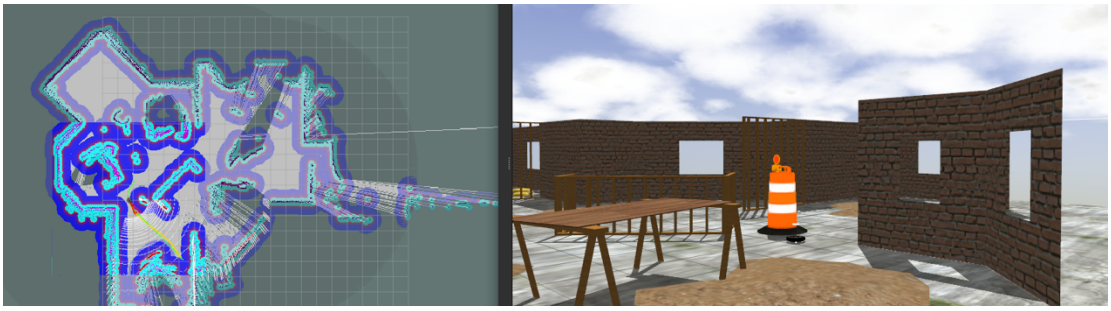


Robot reaches first waypoint, and is then directed to bottom right of environment.

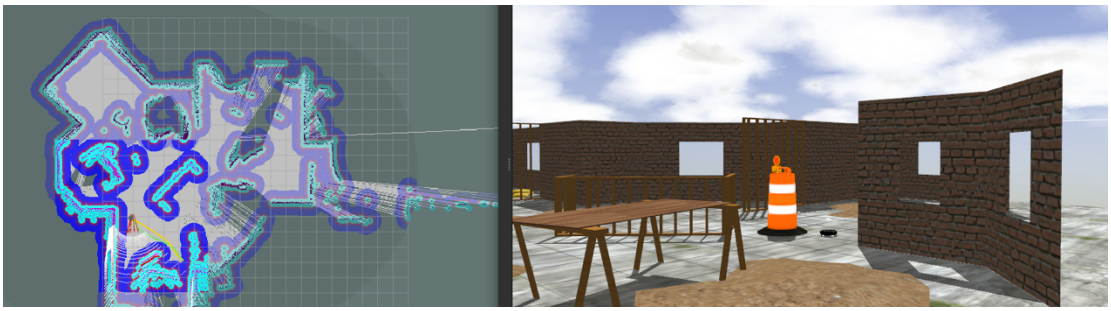


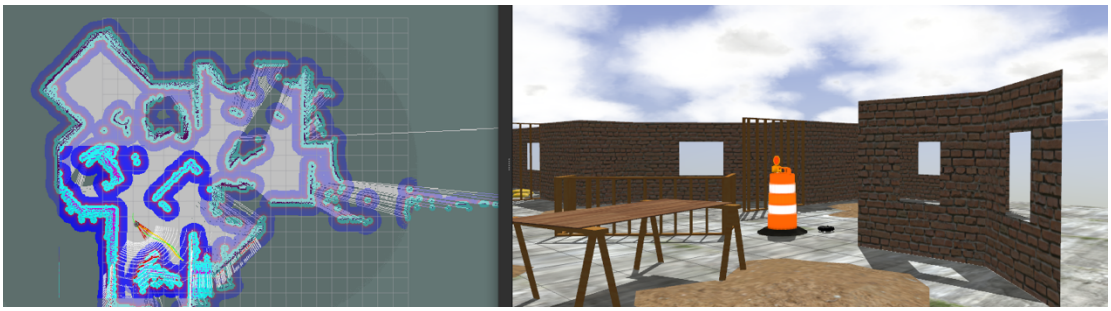


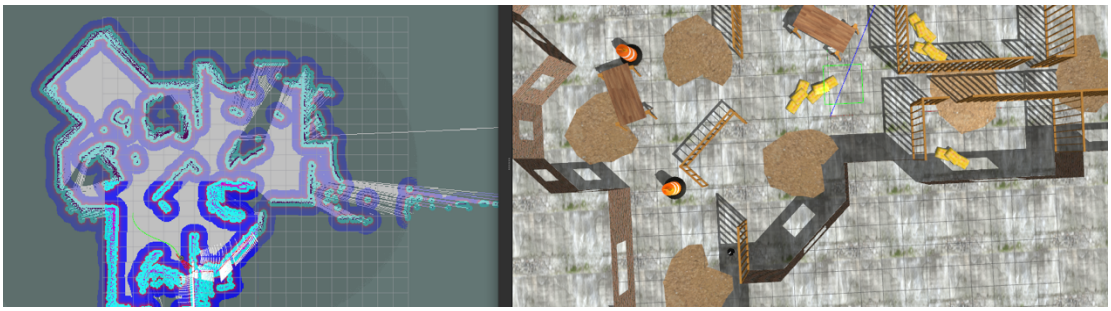
Another traffic cone is placed in the path of the robot.



Robot detours without collision







Robot reaches second waypoint destination, despite dynamic obstacles obstructing its path.

Figure 5.19 Robot performing SLAM and iterative waypoint navigation, enabling it to explore unmapped frontiers

## 5.6 Conclusions in Autonomous Driving

### 5.6.1 Improvements & Contributions Recap

This research outlines the creation of an autonomous driving stack and VLN agent that can simultaneously, and in real-time: map, localize, path plan, execute, and detour. Dynamic obstacle avoidance is implemented by referencing the in situ SLAM map (which has a slow update rate but larger spatial scope), and live data (which has a fast update rate, but smaller spatial scope). Waypoint navigation is performed in unmapped environments by creating different assumptions for unmapped areas in the global map versus the local map. The autonomous driving stack, capable of path planning, localization, and mapping, supports the VLN agent which relies on obstacle-free path execution to the waypoints it specifies.

In work on 2D autonomous driving (driving on a plane), the autonomous driving agent saw a 100% success rate in 100-200 simulated runs, and 50-100 real life hardware demos. The definition of success is that the agent always reached its goal (or got as close as possible to the goal if the goal was impossible to reach), and never crashed into an obstacle.

When driving in 3D terrain (rough, sloped, rocky environments), the autonomy stack was able to create maps of sparse environments that were representative of the terrain, tested over 50 meter stretches in both simulated and real environments, while accurately tracking the 6-DOF pose.

### 5.6.2 Lessons Learned

Through the design, build, and test phases of the autonomous driving stack, there are several important takeaways from this work:

Each iteration of autonomy architectures as outlined in Section 5.3: Prior Architectures: Incremental Improvements highlight different lessons learned and opportunities for improvement. This includes using a local planner in conjunction with a global planner, allowing for dynamic updates to the global map, conditional obstacle clearing, expanded detour areas, and relaxed global localization tolerance.

The Local Planner must integrate information from both the global map to fill in blanks from live sensor data, and the live sensor stream in case a new obstacle appeared between map updates. Fast and safe navigation requires the local planner to adapt based on the most recent sensor data, while also referencing the global map for overall context.

Planners have opposite assumptions for frontier exploration. The global planner must assume that unmapped areas are clear of lethal obstacles, allowing the planner to explore new territories and update the global map as new areas are discovered. In contrast, the local planner must assume unmapped areas contain lethal obstacles. This precautionary approach ensures that the local planner can avoid potential hazards when the traversability of new areas cannot be confirmed through sensor data. For example, if a local planner cannot verify a map's traversability because there is no

point cloud representation of that area, it may be because that area is above a drop-off or cliff, or the depth sensor cannot register a depth value if an obstacle is too close.

Side-facing visual sensors, such as wide FOV lidars and fish-eye cameras, are effective for capturing odometry and improving positioning estimates; and detecting obstacles from the sides which is helpful when passing closely between obstacles. However, higher-resolution and more accurate forward-facing sensors (eg lidar and depth cameras) provide better depth registration, improving both map accuracy and path planning efficacy.

These lessons underscore the importance of integrating global and local information, the need for careful handling of assumptions during exploration, and the critical role of high-quality sensors in achieving accurate and efficient autonomous navigation.

### 5.6.3 Future Work

Future work in autonomous driving could greatly benefit from exploring alternative global planning algorithms like LPA*, D* Lite, and θ* to enhance dynamic obstacle handling and path efficiency. Additionally, advanced localization techniques that leverage elevation maps and sensor data could address challenges in GPS-denied environments, improving overall navigation accuracy and robustness. These advancements will push the boundaries of autonomous driving systems, making them more adaptable and capable in diverse and dynamic environments.

#### 5.6.3.1 Exploring Different Global Planners

Currently, the global planner uses A*, a heuristic-guided informed search algorithm. There are some drawbacks to A*, for example if the global path needs to be replanned (if a new obstacle obstructs it), the entire path needs to be recalculated. Thus it is worth exploring other informed search algorithms for global planning such as LPA* (Lifelong-Planning-A*) [90] and D*Lite [91] [92] which are more efficient at re-planning. It would be valuable to benchmark θ* as well for large, open environments.

D* is a dynamic version of A* in that it can handle dynamic obstacles without having to perform an entirely new search. This would be helpful for more quickly reevaluating the global path if a new obstacle required a detour outside the spatial bounds of the local planner. D*Lite is a faster and simpler version of D*, based on LPA*.

In contrast to A*, D* searches backwards from the goal node, effectively computing the A* optimal path for any possible location on the map, which is helpful for re-routing a portion of the path, or restarting the search from a new location. D* adapts to new (dynamic) obstacles by simply adjusting the weights of the affected edges along the intended path in real-time.

It would be worth exploring θ* [93] (pronounced “theta-star”) – an any-angle planner which is relieved of the 8-way connected limitation that A* (and other grid-based searches) are bound to. This algorithm may find the path more quickly in landscapes with sparse obstacles but may need to be adjusted for working on

costmaps where there are different costs associated with different preferences of traversable area. While a post-smoothed A* can shorten the path after search has completed, θ* effectively searches and smooths at the same time.

#### 5.6.3.2 GPS-Denied Global Localization

This work could be extended to replace GPS signals with other global-localization capabilities in GPS-denied environments. This can be done by implementing IMU-based global localization, eg: TRIAD, point cloud registration, and other approaches outlined in section 5.2.6.1: Methods for Global Localization in GPS denied environments.

#### 5.6.3.3 Motion Planning for High-Speed Off-Road Driving

It would be valuable to create a kino-dynamically feasible motion planner for high-speed autonomous driving, where the vehicle is going sufficiently fast to be concerned about rollover and traction control. Future work may consider fastest path rather than shortest path, stability control and traction control, and braking in advance before anti-banked turns or on a downhill.

# 6 Appendix A: Transition to Industry

*All preceding sections have been approved for public release, under Public Information Release Authorization (PIRA) # SSS2024111443.*

*This following section has been approved for public release, under Public Information Release Authorization (PIRA) # SSS2024091300.*

The figures in this section showcase the transition of academic research to industry applications, where it was adapted to different hardware, with faster onboard compute and higher accuracy sensors.

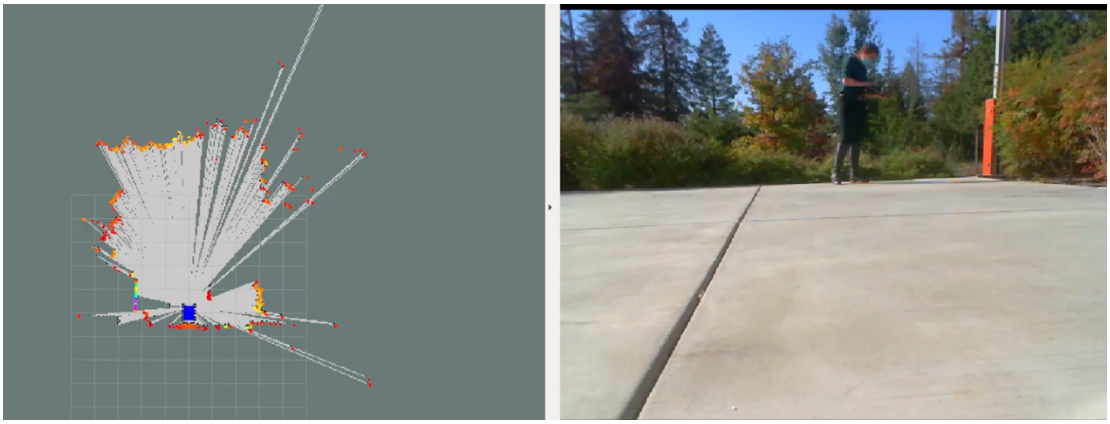
## 6.1 Fully Deployed Rover Mapping Environment

The video frames in Figure 6.1 show a live hardware demonstration of 2D SLAM in an outdoor environment using a Velodyne lidar with 16 channels and panoramic FOV. The rover can be seen mapping its surroundings and localizing within the map it creates.

On the right of the images in Figure 6.1 is the view the rover-mounted webcam. The webcam's livestream gives the operator context into what part of the environment the rover is in, and how it relates to the SLAM performance, but is not itself used for autonomous driving, solely for user insight.

Images on the left side of Figure 6.1 show the rover model (blue chassis and black wheels) and what the robot perceives: rainbow colored pixels show the intensity of the lidar return off local obstacles; black pixels show lidar returns on obstacles within the sensor's depth range; light grey pixels show unobstructed areas as defined by a

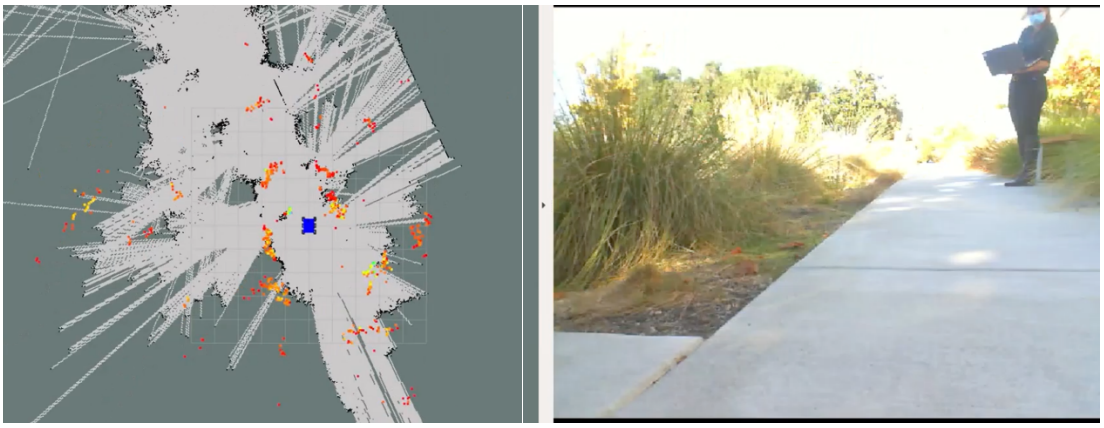
ray being traced through that location to an obstacle behind it; dark grey pixels show unknown areas: unknown either because there was no lidar return from that ray, or because the ray was stopped by a closer obstacle.



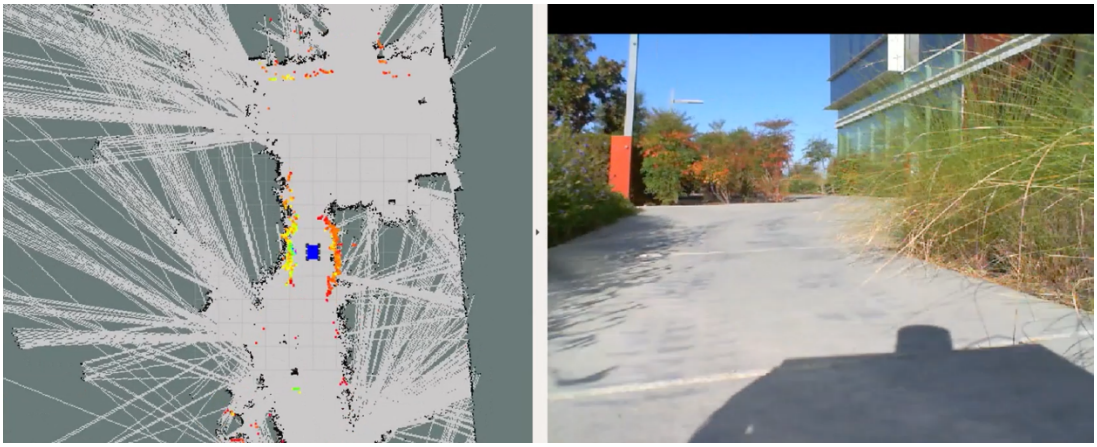
In a new environment, the initial frame of the map is the first frame from the sensors.

Rover drives forward and makes left turn down a sidewalk.

Rover progresses down sidewalk, filling in the map and localizing as it goes.



Rover makes U-turn and backtracks.



Rover passes original starting location.

Rover begins turn onto river stones

Rover continues along river stones, with wheel slip causing the odometry to drift.

Rover returns to starting location, showing the correct position on the map utilizing loop closure, with lidar returns matching mapped obstacles.

Figure 6.1 Rover performing 2D SLAM fused with wheel odometry in real, outdoor environment *(PIRA # SSS2024091300)*

## 6.2 Real-World Testing Path Planning & Dynamic Obstacle Avoidance

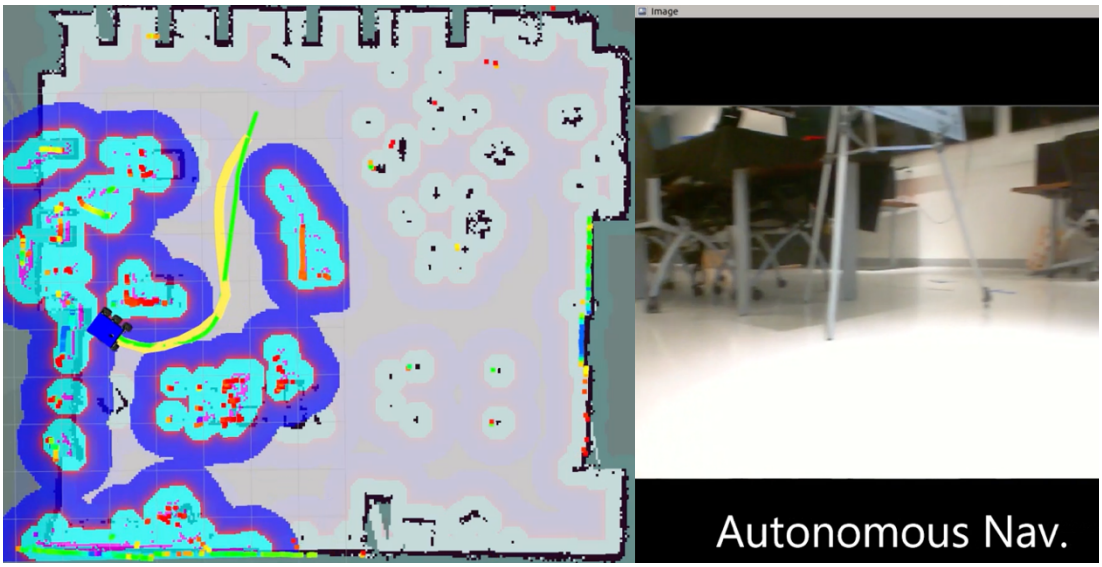
Figure 6.2 shows path planning and dynamic obstacle avoidance as tested in an indoor environment.

The right side of the images shows the rover's view via a webcam; this purely provided context to the operator and was not used for autonomous driving. The left of the images illustrates what the rover is perceiving and how it is planning in its environment as represented by:

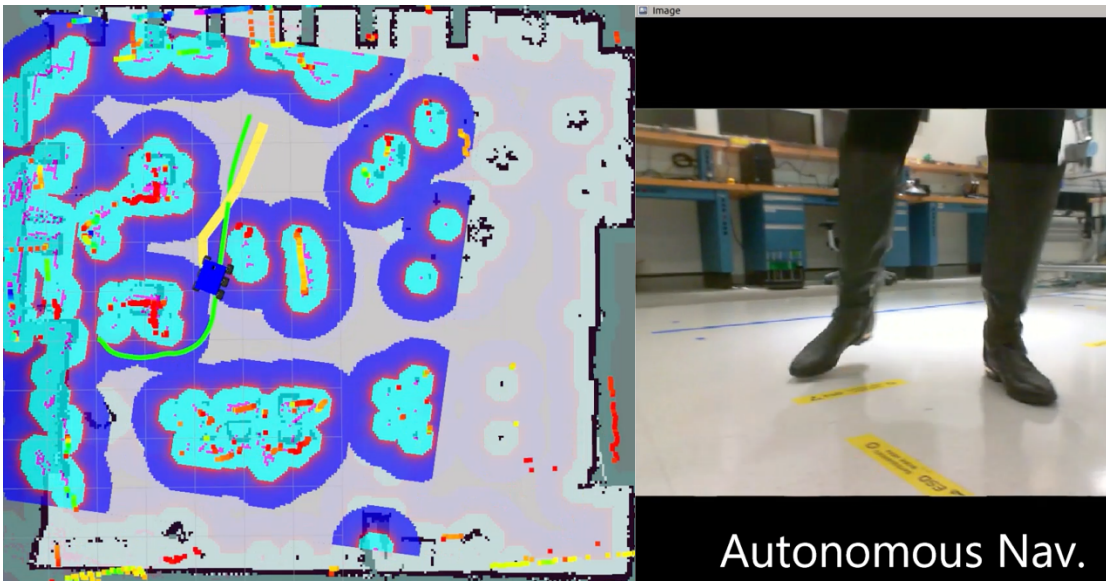
- Green path: Global path which gets rover to its goal; may traverse entire global costmap, which is shown as the larger opaque map in the background
- Yellow path: Local planner which follows global path, while detouring around dynamic obstacles
- Rainbow colored pixels: color coded intensities of lidar laser returns
- Black pixels: lethal obstacles
- Cyan inflation layer with red outline: lethal inflation layer just wider than half the width of the rover, allowing the planned path to be agnostic to the rover's width
- Blue inflation layer: safety layer which adds a low (but not lethal) cost to the costmap. This encourages the planners to give more breadth around an obstacle, but still allows the rover to pass though if necessary (when the

alternative is a significantly larger detour which would result in a larger total path cost).

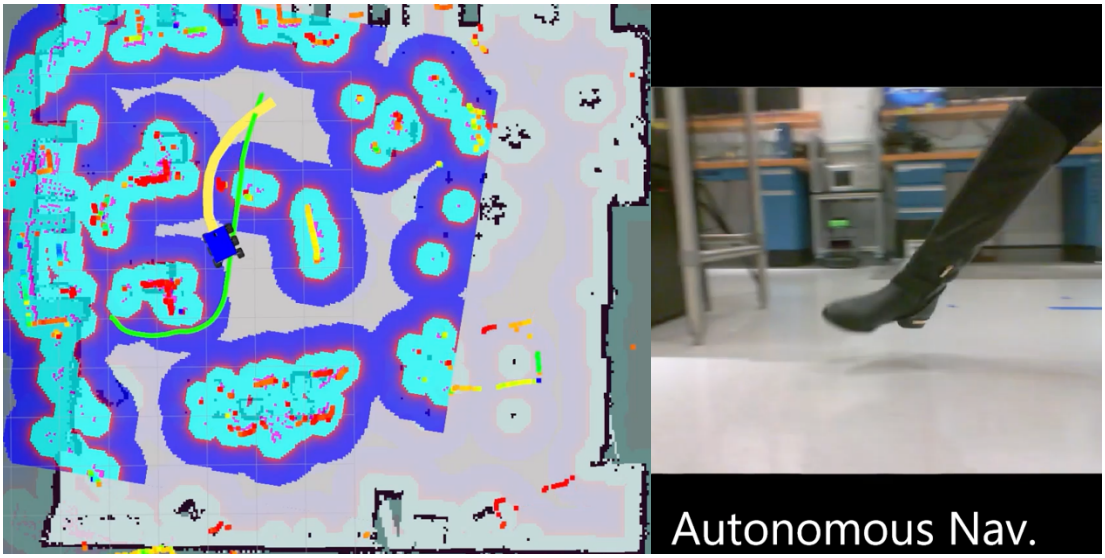
- Large opaque map in background: global costmap
- Smaller, more brightly colored map in foreground: local costmap, which inherits obstacles from global costmap and live sensor stream



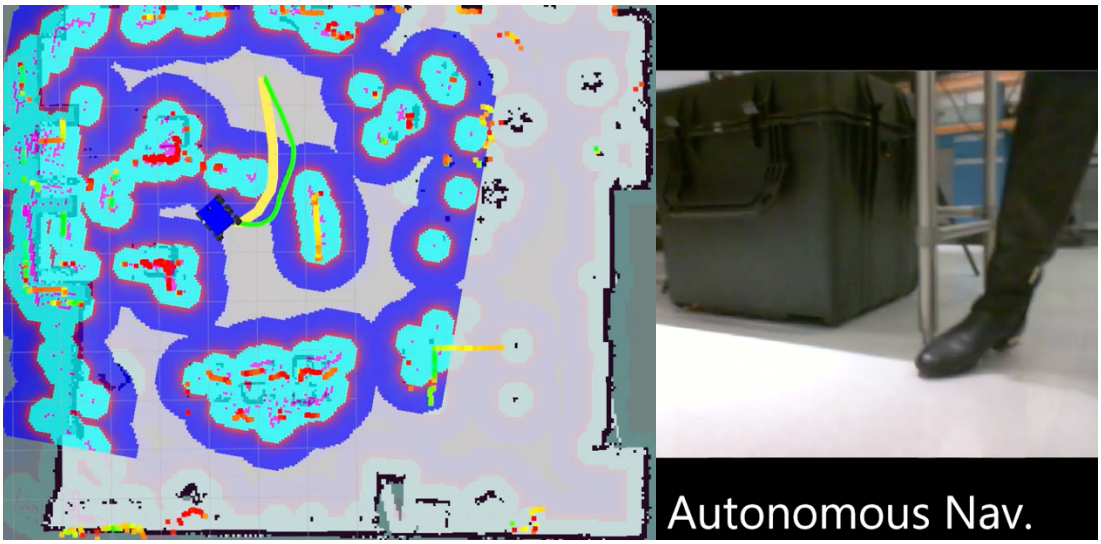

The rover in its starting location; the global planner plans a new path to the goal at the top left of the map, as indicated by the end of the green line. The yellow local path follows the green global path since there are no obstacles in the way.



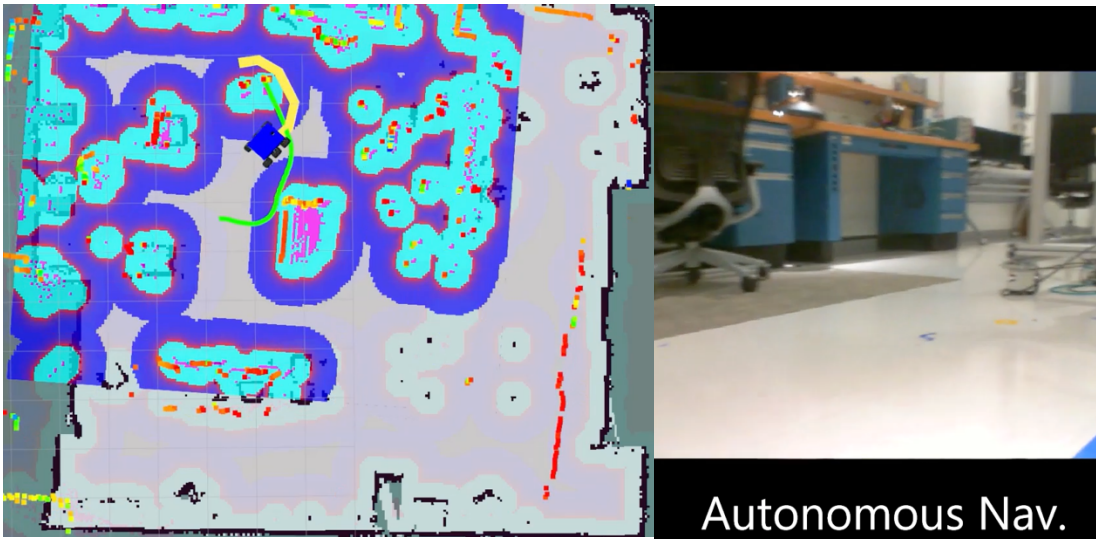

As seen in the webcam view, a (human) obstacle steps in front of rover where it was planning to drive, and a lidar point cloud appear where the person stands on the (green) global path. The yellow local path detours around the person while rendezvousing with the global path later.



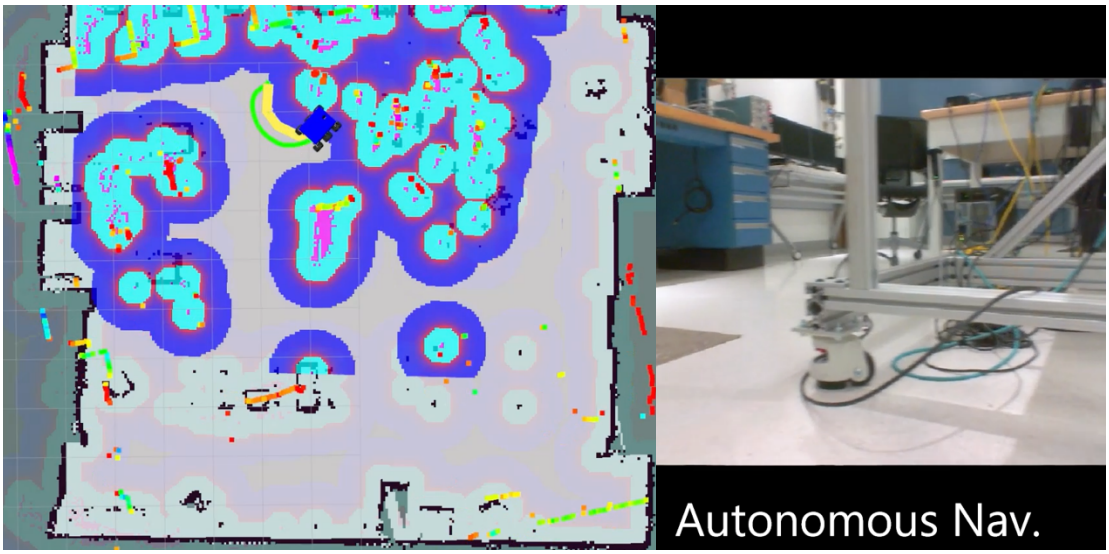

Person steps even further into the local path, and the red lidar points show mark the new the boot as a new obstacle…



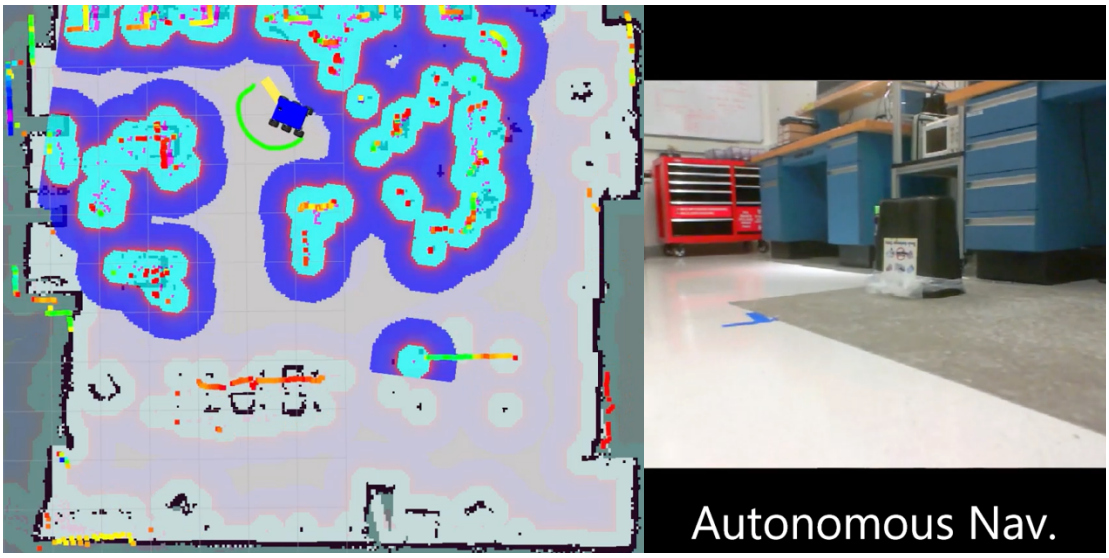

… And 0.05[s] later the local planner deems the detour unsafe and the global path untraversable. The global planner determines a new path (in green), and the local planner (in yellow) follows the new path.




The obstacle (person) moves in front of the rover again, and the yellow local path deflects again.




Person continues moving up and to the right in the map, making the last detour untraversable, so the global planner re-routes yet again.

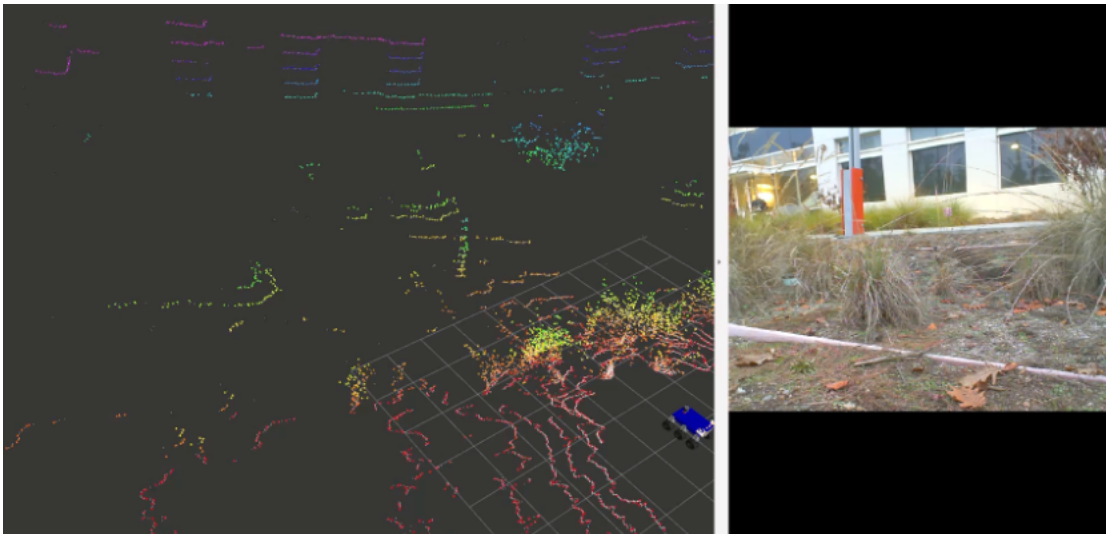



Person moves out of the way of the rover, allowing the local planner to contract its (yellow) path and drive straight to the furthest point on the global path, which is the end destination.

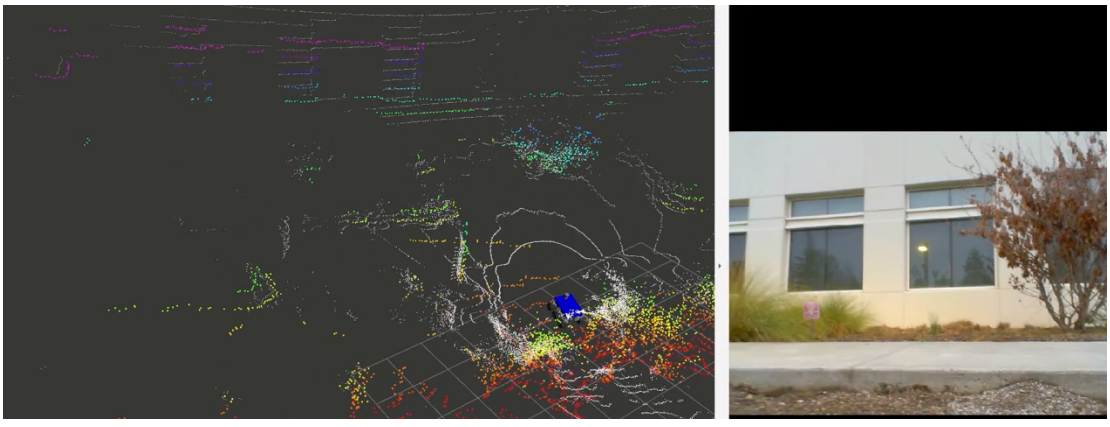
Figure 6.2 Rover performing path planning and dynamic obstacle avoidance in a real environment, as person jumps in front of the rover and blocks path *(PIRA # SSS2024091300)*

## 6.3 Crawling Out of a Ditch with 3D SLAM in a Real Environment

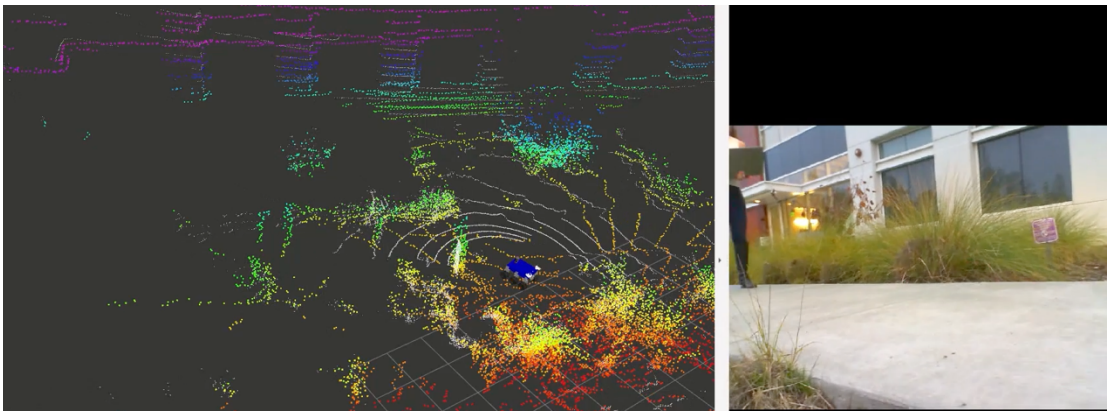
Figure 6.3 shows the rover pitching and rolling as it climbs out of a ditch, while performing SLAM, showcasing the rover's ability to track pose (location and orientation) while mapping.



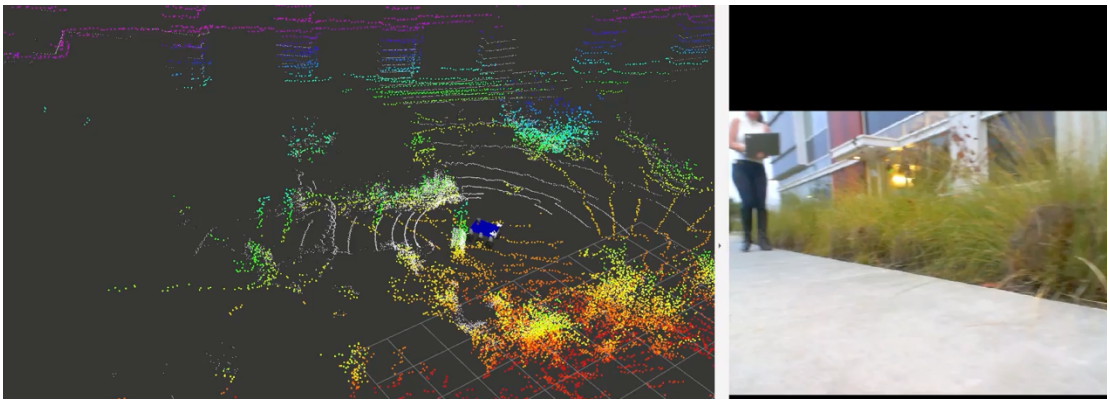
In a new environment, the first frame of the map is the first point cloud received.



Rover is pitching up as it crawls over a piece of wood.



Rover uses features on the building to help align point clouds as it is crawling up a cement curb.



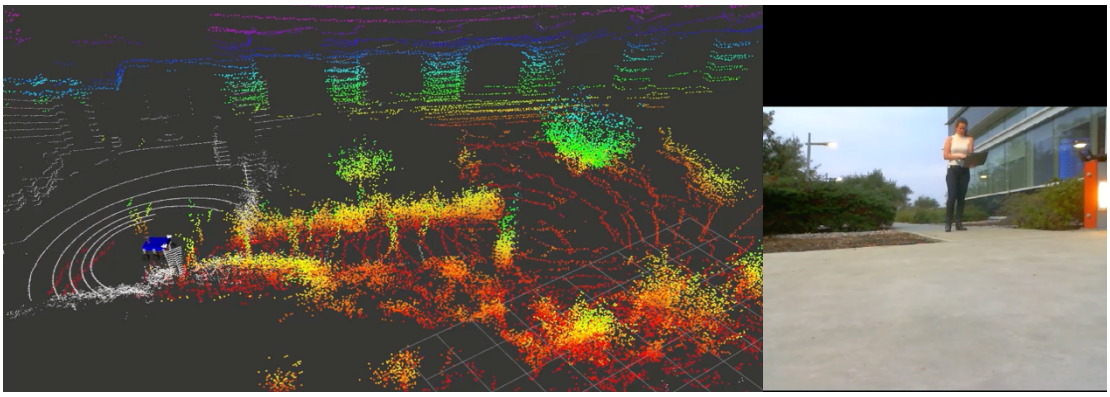


Figure 6.3 Rover performing 3D SLAM while pitching and rolling as it crawls out of a ditch *(PIRA # SSS2024091300)*

# 7 Bibliography


[1] J. Gu, Eliana Stefani, Q. Wu, J. Thomason and X. Wang, "Vision-and-Language Navigation: A Survey of Tasks, Methods, and Future Directions," In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7606–7623, Dublin, Ireland. Association for Computational Linguistics., ACL 2022. [Online]. Available: https://arxiv.org/abs/2203.12667 https://aclanthology.org/2022.acl-long.524/.

[2] J. Krantz, A. Gokaslan, D. Batra, S. Lee and O. Maksymets, "Waypoint Models for Instruction-guided Navigation in Continuous Environments," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 2021, pp. 15142-15151, doi: 10.1109/ICCV48922.2021.01488, [Online]. Available: https://ieeexplore.ieee.org/document/9710310 https://arxiv.org/pdf/2110.02207.

[3] Amazon, "Meet Astro," 28 September 2021. [Online]. Available: https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other. [Accessed 2024 ].

[4] Honda, "What we Learned from ASIMO," [Online]. Available: https://global.honda/en/robotics/asimo/.

[5] Boston Dynamics, "Atlas and beyond: the world's most dynamic robots," [Online]. Available: https://bostondynamics.com/atlas/. [Accessed 2024].

[6] NASA, "NASA Selects Companies to Advance Moon Mobility for Artemis Missions," Apr 03, 2024 . [Online]. Available: https://www.nasa.gov/news-

release/nasa-selects-companies-to-advance-moon-mobility-for-artemis-missions/.

[7] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sunderhauf, I. Reid, S. Gould and A. v. d. Hengel, "Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments," IEEE Conference on Computer Vision and Pat- tern Recognition (CVPR), 2018. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Anderson_Vision-and-Language_Navigation_Interpreting_CVPR_2018_paper.pdf.

[8] P. Anderson, A. Shrivastava, J. Truong, A. Majumdar, D. Parikh, D. Batra and S. Lee, "Sim-to-real transfer for vision-and-language navigation," Conference on Robot Learning (CoRL), 2020. [Online]. Available: https://arxiv.org/abs/2011.03807.

[9] A. Ku, P. Anderson, R. Patel, E. Ie and J. Baldridge, "Room-Across-Room: Multilingual Vision-and-Language Navigation with Dense Spatiotemporal Grounding," Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), November 2022. [Online]. Available: https://aclanthology.org/2020.emnlp-main.356/.

[10] J. Krantz, E. Wijmans, A. Majumdar, D. Batra and S. Lee, "Beyond the Nav-Graph: Vision-and-Language Navigation in Continuous Environments," In: Vedaldi, A., Bischof, H., Brox, T., Frahm, JM. (eds) Computer Vision – ECCV 2020. ECCV 2020. Lecture Notes in Computer Science(), vol 12373. Springer, Cham, 03 November 2020. [Online]. Available: https://arxiv.org/abs/2004.02857v2 https://link.springer.com/chapter/10.1007/978-3-030-58604-1_7.

[11] S. Ross, G. J. Gordon and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," Proceedings of the

Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15:627-635, 2011, [Online]. Available: https://arxiv.org/pdf/1011.0686.

[12] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt and P. Stang, "Stanley: The Robot that Won the DARPA Grand Challenge," Wiley InterScience, 25 September 2006. [Online]. Available: http://robots.stanford.edu/papers/thrun.stanley05.pdf.

[13] NASA JPL, "Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission," [Online]. Available: https://robotics.jpl.nasa.gov/media/documents/MER_ISER2004.pdf.

[14] NASA, "Spirit and Opportunity's Science Instruments," [Online]. Available: https://science.nasa.gov/mission/mars-exploration-rovers-spirit-and-opportunity/science-instruments/.

[15] NASA JPL, "How Perseverance Drives on Mars," [Online]. Available: https://science.nasa.gov/resource/how-perseverance-drives-on-mars/.

[16] NASA, "Perseverance Rover Components," [Online]. Available: https://science.nasa.gov/mission/mars-2020-perseverance/rover-components/.

[17] DARPA, "The Grand Challenge," [Online]. Available: https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles.

[18] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt and P. Stang, "Stanley: The Robot that Won the DARPA Grand Challenge," Journal of Field Robotics, 2006. [Online]. Available: http://robots.stanford.edu/papers/thrun.stanley05.pdf.

[19] DARPA, "DARPA Urban Challenge," [Online]. Available: https://www.darpa.mil/about-us/timeline/darpa-urban-challenge.

[20] M. Montemerlo, e. al. and S. Thrun, "Junior: The Stanford Entry in the Urban Challenge," The DARPA Urban Challenge. Springer Tracts in Advanced Robotics, vol 56. Springer, Berlin, Heidelberg, 2009. [Online]. Available: http://robots.stanford.edu/papers/junior08.pdf.

[21] C. Urmson and e. al., "Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge," 2007. [Online]. Available: https://www.ri.cmu.edu/pub_files/2007/4/Tartan_Racing.pdf.

[22] DARPA, "Subterranean Challenge Urban Circuit," [Online]. Available: https://www.darpa.mil/about-us/subterranean-challenge-urban-circuit.

[23] U.S. Department of Defense, "Subterranean Challenge Robotics Triathlon," [Online]. Available: https://www.defense.gov/Multimedia/Experience/Subterranean-Challenge/.

[24] DARPA, "Team CERBERUS and Team Dynamo Win DARPA Subterranean Challenge Final Event," 24 September 2021. [Online]. Available: https://www.darpa.mil/news-events/2021-09-24a.

[25] Team Cerberus, "Team CERBERUS wins the DARPA Subterranean Challenge!," [Online]. Available: https://www.subt-cerberus.org/. [Accessed 2024].

[26] M. Tranzatto and e. al., "CERBERUS: Autonomous Legged and Aerial Robotic Exploration in the Tunnel and Urban Circuits of the DARPA Subterranean Challenge," Field Robotics, 2022. [Online]. Available: https://arxiv.org/abs/2201.07067.

[27] M. Tranzatto and e. al, "Team CERBERUS Wins the DARPA Subterranean Challenge: Technical Overview and Lessons Learned," 11 July 2022. [Online]. Available: https://arxiv.org/pdf/2207.04914v1.

[28] Jeep, "Jeep AI & Autonomous Off-Road Driving Technology," [Online]. Available: https://www.jeep.co.uk/news/autonomous-driving.

[29] Tesla, "Autopilot," [Online]. Available: https://www.tesla.com/autopilot.

[30] R. Akkaya, O. Aydogdu and S. Canan, "An ANN Based NARX GPS/DR System for Mobile Robot Positioning and Obstacle Avoidance," Journal of Automation and Control 1.1, (2013): 6-13. [Online]. Available: https://pubs.sciepub.com/automation/1/1/2/index.html.

[31] Y. Sun, Y. Qiu, Y. Aoki and H. Kataoka, "Outdoor Vision-and-Language Navigation Needs Object-Level Alignment," *Sensors (Basel, Switzerland), 23(13), 6028.,* vol. https://doi.org/10.3390/s23136028, 2023.

[32] R. S. Sutto and A. G. Barto, Adaptive Computation and Machine Learning series Reinforcement Learning, Second Edition, An Introduction, MIT Press, 2018.

[33] L. Pasqualini and M. Parton, "Pseudo Random Number Generation through Reinforcement Learning and Recurrent Neural Networks," Algorithms 2020, 13(11), 307. [Online]. Available: https://www.mdpi.com/1999-4893/13/11/307.

[34] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva and D. Batra, "DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames," ICLR 2020. [Online]. Available: https://arxiv.org/pdf/1911.00357 https://iclr.cc/virtual_2020/poster_H1gX8C4YPr.html https://wijmans.xyz/publication/ddppo-2019/.

[35] Eval AI, "2018 Vision and Language Navigation Evaluation," [Online]. Available: https://eval.ai/web/challenges/challenge-page/97/evaluation.

[36] G. Ilharco, V. Jain, A. Ku, E. Ie and J. Baldridge, "General Evaluation for Instruction Conditioned Navigation using Dynamic Time Warping," [Online]. Available: https://arxiv.org/abs/1907.05446.

[37] facebookresearch, "Github: Habitat-Sim," [Online]. Available: https://github.com/facebookresearch/habitat-sim.

[38] A. Szot and e. al., "Habitat 2.0: Training Home Assistants to Rearrange their Habitat," Advances in Neural Information Processing Systems (NeurIPS), 2021.

[39] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, 2016. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition. [Online]. Available: https://ieeexplore.ieee.org/document/5206848 https://www.image-net.org/static_files/papers/imagenet_cvpr09.pdf.

[41] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva and D. Batra, "DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames," International Conference on Learning Representations, 2019. [Online]. Available: https://arxiv.org/abs/1911.00357.

[42] NVIDIA, "NVIDIA Jetson Xavier," [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[43] iRobot, "Create 2 Open Interface Spec," [Online]. Available: https://edu.irobot.com/learning-library/create-2-open-interface-spec.

[44] Intel Realsense, "Intel RealSense Depth Camera D455," [Online]. Available: https://www.intelrealsense.com/depth-camera-d455/.

[45] Ricoh Imaging, "Theta V," [Online]. Available: https://us.ricoh-imaging.com/product/theta-v/.

[46] Ricoh Imaging, "THETA Z1 51GB," [Online]. Available: https://us.ricoh-imaging.com/product/theta-z1-51gb/.

[47] Hugging Face, "BERT," [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/bert.

[48] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805.

[49] A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is All you Need," Advances in Neural Information Processing Systems, 2017. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html https://arxiv.org/abs/1706.03762.

[51] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh and Dan, "Language Models

are Few-Shot Learners," Advances in Neural Information Processing Systems 33 (NeurIPS 2020). [Online]. Available: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[52] Y. Hong, Q. Wu, Y. Qi, C. Rodriguez-Opazo and S. Gould, "VLN↻BERT: A Recurrent Vision-and-Language BERT for Navigation," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [Online]. Available: https://ieeexplore.ieee.org/document/9578351 https://openaccess.thecvf.com/content/CVPR2021/papers/Hong_VLN_BERT_A_Recurrent_Vision-and-Language_BERT_for_Navigation_CVPR_2021_paper.pdf.

[53] Ricoh360, "Theta," [Online]. Available: https://www.ricoh360.com/theta/.

[54] Car and Driver, "Polestar Will Build Some Sort of Ride-Hailing Robotaxi with Waymo," [Online]. Available: https://www.caranddriver.com/news/a32988396/polestar-robotaxi-waymo-deal-announced/.

[55] J. Strader, N. Hughes, W. Chen, A. Speranzon and L. Carlone, "Indoor and Outdoor 3D Scene Graph Generation via Language-Enabled Spatial Ontologies," IEEE Robotics and Automation Letters, 2024. [Online]. Available: https://arxiv.org/pdf/2312.11713.

[56] R. Schumann and S. Riezler, "Generating Landmark Navigation Instructions from Maps as a Graph-to-Text Problem," Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1:

Long Papers), pages 489–502. Association for Computational Linguistics., 2021. [Online]. Available: https://aclanthology.org/2021.acl-long.41/.

[57] R. Schumann and S. Riezler, "Analyzing Generalization of Vision and Language Navigation to Unseen Outdoor Areas," Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)", 2022. [Online]. Available: https://aclanthology.org/2022.acl-long.518/.

[58] J. Li, A. Padmakumar, G. Sukhatme and M. Bansal, "VLN-Video: Utilizing Driving Videos for Outdoor Vision-and-Language Navigation," AAAI Conference on Artificial Intelligence. 2024. [Online]. Available: https://arxiv.org/abs/2402.03561.

[59] Open Robotics, "Gazebo Sim," [Online]. Available: https://gazebosim.org/home.

[60] Intel, "Intel® RealSense™ Depth Camera D455," [Online]. Available: https://www.intelrealsense.com/depth-camera-d455/.

[61] Modular Circuits, "Lock Anti-Phase Drive," [Online]. Available: https://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/lock-anti-phase-drive/.

[62] Modular Circuits - tantosonline, "Sign-Magnitude Drive," [Online]. Available: https://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/sign-magnitude-drive/.

[63] Intel RealSense, "librealsense NVIDIA Jetson installation," [Online]. Available: https://github.com/IntelRealSense/librealsense/blob/master/doc/installation_jetson.md.

[64] Intel, "Intel RealSense Documentation," [Online]. Available: https://dev.intelrealsense.com/docs/docs-get-started.

[65] GPS.gov, "Other Global Navigation Satellite Systems (GNSS)," [Online]. Available: https://www.gps.gov/systems/gnss/.

[66] GPS.gov, "The Global Positioning System," [Online]. Available: https://www.gps.gov/systems/gps/.

[67] Cambridge Radio Frequency Systems, "How to address GPS jamming in high-jamming environments," [Online]. Available: https://www.crfs.com/blog/how-to-address-gps-jamming-in-high-jamming-environments.

[68] H. D. BLACK, "A passive system for determining the attitude of a satellite," AIAA Volume 2, Number 7, July 1964. [Online]. Available: https://arc.aiaa.org/doi/10.2514/3.2555 https://apps.dtic.mil/sti/trecms/pdf/AD0624479.pdf.

[69] F. O. Silva, W. C. L. Filho and E. M. Hemerly, "Design of a Stationary Self-Alignment Algorithm for Strapdown Inertial Navigation Systems," IFAC-PapersOnLine, Volume 48, Issue 9, 2015, Pages 55-60, ISSN 2405-8963. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240589631500926X.

[70] S. Wang, G. Yang and L. Wang, "Latitude Determination and Error Analysis for Stationary SINS in Unknow-Position Condition," Sensors (Basel). 2020 May, 20(9): 2558. [Online]. Available: https://www.mdpi.com/1424-8220/20/9/2558.

[71] Britannica, "Solar Compass," [Online]. Available: https://www.britannica.com/technology/magnetic-compass.

[72] The Open University, "DIY: Measuring latitude and longitude," 1 May 2022. [Online]. Available: https://www.open.edu/openlearn/society/politics-policy-people/geography/diy-measuring-latitude-and-longitude#:~:text=Time%20the%20difference%20between%20local,re%20on%20a%20western%20longitude..

[73] M. Izadmehr and M. K. Ghomi, "An accuracy of better than 200 m in positioning using a portable star tracker," New Astronomy, Volume 74, 2020, 101272, ISSN 1384-1076. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1384107618303713.

[74] T. Tuna, J. Nubert, Y. Nava, S. Khattak and M. Hutter, "X-ICP: Localizability-Aware LiDAR Registration for Robust Localization in Extreme Environments," IEEE Transactions on Robotics, Volume 40, [Online]. Available: https://ieeexplore.ieee.org/document/10328716 https://www.research-collection.ethz.ch/handle/20.500.11850/641538 https://www.turcantuna.com/publication/TRO/.

[75] MATLAB, "What Is SLAM?," [Online]. Available: https://www.mathworks.com/discovery/slam.html.

[76] G. Grisetti, C. Stachniss and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," In Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2005. [Online]. Available: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti05icra.pdf.

[77] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," IEEE Transactions on Robotics, Volume 23, pages 34-46, 2007. [Online]. Available: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf.

[78] D. B. Rubin, "The Calculation of Posterior Distributions by Data Augmentation: Comment: A Noniterative Sampling/Importance Resampling Alternative to the Data Augmentation Algorithm for Creating a Few Imputations When Fractions of Missing Information Are Modest: The SIR," Journal of the American Statistical Association Vol. 82, No. 398, pp. 543-546 (4 pages), June, 1987. [Online]. Available: https://www.jstor.org/stable/2289460.

[79] S. Thrun and M. Montemerlo, "The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," Stanford AI Lab, The International Journal of Robotics Research, Vol. 25, No. 5–6, May–June 2006, pp. 403-429. [Online]. Available: http://robots.stanford.edu/papers/thrun.graphslam.pdf.

[80] gtsam.org, "Factor Graphs and GTSAM," [Online]. Available: https://gtsam.org/tutorials/intro.html.

[81] ROS Wiki, "costmap_2d," [Online]. Available: https://wiki.ros.org/costmap_2d.

[82] CMU Howie Choset, "Robotic Motion Planning: Potential Functions," [Online]. Available: https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf.

[83] F. Zhang, "YIP: Generic Environment Models (GEMs) for Agile Marine Autonomy," 2015. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA557086.

[84] CMU: Geoff Gordon, "Planning With RRTs," [Online]. Available: https://www.cs.cmu.edu/~ggordon/780/slides/16-probability.pdf.

[85] Colombia University: Paul Blaer, "Robot Path Planning Using Generalized Voronoi Diagrams," [Online]. Available: https://www.cs.columbia.edu/~pblaer/projects/path_planner/.

[86] S. Quinlan and O. Khatib, "Elastic Bands: Connecting Path Planning and Control," Proceedings IEEE International Conference on Robotics and Automation, Atlanta, GA, USA, 1993, pp. 802-807 vol.2, doi: 10.1109/ROBOT.1993.291936., 1993. [Online]. Available: https://ieeexplore.ieee.org/document/291936 https://khatib.stanford.edu/publications/pdfs/Quinlan_1993_ICRA.pdf.

[87] A. Boeing, S. Pangeni, T. Bräunl and C. S. Lee, "Real-time tactical motion planning and obstacle avoidance for multi-robot cooperative reconnaissance," IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, Korea (South), 2012, pp. 3117-3122, doi: 10.1109/ICSMC.2012.6378270, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/6378270.

[88] A. Boeing, "Elastic Band - Realtime pathfinding deformation," [Online]. Available: http://adrianboeing.blogspot.com/2012/03/elastic-band-realtime-pathfinding.html.

[89] C. Rosmann, F. Hoffmann and T. Bertram, "Timed-Elastic-Bands for Time-Optimal Point-to-Point Nonlinear Model Predictive Control," 2015 European Control Conference (ECC), July 15-17, 2015. Linz, Austria. [Online]. Available: https://rst.etit.tu-dortmund.de/storages/rst-etit/r/Global/Paper/Roesmann/2015_Roesmann_ECC.PDF.

[90] S. Koenig, M. Likhachev and D. Furcy, "Lifelong Planning A∗," Artificial Intelligence, Volume 155, Issues 1–2, Pages 93-146, ISSN 0004-3702, 2004. [Online]. Available: https://doi.org/10.1016/j.artint.2003.12.001.

[91] S. Koenig and M. Likhachev, "D* Lite," 2002. AAAI-02 Proceedings. [Online]. Available: https://cdn.aaai.org/AAAI/2002/AAAI02-072.pdf.

[92] S. Koenig and M. Likhachev, "Fast Replanning for Navigation in Unknown Terrain," Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292), Washington, DC, USA, 2002, pp. 968-975 vol.1, doi: 10.1109/ROBOT.2002.1013481., [Online]. Available: https://www.cs.cmu.edu/~maxim/files/dlite_tro05.pdf https://ieeexplore.ieee.org/document/1013481.

[93] K. Daniel, A. Nash, S. Koenig and A. Felner, "Theta*: Any-Angle Path Planning on Grids," Journal of Artificial Intelligence Research 39 (2010) 533-579, 2010. [Online]. Available: https://arxiv.org/pdf/1401.3843.

[94] Lockheed Martin, "Lunar Mobility Vehicle," [Online]. Available: https://www.lockheedmartin.com/en-us/products/lunar-mobility-vehicle.html.

[95] Lockheed Martin, "Lunar Dawn Team Awarded NASA Lunar Terrain Vehicle Contract," [Online]. Available: https://news.lockheedmartin.com/2024-04-03-Lunar-Dawn-Team-Awarded-NASA-Lunar-Terrain-Vehicle-Contract.

[96] NASA, "Lunar Terrain Vehicle," [Online]. Available: https://www.nasa.gov/extravehicular-activity-and-human-surface-mobility/lunar-terrain-vehicle/.

[97] NASA, "NASA Moves Forward with Development of LTV," 30 April 2024. [Online]. Available: https://appel.nasa.gov/2024/04/30/nasa-moves-forward-with-development-of-ltv/.