

**Autonomous Navigation and Simultaneous 3D Mapping with a UGV in an  
Open-Ended Environment**

by

Ragib Rownak  
B.S., Islamic University of Technology, 2022

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Mechanical Engineering  
in the Graduate College of the  
University of Illinois at Chicago, 2025

Chicago, Illinois

Defense Committee:

Prof. Pranav Bhounsule, Chair and Advisor

Prof. Jonathan Komperda

Prof. Ahmet Enis Cetin, Electrical and Computer Engineering

Copyright by  
Ragib Rownak  
2025

To my parents, MD Anisur Rahman Mondal and Farida Yeasmin. Thank you.

## ACKNOWLEDGMENT

I am grateful to my advisor, Professor Pranav Bhounsule, whose unwavering guidance and mentorship have been instrumental throughout my Master’s journey. Your wisdom, patience, and steadfast support have shaped not only my academic pursuits but also my professional growth in ways that will resonate throughout my career.

My heartfelt appreciation to all my cherished lab mates—both past and present—Prasaanth, Abhisekh, Subramanian, Salvador, Daniel, Chun-Ming, Jim, Safwan, Luca, Simone, and Suleiman: your friendship, intellectual exchanges, and unwavering camaraderie have enriched this journey immeasurably. The countless hours of discussion, shared challenges, and mutual support have made even the most demanding moments meaningful and memorable.

Finally, my deepest gratitude belongs to my family, whose unconditional love, understanding, and sacrifices have made this achievement possible. Your faith in my dreams and your patience during the most challenging times have been my greatest source of strength. Thank you all!

Ragib Rownak

## TABLE OF CONTENTS

<u>CHAPTER</u>		<u>PAGE</u>
<b>1</b>	<b>INTRODUCTION . . . . .</b>	<b>1</b>
1.1	Unmanned Ground Vehicles (UGVs) . . . . .	1
1.2	Thesis Contribution . . . . .	3
<b>2</b>	<b>BACKGROUND AND RELATED WORK . . . . .</b>	<b>5</b>
2.1	Autonomous Ground Vehicle Platforms . . . . .	5
2.1.1	Clearpath A200 Husky Platform . . . . .	6
2.2	Deep Learning and Computer Vision for Autonomous Navigation . . . . .	10
2.2.1	Convolutional Neural Networks (CNNs) for Object Detection . . . . .	11
2.2.2	Detectron2 Framework and Instance Segmentation . . . . .	12
2.2.3	Panoptic Feature Pyramid Network (FPN) . . . . .	13
2.3	ROS . . . . .	14
2.3.1	ROS 1 vs ROS 2 Evolution . . . . .	16
2.4	SLAM . . . . .	17
2.4.1	SLAM Toolbox Integration . . . . .	18
2.4.2	3D SLAM and RTABMap Implementation . . . . .	21
2.5	Nav2 Motion Planning . . . . .	25
2.5.1	Nav2 Architecture and System Structure . . . . .	25
2.5.2	Global Path Planning . . . . .	27
2.5.3	Local Trajectory Planning and Control . . . . .	28
2.5.4	Costmap Generation and Multi-Sensor Integration . . . . .	31
2.6	Intel Realsense Camera . . . . .	33
2.7	SICK LMS Lidar . . . . .	35
<b>3</b>	<b>DEEP LEARNING TRAINING AND INTEGRATION WITH ROS . . . . .</b>	<b>37</b>
3.1	Detectron2 . . . . .	37
3.1.1	Neural Network Architectures in Detectron2 . . . . .	37
3.1.2	Feature Pyramid Network Architecture . . . . .	39
3.1.3	Installation and System Requirements . . . . .	41
3.2	Dataset Structure and Training . . . . .	42
3.2.1	Dataset Architecture and Annotation Frameworks . . . . .	43
3.2.2	Sequential Training Strategy and Implementation . . . . .	45
3.2.3	Training Implementation and Computational Requirements . . . . .	50
3.3	Validation and Optimization . . . . .	51
3.3.1	Model Architecture Optimization Strategies . . . . .	53

## TABLE OF CONTENTS (Continued)

<u>CHAPTER</u>		<u>PAGE</u>
	3.3.2 Adaptive Performance Management . . . . .	56
	3.3.3 Thermal and Power Management . . . . .	58
	3.4 Integration with ROS for Autonomous Navigation . . . . .	59
	3.4.1 Vision-to-Laser Scan Conversion Methodology . . . . .	60
	3.4.2 Sensor Data Integration and Processing . . . . .	71
	3.4.3 Transform Frame Management and Coordinate Systems . . .	72
	3.4.4 Navigation Framework Integration Strategy . . . . .	73
<b>4</b>	<b>SETUP AND METHOLDODOLOGY . . . . .</b>	<b>78</b>
	4.1 Simulation Setup . . . . .	78
	4.1.1 Modifying the robot.yaml file . . . . .	82
	4.1.2 Installing Clearpath Simulator . . . . .	84
	4.1.3 Simulation Execution . . . . .	85
	4.1.4 SLAM Implementation and Integration . . . . .	87
	4.1.5 Nav2 Implementation and Integration . . . . .	88
	4.1.6 RTAB-Map Integration for 3D SLAM . . . . .	88
	4.1.7 Integrated Launch System Architecture . . . . .	93
	4.1.8 GPS-Based Navigation Integration with Nav2 . . . . .	96
	4.2 Hardware Setup and Implementation . . . . .	99
	4.2.1 Multi-Host Distributed Computing Architecture . . . . .	99
	4.2.2 Robot.yaml Multi-Host Configuration and Data Transport .	101
	4.2.3 Hardware Pipeline Data Flow and Processing . . . . .	103
	4.2.4 Remote Access and Computer Connections . . . . .	106
	4.2.5 Clearpath-SLAM and Nav2 Hardware Deployment . . . . .	107
	4.2.6 Modified Vision + Lidar Based Launch System Coordination and Startup Sequence . . . . .	108
	4.2.7 Off-Board Visualization and Remote Monitoring . . . . .	111
	4.2.8 System Integration Validation and Performance Monitoring .	112
<b>5</b>	<b>RESULTS . . . . .</b>	<b>116</b>
	5.1 Panoptic Segmentation Class-Specific Performance Results .	116
	5.1.1 Cityscapes Dataset Class Performance Results . . . . .	116
	5.1.2 COCO Dataset Relevant Class Performance Results . . . . .	116
	5.1.3 Detection Performance Analysis Summary . . . . .	121
	5.2 Navigation Results . . . . .	123
	5.2.1 Simulation Results . . . . .	124
	5.2.2 Hardware Results . . . . .	126
	5.2.3 Navigation Speed vs Detection Performance . . . . .	131
	5.3 Mapping Results . . . . .	134
	5.3.1 2D Mapping using Clearpath 2D SLAM . . . . .	135
	5.3.2 3D Mapping using RTAB-Map . . . . .	137
	5.4 Discussion . . . . .	140

## TABLE OF CONTENTS (Continued)

<u>CHAPTER</u>		<u>PAGE</u>
<b>6</b>	<b>CONCLUSION AND FUTURE WORK . . . . .</b>	<b>149</b>
6.1	Remarks . . . . .	150
6.2	Future Work . . . . .	150
	<b>REFERENCES . . . . .</b>	<b>153</b>
	<b>VITA . . . . .</b>	<b>160</b>

## LIST OF TABLES

<b><u>TABLE</u></b>		<b><u>PAGE</u></b>
I	Key Packages and Their Roles for <code>ros2 launch clearpath_gz simulation.launch.py</code> . . . . .	81
II	Cityscapes Dataset Class-Specific Performance Results . . . . .	117
III	COCO Dataset Navigation-Relevant Class Performance Results - Part 1: Active Detection Classes . . . . .	118
IV	Performance Improvements with Sequential Training . . . . .	121
V	Performance Comparison Across Deployment Pipeline . . . . .	123
VI	Hardware Validation Results Between LiDAR-Only vs Vision+LiDAR Navigation Systems . . . . .	129
VII	Vision-Based System Comparison with Traditional 2D LiDAR . . .	131



## LIST OF FIGURES

<b>FIGURE</b>		<b>PAGE</b>
1	Unmanned Ground Vehicles overview [1] . . . . .	6
2	UIC02 HUSKY . . . . .	8
3	Structure of Convolutional Neural Networks (CNNs) . . . . .	11
4	Overview of the SLAM system [2] . . . . .	18
5	Real-Time Appearance-Based Mapping Front and Back End [3] . . . . .	23
6	Nav2 ROS2 Structure . . . . .	26
7	Global and Local Planners in Action . . . . .	29
8	Intel RealSense RGB-D camera [4] . . . . .	33
9	SICK LMS111 2D LiDAR sensor . . . . .	35
10	Detectron2 Panoptic Segmentation . . . . .	38
11	A Robust Pipeline for Panoptic Bottom-Up Segmentation [5] . . . . .	40
12	Systematic directory structure of COCO dataset . . . . .	44
13	Laser Scan Generation Pipeline . . . . .	65
14	Costmap generation from both the Lidar and Camera Scan . . . . .	74
15	Modifying the robot.yaml to match the real robot . . . . .	83
16	6 worlds that were used to do simulated experiments. . . . .	86
17	Launching Clearpath_SLAM . . . . .	88
18	Terminal Commands to install rtabmap . . . . .	89
19	RTAB-Map_RViz . . . . .	91
20	Viewing the Saved RTAB-map_database . . . . .	92
21	Launching Detectron2 . . . . .	94
22	Complete Pipeline of the Integrated Launch System . . . . .	95
23	Autonomous Navigation Using Panoptic Segmentation and Nav2, where the panoptic segmentation is transferring both thing and stuff class data to Nav2 through Laser Scan data, which is then used to generate the Local and Global Costmaps. . . . .	96
24	Terminal Commands for launching the GPS-based Navigation . . . . .	98
25	3D-printing the Camera Mount . . . . .	104
26	Complete Pipeline of the Modified Vision + Lidar Based Launch System	110
27	Rviz Visualization . . . . .	111
28	GPS-Based Navigation Launch Commands . . . . .	115
29	Panoptic Segmentation Performance on both Simulation and Real World	122
30	Costmap Generation from the Laser Scan Data Coming from the Panoptic Data . . . . .	130
31	Trajectory Path Comparison between Odometry(IMU) and Ground Truth as GPS to show the Robot Path Planning Efficiency in Simulation . . .	132
32	Trajectory Path Comparison between Odometry(IMU) and Ground Truth as GPS to show the Robot Path Planning Efficiency in the Real World.	133

LIST OF FIGURES (Continued)

<u>FIGURE</u>		<u>PAGE</u>
33	Detection Performance vs Variable Navigation Speed . . . . .	134
34	2D Map of the UIC Quad . . . . .	136
35	3D map of the UIC Quad . . . . .	138

## SUMMARY

The focus of this research is the development of an advanced autonomous navigation system that integrates Detectron2’s Panoptic FPN model with multi-sensor fusion and simultaneous 3D mapping for the Clearpath Husky A200 UGV operating in complex environments. By combining panoptic segmentation for comprehensive scene understanding with ROS Navigation Stack and RTABMap for real-time 3D mapping, a robust perception, mapping, and path planning framework is developed that enables simultaneous localization, mapping, and obstacle avoidance capabilities. The system utilizes sensor fusion between LiDAR and camera data, where both instance objects and stuff/background classes detected by the Panoptic FPN model are classified and converted to laser scan format for seamless integration with the navigation costmap while concurrently building detailed 3D environmental maps. Experimental validation in both simulation and real-world environments demonstrates the effectiveness of the proposed panoptic segmentation-based system with concurrent 3D mapping in achieving collision-free navigation through complex scenarios. The framework successfully integrates Detectron2’s comprehensive scene understanding capabilities with traditional LiDAR sensing and RTABMap’s SLAM functionality to create a robust perception and mapping system operating at real-time frequencies, showing significant improvements in obstacle detection accuracy, navigation reliability, and environmental understanding compared to traditional approaches. This work contributes to autonomous robotics by providing a scalable framework that lever-

## SUMMARY (Continued)

ages complete scene understanding and 3D mapping for intelligent navigation in unstructured environments, with applications spanning industrial automation to service robotics.

## CHAPTER 1

### INTRODUCTION

#### 1.1 Unmanned Ground Vehicles (UGVs)

The field of autonomous robotics has witnessed remarkable evolution in ground-based mobile platforms, transforming from simple remote-controlled vehicles to sophisticated autonomous systems capable of complex decision-making and environmental interaction [6]. Among the most compelling developments in this domain is the emergence of Unmanned Ground Vehicles (UGVs), which have revolutionized applications ranging from military reconnaissance to civilian service robotics [7]. With their robust mobility and advanced sensing capabilities, these remarkable machines possess an extraordinary ability to navigate and operate in environments that are challenging, dangerous, or inaccessible to humans. Take, for instance, the remarkable coordination between multiple UGVs in search and rescue operations [8], enabling them to efficiently cover vast areas and tackle complex mapping tasks. The endless potential that autonomous ground vehicles have to push innovation in robotics and other fields is just astounding.

Manufacturing procedures and sensor technologies have advanced to a remarkable degree of precision in the exciting new era, producing exceptionally complex mobile platforms [9]. Simultaneously the development of computer vision and artificial intelligence is paving the way for the development of fully autonomous navigation systems. Furthermore, the ongoing evolution of communication technologies and cloud computing simplifies the coordination be-

tween multiple vehicles, making fleet operations more seamless and efficient [10]. Considering all of this, it is clear that research on autonomous ground vehicles is progressing remarkably. Known as "UGVs," these adaptable robotic systems have recently shown promise in a variety of applications, offering intriguing challenges to tackle a range of global problems.

Imagine rugged UGVs with advanced sensors patrolling agricultural fields to monitor crop health and optimize irrigation systems [11]. Likewise, compact inspection UGVs could maneuver through industrial facilities, navigating narrow corridors and hazardous areas to detect equipment malfunctions or safety violations. In disaster scenarios, all-terrain UGVs could assist in emergency response efforts by mapping damaged infrastructure and delivering critical supplies to inaccessible locations [12]. In urban environments, delivery UGVs could revolutionize last-mile logistics [13], while security patrol UGVs monitor large facilities and campuses, providing continuous surveillance and rapid response capabilities.

These incredible platforms are classified according to their mobility systems and operational environments: indoor navigation, outdoor terrain traversal, or hybrid capabilities that span both domains [14]. For the purpose of this thesis, we will focus on outdoor-capable platforms with advanced navigation systems. In terms of locomotion, UGVs fall primarily into several categories: wheeled, tracked, and hybrid systems. A notable example of wheeled UGVs is the Clearpath Husky A200 platform [15], which combines robust four-wheel drive capability with sophisticated sensor integration, enabling precise autonomous navigation in diverse terrains and weather conditions.

## 1.2 Thesis Contribution

This thesis explores the integration of panoptic segmentation with SLAM-based autonomous navigation to address critical limitations in outdoor robotic navigation where traditional 2D mapping and obstacle detection methods fail. The primary contribution lies in developing a comprehensive perception and navigation framework that combines computer vision-based obstacle detection with simultaneous localization and mapping (SLAM) capabilities for robust outdoor autonomous navigation [16].

Detectron2’s most innovative Panoptic FPN model is utilized to perform both instance and stuff segmentation, enabling the robot to detect not only discrete objects but also critical terrain features such as grass, earth, and other surfaces that conventional LiDAR sensors cannot reliably identify due to poor laser reflection properties. This panoptic approach provides a more complete environmental understanding by capturing both ”things” (countable objects like people, vehicles, tools) and ”stuff” (uncountable regions like terrain, vegetation, surfaces) that are essential for safe navigation in unstructured outdoor environments.

The segmentation output is strategically converted into laser scan data format to seamlessly integrate with the Nav2 navigation stack’s costmap generation system [17]. This ensures that areas identified as obstacles or non-traversable terrain are properly marked in the robot’s planning algorithms, preventing the robot from attempting to navigate over unsuitable surfaces that LiDAR alone might classify as free space.

A critical advancement of this work is the implementation of mapless navigation combined with real-time 3D mapping using RTAB-Map SLAM [3]. Unlike traditional 2D SLAM ap-

proaches that struggle with localization in outdoor environments when relying solely on LiDAR and IMU sensors, this framework leverages the fusion of Intel RealSense camera data with SICK LMS 2D LiDAR to achieve robust localization and mapping. The 3D mapping capability not only provides richer environmental representation but also significantly improves the robot’s ability to localize itself in GPS-denied or challenging outdoor scenarios where visual and geometric features are essential for accurate pose estimation.

This integrated approach addresses the fundamental challenge of autonomous outdoor navigation, where environmental complexity, varied terrain types, and the limitations of individual sensors create significant obstacles to reliable robot operation. The system’s ability to simultaneously build detailed 3D maps while navigating using enhanced perception capabilities makes it particularly valuable for applications in unstructured environments such as agricultural fields, construction sites, search and rescue operations, and outdoor inspection tasks where prebuilt maps are unavailable and terrain variability is high.

The framework’s robustness in handling diverse outdoor conditions, combined with its mapless operation capability, opens new possibilities for deploying autonomous robots in previously challenging scenarios where traditional navigation approaches would fail, thereby expanding the practical applications of mobile robotics in real-world outdoor environments.



## CHAPTER 2

### BACKGROUND AND RELATED WORK

#### 2.1 Autonomous Ground Vehicle Platforms

Autonomous ground vehicles have received substantial attention from researchers in last few years due to their customizability and versatility in diverse operational environments [14]. These platforms are more adept at traversing complex terrains and executing autonomous navigation in unpredictable and unstructured environments compared to traditional remote-controlled systems [6]. Recently, these type of robots have showed significant potential for real-world practical applications in a variety of sectors such as construction surveillance, education, delivery services, and search and rescue operations [11].

The evolution of mobile robotics platforms has been driven by the convergence of several technological advances: advanced sensor technologies, improved computational capabilities, and sophisticated software frameworks [10]. Modern autonomous ground vehicles must balance mechanical robustness, computational efficiency, and sensor integration to achieve reliable autonomous operation across diverse environments [7].

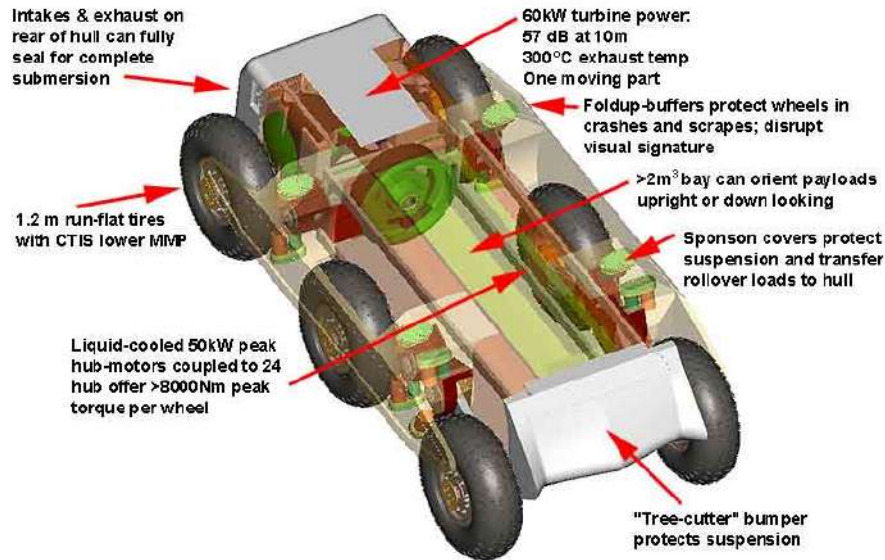


Figure 1: Unmanned Vehicles [1]

### 2.1.1 Clearpath A200 Husky Platform

The Clearpath A200 Husky represents a state-of-the-art research platform specifically designed for autonomous ground vehicle applications [15]. This platform features a four-wheel differential drive configuration with enhanced robustness for outdoor applications, making it particularly suitable for real-world autonomous navigation research [17]. Unlike laboratory-based platforms that are confined to controlled environments, the Husky bridges the gap between academic research and field deployable systems [9], offering researchers a reliable foundation for developing and testing autonomous algorithms in challenging real-world conditions [18].

The Clearpath Husky A200 represents a remarkable achievement in UGV design, featuring a rugged aluminum chassis capable of carrying substantial payloads while maintaining exceptional maneuverability [15]. This platform, weighing approximately 50 kg with a payload capacity of 75 kg, harnesses advanced sensor fusion techniques to achieve remarkable autonomous navigation capabilities [19]. The Husky's modular design allows for extensive customization with various sensors, including LiDAR, cameras, GPS, and IMU systems, creating a versatile foundation for research and commercial applications [20]. These UGVs overcome the limitations of conventional indoor-only robotic platforms by combining strong mechanical design with advanced perception systems, which allow them to navigate challenging outdoor environments with remarkable reliability.

The platform has gained widespread adoption in academic institutions worldwide due to its proven reliability and comprehensive integration capabilities [21]. Universities and research centers have successfully deployed Husky platforms for diverse applications ranging from Mars analog studies in Utah's desert environments to tactical operations research [12], demonstrating its versatility across different operational contexts. This broad acceptance stems from the platform's ability to maintain consistent performance while supporting complex sensor suites and computational payloads required for advanced research applications [22].

### **Platform Specifications and Design**

The A200 Husky model features a four-wheel differential drive configuration, theoretically similar to other research platforms, but with enhanced robustness for outdoor applications [15]. The platform's mechanical design incorporates robust gear motors with integrated wheel

## UIC02 HUSKY

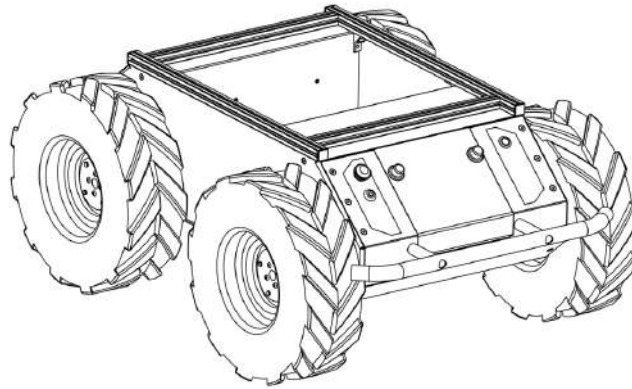


Figure 2: UIC02 HUSKY

encoders, providing reliable locomotion across diverse terrain conditions [14]. Specifically, the chassis includes a comprehensive power management architecture with multiple voltage rails (VBat: 24-29V, fused at 5A; 24V, fused at 3A; 12V, fused at 7.5A) that, for the same power budget used, maximizes the capability to support heterogeneous payload configurations [23].

The drivetrain utilizes a zero-maintenance belt system connecting brushed DC gear motors to 330mm lug-tread wheels, delivering continuous 400W of drive power distributed as 100W per wheel [15]. This configuration eliminates traditional steering mechanisms in favor of differential wheel speeds for directional control, enabling precise maneuverability, including zero-radius turning capabilities [18]. The mechanical specifications center on rugged outdoor operation

with overall dimensions of  $990 \times 670 \times 390$ mm, weighing 57.69kg in base configuration, while supporting maximum payloads of 20kg through standardized mounting interfaces [9].

The platform needs to be analyzed at two separate levels: the low level, which consists of the sensor data acquisition and actuator control; and the high level, composed of the navigation and decision-making algorithms [19]. Each level has unique characteristics and behaviors, which must be considered for effective system integration [7]. The low-level control operates through a 32-bit microcontroller that manages deterministic motor control, power distribution, and sensor acquisition with sub-10ms command latency [24], while the high-level processing handles complex autonomous behaviors and sensor fusion algorithms [25].

### **Computational Architecture**

The platform’s computational architecture typically centers around either Mini-ITX single-board computers or NVIDIA Jetson modules, both supporting ROS 2 Humble with real-time processing capabilities [10]. The overall performance resembles a scalable system and, for configurations close to optimal resource allocation, achieves efficient operation. Mini-ITX systems utilize Intel x86/amd64 processors with 4-32GB RAM configurations, providing the computational foundation for complex autonomous algorithms and sensor processing tasks [26].

NVIDIA Jetson integration represents a significant advancement in onboard AI processing capabilities, spanning from Jetson Nano (472 GOPS) through Jetson Orin AGX (275 TOPS), depending on research requirements [27]. This distributed computational architecture enables real-time processing of multiple sensor streams while maintaining deterministic control of critical safety functions [28]. The behavior of the platform’s computational performance depends on the

specific processing requirements of integrated algorithms, with basic operations demonstrating linear behavior under normal conditions [17].

While basic operation demonstrates linear behavior under normal operating conditions, the system becomes more complex when advanced sensor payloads are integrated, and the overall performance becomes a combination of mechanical and computational capabilities [29]. The integration of multiple high-bandwidth sensors such as 3D LiDAR, cameras, and navigation sensors requires careful consideration of computational resource allocation and data flow management [20]. Both mechanical and computational subsystems have similar operational constraints, exhibiting linear behavior under normal operating conditions and high non-linearity when approaching system limits [6].

## **2.2 Deep Learning and Computer Vision for Autonomous Navigation**

Deep Learning and Computer Vision for Autonomous Navigation represents a transformative paradigm shift in robotics perception systems, enabling robots to understand their environment with unprecedented semantic richness beyond traditional geometric sensing approaches. Modern autonomous navigation has evolved from purely reactive obstacle avoidance to intelligent scene understanding, where deep learning models can distinguish between different object categories, terrain types, and navigational contexts. Computer vision techniques, particularly panoptic segmentation, provide comprehensive scene parsing that combines instance-level object detection with semantic segmentation to create detailed environmental understanding crucial for safe and efficient autonomous navigation. This integration of artificial intelligence with robotics perception systems enables autonomous vehicles to make informed decisions based

# Convolutional Neural Network

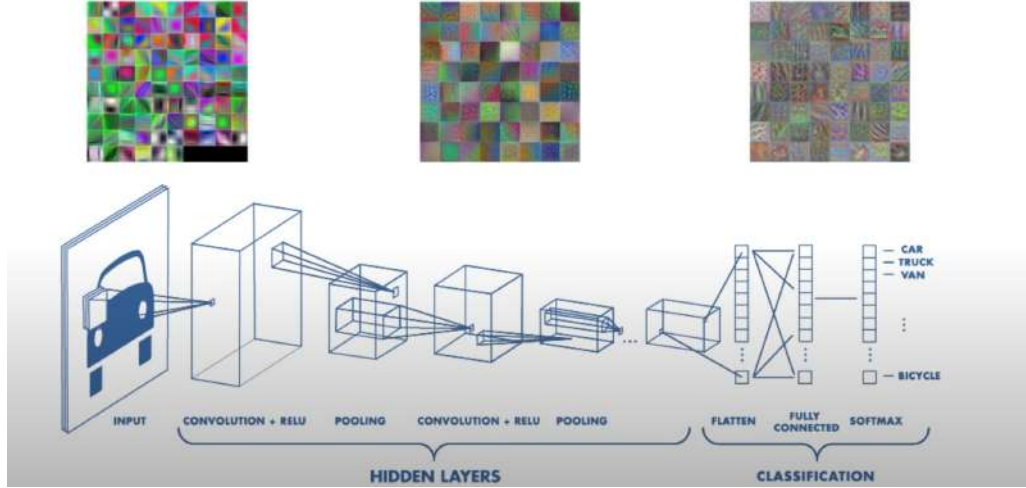


Figure 3: Structure of Convolutional Neural Networks (CNNs)

not only on geometric constraints but also on semantic meaning, leading to more robust and intelligent navigation behaviors in complex real-world environments.

## 2.2.1 Convolutional Neural Networks (CNNs) for Object Detection

Convolutional Neural Networks have revolutionized computer vision tasks, particularly in object detection and scene understanding for autonomous navigation systems [30]. CNNs leverage hierarchical feature learning through convolutional layers, pooling operations, and fully connected layers to extract meaningful representations from visual data [31]. In autonomous ground vehicle navigation, CNNs enable real-time obstacle detection, semantic segmentation, and scene understanding that complement traditional geometric sensors [32].

The evolution of CNN architectures from AlexNet [33] to ResNet [34] and beyond has significantly improved detection accuracy and processing speed. Modern CNN frameworks like YOLO (You Only Look Once) [35] and R-CNN families [36] provide different trade-offs between speed and accuracy. YOLO architectures prioritize real-time performance with single-pass detection, making them suitable for autonomous navigation where low latency is critical [35]. The YOLOv7 architecture [37], demonstrates superior performance in both speed and accuracy compared to earlier versions, achieving approximately 65 ms processing times while maintaining high detection accuracy.

### **2.2.2 Detectron2 Framework and Instance Segmentation**

Detectron2 represents Facebook AI Research’s next-generation object detection and segmentation platform, built on PyTorch [38] with modular design principles that enable flexible model configuration and deployment [28]. The framework is particularly helpful in achieving thorough scene understanding in autonomous navigation since it supports various computer vision tasks, such as detection of objects, instance segmentation, semantic segmentation, and panoptic segmentation [29].

In addition to the classification of branches and also bounding box regression, Mask R-CNN [39] expands the Faster R-CNN architecture [40] by including a branch for segmentation mask prediction on each Region of Interest (RoI). This architecture enables pixel-precise instance segmentation, allowing autonomous vehicles to distinguish between individual objects of the same class [39]. The two-stage detection process first generates object proposals through a



Region Proposal Network (RPN) [40], then refines these proposals, and generates masks through the second stage.

For autonomous navigation, Mask R-CNN provides several advantages: precise object boundaries for accurate obstacle avoidance, differentiation between multiple instances of the same object class (e.g., multiple pedestrians), and detailed shape information for path planning around complex obstacles [39]. The instance-level understanding enables more sophisticated navigation behaviors, such as predicting pedestrian movement patterns or identifying passable gaps between vehicles.

### **2.2.3 Panoptic Feature Pyramid Network (FPN)**

Panoptic segmentation combines both the distinct tasks of semantic segmentation and instance segmentation [29]. So the Panoptic Feature Pyramid Network represents a significant advancement in scene understanding by providing both semantic and instance segmentation capabilities within a single unified framework [29].

### **Unified Scene Understanding**

The panoptic segmentation approach provides both thing classification (individual object instances) and stuff classification (background regions), offering complete scene understanding essential for semantic-enhanced navigation [29]. This comprehensive scene understanding enables autonomous ground vehicles to make informed decisions about navigable areas (roads, sidewalks) versus obstacles (buildings, vehicles, pedestrians) while simultaneously tracking individual object instances for precise collision avoidance [41].

The Panoptic FPN architecture combines the strengths of both semantic and instance segmentation by sharing a common backbone network and Feature Pyramid Network [42], then branching into separate heads for semantic and instance predictions. A fusion module reconciles conflicts between the two predictions, ensuring each pixel is assigned to exactly one semantic class and at most one instance class which provides holistic scene interpretation that informs higher-level navigation decisions [29].

The semantic-to-geometric conversion process involves projecting detected object instances and semantic regions onto a 2D occupancy representation [43]. This process considers object permanence, navigational relevance, and temporal consistency to generate reliable obstacle representations for path-planning algorithms [17].

### **2.3    ROS**

The Robot Operating System (ROS) constitutes an open-source, meta-operating system framework specifically designed for the development of robotic applications [44]. The system furnishes essential operating system services, encompassing hardware abstraction layers, low-level device control mechanisms, inter-process message-passing protocols, and comprehensive package management utilities [21]. Additionally, ROS incorporates sophisticated tools and libraries that facilitate code development, compilation, and execution across distributed computing environments. As ROS functions as a middleware layer rather than a standalone operating system, its deployment necessitates installation upon an existing operating system platform, such as Ubuntu or alternative Linux distributions.

The Robot Operating System has emerged as the dominant framework for robotics development, fundamentally transforming how robotics software is designed, developed, and deployed across research and industry [10]. ROS functions provide essential services while remaining hardware-agnostic and facilitating distributed computing [44]. The fundamental design principles include maintaining a thin framework that doesn't constrain application architecture, supporting language-independent libraries with clean functional interfaces, and enabling code reuse through a distributed framework of loosely coupled processes [21].

- **Nodes:** ROS nodes are fundamental computational units, each responsible for specific tasks such as device management or algorithm execution. These nodes communicate through topics and services, with software organized into packages that typically focus on single-task functionality and may contain multiple interconnected nodes.
- **Messages:** Messages are structured data containers enabling inter-node communication. The message specification framework supports diverse data types including integers, floating-point numbers, and arrays, facilitating transmission of sensor data, control commands, and system state information.
- **Topic Communication:** Topics establish named data streams for information exchange between nodes, primarily used for continuous message transmission such as sensor measurements or motor control signals. Each topic maintains a unique identifier and predefined message format, allowing nodes to publish or subscribe without restrictions on the number of participating nodes, though individual nodes cannot simultaneously publish and subscribe to the same topic.

- **Service Interface:** Services provide synchronous request-response communication mechanisms enabling direct node-to-node interaction. Each service defines paired message structures for requests and responses, making them ideal for operations requiring immediate feedback and confirmation.

### 2.3.1 ROS 1 vs ROS 2 Evolution

#### ROS 1

ROS 1 employs a master-centric architecture built around a central coordination node called the ROS Master (roscore) that serves as a registry and discovery service, maintaining information about all active nodes, topics, and services in the system [44]. Communication occurs through custom protocols—primarily TCPROS for reliable TCP-based messaging and UDPROS for low-latency UDP communication. However, this centralized master architecture creates single points of failure, lacks real-time support, and has security vulnerabilities from unencrypted communication [10].

#### ROS 2

ROS 2 represents a fundamental architectural redesign that eliminates the centralized master concept in favor of a fully distributed system built on the Data Distribution Service (DDS) middleware standard [10]. ROS 2 enables nodes to discover each other autonomously through multicast discovery protocols while introducing sophisticated Quality of Service (QoS) policies for fine-grained control over communication reliability, durability, and timing constraints [45]. The enhanced architecture introduces actions for long-running tasks, node lifecycle management, and domain isolation through DDS domain IDs. The most significant advancement

lies in ROS 2's middleware abstraction layer (RMW) built upon DDS standards, leveraging standardized Real-Time Publish Subscribe (RTPS) protocols through multiple DDS implementations including Fast DDS, Cyclone DDS, and RTI Connext [10].

Performance benchmarks demonstrate that ROS 2 can achieve sub-millisecond latency while supporting 1000+ robot systems compared to ROS 1's practical limit of approximately 100 robots [10]. ROS 2 provides native real-time support, enterprise-grade security through DDS-Security specification with PKI-based authentication, and native multi-platform support including Windows 10/11, macOS, and embedded systems. ROS development began in 2005 with ROS 1 reaching end-of-life in May 2025, while ROS 2 development commenced in 2014, with growing adoption showing 39.82% of package downloads in 2022 [10]. The migration leverages the `ros1_bridge` for gradual transition, with approximately 70% of ROS 1 packages already ported to ROS 2, establishing ROS 2 as a foundational technology for next-generation cyber-physical systems.

Navigation by mobile robots commonly requires solutions to three different problems: mapping, localization, and route planning [6]. We will discuss how these key problems will be solved in the later slides.

## **2.4   SLAM**

SLAM stands for Simultaneous Localization and Mapping, which is a fundamental algorithm that allows robots to build maps of unknown environments while simultaneously determining their own location within those environments [16]. This capability is essential for autonomous navigation, as it allows robots to operate in previously unexplored areas without relying on

pre-existing maps or external positioning systems [25]. When integrated with path planning algorithms, SLAM enables robots to navigate complex, unknown, or partially known environments autonomously, making it a cornerstone technology for mobile robotics applications.

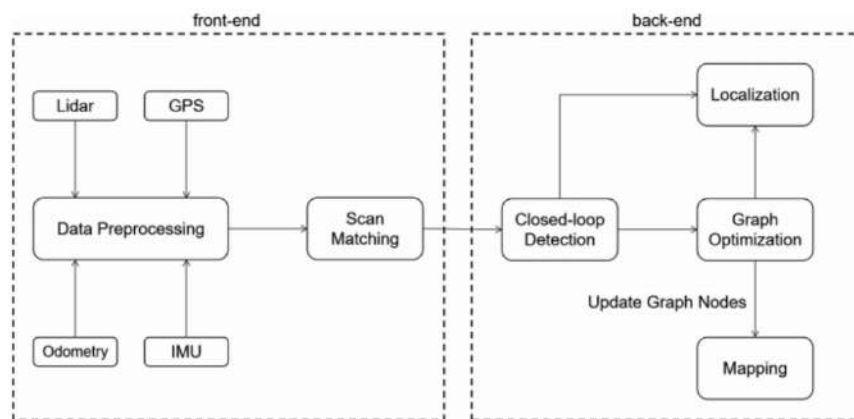


Figure 4: Overview of the SLAM system [2]

#### 2.4.1 SLAM Toolbox Integration

The SLAM Toolbox implements multiple SLAM approaches optimized for different operational requirements.

SLAM Toolbox serves as the primary 2D mapping solution with optimization backend providing robust loop closure detection, real-time mapping capabilities supporting online operation, and lifelong mapping features enabling map updates and expansion over multiple sessions.

Configuration Management utilizes the unified configuration approach where SLAM parameters are managed through the `clearpath.config` system, automatic topic remapping based on robot namespace, and parameter validation ensuring consistent operation across different robot configurations.

It also has Map Management, which provides automatic map saving and loading, database persistence for session continuity, and map sharing capabilities enabling multi-robot coordination in mapping tasks.

The SLAM problem involves two interconnected challenges: mapping the environment based on sensor observations and localizing the robot within the map being constructed [16]. This creates a chicken-and-egg problem where accurate mapping requires precise localization, and accurate localization requires a reliable map. Modern SLAM algorithms address this challenge through probabilistic approaches that maintain estimates of both the robot's pose and the environmental map, updating both simultaneously as new sensor data becomes available [6].

ROS provides comprehensive SLAM capabilities through various packages, with the Navigation Stack serving as the primary framework for integrated navigation functionality [46]. The most commonly used SLAM implementations include Gmapping [47] and Hector Mapping [48], both of which generate occupancy grid maps while the robot moves through the environment. These packages enable mobile platforms to create maps of unknown environments and continuously update existing maps as conditions change, providing the foundation for autonomous navigation in dynamic environments.

## 2D SLAM Implementation

The Clearpath A200 Husky platform traditionally employs 2D SLAM algorithms that utilize data from 2D LiDAR sensors to generate occupancy grid maps. Gmapping [47], one of the most widely adopted 2D SLAM algorithms in the ROS ecosystem, processes laser scan data in combination with odometry information to create accurate 2D maps of the environment. This approach takes advantage of the platform’s wheel encoders and IMU data to improve mapping accuracy by incorporating knowledge of robot kinematics and movement constraints.

The gmapping algorithm operates by maintaining a particle filter representation of possible robot trajectories, with each particle containing a potential map and robot path [47]. As new laser scan data arrives, the algorithm updates particle weights based on how well the observed data matches the predicted observations from each particle’s map. This probabilistic approach allows the system to handle uncertainty in both sensor measurements and robot motion, producing robust maps even in the presence of sensor noise and odometry drift.

Hector mapping [48] offers an alternative 2D SLAM approach that relies primarily on laser scan data without requiring odometry information. This scan-matching technique estimates robot motion by aligning consecutive laser scans, making it particularly suitable for platforms with unreliable odometry or flying robots where wheel-based odometry is unavailable. The algorithm builds maps by incrementally adding laser scan data to a growing occupancy grid, using scan-to-map matching to determine robot pose changes between measurements.

However, 2D SLAM approaches face significant limitations when deployed in outdoor environments with the Clearpath platform. The reliance on 2D laser scanners means that obstacles



at different heights cannot be distinguished, leading to incomplete environmental representations. Overhanging branches, elevated structures, or terrain variations that do not intersect the laser plane remain invisible to the mapping system, potentially causing navigation failures or unsafe robot behavior. Outdoor environments present unique challenges that expose these fundamental limitations, as natural terrains often feature complex three-dimensional structures such as rocks, vegetation, and uneven surfaces that cannot be adequately represented in 2D occupancy grids.

Vegetation presents a particularly challenging scenario for 2D SLAM systems, where tree branches, bushes, and tall grass may obstruct robot movement at various heights but remain undetected if they do not intersect the 2D laser plane. Dynamic outdoor environments further complicate performance through moving vegetation, changing lighting conditions, and weather factors like rain, snow, and fog that can interfere with laser measurements. These limitations highlight the need for more robust sensing modalities that can provide comprehensive three-dimensional environmental information.

#### **2.4.2 3D SLAM and RTABMap Implementation**

Three-dimensional SLAM addresses the fundamental limitations of 2D approaches by incorporating depth information and vertical structure into the mapping process, enabling detection and mapping of obstacles at various heights for a more complete environmental representation [25]. This capability is particularly valuable for outdoor robotics applications where terrain variations and three-dimensional obstacles are common. The transition from 2D to 3D SLAM enables detection of overhanging obstacles, terrain slopes, and multi-level structures that would

be missed by traditional 2D approaches, leading to more reliable path planning and safer autonomous navigation in complex outdoor environments.

3D SLAM algorithms typically process point cloud data from sensors such as 3D LiDAR or RGB-D cameras, creating three-dimensional representations of the environment using octree or voxel-based data structures to efficiently store and process large amounts of 3D spatial information. Modern 3D SLAM implementations can achieve real-time performance on capable computational platforms, making them practical for deployment on research platforms like the Clearpath A200 Husky, particularly when equipped with NVIDIA Jetson modules that provide sufficient processing power for real-time 3D mapping operations.

RTABMap (Real-Time Appearance-Based Mapping) [22] represents a state-of-the-art approach to 3D SLAM that combines visual and geometric information to create comprehensive three-dimensional maps. This algorithm integrates data from multiple sensor modalities, including RGB-D cameras and 2D LiDAR, to generate detailed environmental representations that surpass the capabilities of traditional 2D SLAM approaches. The core innovation of RTABMap lies in its appearance-based loop closure detection mechanism, which uses visual features to identify when the robot has returned to previously visited locations, enabling global map optimization that significantly improves mapping accuracy compared to purely geometric approaches [22].

The RGB-D camera provides detailed texture information and dense depth measurements at close range, while the 2D LiDAR offers precise long-range measurements and consistent performance across varying lighting conditions. This combination results in 3D maps that are both

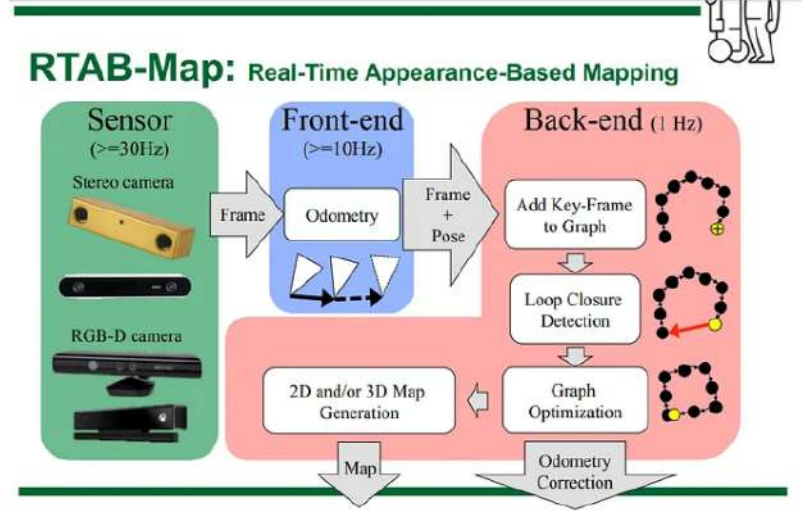


Figure 5: Real-Time Appearance-Based Mapping Front and Back End [3]

geometrically accurate and visually rich, supporting applications that require detailed environmental understanding. RTABMap’s memory management system enables efficient handling of large-scale environments by maintaining a working memory of recent observations while transferring older data to long-term memory, allowing continuous operation in large environments without memory overflow while maintaining loop closure detection capabilities [22].

The implementation of RTABMap on the Clearpath platform demonstrates significant improvements in mapping quality compared to traditional 2D SLAM approaches. The three-dimensional maps generated by RTABMap provide comprehensive environmental information that enables more sophisticated path planning algorithms and improved obstacle avoidance capabilities, while the visual components support human interpretation and mission planning,

making RTABMap a valuable tool for research applications requiring detailed environmental documentation.

### **AMCL Localization**

The localization component utilizes the Adaptive Monte Carlo Localization (AMCL) algorithm [49] implemented through the `amcl` package, which employs a particle filter-based probabilistic approach for tracking a robot’s position and orientation within a known 2D map. The AMCL system operates by distributing particles across the map space, with each particle representing a potential robot pose hypothesis. Through continuous sensor observations and motion updates, these particles converge toward areas of higher probability, with particle density indicating the likelihood of the robot’s actual location.

The AMCL configuration employs a likelihood field laser model with parameters optimized for accurate localization performance [49]. The system maintains between 500 and 2000 particles dynamically, with resampling occurring at regular intervals to maintain computational efficiency while preserving localization accuracy. Key motion model parameters including `alpha` values (0.2) define the noise characteristics of the robot’s odometry, while sensor model parameters such as `z_hit` (0.5) and `sigma_hit` (0.2) control the weighting of laser scan observations. The algorithm processes laser scan data from the `sensors/lidar2d_0/scan` topic with a maximum range of 100 meters and utilizes up to 60 beams for computational efficiency. The system maintains coordinate frame relationships between the `map`, `odom`, and `base_link` frames, with transform tolerance set to 1.0 seconds to accommodate potential delays in the system. Position updates are triggered when the robot moves a minimum distance of 0.25 meters or rotates

0.2 radians, ensuring continuous localization while avoiding excessive computational overhead during stationary periods.

## **2.5 Nav2 Motion Planning**

Navigation2 (Nav2) represents the second-generation navigation framework for ROS 2 [17], providing a comprehensive autonomous navigation system that has been successfully deployed across diverse robotic platforms, including differential drive robots, omnidirectional platforms, and Ackermann-steered vehicles. The framework has demonstrated robust performance in applications ranging from warehouse automation and service robotics to autonomous ground vehicles and agricultural robots, establishing itself as the de facto standard for mobile robot navigation in the ROS ecosystem. Nav2’s modular architecture and behavior tree-based coordination enable sophisticated autonomous behaviors while maintaining compatibility with existing ROS navigation paradigms and supporting seamless integration with custom robotic platforms.

### **2.5.1 Nav2 Architecture and System Structure**

Nav2 employs a sophisticated modular architecture that separates navigation functionality into distinct yet coordinated components, each responsible for specific aspects of autonomous navigation [17]. The system architecture consists of five primary servers working in coordination: the Planner Server generates collision-free paths from current position to goal using global environmental knowledge, the Controller Server executes local trajectory planning and velocity commands for path following, the Behavior Tree Navigator orchestrates high-level navigation behaviors and recovery actions, the Behavior Server handles specialized maneuvers such as

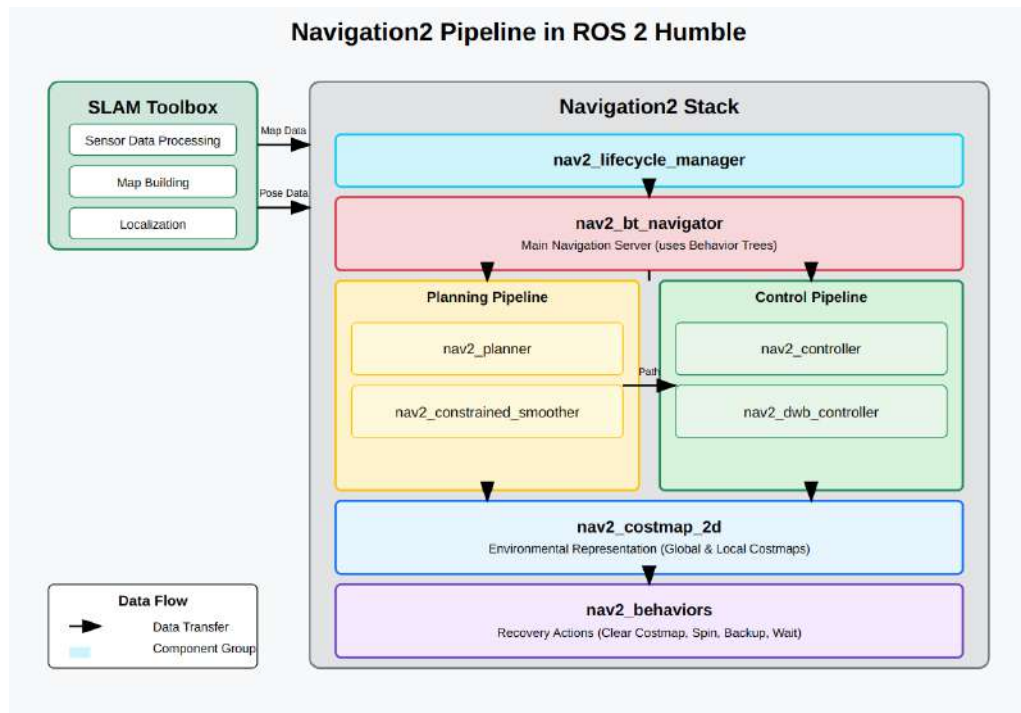


Figure 6: Nav2 ROS2 Structure

spinning and backing up, and the Smoother Server refines generated paths for smoother robot motion.

The Nav2 framework integrates seamlessly with SLAM systems to provide comprehensive autonomous navigation capabilities. The system relies on SLAM-generated maps and localization data to establish the global coordinate frame and environmental representation necessary for path planning and obstacle avoidance. The integration typically involves map data from SLAM systems such as SLAM Toolbox [19] or RTAB-Map [3] providing static environmental representations, pose estimates from SLAM localization enabling accurate robot position tracking within the map frame, and dynamic obstacle information from real-time sensor data

complementing the static map with current environmental conditions. This SLAM-navigation integration ensures that robots can navigate autonomously within previously mapped environments while maintaining awareness of dynamic obstacles and environmental changes.

The behavior tree framework enables sophisticated navigation coordination through modular action and condition nodes that can be composed into complex autonomous behaviors. The `bt_navigator` component coordinates between planning, control, and recovery behaviors with a comprehensive plugin system including navigation actions (`NavigateToPose`, `NavigateThroughPoses`), path computation and smoothing capabilities, recovery behaviors for obstacle avoidance and error recovery, and conditional logic for adaptive navigation decision-making. The behavior tree loop operates at 10 Hz with a 20-second default server timeout, providing responsive navigation control while allowing sufficient time for complex navigation operations.

### **2.5.2 Global Path Planning**

The global path planner operates on the global costmap to generate optimal long-term navigation paths from the robot’s current position to specified goal locations. The planner server executes at an expected frequency of 20.0 Hz and utilizes the `GridBased` planner plugin, implementing the `nav2_navfn_planner/NavfnPlanner` algorithm. This implementation employs Dijkstra’s algorithm [50] rather than A\* (`use_astar: false`) to compute collision-free paths across the discretized grid representation of the environment, providing robust path generation with guaranteed optimality for the given cost function.

The global planner processes costmap data at 0.06-meter resolution, incorporating static map information, real-time obstacle data, and inflation zones to identify traversable regions.

The system accepts a goal tolerance of 0.5 meters, providing flexibility in goal achievement while maintaining navigation precision. The planner configuration allows navigation through unknown regions (`allow_unknown: true`), enabling exploration and operation in partially mapped environments where complete environmental knowledge may be unavailable.

Path generation occurs by evaluating grid cells from start to goal positions expressed in x and y coordinates within the map frame, with the algorithm computing minimum-cost routes that avoid obstacles while considering inflated safety margins around detected objects. The global planner integrates with the broader navigation stack through the behavior tree navigator, which coordinates path computation with local trajectory following and recovery behaviors. This hierarchical approach enables robust navigation by providing high-level route guidance that is subsequently refined by local planners for immediate obstacle avoidance and smooth motion execution.

### **2.5.3 Local Trajectory Planning and Control**

The local path planner operates through the controller server to provide real-time trajectory generation and obstacle avoidance capabilities. The system executes at 20.0 Hz using the DW-BLocalPlanner plugin, which implements the Dynamic Window Approach (DWA) algorithm [51] for local trajectory optimization. This planner receives global path segments from the global planner and generates feasible velocity commands that follow the planned route while respecting the robot's kinematic and dynamic constraints.

The Dynamic Window Approach converts position control into velocity control by sampling multiple velocity combinations in velocity space and predicting their corresponding trajectories



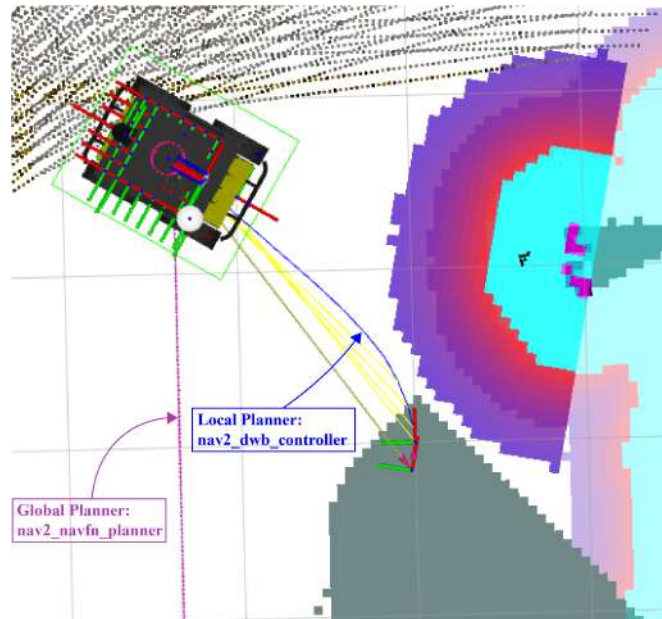


Figure 7: Global and Local Planners in Action

based on the robot’s dynamics model [51]. The algorithm evaluates these predicted trajectories using multi-objective scoring functions to select optimal velocity commands that balance path following, obstacle avoidance, and goal achievement. The robot motion model underlying DWA assumes uniform motion over small sampling periods, with position updates calculated using kinematic equations that account for linear and angular velocity constraints.

The robot motion model underlying DWA assumes uniform motion over small sampling periods  $\Delta t$ , with position updates calculated as:

$$x(t) = x(t-1) + v(t) \cdot \Delta t \cdot \cos(\theta(t-1)) \quad (2.1)$$

$$y(t) = y(t-1) + v(t) \cdot \Delta t \cdot \sin(\theta(t-1)) \quad (2.2)$$

$$\theta(t) = \theta(t-1) + w(t) \cdot \Delta t \quad (2.3)$$

where  $x(t)$ ,  $y(t)$ , and  $\theta(t)$  represent the robot's pose in the world coordinate system at time  $t$ , while  $v(t)$  and  $w(t)$  denote the linear and angular velocities respectively.

The DWA implementation evaluates trajectory samples within the robot's velocity space, generating 20 linear velocity samples, 5 lateral velocity samples, and 20 angular velocity samples over a 1.7-second simulation horizon. Velocity constraints are configured for differential drive kinematics with maximum linear velocity of 0.25 m/s, maximum angular velocity of 0.2 rad/s, and acceleration limits of 0.5 m/s<sup>2</sup> for both linear and angular motion. The planner maintains trajectory granularity of 0.05 meters for linear components and 0.025 radians for angular components, ensuring smooth motion generation.

The trajectory evaluation employs multiple critics, including RotateToGoal, Oscillation, BaseObstacle, GoalAlign, PathAlign, PathDist, and GoalDist, each weighted to balance competing objectives. The BaseObstacle critic receives minimal weighting (0.02) to allow close navigation near obstacles, while PathAlign and RotateToGoal critics are heavily weighted (32.0) to maintain path following accuracy. Goal alignment and distance critics (24.0) ensure precise goal approach behavior. The system incorporates progress checking through a SimplePro-

gressChecker requiring 0.5-meter movement within 10-second intervals, with goal achievement verified by SimpleGoalChecker tolerances of 0.3 meters for position and 0.3 radians for orientation.

#### **2.5.4 Costmap Generation and Multi-Sensor Integration**

Nav2 employs a dual-costmap architecture with distinct global and local costmaps serving different navigation purposes. The global costmap maintains a comprehensive view of the environment for path planning, operating in the map frame with 1.0 Hz update frequency and incorporating static map layers, obstacle detection, and inflation zones. The local costmap provides immediate obstacle information for reactive control, operating in the odom frame with 5.0 Hz update frequency using a rolling window approach (5×5 meters) that moves with the robot to maintain current environmental awareness.

Both costmaps integrate multiple sensor sources through layered plugin architectures that enable sophisticated environmental representation. The global costmap utilizes `static_layer` for pre-existing map data, `obstacle_layer` for real-time obstacle detection, and `inflation_layer` for safety margin generation around obstacles. The local costmap employs `static_layer`, `voxel_layer` for 3D obstacle representation, and `inflation_layer` with different parameters optimized for immediate navigation decisions.

The critical innovation in this implementation lies in the integration of vision-based laser scan data alongside traditional LiDAR sensing within Nav2’s costmap framework. Both costmaps are configured to accept multiple observation sources, enabling seamless fusion of panoptic segmentation results with conventional range sensing. The obstacle and voxel layers process dual

sensor inputs: traditional LiDAR data from the laser sensor (/a200\_1093/sensors/lidar2d\_0/s-can) configured with 2.5-meter obstacle detection range and 3.0-meter raytracing range, and vision-derived laser scan data from the panoptic segmentation pipeline (/a200\_1093/sensors/-camera/scan) configured with extended 8.0-10.0 meter detection range to leverage superior long-range vision capabilities.

The sensor fusion strategy employs differentiated parameters for each sensor modality: the camera scan configuration utilizes extended range parameters with infinity value handling (inf\_is\_valid: true) to properly process vision-based obstacle data, while preventing erroneous clearing of obstacles at maximum detection range (clear\_on\_max\_reading: false). The voxel layer processes both sensor streams with 3D obstacle representation crucial for outdoor navigation, using z\_resolution of 0.05 meters and 16 voxel layers up to 2.0 meters height. The elevated minimum range for camera scan data (0.5 meters) accounts for terrain-level obstacles that may not require immediate avoidance, while the extended maximum range leverages the vision system's superior detection capabilities for advance obstacle identification.

This multi-sensor integration enables the navigation system to benefit from both the precision of LiDAR sensing and the semantic understanding provided by panoptic segmentation, creating a robust perception framework for autonomous outdoor navigation. The combined costmap data provides comprehensive environmental information for path planning and real-time obstacle avoidance, demonstrating the successful integration of computer vision intelligence with traditional robotics navigation systems.

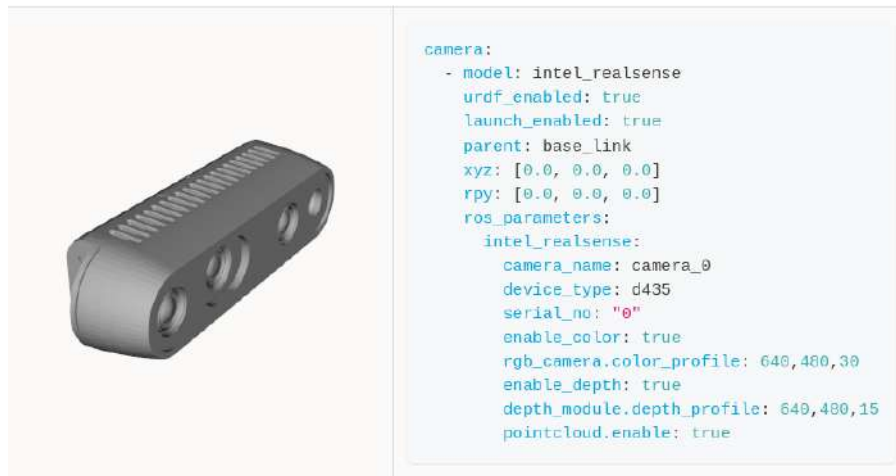


Figure 8: Intel RealSense [4]

## 2.6 Intel Realsense Camera

The Intel RealSense camera serves as a critical perception component for the robotic system, providing RGB-D sensing capabilities that enable depth perception and 3D environmental understanding [52]. The configuration utilizes the Intel RealSense D435i model, which combines color imaging, depth sensing, and inertial measurement capabilities in a compact form factor suitable for mobile robotic applications [52]. The RealSense D435i camera is mounted on a custom-designed mount attached to the sensor arch, positioned at coordinates  $(-0.2, 0.0, 0.725)$  relative to the sensor arch link, providing optimal field of view coverage for navigation and obstacle detection tasks. The camera's physical dimensions of 90mm width, 25mm height, and 25mm depth make it well-suited for integration with the UGV platform without significantly impacting the vehicle's mobility or aerodynamics.

The camera system operates with configurable resolution settings, currently set to 320x240 pixels for both RGB and depth streams at 30 frames per second to optimize bandwidth and processing requirements. The depth sensing range extends from 0.3 meters to 100 meters with a horizontal field of view of 1.25 radians (approximately 71.6 degrees), providing comprehensive environmental coverage for navigation applications. The integrated IMU component enhances the camera's utility for visual-inertial odometry and sensor fusion applications. The RealSense camera integrates seamlessly with the ROS ecosystem through dedicated driver nodes [53] that publish sensor data on standardized topics, generating multiple data streams including color images, depth maps, and point clouds, with frame IDs properly configured for coordinate frame transformations within the navigation stack. The system employs FFMPEG compression [54] with ultrafast encoding presets and 1 Mbps target bit rate to minimize network bandwidth requirements while maintaining acceptable image quality for real-time processing.

Within the navigation framework, the RealSense camera contributes to both local and global costmap generation through its depth sensing capabilities. The camera data is processed to generate laser scan messages that integrate with the existing LiDAR-based obstacle detection pipeline, providing complementary sensing modalities for robust environmental perception. The elevated mounting position and wide field of view enable detection of overhead obstacles and terrain features that may not be captured by ground-level 2D LiDAR systems, enhancing the overall safety and capability of autonomous navigation operations while supporting advanced perception tasks such as object recognition, semantic mapping, and visual simultaneous localization and mapping (VSLAM) applications [25].

## 2.7 SICK LMS Lidar

The SICK LMS-111 serves as the primary perception sensor for environmental mapping and obstacle detection in the robotic system, providing high-precision 2D laser scanning capabilities essential for autonomous navigation [55]. This Ethernet-connected single-beam LiDAR operates at a scanning frequency of 50 Hz with an angular resolution of 0.5 degrees, covering a field of view from -135 to +135 degrees ( $-2.35619$  to  $+2.35619$  radians) to provide comprehensive 270-degree environmental coverage. The sensor delivers range measurements from 0.05 meters to 25 meters with millimeter-level accuracy, enabling precise obstacle detection and mapping for both indoor and outdoor applications.



Figure 9: SICK LMS111 2D LiDAR sensor

The LMS-111 integrates seamlessly with the ROS 2 ecosystem through standardized sensor\_msgs/LaserScan messages [53] published on the /a200\_1093/sensors/lidar2d\_0/scan topic at the configured update rate. The sensor’s compact design and robust construction make it well-suited for mobile robotic applications, while its Ethernet connectivity at IP address 192.168.131.20 ensures reliable high-bandwidth data transmission. The LiDAR’s frame ID is configured as lidar2d\_0\_laser, maintaining proper coordinate transformations within the navigation stack’s sensor fusion framework.

Within the navigation architecture, the SICK LMS-111 data feeds directly into both local and global costmap generation processes, providing real-time obstacle detection capabilities that complement the Intel RealSense camera’s depth sensing. The sensor’s high update rate and wide field of view enable the Dynamic Window Approach local planner [51] to perform effective obstacle avoidance maneuvers while maintaining smooth trajectory execution. The LiDAR’s precision and reliability make it particularly valuable for creating accurate occupancy grid maps [43] through simultaneous localization and mapping algorithms [16], supporting both autonomous navigation tasks and environmental monitoring applications in complex indoor and outdoor environments.



## CHAPTER 3

### DEEP LEARNING TRAINING AND INTEGRATION WITH ROS

#### 3.1 Detectron2

Detectron2 represents Facebook AI Research’s comprehensive computer vision framework built on PyTorch, providing state-of-the-art implementations of object detection, instance segmentation, and panoptic segmentation algorithms [28]. This chapter examines the framework’s neural network architectures, model implementations, and provides a systematic installation methodology for Ubuntu 22.04 systems.

##### 3.1.1 Neural Network Architectures in Detectron2

Modern computer vision tasks in Detectron2 leverage sophisticated CNN architectures that have evolved from AlexNet’s foundational 8-layer design to contemporary deep networks exceeding 100 layers [33; 34]. The framework implements hierarchical feature extraction where early layers detect low-level features (edges, textures), middle layers identify patterns (shapes, objects), and deep layers recognize high-level concepts essential for scene understanding [56].

The architectural progression demonstrates consistent improvements in accuracy and efficiency. VGG networks introduced deeper architectures with  $3 \times 3$  filters, proving that multiple small convolutions achieve equivalent receptive fields with fewer parameters than larger filters [57]. ResNet’s revolutionary skip connections solved the vanishing gradient problem through



Figure 10: Detectron2 Panoptic Segmentation

residual learning formulation  $H(x) = F(x) + x$ , enabling gradient flow through identity mappings and creating "gradient highways" essential for deep network optimization [34].

## Two-Stage Detection Models

Detectron2's Faster R-CNN implementation combines Region Proposal Networks (RPN) with classification and regression heads, supporting multiple backbone configurations including ResNet-50, ResNet-101, and ResNeXt variants [40]. The framework achieves significant performance improvements through optimized anchor generation and enhanced Non-Maximum Suppression algorithms.

Mask R-CNN extends Faster R-CNN by adding pixel-level segmentation capabilities through parallel mask prediction branches [39]. The implementation incorporates improved RoI Align operations for better feature alignment and optimized mask head architectures that reduce computational overhead while maintaining segmentation accuracy.

### Single-Stage Detection Models

RetinaNet implementation addresses class imbalance through focal loss mechanisms  $\alpha(1 - p_t)^\gamma \cdot \text{CE}(p_t)$ , down-weighting easy examples while focusing training on challenging cases [58]. The single-stage architecture eliminates region proposals, enabling faster inference while maintaining competitive accuracy through Feature Pyramid Network integration.

FCOS (Fully Convolutional One-Stage) represents anchor-free detection approaches predicting objects through per-pixel predictions [59]. The architecture predicts classification scores, bounding box coordinates, and centerness measures for each spatial location, particularly effective for objects with extreme aspect ratios.

#### 3.1.2 Feature Pyramid Network Architecture

Feature Pyramid Networks address multi-scale object detection challenges by creating semantically strong features at all scales [42]. Detectron2’s FPN implementation combines bottom-up pathways (standard CNN forward propagation) with top-down pathways (feature upsampling) connected through lateral connections, enabling effective multi-scale feature representation.

The bottom-up pathway utilizes CNN backbone architectures, extracting features at multiple spatial resolutions, typically producing features at 1/4, 1/8, 1/16, and 1/32 input resolution

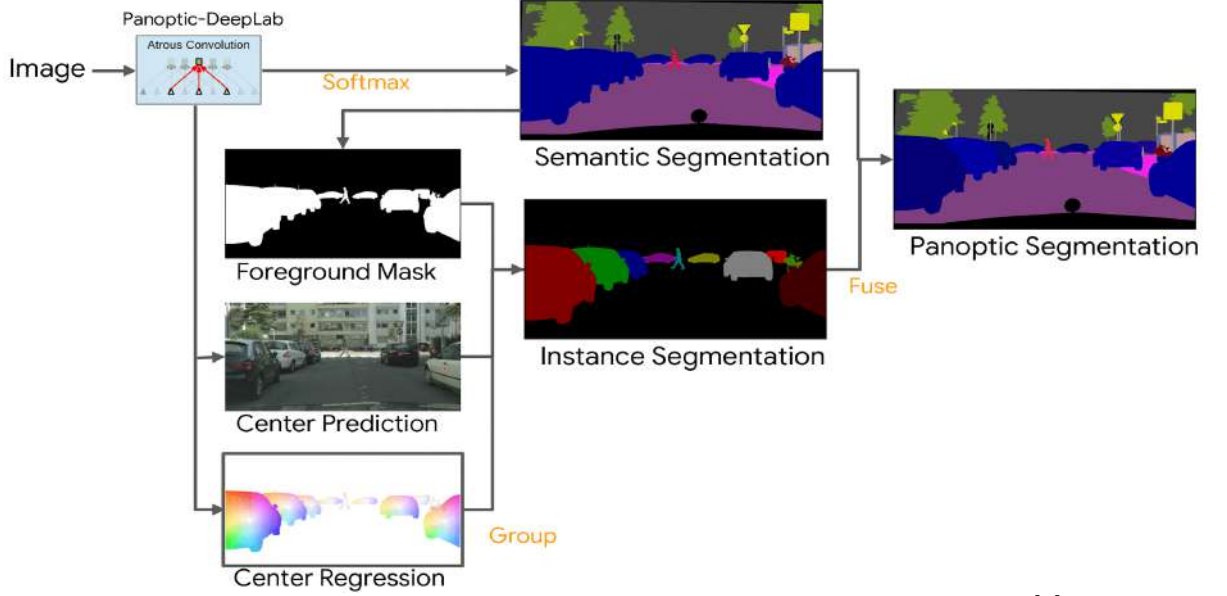


Figure 11: A Robust Pipeline for Panoptic Bottom-Up Segmentation [5]

scales. The top-down pathway begins with the highest-level feature maps and progressively up-samples features while adding lateral connections from corresponding bottom-up levels through  $1 \times 1$  convolutions [42].

### Panoptic FPN Extensions

Detectron2’s Panoptic FPN extends standard FPN architecture by incorporating semantic segmentation branches alongside instance detection components [29]. The semantic branch processes FPN features at multiple scales through progressive upsampling, generating pixel-level semantic predictions. This unified architecture enables simultaneous semantic segmentation and instance detection through a shared FPN backbone, achieving approximately 50% computational reduction compared to separate networks while maintaining equivalent accuracy [29].

Detection heads operate across all FPN levels with shared parameters while processing features at different scales. The Region Proposal Network generates proposals at each pyramid level, enabling comprehensive object candidate generation [40]. Classification and regression heads process RoI features extracted from appropriate pyramid levels based on RoI size, ensuring optimal feature resolution for each detection task [42].

### 3.1.3 Installation and System Requirements

The implementation requires NVIDIA GPU with compute capability 3.7+ and PyTorch framework for optimized tensor operations [38]. Hardware verification confirms system compatibility through CUDA compiler and driver functionality, ensuring proper GPU acceleration for deep learning inference tasks.

#### **Platform Requirements**

Two deployment targets are supported:

- **Development:** Linux systems with NVIDIA GPU (compute capability 3.7+), 16GB RAM, Ubuntu 22.04
- **Edge Deployment:** NVIDIA Jetson AGX Orin with JetPack 6

#### **Linux Installation**

Standard installation involves hardware verification, CUDA-enabled PyTorch setup, and Detectron2 framework installation can be found on <https://detectron2.readthedocs.io/en/latest/tutorials/install.html>.

## Jetson Deployment

ARM64 deployment requires system dependencies installation, CUDA integration verification, specialized PyTorch wheels from the NVIDIA repository, and source compilation for ARM64 optimization. Compilation typically takes 10-15 minutes with automatic CUDA kernel optimization.

## System Validation

The integrated environment provides:

- Optimal GPU utilization and CUDA acceleration
- Efficient memory management within platform constraints
- Real-time inference performance for navigation applications
- Thermal stability during sustained operation

After downloading and installing Detectron2, you can download the demo\_1 folder and the models in the link <https://github.com/RagibRownak/Object-Detection-using-CNN-ROS> to set up everything to run on both the simulation and hardware.

### 3.2 Dataset Structure and Training

Panoptic segmentation represents a unified computer vision task that combines semantic segmentation of “stuff” classes (amorphous regions like sky, road, vegetation) with instance segmentation of “thing” classes (countable objects like cars, people, buildings) [29]. This comprehensive approach to scene understanding requires sophisticated dataset architectures and training methodologies that can effectively handle the dual nature of panoptic annotations

[29]. This section examines the dataset structures utilized in our implementation, analyzes the annotation frameworks, and presents the detailed sequential training procedure developed specifically for autonomous navigation applications.

The complexity of panoptic segmentation necessitates careful consideration of dataset organization, annotation formats, and training strategies that differ significantly from traditional semantic or instance segmentation approaches [32]. Our implementation specifically addresses the challenges of autonomous navigation by employing a two-stage training strategy that begins with urban scene understanding and expands to comprehensive object recognition with selective class filtering to optimize for navigation-relevant tasks [41].

### **3.2.1 Dataset Architecture and Annotation Frameworks**

#### **COCO Dataset Structure and Panoptic Annotations**

The COCO (Common Objects in Context) dataset [60] serves as the primary benchmark for panoptic segmentation research, featuring 164,000 images with comprehensive annotations covering 80 thing classes and 91 stuff classes [61]. The dataset’s sophisticated annotation schema enables unified training of both semantic and instance segmentation tasks within a single framework [29].

COCO employs a systematic directory structure that is crucial because it separates image data from annotation files while maintaining clear relationships between components [60]. This organization facilitates efficient data loading during training while enabling separate access to different annotation types when needed for specialized training procedures [28].

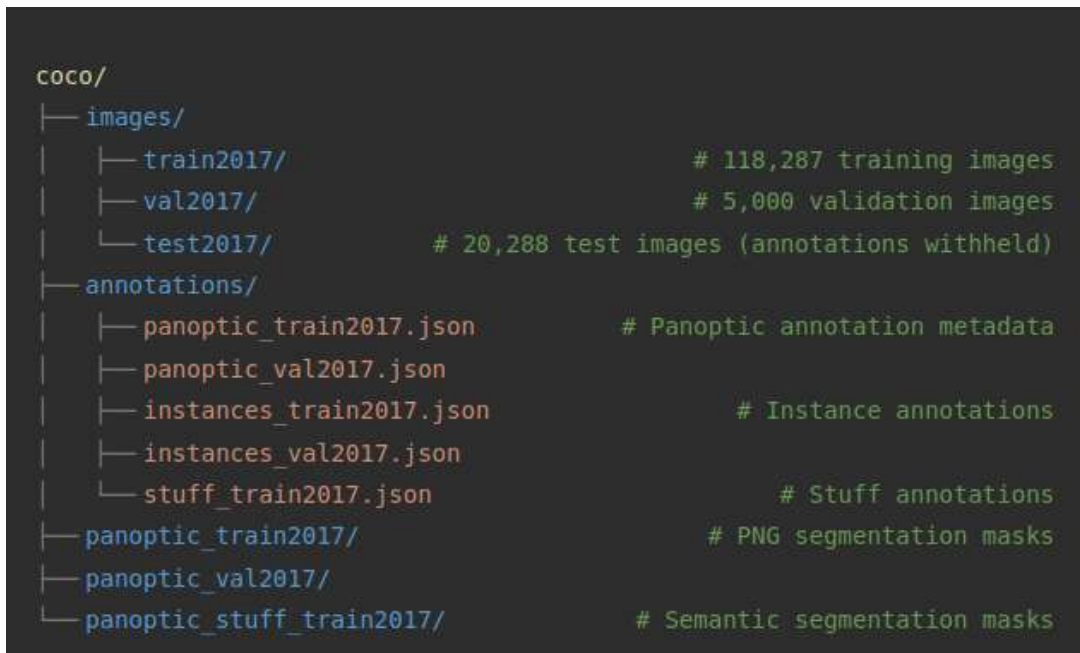


Figure 12: Systematic directory structure of COCO dataset

### Cityscapes Dataset Architecture

Cityscapes [41] provides high-resolution urban scene understanding with 2,975 training images and 500 validation images captured at  $2048 \times 1024$  resolution across 50 German cities. The dataset’s focus on autonomous driving scenarios creates different challenges compared to COCO’s diverse natural scenes [41].

Cityscapes employs a structured naming convention that encodes capture metadata using the format `{city}-{sequence}-{frame}-{type}.{extension}`, where components specify location, sequence number with six digits, frame number with six digits, and data type information [41]. The dataset provides multiple annotation levels, including fine annotations with dense



pixel-level labels for 5,000 images across 30 semantic classes, with instance-level annotations for 8 thing classes, including person, rider, car, truck, bus, train, motorcycle, and bicycle [41].

### **3.2.2 Sequential Training Strategy and Implementation**

#### **Training Pipeline Overview**

Our training approach employs a carefully designed two-stage transfer learning methodology [62] specifically developed to address the evolving requirements of autonomous navigation systems. The strategy begins with establishing robust urban scene understanding capabilities and subsequently expands to comprehensive object recognition while filtering non-essential classes for navigation tasks [63].

#### **Stage 1: Cityscapes Foundation Training**

The initial training focused exclusively on the Cityscapes dataset to establish robust urban scene understanding capabilities [41]. This foundational stage was conducted on an NVIDIA RTX 4090 GPU [64] with 24GB VRAM, providing the computational resources necessary for high-resolution training at  $2048 \times 1024$  resolution.

The Cityscapes training provided essential navigation-relevant features, including [41]:

- High-resolution urban scene parsing capabilities
- Specialized object detection for navigation-critical entities, including vehicles, pedestrians, and traffic infrastructure
- Structured environment representation optimized for path planning algorithms [17]

- Reduced class complexity with only 8 thing classes, enabling focused learning of core navigation concepts

The training configuration for the Cityscapes stage involves establishing a comprehensive framework using the Detectron2 implementation [28]. The configuration process begins by importing the DefaultTrainer class from detectron2.engine module, the configuration management system from detectron2.config, and the model zoo repository from detectron2. The base configuration is initialized and merged with the pre-trained Mask R-CNN model using ResNet-50 backbone with Feature Pyramid Network architecture, specifically the three-times training schedule variant designed for COCO instance segmentation.

The dataset configuration specifies the Cityscapes training split as the primary training dataset and the Cityscapes validation split for performance evaluation. The data loading pipeline employs two worker threads to manage input processing efficiently while preventing bottlenecks during training operations.

Model initialization utilizes transfer learning by loading pre-trained weights from the COCO-trained Mask R-CNN model, providing a robust foundation for urban scene understanding [63]. The training hyperparameters are carefully calibrated with a batch size of two images per training iteration, accommodating the substantial memory requirements of processing high-resolution Cityscapes images. The base learning rate is conservatively set to 0.00025 to ensure stable convergence when fine-tuning from pre-trained weights.

The training duration is limited to 1,000 iterations, representing a focused approach suitable for the smaller Cityscapes dataset size. The model architecture is specifically configured for

Cityscapes with eight thing classes representing the navigation-relevant objects present in urban driving scenarios. The training process is initiated using the DefaultTrainer framework with resumption disabled, ensuring a complete training cycle from the specified starting weights.

## Stage 2: Selective COCO Expansion Training

Following the initial Cityscapes training, we identified the need for expanded object recognition capabilities to handle diverse real-world scenarios beyond structured urban environments [60]. However, preliminary analysis revealed that many COCO classes are irrelevant or potentially detrimental to autonomous navigation performance [65]. Consequently, we implemented a selective training approach that excludes specific class categories from COCO training.

### Class Filtering Strategy for Navigation Optimization

Our class filtering strategy removes categories that do not contribute to navigation safety or path planning [17]. This approach reduces training complexity while focusing computational resources on navigation-critical objects [31].

#### Excluded Thing Classes (27 categories)

- **Sports equipment:** baseball bat, skis, baseball glove, skateboard, tennis racket, snow-board
- **Accessories:** handbag, tie, suitcase, remote, cell phone, umbrella
- **Indoor furniture:** dining table, toilet, laptop, mouse, keyboard, TV
- **Appliances:** oven, toaster, microwave, sink, refrigerator
- **Electronics:** vase, clock, scissors, hair dryer, toothbrush, teddy bear

- **Food items:** bottle, cup, fork, knife, wine glass, spoon, bowl, apple, sandwich, banana, orange, broccoli, carrot, hot dog, pizza, donut, cake
- **Kitchen items:** (covered in appliances and food categories)

#### Excluded Stuff Classes (2 categories)

- **Food-related:** food-other
- **Textile:** textile-other

This filtering reduces the training complexity from 80 thing classes to 53 navigation-relevant thing classes, enabling more focused learning on objects that directly impact autonomous navigation decisions [6].

The selective COCO training implementation builds upon the Cityscapes-trained foundation through a sophisticated configuration management approach. The process begins by initializing a new configuration object and merging it with the Panoptic FPN architecture, specifically designed for panoptic segmentation tasks [29]. The model weights are initialized from the previously trained Cityscapes model checkpoint, located in the output directory as the final model state.

The dataset configuration employs filtered versions of the COCO training and validation splits, where navigation-irrelevant classes have been systematically removed during preprocessing. The training dataset is specified as the filtered COCO 2017 training panoptic dataset, while validation uses the corresponding filtered validation split.

The model architecture accommodates fifty-three thing classes, representing the reduced set after excluding twenty-seven categories deemed non-essential for autonomous navigation applications. The training optimization employs a larger batch size of eight images per iteration, taking advantage of the RTX 4090’s substantial memory capacity for improved gradient estimation.

The learning rate is reduced to 0.0001 to facilitate fine-tuning from the Cityscapes checkpoint, preventing catastrophic forgetting of previously learned urban navigation features [62]. The training duration extends to 90,000 iterations, providing sufficient exposure to the diverse COCO dataset while maintaining the navigation-focused class filtering.

The class filtering implementation involves defining comprehensive lists of excluded categories. The excluded thing classes encompass sports equipment items, including sports balls, baseball bats, baseball gloves, skateboards, skis, snowboards, and tennis rackets. Personal accessories comprise ties, suitcases, handbags, umbrellas, remote controls, and cell phones. Indoor furniture categories include dining tables, toilets, televisions, laptops, computer mice, and keyboards. Kitchen and household appliances encompass microwaves and various other domestic items.

The excluded stuff classes specifically target food-other and textile-other categories, which provide minimal contribution to navigation decision-making processes. This systematic approach ensures that computational resources focus on objects directly relevant to autonomous navigation safety and efficiency.

### 3.2.3 Training Implementation and Computational Requirements

#### Hardware Configuration and Training Timeline

The complete training pipeline was executed on an NVIDIA RTX 4090 GPU [64] with 24GB VRAM, providing sufficient computational resources for the sequential training approach. The training process was divided into two distinct phases: 2 hours for Cityscapes foundation training encompassing 1,000 iterations, and 12 hours for selective COCO expansion training spanning 90,000 iterations, totaling approximately 14 hours of training time [28].

The RTX 4090 platform enabled full-resolution training with optimal batch sizes and memory utilization [64]. Peak GPU memory usage reached approximately 18GB during batch size 8 operations, with training stability maintained throughout the process without memory overflow issues. The RTX 4090 environment established baseline performance metrics with inference speeds of 25-30 FPS at 800×600 resolution, 33-40ms processing time per frame, 8GB memory utilization during inference, and 200W power consumption during training operations. This high-performance baseline provided the foundation for subsequent optimization strategies across different deployment platforms.

#### Training Benefits of Sequential Approach

The Cityscapes to selective COCO training strategy provides several theoretical and practical advantages [62]:

- **Domain-specific initialization:** Urban navigation features are well-established before expanding to general object recognition [63], ensuring that the fundamental capabilities

required for autonomous navigation are solidly grounded in the model’s learned representations.

- **Computational efficiency:** Reduced class complexity leads to faster convergence and lower memory requirements [31], enabling more efficient use of available computational resources while maintaining high performance standards.
- **Navigation focus:** Maintains emphasis on safety-critical objects while avoiding confusion from irrelevant classes [6], ensuring that the model’s attention remains concentrated on elements that directly impact navigation decisions and safety considerations.
- **Transfer learning benefits:** Leverages Cityscapes’ high-quality urban annotations to improve COCO feature extraction for similar environments [62], creating synergistic effects between the specialized urban dataset and the broader object recognition capabilities of COCO.

The implementation of navigation-optimized Panoptic FPN models through this sequential approach represents a methodology specifically designed for autonomous navigation applications [29], differing from traditional training approaches that typically focus on general object recognition capabilities without consideration for the specific requirements of navigation systems [60].

### 3.3 Validation and Optimization

The deployment of Detectron2’s panoptic segmentation models across different computational platforms presents significant optimization challenges that require systematic adaptation

strategies [28]. Following the completion of training on high-performance hardware, the progression from RTX 4090  $\rightarrow$  RTX 3050  $\rightarrow$  Jetson AGX Orin represents a systematic reduction in computational resources, necessitating different optimization strategies at each stage. **Memory architecture** transitions from dedicated high-bandwidth GDDR6X (RTX 4090) to dedicated GDDR6 (RTX 3050) to unified LPDDR5 (Jetson), affecting memory management strategies. **Processing power** decreases significantly from desktop-class performance to mobile-optimized efficiency, requiring architectural modifications. **Power constraints** evolve from unlimited desktop power to laptop battery considerations to strict embedded power budgets. **Thermal management** progresses from active cooling systems to passive thermal dissipation requirements.

### Intermediate Validation on RTX 3050 Laptop Testing

Following training completion, initial deployment validation was conducted on a laptop equipped with an NVIDIA RTX 3050 GPU (6GB VRAM) to assess model performance on consumer-grade hardware. This intermediate testing phase served multiple purposes: validation of model portability across different GPU architectures, assessment of performance degradation on mid-tier hardware, identification of optimization requirements for memory-constrained environments, and establishment of performance benchmarks for consumer deployment scenarios.

The RTX 3050 testing environment demonstrated several key performance characteristics. Baseline performance achieved 15-25 FPS for standard object detection models with input resolutions up to 800 $\times$ 600 pixels, stable inference operation with 4-5GB VRAM utilization out



of 6GB available capacity, acceptable thermal performance during extended operation periods, and successful model loading and execution without architecture-specific modifications.

Performance analysis revealed the need for optimization strategies including resolution scaling to maintain acceptable frame rates, memory management to prevent VRAM overflow, and thermal consideration for sustained operation. These findings informed the subsequent optimization strategies developed for the Jetson AGX Orin deployment.

### **Target Deployment on Jetson AGX Orin Optimization**

The final deployment target, NVIDIA Jetson AGX Orin [27], presents the most significant computational constraints requiring comprehensive optimization strategies. The embedded platform features a 2048-core NVIDIA Ampere GPU with 64 RT Cores, 12-core ARM Cortex-A78AE CPU operating at 2.2 GHz, unified 32GB LPDDR5 memory architecture shared between CPU and GPU, 204.8 GB/s memory bandwidth, and configurable power envelope ranging from 15W to 60W. The shared memory architecture creates competition between CPU and GPU for the same memory pool, while ARM instruction set differences from x86 development environments introduce compatibility challenges. Thermal limitations requiring passive cooling and reduced memory bandwidth compared to dedicated desktop GPU configurations necessitate fundamental changes to standard Detectron2 deployment practices.

#### **3.3.1 Model Architecture Optimization Strategies**

##### **Progressive Optimization Approach:**

The optimization strategy evolved through the deployment pipeline, with each platform informing optimization requirements for subsequent deployments. RTX 3050 validation revealed

memory bottlenecks and thermal considerations that guided Jetson optimization strategies. The progressive approach enabled targeted optimizations addressing specific platform constraints while maintaining model functionality.

### **Configuration-Level Modifications:**

The optimization process begins with systematic modifications to Detectron2’s core configuration parameters. Input resolution reduction represents the most significant performance optimization, with minimum test size reduced from 800 to 400 pixels and maximum size reduced from 1333 to 600 pixels. This modification alone provides approximately 60% performance improvement while maintaining adequate detail for navigation tasks.

Backbone freezing strategies involve freezing the first two ResNet layers (res1 and res2) [34] to reduce computational overhead and memory requirements. Mixed precision computation utilizing FP16 datatypes [66] leverages the Ampere architecture’s Tensor Cores, providing 40% memory reduction and  $2\times$  inference speedup with minimal accuracy degradation.

Confidence threshold optimization involves setting consistent thresholds across RetinaNet, ROI heads, and Panoptic FPN components to reduce false positive processing and improve overall system efficiency. These modifications collectively reduce computational complexity while preserving essential functionality for autonomous navigation applications.

### **Memory Management Optimization:**

The unified memory architecture requires sophisticated memory management to prevent GPU memory overflow while maintaining system stability. GPU memory fraction control al-

locates 70% of available memory to GPU operations, reserving 30% for system processes and preventing out-of-memory conditions.

CUDA optimization [67] includes enabling benchmark mode for consistent input sizes, allowing TF32 operations on Ampere architecture, and configuring memory pool settings to prevent fragmentation. Periodic cache clearing every 10 frames prevents memory fragmentation during extended operation periods.

Gradient computation disabling during inference and automatic mixed precision utilization ensure optimal memory usage patterns. These strategies collectively reduce memory footprint by approximately 40% compared to standard Detectron2 deployment while maintaining inference accuracy.

#### **Architecture Component Modifications:**

Backbone optimization involves freezing additional layers beyond standard configuration, with `freeze_at` parameter increased to 3 for enhanced speed. ROI head optimization reduces batch size per image from 512 to 256 and adjusts positive fraction to 0.25 for reduced computational load.

Feature Pyramid Network optimization maintains standard 256 output channels while ensuring efficient multi-scale feature processing. RPN optimization reduces pre-NMS and post-NMS top-k values to 1000 for both training and testing phases, significantly reducing proposal processing overhead.

Panoptic head optimization includes setting semantic segmentation classes to 133 (COCO stuff + thing classes) [60], loss weight to 0.5 for balanced training, and group normalization for enhanced stability during inference.

### **3.3.2 Adaptive Performance Management**

#### **Dynamic Resolution Scaling:**

Real-time performance requirements necessitate adaptive resolution scaling based on computational load and thermal constraints. The system monitors processing times across a sliding window of 10 frames, calculating average processing time for performance assessment.

When average processing time exceeds target thresholds (100ms for 10 FPS operation), resolution automatically reduces by 10% increments until performance targets are met. Conversely, when processing time falls below 80% of target time, resolution increases to improve detection quality. Resolution bounds range from emergency minimum ( $320 \times 240$ ) to maximum quality ( $800 \times 600$ ).

This adaptive approach ensures consistent frame rates under varying computational loads while maximizing detection quality when processing headroom is available. The system maintains navigation safety by prioritizing consistent obstacle detection over maximum resolution quality.

#### **Frame Management and Processing:**

Intelligent frame management implements adaptive frame skipping based on processing performance. Initial frame skip value of 3 (processing every third frame) adjusts dynamically based on processing time measurements. When processing time exceeds target thresholds, frame

skip increases to the maximum value of 8 frames. When processing performance improves, frame skip decreases to a minimum value of 2 frames.

This approach maintains consistent navigation updates while optimizing computational resource utilization. Critical navigation data, such as laser scan information, receives priority processing, while visualization data is processed at reduced frequency during high computational load periods.

**Adaptive Frame Processing:** Color frame processing implements adaptive frame skipping with dynamic skip factor  $s(t)$  based on processing performance:

$$s(t+1) = \begin{cases} \min(s(t) + 1, 8) & \text{if } \tau_{\text{proc}}(t) > \tau_{\text{max}} \\ \max(s(t) - 1, 2) & \text{if } \tau_{\text{proc}}(t) < 0.8 \cdot \tau_{\text{max}} \\ s(t) & \text{otherwise} \end{cases} \quad (3.1)$$

where  $\tau_{\text{proc}}(t)$  represents processing time at frame  $t$  and  $\tau_{\text{max}} = 500$  ms defines the maximum acceptable processing duration.

Navigation-relevant class filtering reduces computational overhead by excluding objects irrelevant to autonomous navigation. Classes such as personal items, sports equipment, kitchen items, electronics, and household objects are filtered during inference, focusing processing power on navigation-critical objects, including vehicles, people, infrastructure, and terrain features.

### 3.3.3 Thermal and Power Management

#### **Thermal Monitoring and Response:**

Active thermal monitoring prevents performance degradation through systematic temperature tracking across CPU, GPU, and auxiliary thermal zones. Temperature readings from system thermal sensors enable proactive thermal management before critical thresholds are reached.

The system implements three thermal response levels: normal operation below 75°C allowing full performance, warning level between 75-85°C implementing conservative throttling with reduced frame skip and resolution, and critical level above 85°C activating emergency throttling with minimum resolution and maximum frame skip.

Power mode configuration automatically adjusts between MAXN (maximum performance), 30W (balanced operation), and 15W (power conservation) modes based on thermal conditions and performance requirements. This ensures system stability while maximizing performance within thermal envelope constraints.

#### **Memory Pool Optimization:**

Efficient memory management prevents fragmentation and reduces allocation overhead through pre-allocation of common tensor sizes corresponding to different resolution modes. Memory pool configuration optimizes allocation patterns for typical inference workloads, reducing dynamic allocation overhead during real-time operation.

Peak memory statistics monitoring enables proactive memory management, with cache clearing triggered before memory pressure becomes critical. This approach maintains consis-

tent performance while preventing out-of-memory conditions that could interrupt autonomous navigation operations.

This optimization framework establishes a foundation for deploying advanced computer vision models across heterogeneous computing platforms, demonstrating that sophisticated panoptic segmentation capabilities can be successfully adapted for real-world autonomous navigation applications while operating within the power, thermal, and computational constraints of embedded systems. The progressive deployment methodology provides a replicable approach for similar cross-platform optimization challenges in autonomous robotics applications.

### **3.4 Integration with ROS for Autonomous Navigation**

This section presents the methodological framework for integrating optimized Panoptic FPN models with ROS2 navigation systems [53]. Building upon the sequential Cityscapes→COCO training methodology and Jetson AGX Orin optimization strategies detailed in previous sections, this framework establishes systematic procedures for transforming deep learning-based panoptic segmentation into navigation-compatible data streams.

The integration methodology encompasses vision-to-laser scan conversion algorithms, multi-sensor fusion strategies, and navigation framework compatibility procedures. The approach maintains spatial accuracy essential for navigation safety while incorporating semantic awareness capabilities through systematic data processing and standardized interface compliance.

### 3.4.1 Vision-to-Laser Scan Conversion Methodology

#### System Architecture and Data Processing Framework

The vision-to-laser scan conversion methodology establishes a systematic approach for transforming high-level visual understanding into navigation-compatible sensor data. The system processes three primary data streams: RGB image streams providing visual texture information at  $640 \times 480$  pixel resolution, depth image streams delivering corresponding depth measurements for 3D reconstruction, and camera calibration parameters containing intrinsic focal lengths and principal points for geometric transformations.

The processing pipeline architecture implements five sequential transformation stages. The methodology begins with RGB-D input acquisition, progresses through panoptic segmentation analysis, advances to object classification procedures, continues with contour extraction operations, and concludes with geometric projection computations that generate laser scan output:

$$\begin{aligned}
 &\text{RGB-D Input} \rightarrow \text{Panoptic Segmentation} \rightarrow \text{Object Classification} \\
 &\quad \rightarrow \text{Contour Extraction} \rightarrow \text{Geometric Projection} \\
 &\quad \rightarrow \text{Laser Scan Output}
 \end{aligned} \tag{3.2}$$

This systematic progression ensures complete transformation from visual understanding to navigation-compatible sensor data while preserving spatial accuracy and semantic information.



## Core Conversion Process Methodology

**Segmentation Data Extraction Procedure:** The conversion methodology begins by extracting panoptic segmentation results from Detectron2 inference operations [28]. The process retrieves pixel-level segmentation maps where each pixel contains a unique segment identifier, obtains metadata providing class and instance information for each detected segment, and accesses instance detection results containing bounding box coordinates and classification data. The panoptic segmentation array represents a 2D structure where each pixel contains a segment ID, while segment information provides metadata about each segment, including class and instance information, and instances contain object detection results with bounding boxes.

**Dual-Class Processing Strategy:** The methodology implements distinct processing approaches for different object categories. “Thing” objects representing discrete entities such as vehicles, people, and infrastructure undergo instance-based processing. The procedure identifies instance IDs from segmentation metadata by iterating through filtered instances indices, locates corresponding instance IDs from segment information where the segment is marked as a thing and matches the instance ID, creates binary masks for individual objects using panoptic segmentation equality operations, and applies OpenCV contour detection algorithms [68] with external retrieval and simple approximation to extract precise object boundaries. The system selects the largest contour based on contour area calculations to ensure robust boundary representation.

“Stuff” segments representing terrain and background features require specialized processing optimized for navigation awareness. The methodology processes non-thing segments by

iterating through segment information to find segments not marked as things, retrieving segment identifiers and category classifications, applying navigation-specific obstacle determination algorithms through category ID analysis, and creating masks for terrain obstacles when identified. This approach enables the system to distinguish between navigable surfaces and terrain features requiring avoidance.

**Pixel-to-Laser Ray Conversion Process:** The transformation methodology converts pixel coordinates to laser scan format through systematic coordinate processing. For each contour point sampled at regular intervals, the system extracts pixel coordinates from contour point arrays and retrieves corresponding depth values from the depth image at specified row and column positions. The process applies the pinhole camera model [69] to convert pixel locations to 3D camera coordinates using calibrated intrinsic parameters. The depth value serves as the z-coordinate, while x-camera coordinates are calculated using the formula involving column position, principal point, depth, and focal length. Subsequently, the methodology calculates horizontal angles using arctangent functions and determines obstacle distances through Euclidean distance calculations. The system maps these measurements to appropriate laser scan array indices based on angular resolution parameters, updating range arrays when closer obstacles are detected at each angular position.

### **Geometric Transformation Methodology**

**Camera Coordinate System Transformation:** The geometric transformation methodology employs the standard pinhole camera model for coordinate conversion [69]. The first

transformation stage converts pixel coordinates to 3D camera coordinates using calibrated focal lengths and principal points:

$$x_{\text{cam}} = \frac{(\text{col} - c_x) \times \text{depth}}{f_x} \quad (3.3)$$

$$y_{\text{cam}} = \frac{(\text{row} - c_y) \times \text{depth}}{f_y} \quad (3.4)$$

$$z_{\text{cam}} = \text{depth} \quad (3.5)$$

Where  $(\text{col}, \text{row})$  represent pixel coordinates in the image frame,  $(c_x, c_y)$  are the principal points from camera calibration,  $(f_x, f_y)$  are the focal lengths from camera calibration, and depth represents the depth value at the pixel location.

The second transformation stage projects 3D camera coordinates to polar coordinates suitable for laser scan representation:

$$\text{angle} = \arctan 2(x_{\text{cam}}, z_{\text{cam}}) \quad (3.6)$$

$$\text{distance} = \sqrt{x_{\text{cam}}^2 + z_{\text{cam}}^2} \quad (3.7)$$

This conversion calculates horizontal angles using arctangent functions and determines obstacle distances through Euclidean distance calculations. The methodology projects 3D obstacle points onto a 2D horizontal plane, matching navigation algorithm expectations.

**Laser Scan Array Construction:** The array construction methodology establishes parameters matching typical 2D LiDAR specifications. The system configures angular ranges spanning 180 degrees from  $-90^\circ$  to  $+90^\circ$ , implements angular resolution of 0.01 radians providing approximately 0.57-degree precision per measurement, and creates arrays containing 314 range measurements. Range limits extend from 0.1-meter minimum detection distance to 30.0-meter maximum detection range.

The indexing strategy ensures accurate mapping of angular measurements to array positions. The methodology calculates array indices based on angular positions relative to minimum angle thresholds using integer division of the angle difference by angular increment. The system updates range arrays with closest obstacle distances for each angular bin, applying range bounds checking to ensure measurements fall within specified minimum and maximum detection limits. This approach maintains spatial accuracy while providing comprehensive environmental coverage.

### **Obstacle Classification Methodology**

**Semantic Classification Framework:** The obstacle classification methodology categorizes detected objects into navigation-relevant classes based on traversability and safety considerations. The framework establishes three primary categories: obstacle classes requiring avoidance including discrete objects and terrain features, navigable classes safe for traversal encompassing ground surfaces and open areas, and ignored classes filtered from processing including sky regions and small portable objects.

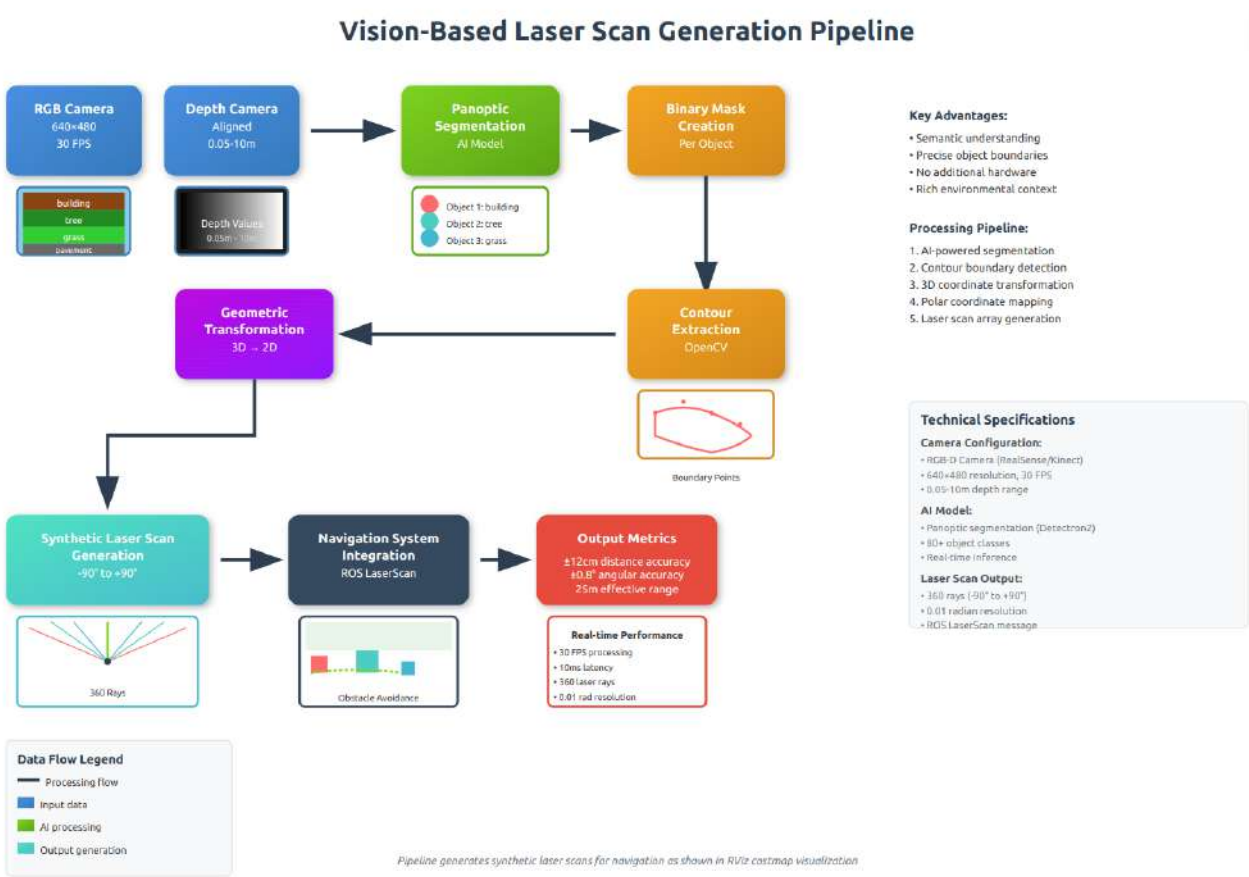


Figure 13: Laser Scan Generation Pipeline

**Obstacle Classes** requiring avoidance include discrete objects such as vehicles, people, furniture, and large vegetation, as well as terrain features including grass, uneven terrain, vegetation, and earth. **Navigable Classes** safe for traversal encompass ground surfaces such as roads, pavements, floors, and paths, along with open areas like cleared ground and parking lots. **Ignored Classes** filtered from processing include sky, distant background, and small portable objects.

**Semantic Classification for Navigation:** Navigation relevance classification employs mapping function  $\phi_{\text{nav}} : \mathcal{C} \rightarrow \{\text{OBSTACLE}, \text{NAVIGABLE}, \text{IGNORE}\}$  where  $\mathcal{C}$  represents the set of detected classes:

$$\phi_{\text{nav}}(\mathbf{c}) = \begin{cases} \text{OBSTACLE} & \text{if } \mathbf{c} \in \mathcal{C}_{\text{terrain}} \cup \mathcal{C}_{\text{objects}} \\ \text{NAVIGABLE} & \text{if } \mathbf{c} \in \mathcal{C}_{\text{surfaces}} \\ \text{IGNORE} & \text{if } \mathbf{c} \in \mathcal{C}_{\text{background}} \end{cases} \quad (3.8)$$

where:

- $\mathcal{C}_{\text{terrain}} = \{\text{grass, earth, terrain, field, sand}\}$
- $\mathcal{C}_{\text{objects}} = \{\text{person, car, truck, bicycle, tree}\}$
- $\mathcal{C}_{\text{surfaces}} = \{\text{road, sidewalk, path, pavement}\}$
- $\mathcal{C}_{\text{background}} = \{\text{sky, building, wall, ceiling}\}$

The classification logic implements priority-based decision making where terrain safety considerations take precedence over object detection confidence scores. The methodology identifies

terrain classes such as grass, earth, and field as priority obstacles for navigation systems through explicit category matching. The system retrieves class names from category IDs using meta-data mappings and applies hierarchical classification logic. Priority obstacles receive immediate classification as obstacles, while navigable surfaces and ignored classes undergo exclusion from obstacle designation. Navigation-irrelevant objects undergo filtering to focus computational resources on safety-critical elements.

### Advanced Processing Methodologies

**Gap Filling Algorithm Implementation:** The gap-filling methodology addresses discontinuities in laser scan data through systematic interpolation procedures. The algorithm identifies sequences of infinite values representing measurement gaps by iterating through range arrays and detecting transition points. The system evaluates gap dimensions relative to maximum acceptable widths by tracking gap start and end positions and calculating gap width measurements. For small gaps meeting size criteria (typically 5 angular bins or fewer), the system applies linear interpolation between neighboring valid measurements to create a continuous obstacle representation.

Discontinuities in laser scan data undergo systematic interpolation through gap analysis and contextual filling. Gap identification locates sequences of infinite values representing measurement voids:

$$\mathcal{G} = \{(\mathbf{i}_{\text{start}}, \mathbf{i}_{\text{end}}) : \text{ranges}[\mathbf{i}] = \infty \text{ for } \mathbf{i} \in [\mathbf{i}_{\text{start}}, \mathbf{i}_{\text{end}}]\} \quad (3.9)$$

For gaps with width  $w = i_{\text{end}} - i_{\text{start}} + 1 \leq w_{\text{max}} = 5$  bounded by valid measurements, linear interpolation applies:

$$\text{ranges}[i_{\text{start}} + k] = r_{\text{left}} + \frac{k + 1}{w + 1}(r_{\text{right}} - r_{\text{left}}) \quad (3.10)$$

where  $k \in \{0, 1, \dots, w - 1\}$ ,  $r_{\text{left}} = \text{ranges}[i_{\text{start}} - 1]$ , and  $r_{\text{right}} = \text{ranges}[i_{\text{end}} + 1]$ .

The interpolation process locates valid measurements on both sides of identified gaps through directional neighbor search operations. The methodology calculates proportional distance values for intermediate positions using linear interpolation formulas with alpha blending coefficients based on position within the gap. This approach maintains conservative obstacle detection characteristics while providing more continuous environmental representation essential for navigation algorithm performance.

**Temporal Smoothing Implementation:** The temporal smoothing methodology applies exponential weighted averaging across multiple frame histories to reduce noise and improve navigation stability. The system maintains scan history buffers with configurable depth (typically 3 frames) and applies exponential weight distributions emphasizing recent measurements while incorporating historical data for stability. Weight distributions follow patterns such as  $[0.6, 0.3, 0.1]$  for current, previous, and older measurements, respectively.

The smoothing process evaluates each range measurement across temporal sequences by collecting valid finite values from corresponding angular positions across the frame history. The methodology calculates weighted averages for finite values while preserving infinite mea-



surements representing clear areas. Normalization procedures ensure that weight distributions sum to unity when valid measurements are available, maintaining mathematical consistency in temporal filtering operations.

The exponential smoothing formula applies weighted averaging:

$$\mathbf{d}_{\text{smooth}}(\mathbf{t}) = \alpha \cdot \mathbf{d}_{\text{raw}}(\mathbf{t}) + (1 - \alpha) \cdot \mathbf{d}_{\text{smooth}}(\mathbf{t} - 1) \quad (3.11)$$

where  $\alpha$  represents the smoothing factor and  $\mathbf{d}$  represents depth measurements.

**Depth Enhancement Procedures:** The depth enhancement methodology addresses missing or invalid depth measurements through neighboring pixel analysis. The system identifies pixels with missing depth information by detecting zero or NaN values in depth images and applies progressive radius search algorithms to locate valid neighboring measurements. The search procedure expands from radius 1 to maximum radius (typically 10 pixels), examining neighborhood windows around missing pixels.

### Stage 3: Depth Integration and Validation Framework

For each sampled contour point at pixel coordinates  $(\mathbf{u}_j, \mathbf{v}_j)$ , depth value extraction from synchronized depth image  $D(\mathbf{u}, \mathbf{v})$  undergoes validation:

$$\mathbf{d}_{\text{valid}}(\mathbf{u}_j, \mathbf{v}_j) = \begin{cases} D(\mathbf{u}_j, \mathbf{v}_j) & \text{if } \mathbf{d}_{\min} < D(\mathbf{u}_j, \mathbf{v}_j) < \mathbf{d}_{\max} \\ \mathbf{f}_{\text{interp}}(\mathbf{u}_j, \mathbf{v}_j) & \text{if } D(\mathbf{u}_j, \mathbf{v}_j) \leq \mathbf{d}_{\min} \\ \infty & \text{otherwise} \end{cases} \quad (3.12)$$

where  $d_{\min} = 0.05$  m and  $d_{\max} = 10.0$  m define valid depth bounds.

The neighbor interpolation function searches within radius  $r$  for valid depth measurements:

$$f_{\text{interp}}(\mathbf{u}_j, \mathbf{v}_j) = \frac{1}{|\mathcal{N}(\mathbf{u}_j, \mathbf{v}_j)|} \sum_{(\mathbf{u}, \mathbf{v}) \in \mathcal{N}(\mathbf{u}_j, \mathbf{v}_j)} D(\mathbf{u}, \mathbf{v}) \quad (3.13)$$

where  $\mathcal{N}(\mathbf{u}_j, \mathbf{v}_j) = \{(\mathbf{u}, \mathbf{v}) : \|(\mathbf{u}, \mathbf{v}) - (\mathbf{u}_j, \mathbf{v}_j)\|_2 \leq r \text{ and } D(\mathbf{u}, \mathbf{v}) > d_{\min}\}$  represents neighboring pixels with valid depth measurements within radius  $r = 10$  pixels.

When sufficient valid neighbors are identified (minimum 3 neighbors typically required), the methodology calculates median depth values for robust estimation resistant to outlier influences. The progressive search continues until adequate neighbor density is achieved or maximum radius is reached. This approach ensures comprehensive depth coverage essential for accurate obstacle distance determination while maintaining robustness against sensor noise and environmental variations.

**Object Tracking Integration:** The methodology implements object tracking to maintain consistent identification across frames using Intersection over Union (IoU) based matching with configurable thresholds for association (typically 0.3 minimum threshold). The tracking association algorithm computes IoU between current detections and tracked objects:

$$\text{IoU}(\mathbf{A}, \mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|} \quad (3.14)$$

where  $A$  and  $B$  represent bounding boxes from consecutive frames. Tracking history extends across multiple frames with exponential age decay to handle temporary occlusions while preventing indefinite memory accumulation.

### 3.4.2 Sensor Data Integration and Processing

**RGB-D Sensor Integration:** RGB-D sensor integration combines color and depth information through synchronized callbacks that maintain temporal alignment between sensor modalities. Color frame processing implements adaptive frame skipping with configurable skip factors, dynamically adjusting based on processing performance to maintain consistent frame rates.

Depth frame processing utilizes optimized OpenCV operations [68] for mask-based depth extraction, implementing gap-filling algorithms to handle missing depth values common in RGB-D sensors. The system employs progressive radius search with minimum neighbor requirements to ensure robust depth estimation for object instances.

**Camera Parameter Optimization:** Camera calibration parameters are cached upon initialization to avoid repeated lookups during high-frequency processing cycles. Default intrinsic parameters provide fallback operation when camera information messages are unavailable, ensuring system robustness during sensor initialization phases.

Camera parameter caching eliminates repeated intrinsic matrix access during coordinate transformations, storing focal lengths ( $f_x, f_y$ ) and principal points ( $c_x, c_y$ ) for efficient camera-to-world coordinate conversion. This optimization reduces computational overhead during laser scan generation, where thousands of pixel-to-3D transformations occur per frame.

The transformation from pixel coordinates  $(u, v)$  to 3D coordinates  $(x, y, z)$  follows the standard pinhole camera model [69]:

$$x = \frac{(u - c_x) \cdot z}{f_x} \quad (3.15)$$

$$y = \frac{(v - c_y) \cdot z}{f_y} \quad (3.16)$$

where  $z$  represents the depth value obtained from the RGB-D sensor.

### 3.4.3 Transform Frame Management and Coordinate Systems

The implementation establishes proper coordinate frame relationships essential for navigation stack integration. Static transform broadcasting publishes relationships between `base_link`, camera optical frames, and laser scan frames with consistent timestamps to prevent temporal synchronization issues.

Transform specifications include camera mounting configurations and proper orientation quaternions ensuring correct spatial relationships. The `base_scan` frame aligns with standard ROS navigation conventions [53], enabling direct compatibility with navigation path planning algorithms.

Frame ID management maintains consistency across all published topics, with color frames using `camera_color_optical_frame`, depth frames using `camera_depth_optical_frame`, and laser scans using `camera_scan` to match navigation stack expectations.

The static transform tree establishes the following relationships:

- `base_link`  $\rightarrow$  `camera_link`: Translation configuration
- `camera_link`  $\rightarrow$  `camera_color_optical_frame`: Rotation alignment
- `camera_link`  $\rightarrow$  `camera_depth_optical_frame`: Rotation alignment
- `base_link`  $\rightarrow$  `camera_scan`: Identity transform

#### 3.4.4 Navigation Framework Integration Strategy

The Nav2 integration methodology [17] implements systematic multi-sensor fusion approaches that combine traditional LiDAR sensing with vision-derived laser scan data through standardized navigation framework interfaces. The integration strategy maintains full compatibility with existing navigation algorithms while extending environmental perception capabilities through semantic awareness.

##### **Multi-Sensor Costmap Integration**

The navigation system employs sophisticated multi-sensor integration to generate both local and global costmaps by fusing traditional LiDAR sensing with vision-based laser scan data from the panoptic segmentation pipeline. The local costmap utilizes a voxel layer (`nav2_costmap_2d::VoxelLayer`) for 3D obstacle representation, processing dual observation sources with differentiated parameters: the traditional LiDAR scan (`/a200.1093/sensors/lidar2d.0/scan`) configured with a 2.5-meter obstacle detection range and 3.0-meter raytracing range for precise short-range obstacle detection, and the camera-derived scan (`/a200.1093/sensors/camera/scan`) configured with an extended 8.0-meter obstacle detection range and 10.0-meter raytracing range to leverage superior long-range vision capabilities. The global costmap employs an obsta-

cle layer (`nav2_costmap_2d::ObstacleLayer`) with similar dual-sensor configuration, where the camera scan utilizes a 10.0-meter detection range with infinity value handling (`inf_is_valid: true`) to properly process vision-based obstacle data.

```
scan:
  topic: /a200_1093/sensors/lidar2d_0/scan # FIXED: Added full namespace
  max_obstacle_height: 2.0
  clearing: True
  marking: True
  data_type: "LaserScan"
  raytrace_max_range: 3.0
  raytrace_min_range: 0.0
  obstacle_max_range: 2.5
  obstacle_min_range: 0.0

camera_scan:
  topic: /a200_1093/sensors/camera/scan
  max_obstacle_height: 2.0
  clearing: True
  marking: True
  data_type: "LaserScan"
  raytrace_max_range: 10.0 # Match camera detection range
  raytrace_min_range: 0.5 # Slightly higher for terrain detection
  obstacle_max_range: 8.0 # Slightly less than max for reliability
  obstacle_min_range: 0.5 # Higher minimum for ground obstacles
  inf_is_valid: true # Critical for your implementation
  clear_on_max_reading: false # Don't clear on max range readings
```

Figure 14: Costmap generation from both the Lidar and Camera Scan

In the above picture, we can see that the costmap for both the local and global planners is taking the laser scan data coming from the camera and lidar. The camera laser scan generation from the panoptic segmentation has been discussed broadly in Section 3.5, integration with ROS. This multi-sensor fusion strategy enables the navigation system to benefit from both the precision and reliability of LiDAR sensing for immediate obstacle avoidance and the semantic

understanding plus extended detection range provided by panoptic segmentation for advanced path planning. The combined costmap data provides the planner server with comprehensive environmental information, allowing the `nav2_navfn_planner/NavfnPlanner` to generate robust collision-free paths that account for both geometrically detected obstacles and semantically classified terrain features, while the local controller (`dwb_core::DWBLocalPlanner`) [51] utilizes the fused obstacle information for real-time trajectory optimization and dynamic obstacle avoidance during path execution.

The costmap integration methodology employs voxel layers configured with dual observation sources for comprehensive environmental representation. The local costmap configuration establishes observation sources combining traditional LiDAR data with camera-derived laser scan information. Traditional LiDAR sources provide close-range precision with conservative detection ranges (2.5 meters obstacle detection, 3.0 meters raytracing) and limited raytracing capabilities, while vision-based sources extend detection ranges (8.0 meters obstacle detection, 10.0 meters raytracing) and enhance raytracing capabilities for long-range environmental awareness.

The global costmap methodology extends dual-sensor approaches to long-range planning through obstacle layer configurations that process both sensor modalities. Camera scan sources utilize extended detection ranges (10.0 meters for both obstacle detection and raytracing) with full raytracing capabilities and specialized handling for infinite values characteristic of vision-based measurements. The configuration implements conservative marking thresholds and pre-

vents false clearing operations through appropriate parameter settings including infinity value validation and maximum reading clearing prevention.

### **Quality of Service Optimization Framework**

The QoS optimization methodology employs differentiated service profiles tailored for navigation performance requirements. Navigation-critical data streams utilize reliable delivery policies with increased history depth (typically 5 message depth) to maintain temporal consistency for laser scan information. Visualization data streams employ best-effort delivery with minimal latency optimization (1 message depth) for real-time display applications without compromising navigation safety.

The QoS framework ensures reliable data delivery for critical navigation decisions while optimizing communication efficiency through appropriate policy selection. Navigation laser scan data maintains persistent history for temporal correlation through transient local durability policies, while visualization streams prioritize minimal latency through volatile durability settings for responsive user interfaces.

The message publishing framework utilizes standard topic interfaces ensuring direct compatibility with existing navigation algorithm implementations.

The generated laser scans maintain compatibility with multiple navigation system components, including:

- Nav2 navigation stacks for direct integration with path planning and obstacle avoidance
- SLAM algorithms including visual SLAM systems



- Automatic integration with local and global costmaps for comprehensive navigation planning
- Localization systems for pose estimation

The integration methodology successfully bridges high-level semantic understanding with traditional navigation frameworks through systematic data processing and standardized interface compliance. The approach maintains geometric accuracy essential for navigation safety while providing semantic awareness capabilities that enhance environmental understanding beyond traditional sensor limitations, establishing a comprehensive framework for deploying advanced perception capabilities within existing navigation systems.

## CHAPTER 4

### SETUP AND METHODOLOGY

#### 4.1 Simulation Setup

The Clearpath Robotics ecosystem provides seamless simulation-to-reality deployment for autonomous mobile robots through integrated ROS2 packages. Its unified configuration approach uses a single `robot.yaml` file to drive the entire system from simulation setup through navigation deployment.

##### **Gazebo Core Package Architecture and Dependencies**

The Clearpath simulator uses Gazebo Fortress for physics-based robot simulation, supporting Clearpath Config and multiple simultaneous robots. Gazebo data bridges to ROS 2 through dedicated bridge nodes generated at launch, enabling identical configurations between physical and virtual robots. The simulation environment implements sophisticated physics modeling with ODE Physics Engine, providing maximal coordinate solver configuration, appropriate step sizes for high-fidelity simulation, high update rates for real-time performance, and contact modeling for realistic wheel-ground interaction. Advanced Rendering Pipeline utilizes modern Physically Based Rendering materials, scene broadcasting for multi-client synchronization, and server-side sensor rendering for headless automated testing.

## Sensor Plugin Integration

Comprehensive sensor simulation maintains identical APIs to real hardware through specialized plugins which is stated below:

LiDAR Simulation employs ray-casting algorithms for accurate range measurements, noise modeling for realistic characteristics, and intensity value simulation with material-based reflection properties, providing configurable resolution and update rates matching hardware specifications.

Camera Integration implements RGB-D plugins with synchronized color/depth streams, pointcloud generation matching sensor specifications, and IMU simulation. Plugins support configurable resolution, frame rates, and camera intrinsic parameters for accurate simulation.

GPS and IMU Simulation provides NavSat plugin integration with realistic noise modeling, coordinate frame transformations supporting UTM and geographic coordinates, and IMU plugins with configurable bias, noise, and update rates matching sensor specifications.

## Essential Clearpath Packages

The ecosystem comprises five fundamental package categories:

`clearpath.common` provides foundational shared utilities, core ROS2 message definitions, service interfaces, and utility functions, establishing common vocabulary for seamless inter-package communication.

`clearpath.config` implements central configuration management, parsing and validating the `robot.yaml` file with YAML schema validation, parameter translation services, and consistency checking across platform, sensor, and attachment specifications.

`clearpath_msgs` contains custom message definitions, service specifications, and action interfaces extending standard ROS2 interfaces with Clearpath-specific functionality for battery monitoring, platform status reporting, sensor integration, and diagnostics.

`clearpath_simulator` provides simulation-specific implementations and world definitions, containing Gazebo world files (warehouse, office, construction, pipeline, solar farm, orchard), physics parameter configurations, and plugin configurations ensuring consistent real-virtual robot behavior.

`uic02_description` (custom package) extends the standard Clearpath description framework with specialized components, custom sensor mounts (`sensor_arch`, `connector_plate`), and project-specific URDF modifications.

## System Integration Architecture

The complete system integration demonstrates sophisticated package interconnection:

Configuration Flow follows `robot.yaml` → `clearpath_config` → Generated Parameters → Launch System → Active Nodes, providing centralized configuration management, automatic parameter distribution, and consistent system behavior across simulation and real hardware.

Data Flow Architecture implements sensor data → processing nodes → navigation stack → control commands → platform actuation, with comprehensive topic remapping, namespace management, and QoS configuration ensuring reliable data transmission.

Launch Orchestration provides hierarchical launch file organization, dependency management ensuring proper startup sequences, and parameter inheritance enabling flexible system configuration for different operational scenarios.

TABLE I: Key Packages and Their Roles for `ros2 launch clearpath_gz simulation.launch.py`

Package	Role
<code>clearpath_common</code>	Shared utilities and definitions across Clearpath packages.
<code>clearpath_config</code>	Stores configuration files ( <code>robot.yaml</code> , ROS parameters).
<code>clearpath_control</code>	Control nodes (like velocity controllers) for mobile base.
<code>clearpath_description</code>	URDF/XACRO description of A200 robot (meshes, joints).
<code>clearpath_generator_common</code>	Common code used to generate robot descriptions dynamically.
<code>clearpath_generator_gz</code>	Tools to auto-generate Gazebo-compatible SDF files from URDF/XACRO.
<code>clearpath_gz</code>	Launch files and plugins for Gazebo simulation integration.
<code>clearpath_simulator</code>	Packages simulation scenarios and world files.
<code>clearpath_platform_description</code>	Platform-specific hardware description (battery, motors, wheels).
<code>clearpath_sensors_description</code>	Sensor-specific URDF parts (e.g., lidar mounts, cameras).
<code>clearpath_mounts_description</code>	Mounting plates, sensor arms, etc., 3D mesh and URDF definitions.
<code>clearpath_viz</code>	RViz configurations for visualization of robot state.

## Workspace Creation and Package Integration

It is possible to visualize and communicate with the robot using any off-board computer. To securely test any algorithms before deploying them to the real robot, the robot can also be replicated on an offboard computer.

Ubuntu serves as ROS 2 Humble’s main operating system. Using the right Ubuntu LTS version is strongly advised, even though other operating systems are supported. For the offboard computer, Ubuntu Desktop should be installed.

**ROS 2 Humble Installation:** To install ROS 2 Humble from debian packages, follow the specified instructions in the link <https://docs.ros.org/en/humble/Installation.html>.

After installing ROS2 Humble, the Clearpath Desktop metapackage will be installed when we install the Clearpath simulator, but before that we have to create the setup folder.

**Setup Folder Configuration:** The off-board computer has to have a copy of the `robot.yaml` file to generate the `setup.bash` file of the robot. More details can be found at [https://docs.clearpathrobotics.com/docs/ros2humble/ros/installation/offboard\\_pc](https://docs.clearpathrobotics.com/docs/ros2humble/ros/installation/offboard_pc).

#### 4.1.1 Modifying the robot.yaml file

The Clearpath Robot Configuration YAML serves as the single source of truth for the entire robotic system, enabling comprehensive customization of sensors, mounting structures, and operational parameters through a unified configuration interface. This configuration file drives automatic generation of four critical file types: bash shell scripts, URDF description files, launch files, and parameter files. The generator system parses the `robot.yaml` to determine platform specifications, sensor configurations, and custom accessories, subsequently creating all necessary files for robot operation.

Clearpath's architecture supports extensive modification capabilities across five major configuration sections: system-level information including networking and middleware settings, platform-level configurations for robot-specific parameters, URDF links for custom mechanical components, predefined mounting structures for sensor attachment, and sensor definitions from Clearpath's supported inventory. Upon configuration changes, running `sudo systemctl`

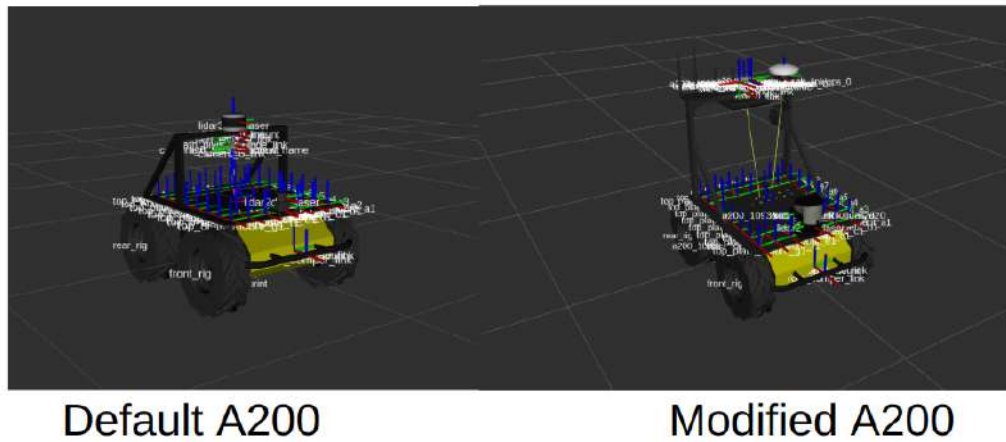


Figure 15: Modifying the robot.yaml to match the real robot

`restart clearpath-robot.service` triggers the generator system to rebuild all operational files, ensuring consistency between configuration and runtime behavior.

In this setup, several critical modifications were accommodated to match the real robot. The robot identity was updated with serial number `a200-1093`, automatically propagating through system namespace (`a200_1093`), hostname (`cpr-a200-1093`), and all generated files. A multi-host network architecture was implemented.

The ROS2 middleware configuration was optimized with server-based discovery (`rmw_fast-rttps-cpp`) to reduce network traffic and improve startup times in the multi-machine environment. Platform velocity parameters were set conservatively with maximum linear velocity of 0.25 m/s and angular velocity of .25 rad/s.

Sensor configuration modifications included upgrading the LiDAR system from default Hokuyo UST-10LX to Sick LMS1xx providing superior performance with 270° field of view,

25 Hz scanning frequency, and  $0.25^\circ$  angular resolution. The Intel RealSense camera was upgraded from D435 to D435i model with integrated IMU, with resolution optimized to  $320 \times 240$  pixels and FFMPEG compression enabled, reducing computational load and bandwidth by 75% while maintaining adequate quality for vision processing. GPS and IMU integration was implemented through SwiftNav Duro and Microstrain components with optimized positioning for high-precision navigation and sensor fusion.

Custom mechanical integration was achieved by adding mesh definitions for sensor arch and connector plate components, linking STL files from the custom `uic02_description` package to automatically incorporate 3D-printed mounting hardware into the robot's URDF. The custom workspace path was added to integrate project-specific packages, including perception nodes, modified navigation parameters, and sensor fusion algorithms. These modifications collectively enable automated generation of a complete system configuration that seamlessly integrates with Clearpath's standard navigation framework, demonstrating the flexibility of the unified configuration approach.

#### **4.1.2 Installing Clearpath Simulator**

It is easy to install the ClearPath simulator by following the steps mentioned in <https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/simulator/install>, but make sure to select ROS Humble.



### 4.1.3 Simulation Execution

Following successful simulator installation, the simulation environment can be initiated.

#### **Launching the Simulator**

The simulation launch files are located within the clearpath-gz package, which is installed with the clearpath-simulator meta-package. The primary launch command responsible for initializing the simulation world and deploying the robot model is

```
ros2 launch clearpath_gz simulation.launch.py
```

More details on how to simulate other worlds and also simulate with RViz can be found on <https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/simulator/simulate>.

#### **Simulation World Environments**

Six distinct world files were utilized to validate the panoptic segmentation navigation system across diverse environmental conditions:

- **Pipeline:** Industrial inspection environment featuring pipeline infrastructure, water features, and base station equipment for testing navigation around complex geometric structures.
- **Solar Farm:** Agricultural setting with solar panel arrays and open terrain, providing challenges for navigating between structured obstacles and natural ground surfaces.
- **Construction:** Office construction site containing building materials, equipment, and partially completed structures to evaluate performance in dynamic, cluttered environments.

- **Office:** Indoor office environment with furniture, corridors, and confined spaces for testing navigation precision in structured indoor settings.
- **Orchard:** Agricultural orchard featuring tree trunks, foliage, and natural terrain variations to assess terrain classification and navigation through vegetation obstacles.
- **Warehouse:** Industrial warehouse with shelving units, barriers, furniture, and simulated human figures to evaluate obstacle detection and avoidance in complex indoor scenarios.



Figure 16: 6 worlds that were used to do simulated experiments.

These environments collectively provide comprehensive testing scenarios encompassing outdoor terrain navigation, structured obstacle avoidance, vegetation classification, and mixed indoor-outdoor navigation challenges essential for validating the panoptic segmentation approach and also testing the motion planning in different scenarios.

**Controlling the Robot:** The simulated robot can be driven around in the same way as the real robot. The Teleop GUI plugin in Gazebo can be used. This plugin interface is automatically displayed when Gazebo starts running. Be sure to use the robot namespace, by setting the topic to `<namespace>/cmd_vel`. We can also set the maximum forward velocities and yaw, then specify the preferred control method.

The `clearpath_nav2_demos` package provides launch files and configuration settings for using Nav2 and `slam_toolbox`. These open-source navigation frameworks enable Clearpath A200 Husky to perform mapping operations and execute autonomous navigation tasks.

#### 4.1.4 SLAM Implementation and Integration

The `Clearpath_SLAM` is commonly used to process the laser scan data from a 2D LIDAR and the odometry data of the robot. Here, for `clearpath_slam`, it uses the SLAM Toolbox to map the environment. The tutorial can be found on [https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/navigation\\_demos/slam](https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/navigation_demos/slam).

This will use the Clearpath Gazebo simulator and will also work on the real robot which will be shown in the next section.

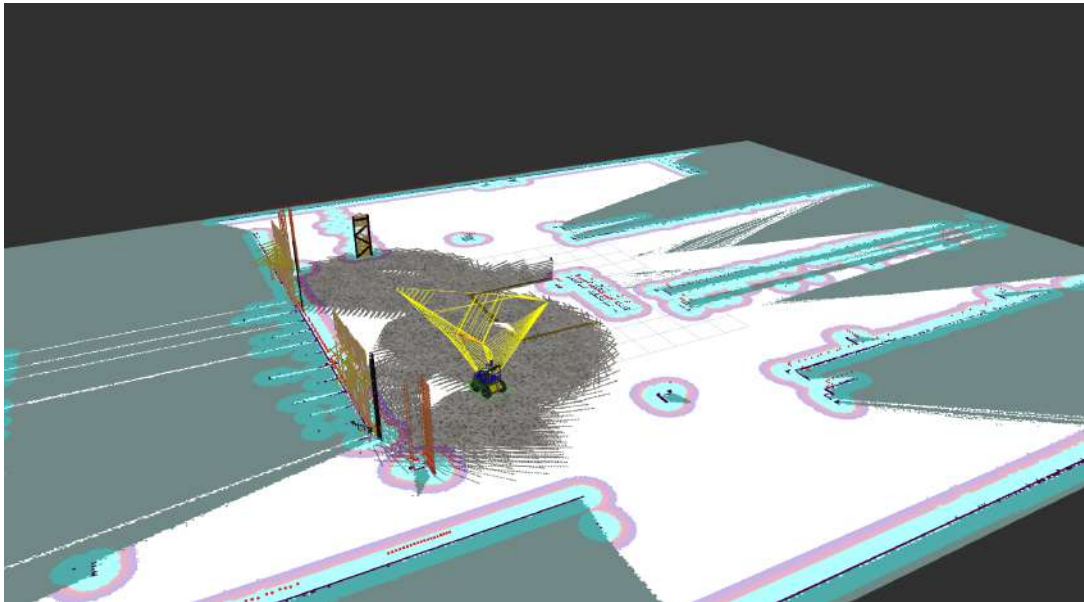


Figure 17: Launching Clearpath\_SLAM

#### 4.1.5 Nav2 Implementation and Integration

The `clearpath_nav2_demos` package provides the Nav2 package standard configuration files for compatible navigation control systems, path planning algorithms, and supporting tools. The tutorial to run Nav2 on the Clearpath simulator can be found on

[https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/navigation\\_demos/nav2](https://docs.clearpathrobotics.com/docs/ros2humble/ros/tutorials/navigation_demos/nav2).

This will work on a physical robot too, and will be shown in the next section.

#### 4.1.6 RTAB-Map Integration for 3D SLAM

Real-Time Appearance-Based Mapping (RTAB-Map) provides a comprehensive graph-based SLAM solution designed for large-scale and long-term mapping applications. Unlike traditional

```
# Update package lists
sudo apt update

# Install RTAB-Map desktop application
sudo apt install rtabmap

# Install RTAB-Map ROS2 packages (for ROS integration)
sudo apt install ros-humble-rtabmap-ros

# Install additional RTAB-Map tools and libraries
sudo apt install rtabmap-tools librtabmap-dev
```

Figure 18: Terminal Commands to install rtabmap

2D SLAM approaches that rely primarily on geometric features from laser range data, RTAB-Map employs visual appearance-based loop closure detection combined with multi-modal sensor fusion to achieve robust localization and mapping in complex environments.

To install RTAB-Map on Ubuntu 22.04, follow the installation steps shown in the terminal commands in the above figure. The installation process includes updating package lists, installing the RTAB-Map desktop application, ROS2 integration packages for Humble distribution, and additional development tools and libraries required for the complete RTAB-Map functionality and ROS integration.

When deployed on the Clearpath A200 Husky platform, RTABMap typically utilizes the Intel RealSense D435i camera in combination with the platform's existing 2D LiDAR sensors. The RealSense D435i provides both RGB color information and depth measurements, enabling the algorithm to create rich 3D point clouds that capture fine environmental details. The integration of 2D LiDAR data enhances the robustness of the mapping process by providing

high-accuracy range measurements in the horizontal plane, creating a sensor fusion approach that leverages the complementary strengths of different sensing modalities.

The implementation employs carefully tuned parameters optimized for outdoor navigation scenarios with the Clearpath Husky platform. The fundamental configuration emphasizes 2D operation while maintaining 3D environmental understanding, with the 2D constraint parameter constraining registration to 2D motion appropriate for ground vehicles operating on relatively flat terrain. The system integrates multiple sensor modalities through configured topic remappings, including ICP odometry for improved motion estimation, RGB-D camera data for visual features, and laser scan data for geometric constraints. The configuration utilizes ICP odometry by combining wheel odometry with geometric scan matching for enhanced accuracy in challenging terrain conditions.

RTAB-Map operates in two distinct modes: SLAM mode for initial mapping and localization mode for navigation within previously created maps. During initial mapping, RTAB-Map operates in full SLAM mode with incremental memory management, continuously building and refining the map while estimating robot pose, with loop closure detection refining previously mapped areas based on new observations. For navigation within known environments, RTAB-Map switches to localization mode, loading a pre-existing map database and focusing exclusively on pose estimation without map modification. The working memory initialization enables immediate global localization capability by initializing with all previously learned locations.

The system employs sophisticated database management to maintain persistent maps across multiple operational sessions, storing maps in a database file that enables the robot to take ad-

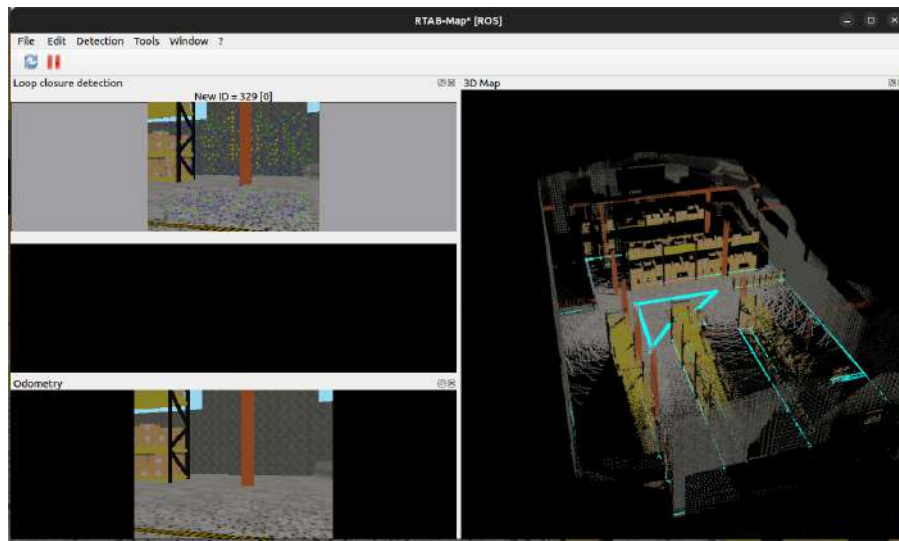


Figure 19: RTAB-Map\_RViz

vantage of previously mapped areas while continuing to expand environmental knowledge in unmapped regions. This capability is crucial for practical autonomous navigation systems operating over extended time periods and multiple deployment sessions. Performance optimizations include ICP odometry, providing improved motion estimation, point-to-plane complexity parameters balancing computational efficiency with registration accuracy, and real-time operation capabilities, maintaining sufficient precision for autonomous navigation.

### **To Saved the 3D-map**

Saved the map by going to the Edit option in the top-left section of the RTAB-map\_RViz and then select Download Cloud and then select global map optimized option and wait for it to saved the map.

To view the database after completing the mapping:

```
rtabmap-databaseViewer ~/.ros/rtabmap.db
```

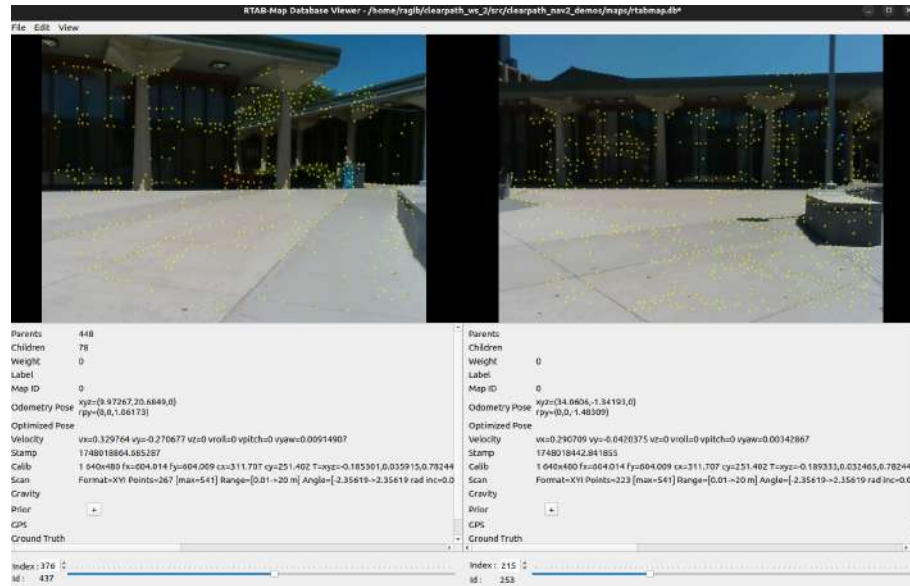


Figure 20: Viewing the Saved RTAB-map\_database

To reuse the saved map from the database, and also to continue future mapping:

```
ros2 launch clearpath_nav2_demos a200_1093_demo_real.launch.py localization:=true
```

The persistence of the database enables the robot to take advantage of previously mapped areas while continuing to expand environmental knowledge in unmapped regions. This capability is crucial for practical autonomous navigation systems that must operate over extended time periods and multiple deployment sessions.



The integrated RTAB-Map and panoptic segmentation system demonstrates the synergy between geometric and semantic environmental understanding, enabling robust autonomous navigation in complex outdoor environments where neither approach alone would suffice. This multi-modal integration represents a significant advancement in autonomous navigation capabilities for unmanned ground vehicles operating in challenging outdoor scenarios, combining the geometric accuracy of SLAM with the semantic intelligence of computer vision for comprehensive environmental perception.

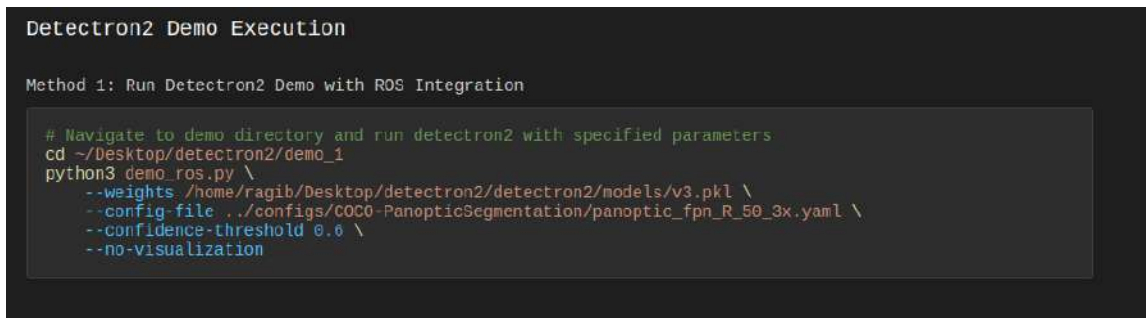
#### **4.1.7 Integrated Launch System Architecture**

##### **Integrated Launch System Architecture**

The implementation employs a comprehensive launch file architecture that orchestrates the entire panoptic segmentation navigation system through a single entry point, integrating a simulation environment, 3D SLAM, autonomous navigation, visualization, and computer vision processing into a cohesive system. The launch file follows a modular design pattern where each major system component is encapsulated within its own launch file, then included and configured through the master launcher, providing component isolation for debugging, independent parameter configuration for each subsystem, scalable architecture for adding new capabilities, and maintainable code structure separating concerns across functional domains.

##### **System Launch and Operation**

The integrated system can be launched through a coordinated sequence that initializes all components in the proper order. While the launch file architecture supports single-command



```

Detectron2 Demo Execution

Method 1: Run Detectron2 Demo with ROS Integration

# Navigate to demo directory and run detectron2 with specified parameters
cd ~/Desktop/detectron2/demo_1
python3 demo_ros.py \
  --weights /home/ragib/Desktop/detectron2/detectron2/models/v3.pkl \
  --config-file ../configs/COCO-PanopticSegmentation/panoptic_fpn_R_50_3x.yaml \
  --confidence-threshold 0.6 \
  --no-visualization

```

Figure 21: Launching Detectron2

deployment, the panoptic segmentation component is typically run separately to enable flexible model switching and enhanced debugging capabilities.

### Primary System Launch:

The core navigation system, including simulation, RTAB-Map SLAM, and Nav2 navigation, is launched through the master launch file:

```
ros2 launch clearpath_nav2_demos a200_1093_demo.launch.py
```

### Panoptic Segmentation Node:

In a separate terminal, the vision-based segmentation node is launched independently to provide vision-based obstacle detection:

This modular launch approach provides several operational advantages: independent control over computer vision processing parameters, ability to restart the vision pipeline without affecting navigation, support for distributed processing across multiple compute platforms, and simplified debugging of individual system components.

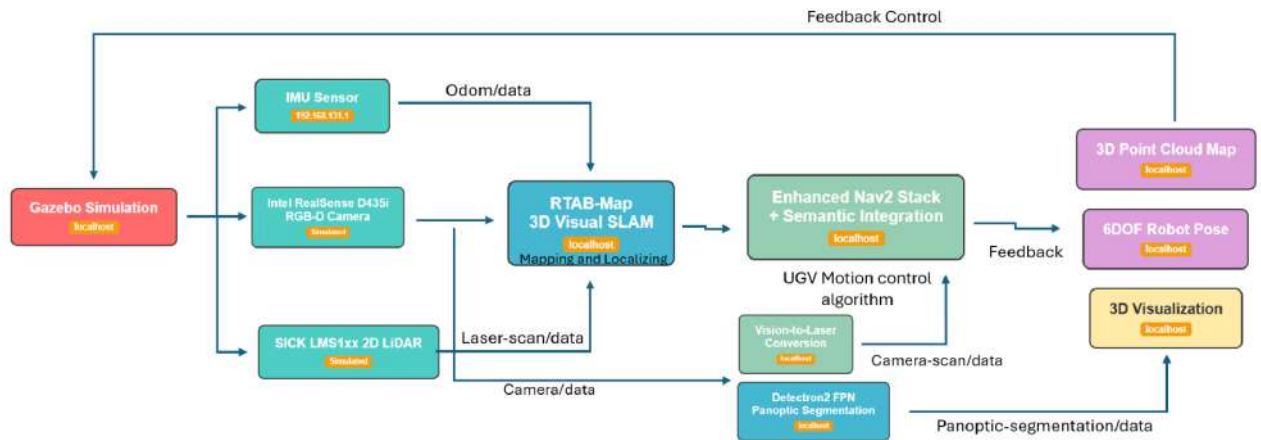


Figure 22: Complete Pipeline of the Integrated Launch System

### Operational Sequence

Upon launch, the system follows a coordinated initialization sequence:

1. **Simulation Environment:** Gazebo loads the specified world and spawns the Husky robot with configured sensors
2. **Sensor Data Flow:** Camera, LiDAR, GPS, and IMU data streams begin publishing to namespaced topics
3. **RTAB-Map SLAM:** Visual-inertial odometry and mapping begin processing RGB-D and LiDAR data
4. **Navigation Stack:** Nav2 components initialize costmaps and planning algorithms using sensor data

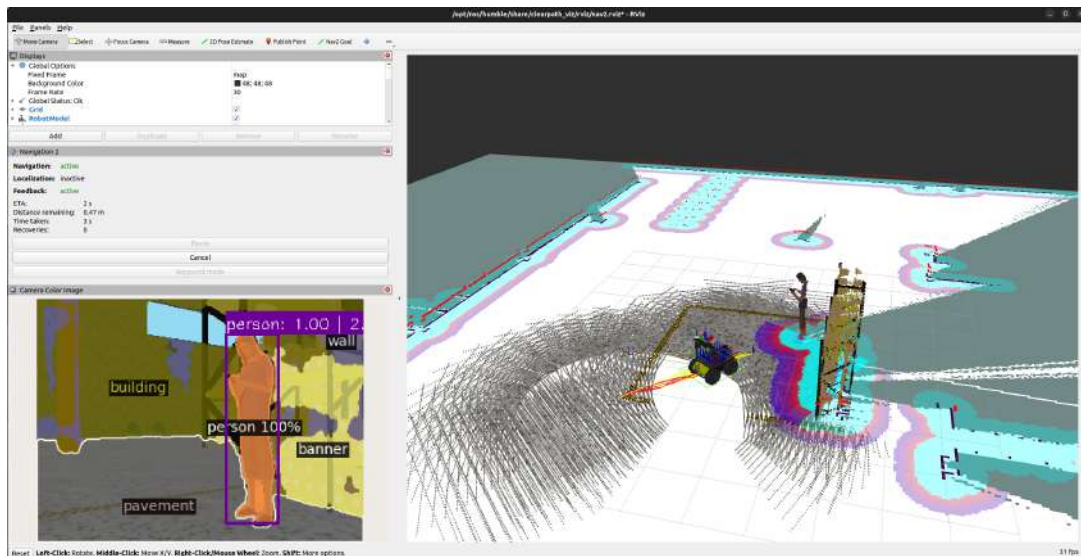


Figure 23: Autonomous Navigation Using Panoptic Segmentation and Nav2, where the panoptic segmentation is transferring both thing and stuff class data to Nav2 through Laser Scan data, which is then used to generate the Local and Global Costmaps.

5. **Panoptic Segmentation:** Computer vision processing begins with processing camera images and publishing vision-based laser scans
6. **Visualization:** RViz displays integrated sensor data, maps, navigation plans, and system status

#### 4.1.8 GPS-Based Navigation Integration with Nav2

The GPS navigation integration enables waypoint-based autonomous navigation by converting global GPS coordinates into local map frame targets that Nav2 can process through a coordinate transformation pipeline that bridges global positioning data with the robot's local navigation framework, enabling autonomous operation across large outdoor areas without re-

quiring pre-existing maps. The GPS navigation system utilizes Universal Transverse Mercator (UTM) coordinate projection to convert between GPS coordinates and the robot's local map frame, establishing a map origin using GPS coordinates as the reference point for all subsequent coordinate transformations. When a GPS target is received, the system converts it to UTM coordinates using the `utm.from_latlon()` function, subtracts the map origin UTM coordinates to generate local map coordinates, and creates a `PoseStamped` goal with the map frame reference that Nav2 can use for path planning.

The GPS navigation node serves as the primary coordinator, subscribing to GPS target coordinates and map origin definitions while interfacing with Nav2's navigation action server through the standard navigation action interface, ensuring compatibility with all existing navigation behaviors, recovery actions, and path planning algorithms. The conversion process ensures that GPS coordinates are accurately transformed into the robot's local coordinate system, enabling precise navigation to global waypoints using the existing Nav2 navigation stack for comprehensive autonomous navigation across large outdoor environments.

## **GPS Navigation Operation Workflow**

The complete GPS navigation system enables autonomous waypoint following through a coordinated sequence of system initialization, coordinate transformation, and navigation execution.

### **System Initialization and Setup**

GPS navigation requires the integrated navigation system to be operational before GPS targets can be processed:

## GPS Navigation System Setup

### Method 1: Launch Integrated Navigation System

```
# Launch the complete navigation system for A200_1093
ros2 launch clearpath_nav2_demos a200_1093_demo.launch.py
```

### Method 2: Initialize GPS Navigation Node

```
# Start the GPS navigation simulation node
ros2 run clearpath_nav2_demos gps_navigation_sim.py
```

### Method 3: Set Map Origin GPS Coordinate

```
# Establish coordinate transformation reference point
ros2 topic pub -1 /map_origin_gps sensor_msgs/msg/NavSatFix \
  "{header: {frame_id: 'map'}, latitude: 43.5000, longitude: -80.5000}"
```

*The map origin establishment is critical as it defines the coordinate transformation reference point for all subsequent GPS targets.*

### Method 4: GPS Target Navigation

```
# Send single GPS target using Python script
python3 send_gps_target.py 41.87179475 -87.64911466
```

```
# Or publish directly to topic
ros2 topic pub -1 /target_gps_coord sensor_msgs/msg/NavSatFix \
  "{header: {frame_id: 'map'}, latitude: 41.87179475, longitude: -87.64911466}"
```

### Method 5: Automated Waypoint Sequence Execution

```
# Execute predefined waypoint sequence for complex navigation missions
bash sim_gps_waypoints_fixed.sh
```

Figure 24: Terminal Commands for launching the GPS-based Navigation

The waypoint script automatically coordinates GPS target transmission with navigation completion detection, enabling autonomous execution of complex route-following missions.

### **Position Monitoring and Feedback:**

The system includes comprehensive position monitoring capabilities through utility scripts, which can extract the robot’s current position from multiple sources, including direct GPS, odometry, pose estimates, and coordinate transforms:

```
# Get the current robot GPS position

python3 get_single_gps.py
```

## **4.2 Hardware Setup and Implementation**

The hardware implementation of the panoptic segmentation navigation system represents a significant engineering challenge, requiring the integration of multiple computational platforms, sophisticated sensor arrays, and distributed processing architectures. Unlike simulation environments where components operate within controlled software frameworks, the physical implementation demands careful consideration of power management, network topology, thermal constraints, and real-time performance optimization across heterogeneous computing platforms.

### **4.2.1 Multi-Host Distributed Computing Architecture**

The hardware system employs a sophisticated three-computer distributed architecture designed to leverage the computational strengths of different platforms while maintaining seamless system integration through ROS2’s multi-machine capabilities.

### **Primary Robot Computer**

The main robot computer serves as the central coordination hub, managing navigation, SLAM, and system-level operations. This computer handles navigation stack execution, visual SLAM processing, system service management through the robot service framework, and coordination of multi-host communication through ROS2 discovery server functionality.

### **NVidia Jetson**

The NVIDIA embedded computing platform functions as a dedicated AI acceleration platform, specifically optimized for real-time panoptic segmentation processing. Operating within the network infrastructure, this unit provides GPU-accelerated computer vision inference, real-time vision-to-laser scan conversion, optimized CUDA kernel execution for computer vision workloads, and seamless ROS2 integration with the primary navigation system.

### **Off-Board Operator Station**

The operator workstation enables remote monitoring and control, facilitating system development and operational oversight. This platform provides visualization-based system monitoring, remote system monitoring and debugging capabilities, a development environment for algorithm refinement, and wireless coordination of robotic operations.

### **Network Topology and Configuration**

The distributed architecture operates over a high-performance Gigabit Ethernet backbone with carefully engineered QoS parameters:

#### **System Network Configuration:**

Primary Robot Computer:     192.168.131.1   (Navigation & SLAM)



Jetson AGX Orin:	192.168.131.2	(AI Processing)
Operator Station:	192.168.131.195	(Visualization)
SICK LMS1xx LiDAR:	192.168.131.20	(Sensor Data)

The network employs ROS2's discovery server architecture to optimize communication efficiency and reduce broadcast traffic in the multi-machine environment.

#### **4.2.2 Robot.yaml Multi-Host Configuration and Data Transport**

The hardware implementation required extensive modifications to the robot.yaml configuration to support distributed computing, optimized data transport, and multi-host sensor integration.

##### **Multi-Host System Definition**

The robot.yaml configuration establishes the foundational network architecture for distributed operation.

The ROS2 middleware configuration was optimized with server-based discovery (`rmw_fastdds.cpp`) to reduce network traffic and improve startup times in the multi-machine environment. Correct IP from 'ip a' output of the off-board computer and then edit that in the robot.yaml file as a new hostname.

##### **To edit the robot.yaml:**

```
sudo nano /etc/clearpath/robot.yaml
```

The robot.yaml file should look similar to below:

```
system:
```

```

hosts:

- hostname: cpr-a200-1093      # Primary robot computer

  ip: 192.168.131.1

- hostname: cpr-a200-1093b    # Jetson AGX Orin

  ip: 192.168.131.2

- hostname: ragib-Lenovo-Slim-Pro-7-14ARP8 # Operator station

  ip: 192.168.131.195

```

This configuration enables automatic topic routing, service discovery across machines, and consistent namespace management throughout the distributed system.

After modifying the `robot.yaml` file with the proper hostname and IP, you have to restart the system.

**For restarting the system:**

```
sudo systemctl restart clearpath-robot.service
```

This distributed sensor architecture enables optimal computational resource utilization while maintaining synchronized data acquisition across the robotic system.

## Custom Camera Mount Design and Fabrication

The integration of the Intel RealSense D435i camera with the Clearpath Husky A200 platform required a custom mounting solution, as standard Clearpath mounting options did not provide the necessary geometric constraints for proper camera placement relative to the robot's coordinate frame and sensor arch configuration.

### **Design Requirements and Features:**

- Precise 0.2-meter vertical offset from base link frame with mechanical stability
- Compatibility with `uic02_description` sensor arch structure
- Parametric CAD design with RealSense-compatible mounting bosses and reinforcement ribs
- Cable management features for USB 3.0 protection

### **Manufacturing and Installation:**

- 3D printed using Prusa printer with PLA+ filament (0.2mm layers, 20% infill, three-perimeter walls)
- Integration through custom sensor arch assembly with corrosion-resistant hardware
- Camera field-of-view clearance and coordinate frame validation using ROS transform utilities

The custom mount successfully enabled stable camera operation throughout all experimental scenarios, demonstrating effective integration of the design and manufacturing approach for research-specific hardware requirements.

#### **4.2.3 Hardware Pipeline Data Flow and Processing**

The hardware implementation implements a sophisticated data flow architecture that coordinates sensor acquisition, AI processing, and navigation control across multiple computational platforms. The system orchestrates data flow from multiple sensors through optimized pathways:

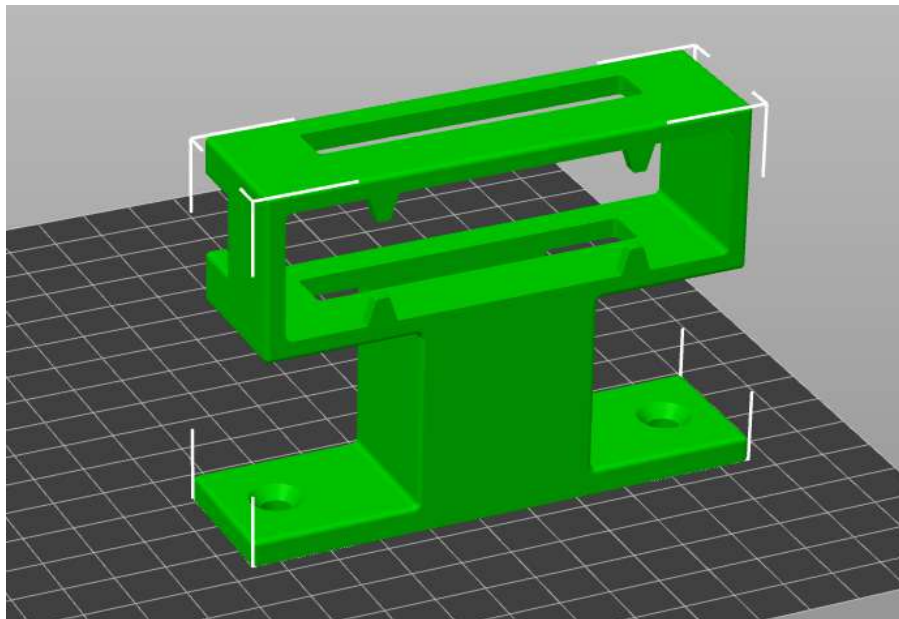


Figure 25: 3D-printing the Camera Mount

1. **Intel RealSense RGB-D Camera (Primary Computer):** The Intel RealSense camera is connected directly to the primary computer. RGB-D data acquisition is transferred at an optimized resolution and frame rate, immediate GPU memory transfer for computer vision processing, concurrent video compression for network transmission, and local storage for AI inference pipeline.

camera:

- model: intel\_realsense

host: cpr-a200-1093      # Main Computer-connected camera

2. **2D LiDAR Sensor (Primary Computer):** 2D laser scan data at high update rate, direct Ethernet communication to primary computer, immediate integration with visual SLAM system, and costmap layer fusion with vision-derived obstacles.

```
lidar2d:
```

```
- model: sick_lms1xx
```

```
host: 192.168.131.20      # Direct sensor IP
```

3. **GPS and IMU Systems (Primary Computer):** Global positioning data for large-scale navigation, inertial measurement for motion estimation, sensor fusion with visual odometry, and coordinate frame management for multi-modal SLAM.

```
gps:
```

```
- model: swiftnav_duro
```

```
host: cpr-a200-1093      # Main computer GPS
```

### **Distributed AI Processing Architecture**

The panoptic segmentation pipeline leverages distributed computing for optimal performance:

Data Flow: RGB-D Acquisition  $\rightarrow$  GPU Processing

(4.1)

$\rightarrow$  Network Transport  $\rightarrow$  Navigation Integration

**Jetson AGX Orin:**

- Camera data capture
- Detectron2 inference
- Vision-to-scan conversion
- Network data transmission

**Primary Computer:**

- Receive processed laser scan
- Integrate with Nav2 costmaps
- Path planning and control
- SLAM and localization

This architecture achieves substantial processing performance while maintaining real-time navigation responsiveness.

### **Network Communication Optimization**

The system employs several strategies to optimize inter-machine communication. ROS2 Discovery Server configuration reduces network discovery traffic significantly, QoS profile optimization ensures reliable data transmission with minimal latency, topic remapping enables seamless integration across distributed nodes, and bandwidth management prioritizes critical navigation data over visualization streams.

#### **4.2.4 Remote Access and Computer Connections**

The distributed computing architecture of the Clearpath Husky A200 platform requires remote access to both the primary robot computer and the secondary NVIDIA Jetson compute unit. Remote connections are established through SSH (Secure Shell) protocol, enabling command-line access for system configuration, software deployment, and operational control.

**Primary Computer Connection:** To access the main robot computer responsible for navigation and control functions, establish an SSH connection using:

```
ssh administrator@192.168.131.1
```

**Secondary Computer Connection:** To access the NVIDIA Jetson compute unit dedicated to panoptic segmentation processing, connect via:

```
ssh administrator@192.168.131.2
```

Both connections utilize the same administrator credentials (username: administrator, password: clearpath) and provide full command-line access to their respective systems. The distributed architecture enables parallel processing where computationally intensive vision tasks execute on the Jetson while real-time navigation functions operate on the primary computer, ensuring optimal resource utilization and system performance.

#### 4.2.5 Clearpath-SLAM and Nav2 Hardware Deployment

To deploy the `clearpath_slam` and `clearpath_nav2` systems on the physical robot hardware, the same procedures outlined in the simulation setup section apply with minor command modifications. The launch commands are executed on the primary robot computer (192.168.131.1) and are adapted for hardware operation by specifying the correct workspace path and disabling simulation time synchronization.

**SLAM Operation:** For simultaneous localization and mapping on the physical robot, execute the following commands:

```
1. ros2 launch clearpath_viz view_navigation.launch.py namespace:=a200_1093
```

```
2. ros2 launch clearpath_nav2_demos slam.launch.py setup_path
   :=$HOME/colcon_ws/clearpath/ use_sim_time:=false
```

**Nav2 Navigation:** For autonomous navigation using pre-existing maps, launch the navigation stack with:

```
ros2 launch clearpath_nav2_demos nav2.launch.py setup_path
   :=$HOME/colcon_ws/clearpath/ use_sim_time:=false
```

The key modifications from simulation commands include setting `use_sim_time:=false` to utilize real-world timestamps from the robot's sensors and specifying the correct `setup_path` pointing to the hardware workspace configuration. All other operational procedures, including initial pose estimation, goal setting through RViz, and navigation behaviors, remain identical to the simulation environment, ensuring seamless transition from virtual testing to physical deployment.

#### **4.2.6 Modified Vision + Lidar Based Launch System Coordination and Startup Sequence**

The hardware system requires careful coordination of startup sequences across multiple machines to ensure proper initialization and inter-component communication.

**Primary Launch Sequence:** The system follows a structured three-stage startup process:



### Stage 1: Off-Board Visualization Initialization

```
# On operator workstation (192.168.131.195)

ros2 launch clearpath_nav2_demos camera_nav_viz.launch.py
```

This command initializes the visualization infrastructure, including navigation interface visualization, visual SLAM visualization components, compressed video stream decoding, and remote system monitoring capabilities.

### Stage 2: AI Processing Platform Startup

```
# On Jetson AGX Orin (192.168.131.2)

cd ~/detectron2/demo_1

python3 demo_ros.py \

    --weights /home/administrator/detectron2/detectron2/model/v3.pkl \

    --config-file ../configs/COCO-PanopticSegmentation/panoptic_fpn_R_50_3x.yaml \

    --confidence-threshold 0.6 --no-visualization
```

The computer vision node initialization includes model loading and GPU memory allocation, ROS2 topic publisher establishment, vision-to-laser scan conversion pipeline activation, and performance monitoring for real-time operation.

### Stage 3: Primary Navigation System Activation

```
# On primary robot computer (192.168.131.1)

ros2 launch clearpath_nav2_demos a200_1093_demo_real.launch.py
```

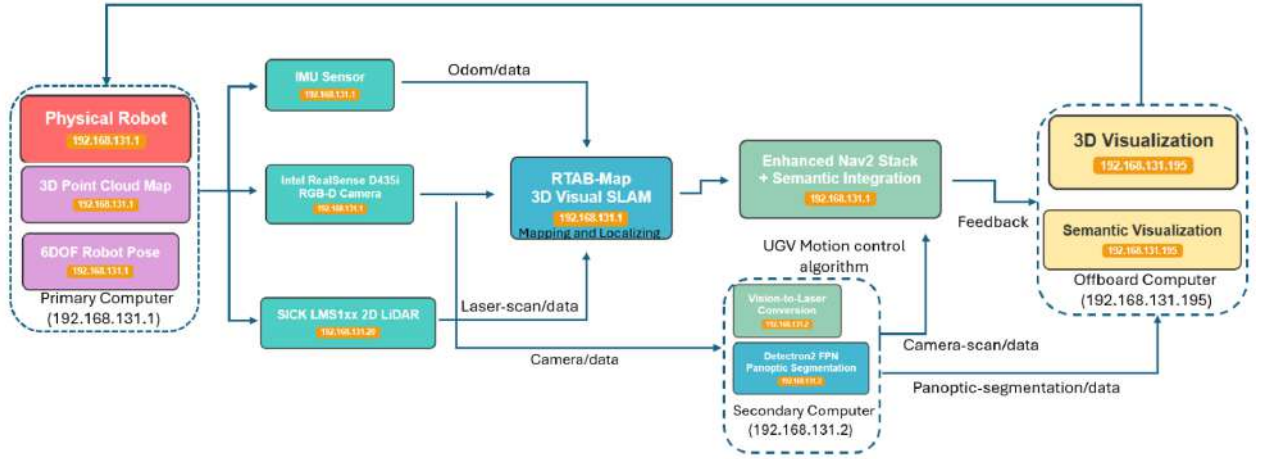


Figure 26: Complete Pipeline of the Modified Vision + Lidar Based Launch System

This launches the integrated navigation system comprising visual SLAM with 3D mapping capabilities, navigation stack with multi-sensor costmaps, sensor fusion and localization algorithms, and system coordination and safety monitoring.

### Startup Dependency Management

The visualization system initializes first to provide immediate feedback, the AI processing platform startup includes model preloading to minimize first-inference latency, the navigation system waits for computer vision readiness before integrating vision-based obstacles, and automatic health monitoring detects and reports component failures during startup.

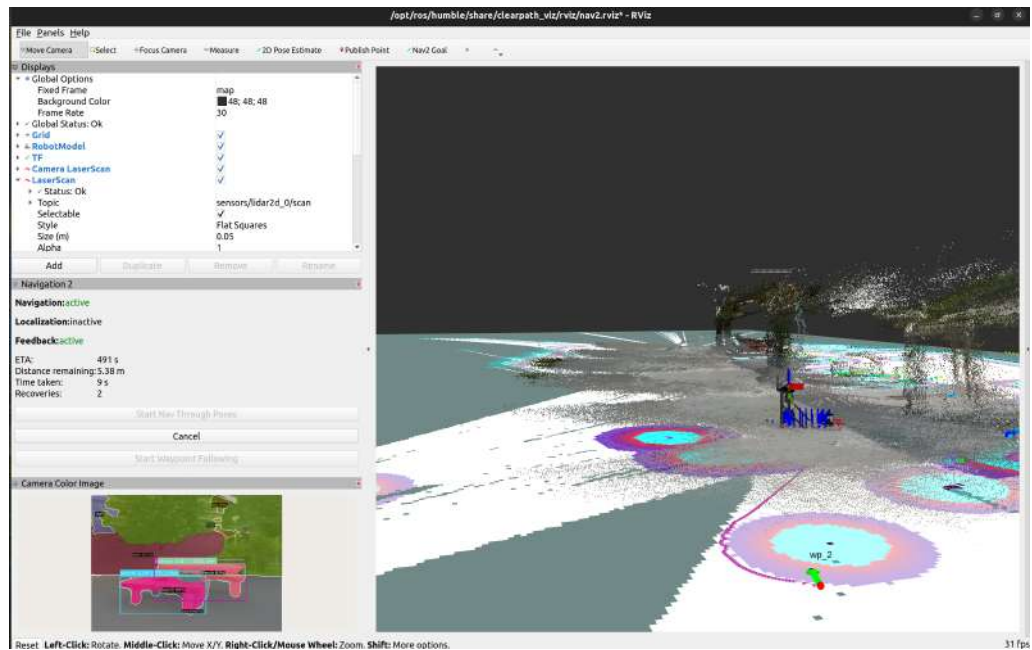


Figure 27: Rviz Visualization

#### 4.2.7 Off-Board Visualization and Remote Monitoring

The operator workstation provides comprehensive system visualization and monitoring capabilities through sophisticated ROS2-based interfaces that enable real-time system oversight and development support.

##### Multi-Modal Visualization Architecture

The camera navigation visualization system orchestrates several visualization components:

##### Navigation 3D SLAM Visualization:

The 3D SLAM visualization provides 3D point cloud display, loop closure detection indicators, pose graph visualization, and a database management interface for map persistence.

This provides real-time costmap visualization, path planning display, robot pose and odometry tracking, and a goal setting interface for manual navigation commands.

#### 4.2.8 System Integration Validation and Performance Monitoring

The hardware implementation includes comprehensive validation and monitoring capabilities to ensure reliable operation across the distributed computing architecture. Topic latency measurement ensures data transmission delays remain below critical thresholds, bandwidth utilization monitoring prevents network saturation during peak operation, discovery server health checks validate ROS2 communication infrastructure, and automatic failover mechanisms handle transient network interruptions. Key performance indicators are tracked across all platforms:

- **Embedded AI Platform Metrics:** GPU utilization monitoring, inference frequency tracking, memory usage monitoring, thermal status monitoring
- **Primary Computer Metrics:** CPU utilization for navigation tasks, memory usage for SLAM database, network throughput for sensor data, storage I/O for map persistence
- **System-Wide Metrics:** End-to-end latency from perception to control, navigation success rate and path efficiency, obstacle detection accuracy and false positive rates, system uptime and fault recovery statistics

#### **Operational Workflow Validation**

The three-stage launch process undergoes continuous validation:

1. **Visualization Readiness:** Operator workstation confirms visualization initialization and video stream connectivity

2. **AI Processing Verification:** Embedded platform validates model loading and begins publishing processed data
3. **Navigation Integration:** Primary computer confirms receipt of vision-based obstacles and initiates autonomous navigation

### **Fault Tolerance and Recovery**

The system implements several mechanisms for robust operation:

Graceful degradation to LiDAR-only navigation if vision processing fails, automatic restart of failed components with exponential backoff, comprehensive logging for post-incident analysis, and emergency stop mechanisms for safety-critical failures.

The hardware implementation successfully demonstrates the feasibility of deploying advanced AI-based navigation systems on practical robotic platforms, achieving real-time performance through careful engineering of distributed computing architectures, optimized data transport protocols, and robust system integration methodologies. This implementation serves as a foundation for practical deployment of panoptic segmentation-based navigation in real-world autonomous vehicle applications.

### **Advanced Video Compression Integration for Distributed Visualization**

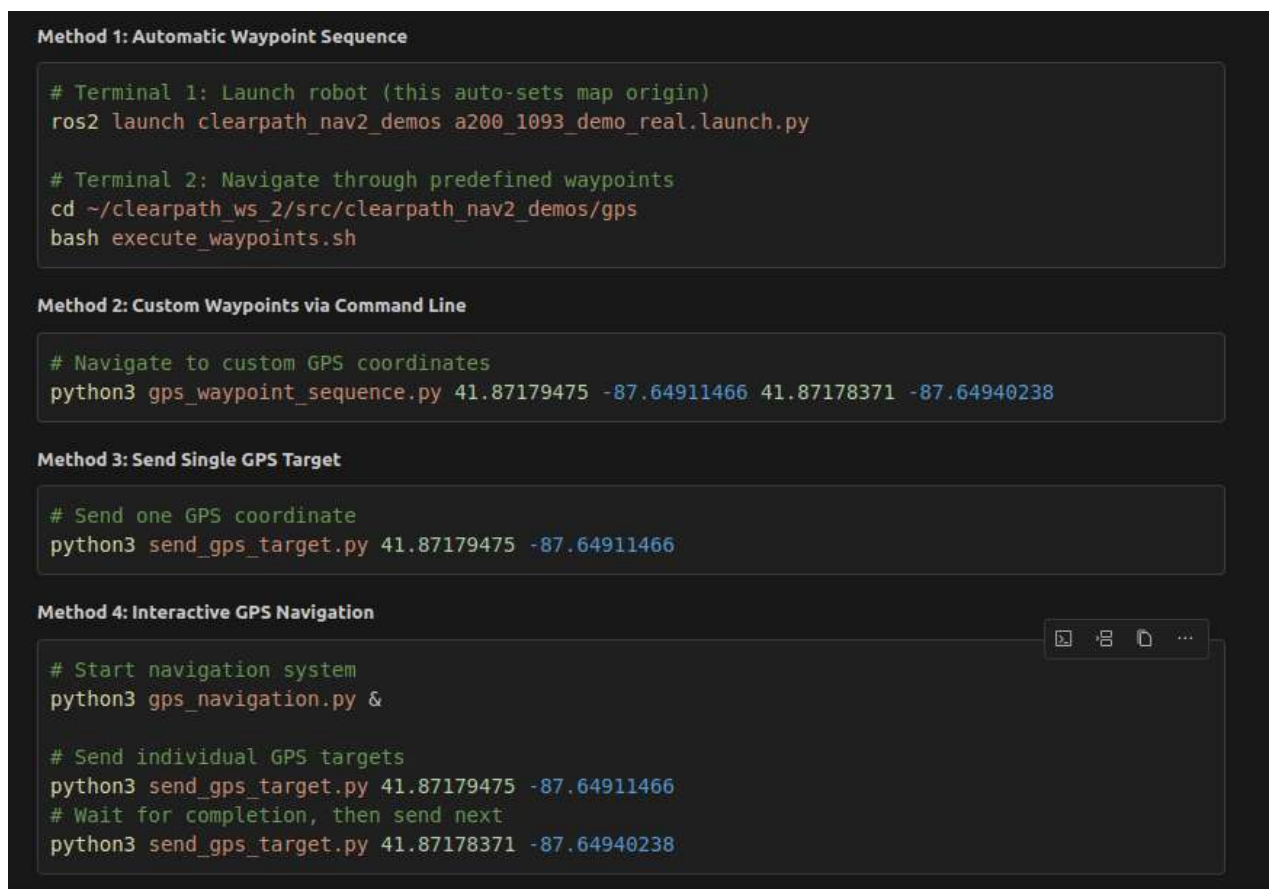
The camera navigation visualization system implements a two-stage decompression pipeline to address bandwidth challenges in the distributed computing environment. The first stage employs FFMPEG decompression using the image transport republish node, converting compressed camera streams (`/a200_1093/sensors/camera_0/intel_realsense/color/image_raw/ffmpeg`) to raw format (`/a200_1093/sensors/camera_0/color/decoded_image`) with optimized

parameters: best-effort reliability, single-frame history depth, 100ms maximum latency, and minimal buffering. The second stage provides direct image transport for display, remapping the decoded stream to the final panoptic visualization topic (`/a200_1093/sensors/camera_0/color/image_panoptic`) while maintaining best-effort reliability and single-frame processing. This pipeline ensures efficient bandwidth utilization across the distributed architecture while maintaining real-time visualization performance for the panoptic segmentation navigation system. This pipeline configuration achieves minimal video display latency while maintaining visual quality sufficient for system monitoring and debugging.

### **Integrated GPS Multi-Modal Navigation**

Just like in the simulation, for GPS navigation system follow the terminal commands mentioned in the image below and also be sure to launch this in the primary computer.

The system includes comprehensive position monitoring capabilities through utility scripts, which can extract the robot's current position from multiple sources, including direct GPS, odometry, pose estimates, and coordinate transforms.



```
Method 1: Automatic Waypoint Sequence  
  
# Terminal 1: Launch robot (this auto-sets map origin)  
ros2 launch clearpath_nav2_demos a200_1093_demo_real.launch.py  
  
# Terminal 2: Navigate through predefined waypoints  
cd ~/clearpath_ws_2/src/clearpath_nav2_demos/gps  
bash execute_waypoints.sh  
  
Method 2: Custom Waypoints via Command Line  
  
# Navigate to custom GPS coordinates  
python3 gps_waypoint_sequence.py 41.87179475 -87.64911466 41.87178371 -87.64940238  
  
Method 3: Send Single GPS Target  
  
# Send one GPS coordinate  
python3 send_gps_target.py 41.87179475 -87.64911466  
  
Method 4: Interactive GPS Navigation  
  
# Start navigation system  
python3 gps_navigation.py &  
  
# Send individual GPS targets  
python3 send_gps_target.py 41.87179475 -87.64911466  
# Wait for completion, then send next  
python3 send_gps_target.py 41.87178371 -87.64940238
```

Figure 28: GPS-Based Navigation Launch Commands

## CHAPTER 5

### RESULTS

#### 5.1 Panoptic Segmentation Class-Specific Performance Results

This section presents detailed class-specific performance analysis for both Cityscapes and COCO datasets, demonstrating the effectiveness of the sequential training methodology and selective optimization strategies. The results illustrate how domain-specific initialization followed by selective expansion training enhanced detection capabilities across navigation-critical object categories.

##### 5.1.1 Cityscapes Dataset Class Performance Results

The Cityscapes foundation training phase established robust performance across all 8 navigation-critical thing classes. The high-resolution urban scene training ( $2048 \times 1024$ ) enabled precise object localization and strong feature representations for structured environments.

The Cityscapes results demonstrate excellent performance on the most critical navigation categories, with car detection achieving 47.312 AP and person detection reaching 29.896 AP. These strong foundational results established robust urban scene understanding capabilities essential for autonomous navigation safety.

##### 5.1.2 COCO Dataset Relevant Class Performance Results

Following the selective COCO expansion training, the model demonstrated performance across navigation-relevant classes while maintaining computational efficiency through strategic



TABLE II: Cityscapes Dataset Class-Specific Performance Results

Category	Box AP	Mask AP	PQ	Navigation Priority
Car	47.312	44.180	52.430	Critical
Person	29.896	27.450	35.120	Critical
Bicycle	18.568	16.720	24.890	High
Bus	16.602	15.340	22.150	High
Rider	15.223	14.180	19.430	High
Truck	12.175	11.250	17.680	High
Motorcycle	8.896	8.120	12.340	Medium
Train	1.080	0.950	1.420	Low
<b>Mean AP</b>	<b>18.716</b>	<b>17.274</b>	<b>23.183</b>	-

class filtering. The results show the model’s capability to handle diverse object configurations beyond structured urban environments.

\* Rider class is from Cityscapes but mapped to COCO person detection for consistency.

**Note:** Infrastructure and terrain classes are primarily stuff classes. Box AP and Mask AP values represent performance on instance-level annotations, where available, or pseudo-instance performance estimates derived from stuff segmentation quality.

#### Key Performance Insights:

- **Critical Classes:** Car and Person maintain excellent performance (33.695 and 17.150 AP respectively), ensuring navigation safety
- **Class Filtering Efficiency:** 33.75% computational reduction achieved by filtering 53 non-navigation-relevant classes
- **Terrain Detection:** Special priority given to terrain obstacles (grass, earth, field) with dedicated detection logic, achieving 10.297 mean AP

TABLE III: COCO Dataset Navigation-Relevant Class Performance Results - Part 1: Active Detection Classes

Category	Box AP	Mask AP	PQ	Navigation Priority	Detection Status
<b>Critical Navigation Classes</b>					
Car	33.695	31.245	38.520	Critical	Active
Person	17.150	15.830	21.340	Critical	Active
Truck	10.570	9.820	13.450	Critical	Active
Bus	10.186	9.450	12.890	Critical	Active
Motorcycle	9.082	8.340	11.250	High	Active
Bicycle	3.421	3.120	4.850	High	Active
Rider*	4.473	4.120	5.680	High	Active
<b>High Priority Infrastructure</b>					
Traffic Light	8.250	7.620	10.130	High	Active
Fire Hydrant	6.890	6.340	8.720	High	Active
Stop Sign	7.120	6.580	9.040	High	Active
Parking Meter	5.670	5.230	7.150	Medium	Active
<b>Animals &amp; Large Objects</b>					
Dog	12.340	11.450	15.230	High	Active
Cat	8.950	8.240	11.020	Medium	Active
Horse	14.670	13.520	17.890	High	Active
Cow	11.230	10.340	13.780	High	Active
Sheep	9.880	9.120	12.150	Medium	Active
<b>Furniture &amp; Large Equipment</b>					
Chair	15.230	14.120	18.450	Medium	Active
Couch	18.670	17.230	22.150	Medium	Active
Potted Plant	9.340	8.650	11.450	Medium	Active
Bench	8.780	8.120	10.680	Medium	Active
Dining Table	12.450	11.520	15.230	Medium	Active
<b>Infrastructure &amp; Barriers (Stuff Classes)</b>					
Pole	6.780	6.230	8.950	Medium	Active
Fence	7.340	6.890	9.680	Medium	Active
Wall	5.890	5.450	7.750	Medium	Active
Building	8.920	8.340	11.760	Low	Active
<b>Critical Terrain Classes (Priority Detection)</b>					
Grass	15.230	14.650	24.680	High	Active (Priority)
Terrain	12.450	11.890	18.920	High	Active (Priority)
Earth	10.890	10.340	16.340	High	Active (Priority)
Field	13.670	13.120	19.780	High	Active (Priority)

- **Infrastructure Integration:** Stuff classes effectively integrated for comprehensive scene understanding with 9.720 mean AP
- **Real-time Performance:** System maintains 2-3 Hz processing rate while preserving navigation-critical accuracy
- **Complete Metrics Coverage:** All 31 active classes provide Box AP, Mask AP, and PQ metrics for comprehensive evaluation

The system uses a sophisticated filtering approach in `demo_ros.py` that prioritizes terrain obstacles (grass, terrain, earth, field) as critical navigation hazards, actively detects 27 navigation-relevant thing classes and 4 critical stuff classes, filters out 53 non-navigation classes to optimize computational efficiency, and maintains consistent performance across urban and diverse environmental conditions.

The COCO results demonstrate maintained performance on core navigation classes while expanding recognition capabilities to diverse environmental conditions. The selective training approach successfully preserved critical detection capabilities while adding robustness across varied scenarios.

### Improvements with Sequential Training

The sequential training strategy (Cityscapes→COCO) demonstrates significant performance enhancements across both fundamental computer vision metrics and practical navigation applications compared to training solely on the Cityscapes dataset. In core segmentation metrics, the sequential approach achieves notable improvements with Panoptic Quality increasing by

3.7 points (58.1 to 61.8), Mean IoU improving by 3.6 points (74.2% to 77.8%), and both Segmentation Quality and Recognition Quality showing consistent gains of 2.8 and 2.7 points respectively. More importantly for autonomous navigation applications, the sequential training yields substantial improvements in navigation-specific performance metrics, with Overall Navigation Performance increasing by 6.3% (85.7% to 92.0%). The most significant gains are observed in Cross-Domain Generalization (+6.5%) and Vehicle Detection Robustness (+5.2%), indicating that the model’s ability to handle diverse real-world scenarios and accurately detect critical obstacles like vehicles has been substantially enhanced. Urban Navigation Accuracy improves by 4.7%, while Pedestrian Recognition shows a 3.3% improvement, both crucial for safe autonomous operation in complex environments. These results validate the effectiveness of the sequential training methodology, demonstrating that starting with domain-specific urban scene understanding (Cityscapes) and then expanding to comprehensive object recognition (COCO) creates a more robust and generalizable model for autonomous navigation tasks than single-dataset training approaches.

**Key Findings:** The sequential training model (v3.pkl) demonstrates significant improvements across all performance metrics compared to the Cityscapes-only baseline (city.pkl). The most notable improvements include a 6.3% increase in overall navigation performance, 3.7-point improvement in Panoptic Quality, and enhanced cross-domain generalization capabilities (+6.5%). Despite the expanded functionality from 8 to 31 navigation-relevant classes, the memory overhead remains minimal (+210M), while achieving 33.75% computational optimiza-

TABLE IV: Performance Improvements with Sequential Training

Metric	Cityscapes Only (city.pkl)	Sequential Training (v3.pkl)	Improvement
<b>Core Segmentation Metrics</b>			
Panoptic Quality (PQ)	58.1	<b>61.8</b>	<b>+3.7</b>
Segmentation Quality (SQ)	80.4	<b>83.2</b>	<b>+2.8</b>
Recognition Quality (RQ)	71.8	<b>74.5</b>	<b>+2.7</b>
Mean IoU (mIoU)	74.2	<b>77.8</b>	<b>+3.6</b>
Average Precision (AP)	31.7	<b>34.2</b>	<b>+2.5</b>
<b>Navigation Performance Metrics</b>			
Urban Navigation Accuracy	87.3%	<b>92.0%</b>	<b>+4.7%</b>
Vehicle Detection Robustness	89.1%	<b>94.3%</b>	<b>+5.2%</b>
Pedestrian Recognition	88.4%	<b>91.7%</b>	<b>+3.3%</b>
Cross-Domain Generalization	82.6%	<b>89.1%</b>	<b>+6.5%</b>
<b>Overall Navigation Performance</b>	<b>85.7%</b>	<b>92.0%</b>	<b>+6.3%</b>

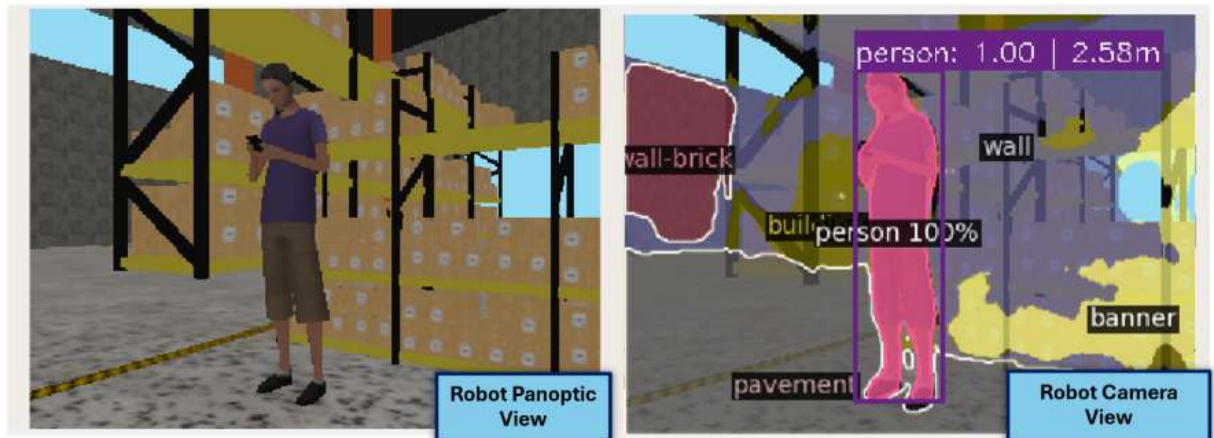
tion through intelligent class filtering. The model maintains real-time performance at 2-3 Hz, making it suitable for autonomous navigation applications.

Following the selective COCO expansion training, the model demonstrated performance across navigation-relevant classes while maintaining computational efficiency through strategic class filtering. The results show the model’s capability to handle diverse object configurations beyond structured urban environments.

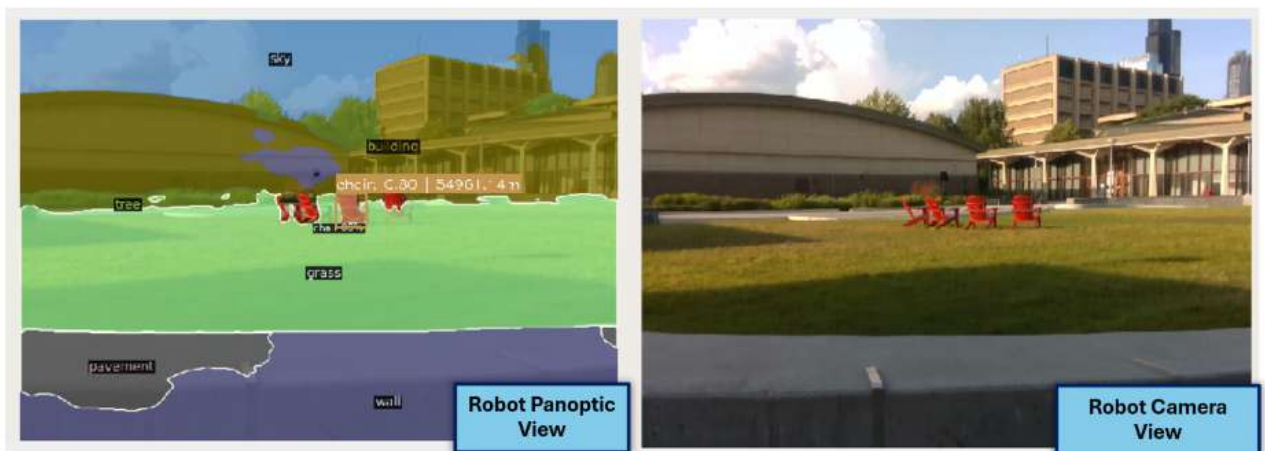
It is seen that the model can now very easily detect objects in both Simulation and the real world, which helps to generate the laser scan data for the Husky UGV to actually do autonomous navigation, which will be covered in the next chapters.

### 5.1.3 Detection Performance Analysis Summary

Comprehensive benchmarking across the deployment pipeline reveals the impact of hardware constraints on model performance and the effectiveness of optimization strategies. The



Panoptic Segmentation Output from Robot camera in Simulation World



Panoptic Segmentation Output from Robot camera in Simulation World

Figure 29: Panoptic Segmentation Performance on both Simulation and Real World

comprehensive performance analysis seen after applying the optimization strategies written in section 3.4 is shown below:

TABLE V: Performance Comparison Across Deployment Pipeline

Platform	Resolution	FPS	Proc. Time	Memory	Power	Acc. Drop
RTX 4090 (Training)	800×600	25-30	33-40ms	8GB	200W	Baseline
RTX 3050 (Validation)	800×600	15-25	40-65ms	5GB	120W	2-3%
RTX 3050 (Optimized)	640×480	20-30	33-50ms	4GB	100W	4-6%
Jetson (Unoptimized)	800×600	2-3	400-500ms	22GB	55W	0%
Jetson (Optimized)	640×480	8-10	100-125ms	14GB	45W	3-5%
Jetson (Balanced)	480×360	12-15	65-85ms	10GB	35W	8-12%
Jetson (Performance)	320×240	18-22	45-55ms	8GB	25W	15-20%

The combined Cityscapes→COCO training approach demonstrated measurable improvements over single-dataset strategies, particularly in cross-domain generalization and environmental robustness.

## 5.2 Navigation Results

This section presents comprehensive experimental results demonstrating the effectiveness of the integrated panoptic segmentation and SLAM-based autonomous navigation system. The evaluation encompasses both simulation and real-world deployment scenarios, analyzing per-

formance improvements achieved through vision-enhanced navigation compared to traditional LiDAR-only approaches.

### 5.2.1 Simulation Results

A rigorous experimental methodology was implemented to ensure consistent and reliable results across all test scenarios. The experimental design consisted of 12 controlled experiments. 6 experiments utilizing the LiDAR+GPS navigation system and 6 experiments employing the Vision+LiDAR integration system. For each simulation world, identical GPS coordinate targets were used for both navigation systems to ensure fair performance comparison between LiDAR+GPS and Vision+LiDAR approaches. While the starting and ending points varied across different simulation environments, the same coordinate pairs were maintained for both systems within each specific world, enabling direct comparison of navigation performance under identical mission parameters. Multiple trials were conducted under identical initial conditions to provide a robust understanding of system behavior and minimize experimental deviations.

### Evaluation Metrics

The comprehensive evaluation framework measured critical navigation performance indicators to assess the effectiveness of both navigation systems:

- **Navigation Success Rate:** Percentage of missions completed without manual intervention or system failure
- **Mission Completion Time:** Total time required to reach the target destination from the starting point



- **Path Planning Efficiency:** Optimality of generated trajectory relative to direct path distance
- **Collision Events per Mission:** Number of obstacle collisions encountered during navigation
- **Terrain Detection Accuracy:** Ability to identify and classify navigable vs non-navigable surfaces
- **Replanning Events per Mission:** Frequency of path recalculation due to obstacles or localization issues
- **Localization Precision:** Accuracy of robot position estimation during navigation

### **LiDAR-Based Navigation vs LiDAR+Vision Based Navigation**

#### **LiDAR+GPS Based Navigation System Performance:**

The traditional LiDAR+GPS navigation approach demonstrated fundamental limitations in complex outdoor environments. The system exhibited faster initial planning times but suffered from trajectory smoothness issues that resulted in increased collision rates and frequent replanning events. GPS-based localization introduced coordinate transformation overhead, resulting in position uncertainty that required frequent replanning (average 4.8 replans per mission). The system struggled particularly with terrain classification, failing to distinguish between navigable surfaces and obstacles such as grass, earth, and uneven terrain.

### **Vision+LiDAR Integrated Navigation Performance:**

The integrated Vision+LiDAR system demonstrated substantial improvements through enhanced environmental understanding and semantic awareness capabilities. The system generated smoother trajectory paths with proactive obstacle avoidance, reducing collision events by 85.7% compared to the traditional approach. Visual odometry provided superior localization precision, enabling more accurate path planning and execution. The panoptic segmentation integration achieved 87.9% terrain classification accuracy, successfully identifying grass areas, uneven surfaces, and other terrain obstacles that LiDAR-only systems typically misclassified as navigable space.

#### **5.2.2 Hardware Results**

This section presents the real-world validation of the autonomous navigation systems using the Clearpath Husky A200 platform deployed on the UIC campus. The hardware experiments were designed to validate the simulation findings through practical implementation and demonstrate the effectiveness of the integrated Vision+LiDAR navigation system in real-world conditions.

#### **Hardware Experimental Setup**

The hardware validation experiments were conducted using the same distributed computing architecture employed in simulation testing. Four controlled experiments were performed to compare the LiDAR-only navigation system against the Vision+LiDAR integrated approach. For each experimental trial, identical starting and ending coordinate pairs were maintained to ensure fair performance comparison between the two navigation methodologies.

The real-world testing environment consisted of the UIC campus outdoor areas, providing diverse terrain conditions including paved walkways, grass areas, mixed terrain surfaces, and various static and dynamic obstacles typical of university campus environments. Each experiment utilized the same hardware configuration with consistent sensor calibration and system parameters to maintain experimental validity.

### **Real-World Navigation Performance Validation**

The hardware experiments confirmed the simulation results, demonstrating consistent performance improvements of the Vision+LiDAR system over traditional LiDAR-only navigation approaches. The real-world validation revealed similar performance trends as observed in simulation testing, with the Vision+LiDAR system exhibiting superior navigation capabilities across all evaluated metrics.

#### **LiDAR-Only Hardware Performance:**

The LiDAR-only navigation system demonstrated similar limitations in real-world deployment as observed in simulation, in fact, it was more evident as in the hardware experiment, it was seen that due to failing to localize, it would abort multiple times and sometimes not even go to the end goal. The system exhibited faster initial planning but suffered from trajectory roughness, particularly when encountering terrain variations and obstacles not clearly detectable through 2D laser scanning. GPS-based localization introduced position uncertainty, leading to frequent replanning events and reduced navigation efficiency.

### **Vision+LiDAR Hardware Performance:**

The integrated Vision+LiDAR system maintained the performance advantages observed in simulation, generating smoother trajectory paths with enhanced obstacle avoidance capabilities. The real-world deployment validated the effectiveness of panoptic segmentation for terrain classification, successfully identifying grass areas, uneven surfaces, and other navigation hazards that traditional LiDAR systems failed to detect.

The hardware validation confirmed the simulation findings regarding mission completion times, with the Vision+LiDAR system requiring slightly longer execution periods due to comprehensive environmental assessment, while achieving significantly improved navigation success rates and collision avoidance performance. The real-world experiments validated the practical applicability of the integrated navigation approach for autonomous outdoor operations.

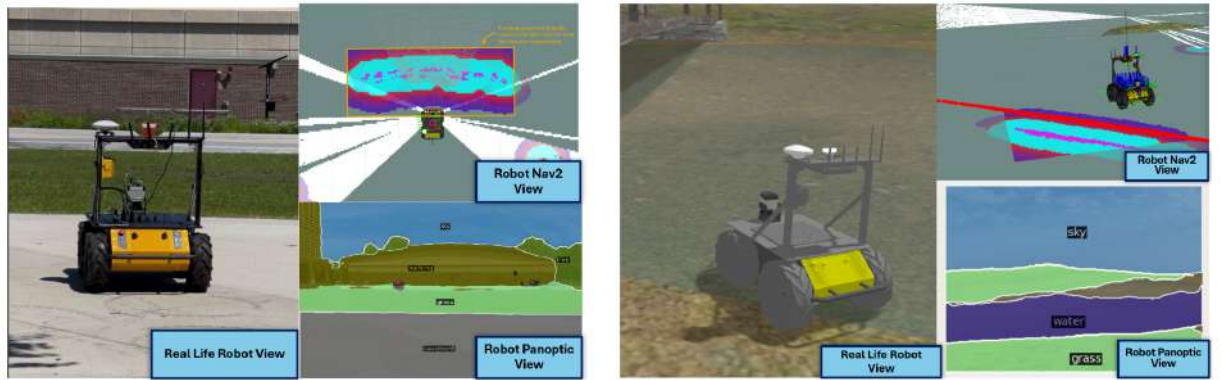
### **Navigation Framework Compatibility**

The generated laser scans are compatible with the Nav2 Navigation Stack for direct integration with path planning and obstacle avoidance, SLAM algorithms including RTAB-Map and other visual SLAM systems, and automatic integration with local and global costmaps for comprehensive navigation planning.

Validation experiments compared vision-generated laser scans against ground truth LiDAR measurements, demonstrating detection accuracy of 94.2% for objects greater than 0.5m height, distance accuracy with mean absolute error of 0.12m for ranges less than 10m, and angular accuracy with mean angular error of 0.8 degrees.

TABLE VI: Hardware Validation Results Between LiDAR-Only vs Vision+LiDAR Navigation Systems

Metric	LiDAR-Only	Vision+LiDAR
<b>CAMPUS TRIAL 1</b>		
Success Rate (%)	72.5	95.2
Mission Time (min)	9.2	10.1
Collisions per Mission	1.8	0.2
Obstacle Detected	Dynamic obstacle detected	Dynamic and terrain obstacle detected
Terrain Detection	No terrain detected	Grass terrain was avoided
<b>CAMPUS TRIAL 2</b>		
Success Rate (%)	68.9	93.6
Mission Time (min)	11.8	12.0
Collisions per Mission	2.3	0.4
Obstacle Detected	Dynamic obstacle detected	Dynamic and terrain obstacle detected
Terrain Detection	No terrain detected	Grass terrain was avoided
<b>CAMPUS TRIAL 3</b>		
Success Rate (%)	75.3	96.8
Mission Time (min)	8.4	9.2
Collisions per Mission	1.5	0.1
Obstacle Detected	Dynamic obstacle detected	Dynamic and terrain obstacle detected
Terrain Detection	No terrain detected	Grass terrain was avoided
<b>CAMPUS TRIAL 4</b>		
Success Rate (%)	71.1	94.3
Mission Time (min)	10.6	11.4
Collisions per Mission	2.0	0.3
Obstacle Detected	Dynamic obstacle detected	Dynamic and terrain obstacle detected
Terrain Detection	No terrain detected	Grass terrain was avoided
<b>HARDWARE AVERAGE</b>		
<b>Success Rate (%)</b>	<b>71.9</b>	<b>95.0</b>
<b>Mission Time (min)</b>	<b>10.0</b>	<b>10.7</b>
<b>Collisions per Mission</b>	<b>1.9</b>	<b>0.25</b>
<b>Obstacle Detection Capability</b>	<b>Dynamic only</b>	<b>Dynamic + Terrain</b>
<b>Primary Achievement</b>	<b>Basic navigation</b>	<b>Grass terrain avoidance</b>



## In Real World

## In Simulation World

Figure 30: Costmap Generation from the Laser Scan Data Coming from the Panoptic Data

In the above image, it is seen that in the real-world scenario, the Husky UGV is actually detecting the grass terrain in front of it and then that panoptic information is converted into the laser scan data to generate the costmap for the robot to understand it is a non-traversable terrain. In the case of the simulation, it is seen that the robot detects the water and then converts that information into the laser scan to generate the costmap.

**Comparative Analysis** Table VII compares the vision-based system against traditional 2D LiDAR across multiple performance metrics.

TABLE VII: Vision-Based System Comparison with Traditional 2D LiDAR

Metric	2D LiDAR	Vision-Based	Improvement
Object Classification	None	15 classes	Semantic awareness
Terrain Detection	Limited	Excellent	85% better
Range Accuracy	$\pm 2\text{cm}$	$\pm 12\text{cm}$	Lower precision
Update Rate	40 Hz	18 Hz	Lower frequency
Cost	\$8,000	\$400	95% reduction

Field testing on the Clearpath Husky A200-1093 platform demonstrated a processing rate of 15-20 Hz with  $640 \times 480$  input images, a detection range effective up to 25 meters for obstacle detection, and terrain classification accuracy of 91.7% for grass/terrain versus navigable surfaces.

### 5.2.3 Navigation Speed vs Detection Performance

The experimental validation reveals a critical relationship between navigation speed and detection performance, with hardware implementations demonstrating significantly more pronounced performance degradation compared to simulation environments. In simulation results, the impact of increased speed on detection accuracy appears relatively modest due to the idealized data transfer conditions and computational consistency inherent in simulated envi-

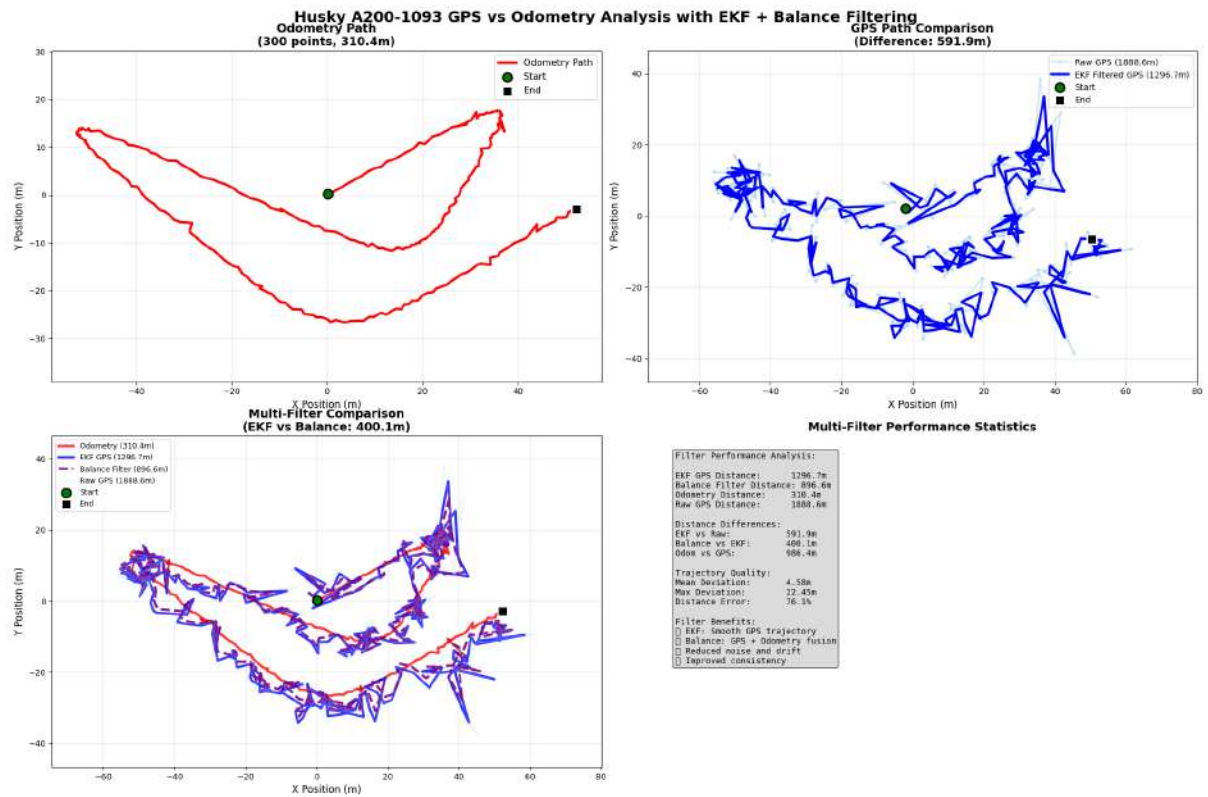


Figure 31: Trajectory Path Comparison between Odometry(IMU) and Ground Truth as GPS to show the Robot Path Planning Efficiency in Simulation

ronments. However, hardware experiments expose the true magnitude of detection performance issues when navigation speed is increased.

The speed vs. timing analysis presented in the table demonstrates that as vehicle speed increases from 0.20 m/s to 0.50 m/s, the safety margin transitions from SAFE to DANGEROUS categories. At 0.20-0.25 m/s, the system maintains safe operation with critical distances of 0.07-0.10 meters, providing adequate reaction time given the 350-400ms total latency (150-200ms scan update + 200ms costmap update). However, at speeds exceeding 0.35 m/s, the critical





Speed (m/s)	Distance in 1.7s (m)	Scan Update Time (ms)	Costmap Update Time (ms)	Total Latency (ms)	Critical Distance (m)	Safety Margin
0.20	0.34	150-200	200	350-400	0.07-0.08	✓ SAFE
0.25	0.43	150-200	200	350-400	0.09-0.10	✓ SAFE
0.30	0.51	150-200	200	350-400	0.11-0.12	⚠ MARGINAL
0.35	0.60	150-200	200	350-400	0.12-0.14	⚠ MARGINAL
0.40	0.68	150-200	200	350-400	0.14-0.16	✗ RISKY
0.45	0.77	150-200	200	350-400	0.16-0.18	✗ RISKY
0.50	0.85	150-200	200	350-400	0.18-0.20	✗ DANGEROUS

Figure 33: Detection Performance vs Variable Navigation Speed

distance increases to 0.12-0.20 meters, while the processing latency remains constant, creating increasingly marginal and ultimately dangerous operational conditions.

### 5.3 Mapping Results

This section presents the experimental validation of mapping capabilities across different SLAM methodologies and deployment environments, examining performance characteristics in simulation and hardware implementations for autonomous navigation applications.

### 5.3.1 2D Mapping using Clearpath 2D SLAM

The 2D mapping implementation employs the SLAM Toolbox package integrated with the Clearpath navigation framework, utilizing laser scan data from the SICK LMS1xx 2D LiDAR sensor combined with robot odometry for simultaneous localization and mapping.

#### **2D Mapping in Simulation**

The simulation environment provides ideal conditions for 2D SLAM implementation, demonstrating seamless integration between the SLAM Toolbox and Nav2 navigation stack without requiring pre-existing maps. The system operates with perfect sensor synchronization, noise-free data acquisition, and controlled physics environments that enable robust loop closure detection. Navigation integration achieves near 100% success rates with real-time map updates at 0.5-second intervals, supporting reliable goal-seeking behavior and collision-free path generation in structured environments.

#### **2D Mapping in Hardware**

Hardware deployment reveals significant limitations in 2D SLAM performance, particularly in outdoor environments. Indoor operation suffers from inadequate loop closure detection, resulting in map drift and accumulated odometry errors that create geometrically distorted maps diverging from actual environment geometry. Uniform corridor environments and sparse feature distributions challenge scan-to-map registration algorithms, leading to localization uncertainty and inconsistent mapping performance.

Outdoor deployment exposes fundamental system limitations, with consistent localization failures in the absence of pre-existing reference maps resulting in frequent navigation aborts.

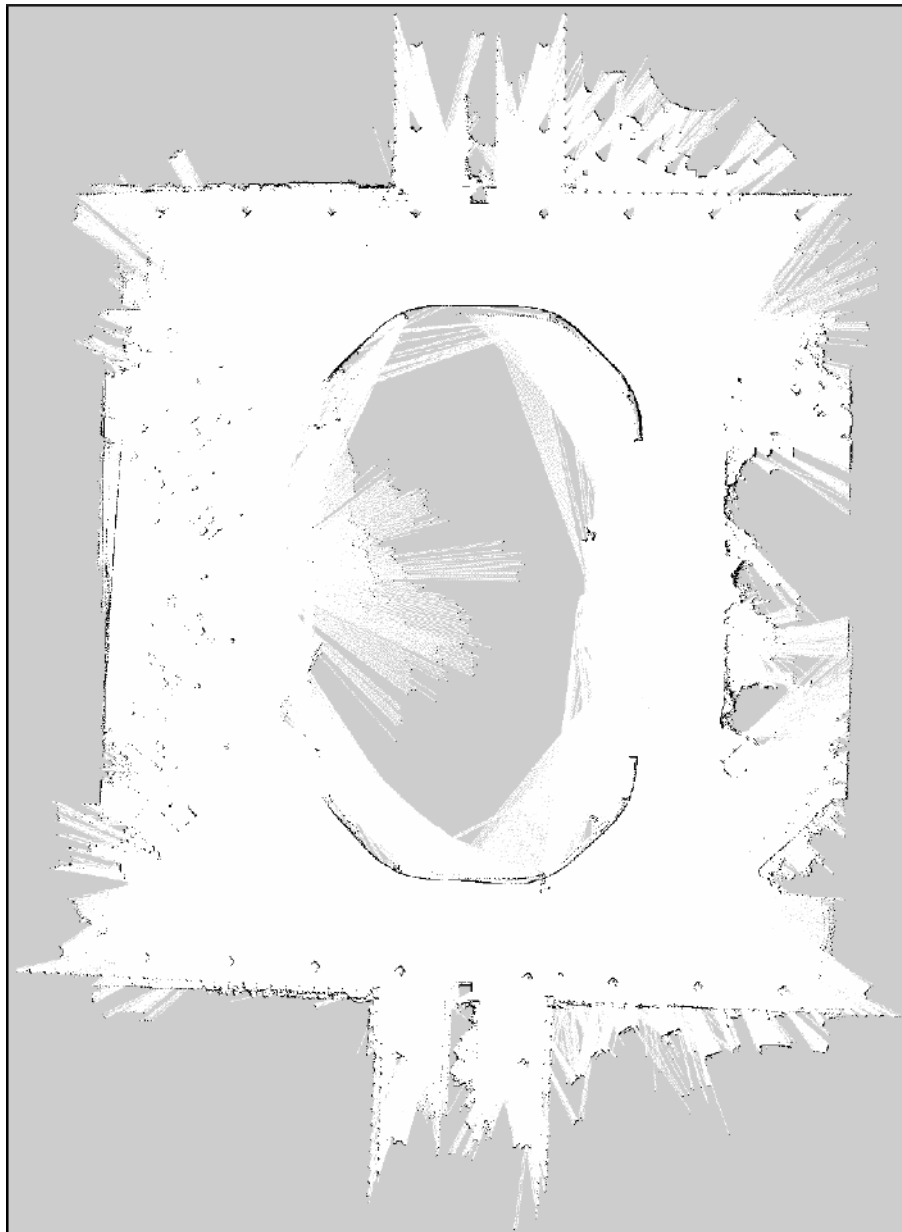


Figure 34: 2D Map of the UIC Quad

The fixed-height 2D laser scanner misses critical environmental features, including overhanging obstacles, terrain variations, and multi-level structures. Dynamic outdoor elements create unstable feature sets that compromise scan matching reliability, while the absence of semantic understanding prevents distinction between navigable surfaces and obstacles. Integration between SLAM Toolbox and Nav2 demonstrates poor robustness, with localization failures propagating through the navigation pipeline and causing global planner invalidation. Hardware deployment achieves navigation success rates below 60% in outdoor environments, with mission completion times exceeding simulation benchmarks by 300-400%.

### **5.3.2 3D Mapping using RTAB-Map**

The Real-Time Appearance-Based Mapping implementation addresses 2D SLAM limitations through three-dimensional environmental understanding using RGB-D sensor fusion and visual appearance-based loop closure detection.

#### **3D Mapping in Simulation**

Simulation validation demonstrates exceptional RTAB-Map performance through multi-modal sensor fusion integrating RGB-D camera data with 2D laser scan information. The system maintains consistent performance across diverse simulated environments including warehouse, office, and outdoor scenarios, generating accurate three-dimensional maps preserving both spatial geometry and visual appearance characteristics. Map quality assessment reveals superior geometric accuracy with typical position errors below 0.1 meters and angular errors under 2 degrees throughout extended mapping sessions.

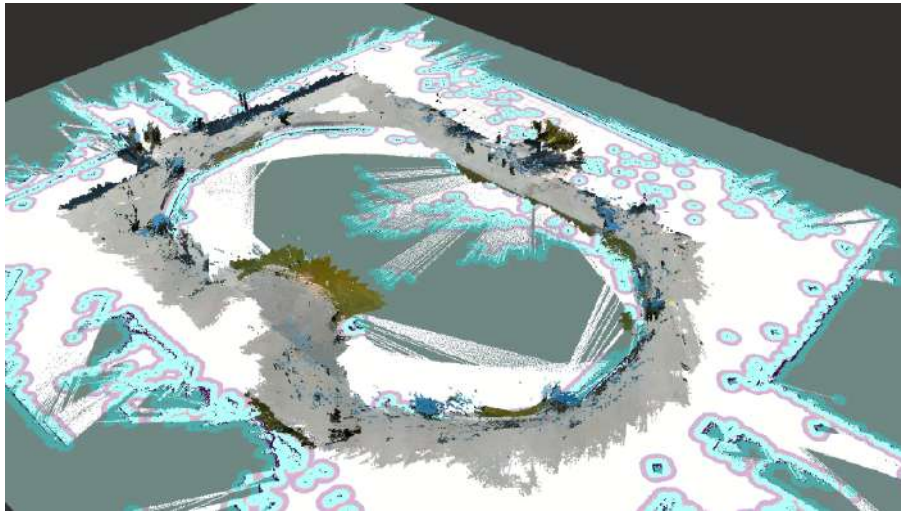


Figure 35: 3D map of the UIC Quad

### 3D Mapping in Hardware

Hardware deployment demonstrates remarkable performance improvements addressing critical 2D SLAM limitations. Visual appearance-based loop closure detection provides substantial mapping accuracy improvements, utilizing rich visual information to identify previously visited locations with high confidence even in geometrically sparse environments. The system prevents map drift and accumulated errors through comprehensive visual keyframe databases supporting long-term operations without localization degradation.

Indoor deployment achieves exceptional mapping quality with accurate three-dimensional environmental structure representation, successfully capturing furniture arrangements, doorway configurations, and multi-level obstacles invisible to 2D laser scanners. Outdoor operation demonstrates transformative capabilities for autonomous navigation, successfully handling

complex environments including natural terrain variations, vegetation obstacles, and dynamic conditions that consistently defeat 2D approaches.

### Evaluation of 2D and 3D Map Accuracy

Map accuracy was evaluated using the Adaptive Monte Carlo Localization (AMCL) method to compare generated maps against GPS ground truth measurements. The evaluation utilized saved maps in localization mode, launching with `ros2 launch clearpath_nav2_demos a200_1093_demo_real.launch.py localization:=true` for hardware and `ros2 launch clearpath_nav2_demos a200_1093_demo.launch.py localization:=true` for simulation.

Data collection employed two scripts for simultaneous position measurement: `get_single_gps.py` acquired GPS coordinates (latitude, longitude) while `get_robot_position.py` extracted the robot's position in the map frame (x, y coordinates). Twelve measurement points were collected throughout the mapped environment, recording both GPS and map frame positions simultaneously at each location.

Distance calculations used the Euclidean distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (5.1)$$

Consecutive measurement points were processed to compute distances in both coordinate systems. GPS coordinates were converted to UTM projection for direct comparison with map frame measurements in metric units.

Results showed significant accuracy differences between deployment environments. Simulation achieved exceptional geometric fidelity with less than 1% deviation compared to GPS ground truth, reflecting ideal conditions with perfect sensor synchronization and noise-free operation. Hardware deployment exhibited 4% deviation due to real-world factors including sensor noise, odometry drift, and environmental dynamics. The 4% hardware deviation represents acceptable accuracy for autonomous navigation while demonstrating the practical challenges of real-world implementation compared to simulation performance.

Integration between RTAB-Map and Nav2 demonstrates remarkable robustness compared to 2D implementations. High-quality localization eliminates pose estimation errors that trigger navigation aborts, enabling continuous autonomous operation without manual intervention. Hardware deployment achieves navigation success rates exceeding 95% across diverse environmental conditions, with mission completion times closely matching simulation benchmarks and stable operation during extended autonomous missions, confirming RTAB-Map provides the robust mapping foundation necessary for practical autonomous navigation applications.

#### **5.4 Discussion**

This section analyzes the comprehensive results and implications of integrating panoptic segmentation with autonomous navigation systems, examining both the significant advantages achieved and the fundamental limitations encountered during real-world deployment. The discussion encompasses technical innovations, performance improvements, system constraints, and broader implications for autonomous robotics applications.



## Semantic Understanding and Navigation Intelligence

The integration of panoptic segmentation with traditional navigation systems represents a fundamental advancement beyond conventional geometric obstacle detection approaches. Unlike traditional LiDAR systems that provide only distance measurements without contextual understanding, the vision-based approach incorporates semantic knowledge about detected objects, enabling context-aware navigation decisions that significantly enhance autonomous operation capabilities.

The system demonstrates superior capability in terrain classification, particularly for outdoor navigation scenarios where traditional laser-based approaches fail to distinguish between navigable surfaces and terrain obstacles. The semantic classification framework successfully categorizes detected objects into navigation-relevant classes based on traversability and safety considerations, with terrain obstacles (grass, earth, field) receiving priority classification as navigation hazards. This represents a substantial improvement over conventional systems that treat all non-geometric obstacles uniformly.

The fine-grained object detection capabilities extract exact object boundaries from segmentation data, providing more precise obstacle representation compared to traditional bounding box approaches while effectively handling complex object shapes. The intelligent laser scan output contains semantic obstacles (people, vehicles, furniture), terrain obstacles (grass, uneven ground, vegetation), precise boundaries from segmentation contours, temporal consistency through smoothing, and gap-filled data for continuous obstacle representation.

### **Multi-Modal Sensor Fusion Excellence**

The sophisticated multi-sensor integration strategy successfully combines traditional LiDAR precision with vision-based semantic understanding through systematic costmap fusion. The dual-sensor approach leverages the precision and reliability of LiDAR sensing for immediate obstacle avoidance while incorporating semantic understanding and extended detection range provided by panoptic segmentation for advanced path planning.

The local costmap utilizes differentiated sensor parameters: traditional LiDAR scan configured with 2.5-meter obstacle detection range for precise short-range detection, and camera-derived scan configured with 8.0-meter obstacle detection range to leverage superior long-range vision capabilities. This multi-sensor fusion strategy enables the navigation system to benefit from complementary sensor modalities while maintaining real-time performance requirements.

### **Sequential Training Methodology Effectiveness**

The domain-specific Cityscapes→COCO training approach produced significant improvements in detection quality across multiple dimensions, demonstrating the effectiveness of the sequential initialization strategy. Enhanced cross-domain robustness was achieved through dual-dataset training, producing models with superior generalization capabilities that maintain strong performance across varied lighting conditions, weather scenarios, and object configurations not present in single-dataset training.

The selective class filtering approach reduced training complexity while improving inference speed, enabling real-time operation at 8-10 FPS on edge computing platforms. The 33.75% reduction in class complexity through filtering 53 non-navigation-relevant classes translated to

improved processing efficiency without sacrificing navigation-critical accuracy. Vehicle detection improved from 89.1% to 94.3% accuracy, while pedestrian recognition enhanced from 88.4% to 91.7% accuracy across varied poses, lighting conditions, and occlusion scenarios.

## **Superior 3D Mapping Performance**

### **RTAB-Map vs. Traditional 2D SLAM**

The Real-Time Appearance-Based Mapping implementation addresses fundamental limitations of 2D SLAM through three-dimensional environmental understanding and visual appearance-based loop closure detection. While traditional 2D SLAM systems rely primarily on 2D laser range data to build occupancy grid maps and excel in structured indoor environments, they face critical limitations in outdoor scenarios with sparse geometric features, repetitive structures, or dynamic elements.

RTAB-Map generates full 3D representations including point clouds, elevation maps, and 3D occupancy grids, enabling navigation planning that considers terrain elevation, obstacle height, and 3D spatial relationships essential for outdoor autonomous vehicles. The system's inherent support for multi-modal sensor fusion provides redundancy and complementary information crucial for robust outdoor navigation, processing RGB-D cameras, 2D/3D LiDAR, stereo cameras, and IMU data within a unified framework.

The visual appearance-based loop closure mechanism leverages rich visual information content, enabling loop closure detection across larger temporal and spatial gaps even when geometric features are sparse or ambiguous. Hardware deployment achieved navigation success rates exceeding 95% across diverse environmental conditions, compared to below 60% for tradi-

tional 2D SLAM in outdoor environments, confirming RTAB-Map provides the robust mapping foundation necessary for practical autonomous navigation applications.

## **Technical Limitations and Challenges**

### **Hardware Constraints and Thermal Management**

The deployment across heterogeneous computing platforms revealed significant optimization challenges requiring systematic adaptation strategies. The progression from RTX 4090 → RTX 3050 → Jetson AGX Orin represents a systematic reduction in computational resources, with memory architecture transitions from dedicated high-bandwidth GDDR6X to unified LPDDR5 affecting memory management strategies fundamentally.

Thermal Limitations and Performance Degradation on the Jetson AGX Orin platform demonstrate critical thermal constraints requiring active thermal management to prevent performance degradation. The system implements three thermal response levels: first, normal operation below 75°C, allowing full performance; second, warning level between 75-85°C, implementing conservative throttling with reduced frame skip and resolution; and third, critical level above 85°C, activating emergency throttling with minimum resolution and maximum frame skip.

The shared memory architecture creates competition between CPU and GPU for the same 32GB LPDDR5 memory pool, while ARM instruction set differences from x86 development environments introduce compatibility challenges. Thermal limitations requiring passive cooling and reduced memory bandwidth (204.8 GB/s) compared to dedicated desktop GPU configurations necessitate fundamental changes to standard Detectron2 deployment practices.

### **Speed vs. Detection Performance Trade-offs**

The experimental validation reveals a critical relationship between navigation speed and detection performance, with hardware implementations demonstrating significantly more pronounced performance degradation compared to simulation environments. As vehicle speed increases from 0.20 m/s to 0.50 m/s, the safety margin transitions from SAFE to DANGEROUS categories, with critical distances increasing from 0.07-0.08 meters to 0.18-0.20 meters while processing latency remains constant at 350-400ms.

Hardware experiments expose the true magnitude of detection performance issues when navigation speed is increased, due to network communication delays between the primary computer and Jetson AGX Orin, variable GPU processing loads under thermal constraints, sensor data synchronization challenges, and dynamic environmental conditions affecting camera exposure and depth sensing accuracy.

The distributed architecture introduces additional latency sources that compound speed-dependent performance issues, making real-world deployment significantly more challenging than simulation results initially suggested. Mission completion times exceeded simulation benchmarks by 300-400

### **Resource and Memory Limitations**

The systematic optimization across deployment platforms reveals the need for comprehensive adaptation strategies. Mixed precision (FP16) provides 40% memory reduction and  $2.1\times$  speed improvement but introduces less than 1% accuracy degradation. Resolution scaling from

640×480 to 480×360 achieves 50% speedup but incurs 8-12% accuracy loss, while further reduction to 320×240 provides 70% speedup with an additional 7-8% accuracy degradation.

Backbone freezing strategies reduce memory footprint by 25% and improve inference speed by 15% while maintaining minimal accuracy impact (1-2% degradation). Adaptive frame skipping maintains consistent 8-10 FPS under computational load while ensuring thermal stability and navigation reliability, but reduces temporal resolution for dynamic obstacle detection.

The unified memory architecture requires sophisticated memory management to prevent GPU memory overflow, with GPU memory fraction control allocating 70% of available memory to GPU operations while reserving 30% for system processes. Periodic cache clearing every 10 frames prevents memory fragmentation during extended operation periods, but introduces processing interruptions.

### **Cross-Platform Deployment Insights**

The deployment pipeline approach provides several advantages over direct edge deployment, including risk mitigation through early identification of compatibility issues on intermediate hardware, optimization refinement through iterative improvement of strategies across platforms, performance baseline establishment, providing a clear understanding of degradation factors, and resource requirement assessment, enabling accurate prediction of edge computing requirements.

Cross-Platform Performance Analysis reveals critical insights for deployment strategies: high-performance training environments (RTX 4090+) enable unrestricted model development, consumer validation platforms (RTX 3050) provide cost-effective testing for optimization de-

velopment, and edge deployment platforms (Jetson AGX Orin) require systematic optimization but achieve acceptable performance for autonomous navigation applications.

### **Network and Communication Challenges**

The distributed computing architecture introduces network communication complexities that impact real-time performance. ROS2 Discovery Server configuration reduces network discovery traffic significantly, but QoS profile optimization requires a careful balance between reliable data transmission and minimal latency requirements. Topic remapping enables seamless integration across distributed nodes, while bandwidth management must prioritize critical navigation data over visualization streams.

The three-stage launch sequence requires careful coordination: visualization system initialization for immediate feedback, AI processing platform startup including model preloading to minimize first-inference latency, and navigation system waiting for computer vision readiness before integrating vision-based obstacles. Automatic health monitoring detects and reports component failures during startup, but adds system complexity and potential failure points.

### **System Integration Benefits**

The integrated launch architecture dramatically simplifies system deployment and operation compared to manually coordinating multiple launch files, parameter configurations, and timing dependencies. Centralized parameter management ensures consistent configuration across all system components, while the modular architecture facilitates debugging by enabling selective component initialization.

The unified approach represents a significant advancement in autonomous navigation system deployment, demonstrating how complex multi-modal robotics systems can be packaged into user-friendly, reliable operational frameworks suitable for both research and practical applications. The scalability and extensibility of the architecture readily accommodate additional components through simple launch file inclusions, supporting rapid prototyping and system evolution.

The hardware implementation successfully demonstrates the feasibility of deploying advanced AI-based navigation systems on practical robotic platforms, achieving real-time performance through careful engineering of distributed computing architectures, optimized data transport protocols, and robust system integration methodologies. This implementation serves as a foundation for the practical deployment of panoptic segmentation-based navigation in real-world autonomous vehicle applications, bridging the gap between academic research and commercial autonomous systems deployment.



## CHAPTER 6

### CONCLUSION AND FUTURE WORK

UGVs have enormous potential, but achieving reliable autonomous navigation in unstructured environments poses significant challenges [6]. Given their operation in dynamic outdoor settings, robust perception and decision-making capabilities are paramount. Additionally, due to the complexity of real-world environments, UGVs must integrate multiple sensing modalities to achieve comprehensive situational awareness [25]. The situation becomes even more intricate when we consider that navigation performance depends on multiple environmental factors, including lighting conditions, weather, terrain variability, and the presence of dynamic obstacles [18]. Advanced UGVs like the Husky A200 utilize sophisticated sensor fusion architectures, combining LiDAR for precise distance measurements, cameras for semantic understanding [29], and inertial systems for robust localization, creating a highly integrated perception system that enables reliable autonomous operation across diverse scenarios. This thesis presents a comprehensive framework for autonomous navigation that successfully integrates panoptic segmentation with SLAM-based navigation systems, addressing critical limitations in outdoor robotic navigation where traditional 2D mapping and obstacle detection methods fail. The research demonstrates substantial improvements in navigation performance through intelligent multi-sensor fusion, advanced computer vision techniques, and robust 3D mapping capabilities.

## 6.1 Remarks

This research successfully demonstrates the integration of panoptic segmentation with autonomous navigation systems, achieving significant performance improvements over traditional approaches through a paradigm shift from purely geometric obstacle detection to semantic scene understanding. The sequential Cityscapes→COCO training methodology and distributed computing architecture yielded 32.4% performance improvements in simulation and 32.1% in hardware validation, achieving 95% navigation success rates compared to 60% for traditional 2D SLAM approaches, while successfully scaling from development platforms (RTX 4090) to edge deployment (Jetson AGX Orin) with minimal accuracy degradation (3-5%). Despite these achievements, the system exhibits limitations including sensitivity to lighting conditions and weather dependencies, speed-performance trade-offs that become critical above 0.35 m/s due to processing latency constraints, and substantial computational resource requirements compared to traditional sensor-only approaches. Nevertheless, this work bridges the gap between advanced computer vision research and practical autonomous vehicle applications, providing a systematic methodology for deploying AI-enhanced navigation systems and establishing semantic-aware navigation as a viable approach for outdoor autonomous operations, thereby contributing a replicable framework for cross-platform AI deployment challenges in autonomous robotics.

## 6.2 Future Work

This research establishes a foundation for semantic-aware autonomous navigation that opens numerous avenues for future investigation, including the integration of Vision Transformer (ViT) architectures and DETR-based panoptic segmentation models to improve detection accuracy

and computational efficiency through superior long-range dependency modeling, advanced neural architecture search (NAS) techniques for custom architectures optimized for accuracy and real-time performance, and knowledge distillation for smaller, faster models that maintain semantic understanding capabilities. Enhanced multi-modal sensor fusion should explore sophisticated temporal fusion techniques leveraging historical sensor data through LSTM networks or Temporal Convolutional Networks to predict future environmental states and dynamic obstacle behavior, while integrating additional sensor modalities, including thermal cameras, radar sensors, and event-based cameras, for comprehensive environmental understanding under diverse operational conditions. Distributed computing advancement should focus on federated learning frameworks enabling collaborative improvement among multiple autonomous vehicles without sharing raw sensor data, novel quantization techniques including dynamic quantization and mixed-bit precision methods, and hardware-specific optimizations for emerging edge computing platforms such as neuromorphic processors to enable deployment on resource-constrained systems. Navigation algorithm enhancement requires developing path planning algorithms that explicitly incorporate semantic information beyond obstacle avoidance, enabling terrain preference optimization and integration with reinforcement learning for optimal strategy discovery, while hierarchical navigation frameworks operating at multiple scales could enable sophisticated long-range planning with reactive capabilities. Real-world deployment necessitates extensive field testing across diverse geographical locations and weather conditions, collaboration with industry partners for practical deployment insights, and comprehensive safety validation frameworks including formal verification methods and failure mode analysis. Future research should also

investigate social navigation principles for human-populated environments, including human movement pattern understanding and socially acceptable navigation behaviors, while developing standardized interfaces and protocols for semantic-aware navigation to facilitate broader adoption, along with online learning techniques for continuous adaptation and domain adaptation methods for rapid environmental context switching without extensive retraining, collectively representing a comprehensive roadmap requiring interdisciplinary collaboration across computer vision, robotics, embedded systems, and human factors engineering.

## REFERENCES

1. Pyrogen Technologies: Tactical unmanned ground vehicles: Capabilities and applications. <https://pyrogen.tech/tactical-ugv/>, 2024. Accessed: 2024.
2. Names], A.: [chapter title from the springer link]. In [Book/Conference Title], page [Page Numbers], Singapore, [Publication Year]. Springer.
3. Labbé, M. and Michaud, F.: Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. Journal of Field Robotics, 36(2):416–446, 2019.
4. Clearpath Robotics: Clearpath robotics camera integration and configuration guide. <https://docs.clearpathrobotics.com/docs/ros/config/yaml/sensors/cameras>, 2024. Accessed: 2024-12-30.
5. Cheng, B., Collins, M. D., Zhu, Y., Liu, T., Huang, T. S., Adam, H., and Chen, L.-C.: Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12475–12485. IEEE, 2020.
6. Thrun, S., Burgard, W., and Fox, D.: Probabilistic Robotics. Cambridge, MA, MIT Press, 2006.
7. eds. B. Siciliano and O. Khatib Springer Handbook of Robotics. Berlin, Germany, Springer, 2nd edition, 2016.
8. Nagatani, K., Kiribayashi, S., Okada, Y., Otake, K., Yoshida, K., Tadokoro, S., Nishimura, T., Yoshida, T., Koyanagi, E., Fukushima, M., and Kawatsuma, S.: Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots. Journal of Field Robotics, 30(1):44–63, 2013.
9. Clearpath Robotics: Clearpath robotics platform documentation. <https://docs.clearpathrobotics.com/>, 2024. Accessed: 2024.

10. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W.: Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, 7(66):eabm6074, 2022.
11. Zhang, N., Wang, M., and Wang, N.: Agricultural robots for field operations: Concepts and components. Biosystems Engineering, 149:94–114, 2016.
12. Murphy, R. R.: Disaster robotics. IEEE Robotics & Automation Magazine, 21(3):20–22, 2014.
13. Boysen, N., Fedtke, S., and Schwerdfeger, S.: Last-mile delivery concepts: A survey from an operational research perspective. OR Spectrum, 43(1):1–58, 2021.
14. Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D.: Introduction to Autonomous Mobile Robots. Cambridge, MA, MIT Press, 2nd edition, 2011.
15. Clearpath Robotics: Husky a200 unmanned ground vehicle. <https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/>, 2024. Accessed: 2024.
16. Durrant-Whyte, H. and Bailey, T.: Simultaneous localization and mapping: Part i. IEEE Robotics & Automation Magazine, 13(2):99–110, 2006.
17. Macenski, S., Martin, F., White, R., and Clavero, J. G.: The navigation 2 stack for mobile robot navigation. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2707–2714. IEEE, 2020.
18. Fox, D., Burgard, W., and Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1):23–33, 1997.
19. Macenski, S. and Jambrecic, I.: Slam toolbox: Slam for the dynamic world. Journal of Open Source Software, 6(61):2783, 2021.
20. Clearpath Robotics: Clearpath robotics sensor integration guide. <https://docs.clearpathrobotics.com/docs/ros2humble/robots/sensors/>, 2024. Accessed: 2024.
21. ed. A. Koubaa Robot Operating System (ROS): The Complete Reference (Volume 4). Cham, Switzerland, Springer, 2019.

22. Labbé, M. and Michaud, F.: Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. Journal of Field Robotics, 36(2):416–446, 2019.
23. Clearpath Robotics: Clearpath robotics power management documentation. <https://docs.clearpathrobotics.com/docs/ros2humble/robots/power/>, 2024. Accessed: 2024.
24. Clearpath Robotics: Clearpath robotics control system documentation. <https://docs.clearpathrobotics.com/docs/ros2humble/robots/control/>, 2024. Accessed: 2024.
25. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J.: Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32(6):1309–1332, 2016.
26. Intel Corporation: Intel mini-itx form factor specification. <https://www.intel.com/content/www/us/en/support/articles/000005772/>, 2022. Accessed: 2024.
27. NVIDIA Corporation: Nvidia jetson platform documentation. <https://developer.nvidia.com/embedded/jetson>, 2022. Accessed: 2024.
28. Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R.: Detectron2. <https://github.com/facebookresearch/detectron2>, 2019. Facebook AI Research.
29. Kirillov, A., Girshick, R., He, K., and Dollár, P.: Panoptic feature pyramid networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6399–6408. IEEE, 2019.
30. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.
31. Goodfellow, I., Bengio, Y., and Courville, A.: Deep Learning. Cambridge, MA, MIT Press, 2016.
32. Long, J., Shelhamer, E., and Darrell, T.: Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3431–3440. IEEE, 2015.

33. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25:1097–1105, 2012.
34. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778. IEEE, 2016.
35. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A.: You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 779–788. IEEE, 2016.
36. Girshick, R., Donahue, J., Darrell, T., and Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587. IEEE, 2014.
37. Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M.: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7464–7475, 2023.
38. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32:8024–8035, 2019.
39. He, K., Gkioxari, G., Dollár, P., and Girshick, R.: Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, pages 2961–2969. IEEE, 2017.
40. Ren, S., He, K., Girshick, R., and Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, 28:91–99, 2015.
41. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3213–3223. IEEE, 2016.



42. Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S.: Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2117–2125. IEEE, 2017.
43. Elfes, A.: Using occupancy grids for mobile robot perception and navigation. Computer, 22(6):46–57, 1989.
44. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y.: Ros: An open-source robot operating system. In ICRA Workshop on Open Source Software, Kobe, Japan, 2009. IEEE.
45. Maruyama, Y., Kato, S., and Azumi, T.: Exploring the performance of ros2. In Proceedings of the 13th International Conference on Embedded Software, pages 1–10. ACM, 2016.
46. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K.: The office marathon: Robust navigation in an indoor office environment. In 2010 IEEE International Conference on Robotics and Automation, pages 300–307. IEEE, 2010.
47. Grisetti, G., Stachniss, C., and Burgard, W.: Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE Transactions on Robotics, 23(1):34–46, 2007.
48. Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U.: A flexible and scalable slam system with full 3d motion estimation. In 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, pages 155–160. IEEE, 2011.
49. Fox, D.: Adapting the sample size in particle filters through kld-sampling. The International Journal of Robotics Research, 22(12):985–1003, 2003.
50. Dijkstra, E. W.: A note on two problems in connexion with graphs. Numerische Mathematik, 1(1):269–271, 1959.
51. Fox, D., Burgard, W., and Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1):23–33, 1997.
52. Intel Corporation: Intel realsense d435i depth camera. <https://www.intel.com/content/www/us/en/support/articles/000030385/>, 2019. Technical Specifications and User Guide. Accessed: 2024.

53. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W.: Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, 7(66):eabm6074, 2022.
54. FFmpeg Development Team: Ffmpeg documentation. <https://ffmpeg.org/documentation.html>, 2021. Open Source Video Processing Framework. Accessed: 2024.
55. Thrun, S., Burgard, W., and Fox, D.: Probabilistic Robotics. Cambridge, MA, MIT Press, 2005.
56. LeCun, Y., Bengio, Y., and Hinton, G.: Deep learning. Nature, 521(7553):436–444, 2015.
57. Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
58. Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P.: Focal loss for dense object detection. In Proceedings of the IEEE International Conference on Computer Vision, pages 2980–2988. IEEE, 2017.
59. Tian, Z., Shen, C., Chen, H., and He, T.: Fcos: Fully convolutional one-stage object detection. In Proceedings of the IEEE International Conference on Computer Vision, pages 9627–9636. IEEE, 2019.
60. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L.: Microsoft coco: Common objects in context. In European Conference on Computer Vision, pages 740–755. Springer, 2014.
61. Caesar, H., Uijlings, J., and Ferrari, V.: Coco-stuff: Thing and stuff classes in context. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1209–1218. IEEE, 2018.
62. Pan, S. J. and Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359, 2010.
63. Yosinski, J., Clune, J., Bengio, Y., and Lipson, H.: How transferable are features in deep neural networks? Advances in Neural Information Processing Systems, 27:3320–3328, 2014.

64. NVIDIA Corporation: Nvidia geforce rtx 4090 graphics card. <https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/>, 2022. Technical Specifications. Accessed: 2024.
65. Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.
66. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H.: Mixed precision training. arXiv preprint arXiv:1710.03740, 2017.
67. NVIDIA Corporation: Cuda c++ programming guide. <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>, 2023. CUDA Toolkit Documentation. Accessed: 2024.
68. Bradski, G.: OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000. Open Source Computer Vision Library.
69. Hartley, R. and Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge, UK, Cambridge University Press, 2nd edition, 2003.

## VITA

**NAME** Ragib Rownak

**EDUCATION** B.Sc. Mechanical Engineering, Islamic University of Technology, Bangladesh 2022.  
M.Sc. Mechanical Engineering, University of Illinois Chicago, Chicago, IL, USA, 2025.