Autonomous Pothole Detection Rover

Document Version: 4.1

Authors:
Louis Marleau
Maxime Couture
Nathan Gawargy
Syed Yead Zaman

University of Ottawa
Department of Electrical Engineering and Computer Science
Fall 2025

1 Document Control Information

1.1 Document Approval

Name	Title	E-Mail	Date
Louis Marleau	Hardware & Robotics Developer	lmarl090@uottawa.ca	12/12/2025
Maxime Couture	Hardware & Robotics Developer	mcout088@uottawa.ca	12/12/2025
Nathan Gawargy	Full Stack & AI Developer	ngawa073@uottawa.ca	12/12/2025
Syed Yead Zaman	Software & DevOps Developer	szama023@uottawa.ca	12/12/2025

1.2 Change History

Version	Date	Change Description
1.0	30/01/2025	Original CEG4912 Midterm Report Draft
1.1	22/02/2025	Revised CEG4912 Midterm Report
2.0	10/04/2025	CEG4912 Final Report Draft
2.1	11/04/2025	Revised CEG4912 Final Report
3.0	16/10/2025	Original CEG4913 Midterm Report
3.1	18/10/2025	Revised CEG4913 Midterm Report
4.0	10/12/2025	Initial CEG4913 Final Report Draft
4.1	12/12/2025	Final Report

2 Table of Contents

1 DOCUMENT CONTROL INFORMATION 2

1.1 Document Approval 2
1.2 Change History 2

2 TABLE OF CONTENTS 3

3 INTRODUCTION 5

3.1 Project Description 5
3.2 Goals, Objectives & Scope 5
3.3 Assumptions, Constraints & Risks 5
3.4 Project Deliverables 6
3.5 Schedule & Budget Summary 6
3.6 Acronyms, Terms and Definitions 7
3.7 References 8
3.8 Limitations, Issues and Concerns 10
3.9 Change of information 10
3.10 Confidential Information 10
3.11 Third Party Confidentiality Restriction 10

4 ROLES & RESPONSIBILITIES 11

4.1 Objective 11
4.2 Project Stages 11
4.3 Clients 11
4.4 Participants 12
4.5 Market Space and Industry Sector 12

5 REQUIREMENTS 12

5.1 Functional Requirements 12
5.2 Non-Functional Requirements 13
5.3 Constraints 13
5.4 Risk Analysis 13

6 SOFTWARE ARCHITECTURE 14

6.1 High Level Design 14
6.2 Front-End (Graphical User Interface) 16
6.2.1 Overview 16
6.2.2 Figma UI Mockup 16
6.2.3 Implementation of UI Pages 17
6.3 Back-End 21
6.3.1 Rover Operations 21
6.3.2 Communication Between Web Application & Rover 23
6.3.3 Autonomous Navigation 24
6.3.4 Database Architecture & Interface 25
6.3.5 AI Detection & Analysis Pipeline 25
6.3.6 CI/CD Pipeline & Deployment with Docker 33

7 HARDWARE DESIGN 34

7.1 Hardware Components 34
7.2 Hardware Specifications 36
7.3 Power Consumption Breakdown 37

7.4 HARDWARE BLOCK DIAGRAM 38
7.5 HARDWARE DEVELOPMENT 38
7.5.1 Design & Planning 38
7.5.2 Hardware Integration 40
8 TESTING 41
8.1 SOFTWARE 41
8.2 HARDWARE TESTING 43
8.2.1 Camera Tests 44
8.2.2 Hardware Mount and Fit Test 45
8.2.3 Outdoor Tests 45
9 PROJECT MANAGEMENT 46
9.1 PROTOCOLS AND PROCEDURES 46
9.2 TASKS AND TIMELINES 46
9.3 WORK BREAKDOWN STRUCTURE 48
9.4 MANAGEMENT BOARDS 49
10 FUTURE WORKS & CLOSING REMARKS 50
11 PROJECT CODE 50

3 Introduction

3.1 Project Description

This document outlines the architecture and high-level design of the Capstone Project developed by the members of this team. It is intended for the course professor and the supervisors of the project. Over the course of two terms, the team developed an Autonomous Pothole Detection Rover which is designed to autonomously navigate a predefined path while identifying street damages, specifically potholes. The following sections of the document will describe the key components and functionality of the project in detail.

3.2 Goals, Objectives & Scope

The goal of the project is to help the City of Ottawa plan renovations ahead of time by detecting potential risks along roads and sidewalks faster, by developing an autonomous rover using artificial intelligence to detect and categorize potholes. Traditional methods of identifying and repairing such defects are labor-intensive, time-consuming, and often rely on individuals in the population to submit complaints or requests for repairs. The proposed solution seeks to automate the process of road condition monitoring in a proactive fashion, reducing the need for human interaction and manual inspections. The ideal use case for this project is to deploy multiple autonomous rovers throughout the city to detect potholes and transmit real-time updates. Each rover will provide real-time updates about its findings, location and progress along the path, enabling remote monitoring and control. The project aims to incorporate and fulfill the following high-level objectives:

- Implement an autonomous navigation algorithm and logic on a rover to follow a predefined path and perform obstacle avoidance.
- Design and train an AI computer vision model to detect potholes in real-time.
- Develop categorization and image processing ML pipeline to process and categorize pothole images to facilitate planning for repairs.
- Build a web-based application for remote control and monitoring, providing live metrics for the user.
- Provide real-time communication between the rover and the software.

The scope of this document is to detail the design and interactions of the software and hardware components used in the system.

3.3 Assumptions, Constraints & Risks

The project is intended to be an advanced functional prototype, and due to limitations in time and budget, the chosen hardware such as the Raspberry Pi is sufficient to demonstrate the core functionality. The ideal solution for wireless communication in this project would be cellular communication; however, this option is not feasible due to the previously mentioned limitations. As an alternative, a Wi-Fi module has been selected to enable communication between the rover and the software system. It is assumed that the hardware selected for the system will perform adequately, nevertheless for a production-ready system, higher performance hardware may be required to ensure faster real-time processing and better overall results. Therefore, there are risks associated with testing the final prototype in real-world environments, as factors such as dirt, weather conditions, and hardware limitations could affect the accuracy and reliability of the system. These risks may lead to the prototype underperforming or encountering issues not evident during controlled testing. It is important to note that the rover is not designed to be driven on roads and does not fall under the Motor Vehicle Safety Act [1].

3.4 Project Deliverables

Table 1: Project Deliverables

Deliverable	Description	Date
Project Proposal	Document describing the project, rationale, requirements, high-level architecture, and planning diagrams.	17/01/2025
Project Proposal Presentation	Presentation of contents of the project proposal document.	22/01/2025
CEG4912 Midterm Presentation & Demonstration	Presentation of project progress with details on planning and management. Demonstration of some software and hardware components working in isolation.	10/02/2025
CEG4912 Midterm Report	Document presenting the developments of the project at the midpoint of the 1st term.	24/02/2025
CEG4912 Final Presentation & Demonstration	Presentation of the state of the project at the end of the term. Demonstration of a basic functional prototype at the end of the 1st term.	01/04/2025
CEG4912 Final Report	Document presenting the developments of the project throughout the 1st term.	14/04/2025
CEG4913 Midterm Presentation & Demonstration	Presentation of the state of the project and emphasis on integration of hardware and software components. Demonstration of progress made in 2nd term.	10/20/2025
CEG4913 Midterm Report	Document presenting the developments of the project at the midpoint of the 2nd term.	20/10/2025
CEG4913 Final Presentation & Demonstration	Presentation of the final product and details of the architecture between the integrated components. Demonstration of the final product.	28/11/2025
CEG4913 Final Report	Document presenting the work completed during the whole project.	15/12/2025

3.5 Schedule & Budget Summary

The tables below provide insights into the projected schedule for the CEG4912 and CEG4913 terms, as well as the estimated budget required for the entire project.

Table 2: Tentative Schedule for CEG4912 and CEG4913

Item	Estimated Completion Date
Requirements Gathering *(CEG4912 – 1st Work Term)*	31/01/2025
Architecture Design	03/02/2025
Hardware Procurement	07/02/2025
Basic Hardware Setup	14/02/2025
AI Pothole Detection Model	14/02/2025
LiDAR Obstacle Detection & GPS Setup	21/02/2025
Path Planning Algorithm	28/02/2025
Basic Autonomous Navigation	07/03/2025
AI Pothole Assessment Model	14/03/2025
Basic Front-end UI	21/03/2025
Communication between Rover and Software	28/03/2025
Full Autonomous Navigation *(CEG4913 – 2nd Work Term)*	29/09/2025
Final Front-end Design	29/09/2025
Installation of Camara and LiDAR Mounts	06/10/2025
System Integration (Back-end, Front-end & Hardware)	03/11/2025
Final Deployment and Testing	17/11/2025

Table 3: Budget Estimate for the Project

Item	Cost
LiDAR	$0 (borrowed from Lab Technician)
Power Supply (Batteries)	$35
Raspberry Pi 5 full kit	$200
USB Webcam	$30
IMX219 Camera Module x2	$50 ($25 per camera)
GPS module	$20
Rover with onboard ESP32	$150
Total	**$485**

3.6 Acronyms, Terms and Definitions

Table 4: Table of Definitions, Acronyms and Terms

Terms, acronyms, abbreviations	Definitions
ADC	Analog to Digital Converter
AI	Artificial Intelligence
API	Application Programming Interface
AV	Autonomous Vehicle
CI/CD	Continuous Integration and Deployment
CSI	Camera Serial Interface
DAC	Digital to Analog Converter
FOV	Field of View
FPS	Frames Per Second
GPS	Global Positioning System
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
ML	Machine Learning
m	Meter
NMEA	National Marine Electronics Association
ORM	Object-Relational Mapping
ROS2	Robot Operating System
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
UI	User Interface
UE5	Unreal Engine 5
USB	Universal Serial Bus
WIP	Work In Progress
YOLOv5	Computer vision model used for object detection
3D	Three dimensional

3.7 References

[1] Governement of Canada, "Motor Vehicle Safety Act," 6 May 1993. [Online]. Available: https://laws-lois.justice.gc.ca/eng/acts/m-10.01/page-1.html.

[2] GNU, "GNU AFFERO GENERAL PUBLIC LICENSE," 19 November 2007. [Online]. Available: https://www.gnu.org/licenses/agpl-3.0.en.html.

[3] R. d. Boer, "Leaflet maps marker power [Image Taken From Reference]," 21 December 2019. [Online]. Available: https://rikdeboer.medium.com/leaflet-maps-marker-fun-games-53d81fdd2f52.

[4] Storyblok, "Integrate Storyblok into your SvelteKit application [Image Taken From Reference]," [Online]. Available: https://www.storyblok.com/tc/sveltekit.

[5] L. Marx, "Learn how to use TypeScript to build Angular Apps [Image Taken From Reference]," 21 August 2017. [Online]. Available: https://malcoded.com/posts/get-started-typescript/.

[6] B. Morelli, "25 HTML & CSS Tutorials [Image Taken From Reference]," 19 March 2018. [Online]. Available: https://codeburst.io/25-html-css-tutorials-6a864f387185.

[7] Wikipedia, "PostgreSQL [Image Taken From Reference]," [Online]. Available: https://en.wikipedia.org/wiki/PostgreSQL.

[8] "[Image Taken From Reference]," [Online]. Available: https://worldvectorlogo.com/logo/python-3.

[9] S. Macenski, "All-New ROS2 Navigation Logo! [Image Taken From Reference]," November 2020. [Online]. Available: https://discourse.ros.org/t/all-new-ros2-navigation-logo/17262.

[10] J. Houston, "ROS 2: The Transition from Research to Production [Image Taken From Reference]," 6 July 2022. [Online]. Available: https://www.freshconsulting.com/insights/blog/ros-2-the-transition-from-research-to-production/.

[11] N. Emadamerho-Atori, "Beginner's Guide to Svelte and SvelteKit," 4 October 2023. [Online]. Available: https://prismic.io/blog/svelte-and-sveltekit.

[12] E. Uchenna, "Choosing between React and Svelte," Prismic, 20 October 2023. [Online]. Available: https://prismic.io/blog/svelte-vs-react.

[13] J. Glenn, "yolov5 (GitHub Repository)," Ultralytics, [Online]. Available: https://github.com/ultralytics/yolov5.

[14] G. Fan, T. Lim, M. Xinyin, D. Komorowicz, B. Zhang and X. Ao, "DeepLabV3Plus-Pytorch (GitHub Repository)," [Online]. Available: https://github.com/VainF/DeepLabV3Plus-Pytorch.

[15] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng and H. Zhao, "Depth Anything V2 (GitHub Repository)," DepthAnything, [Online]. Available: https://github.com/DepthAnything/Depth-Anything-V2.

[16] Raspberry Pi Ltd, "Raspberry Pi 4 Model B," April 2024. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.

[17] Raspberry Pi Ltd, "Raspberry Pi 5," January 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.

[18] Nvidia Corporation, "Jetson Nano," [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano.

[19] Nabto Company, "ESP32 for IoT: A Complete Guide," [Online]. Available: https://www.nabto.com/guide-to-iot-esp-32/.

[20] STMicroelectronics, "STM32F103ZE," [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f103ze.html#overview.

[21] Arduino, "Arduino UNO R3," 5 February 2025. [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf.

[22] Waveshare Electronics, "WAVE ROVER Flexible And Expandable 4WD Mobile Robot

Chassis, Full Metal Body, Multiple Hosts Support, With Onboard ESP32 Module," [Online]. Available: https://www.waveshare.com/wave-rover.htm.

[23] Freenove, "Freenove 4WD Car Kit (Compatible with Arduino IDE)," [Online]. Available: https://store.freenove.com/products/fnk0041.

[24] Elegoo, "ELEGOO UNO R3 Project Smart Robot Car Kit V4 with UNO R3," Amazon, [Online]. Available: https://www.amazon.ca/ELEGOO-Ultrasonic-Bluetooth-Intelligent-Educational/dp/B07485YQP8.

[25] RobotShop, "STM32 Smart Car with Multifunctional Development Board," [Online]. Available: https://ca.robotshop.com/products/yahboom-stm32-smart-car-with-multifunctional-development-board.

[26] Slamtec, "RPLIDAR A2," [Online]. Available: https://www.slamtec.com/en/Lidar/a2.

[27] Slamtec, "RPLIDAR A1," [Online]. Available: https://www.slamtec.com/en/lidar/a1.

[28] Beitian, "GNSS Module BN-220," [Online]. Available: https://www.beitian.com/en/sys-pd/863.html.

[29] Beitian, "Beitian BN-880 GPS," Maxterdrone, [Online]. Available: https://maxterdrone.com/en/gps-modules/1193-beitian-bn-880-gps.html.

[30] u-blox Holding, "NEO-6 series Versatile u-blox 6 GPS modules," [Online]. Available: https://www.u-blox.com/en/product/neo-6-series.

[31] Waveshare, "IMX219-120 Camera," [Online]. Available: https://www.waveshare.com/wiki/IMX219-120_Camera.

[32] Raspberry Pi Ltd, "About the Camera Modules," [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[33] Waveshare, "General Driver board for Robots, Based on ESP32, multi-functional, supports WIFI, Bluetooth and ESP-NOW communications [Image Taken From Reference]," [Online]. Available: https://www.waveshare.com/general-driver-for-robots.htm.

[34] SparkFun, "RPLIDAR A3M1 360° Laser Range Scanner [Image Taken From Reference]," Core Electronics, [Online]. Available: https://core-electronics.com.au/rplidar-a3m1-360-laser-range-scanner.html.

[35] Cirkit Designer, "How to Use BN- 220 GPS: Examples, Pinouts, and Specs [Image Taken From Reference]," [Online]. Available: https://docs.cirkitdesigner.com/component/8ec5701f-60ff-46f9-b19e-a1b1737e2bdf/bn-220-gps.

[36] Raspberry Pi, "Raspberry Pi 5/8GB [Image Taken From Reference]," PiShop, [Online]. Available: https://www.pishop.ca/product/raspberry-pi-5-8gb/.

[37] Elecrow, "IMX219 8MP Camera For Raspberry Pi 5, CSI Camera Module sensor with 77°/120°/160° [Image Taken From Reference]," [Online]. Available: https://www.elecrow.com/imx219-8mp-camera-for-raspberry-pi-5-csi-camera-module-sensor-with-77-120-160.html.

[38] Zootealy, "Full HD 1080P Wide Angle USB Webcam USB2.0 Drive-Free With Mic Web Cam Laptop Online [image Taken From Reference]," Walmart, [Online]. Available: https://www.walmart.ca/en/ip/Full-HD-1080P-Wide-Angle-USB-Webcam-USB2-0-Drive-Free-With-Mic-Web-Cam-Laptop-Online/0S1I0FZH9MV9.

[39] Svenirven, "2 Packs 18650 3.7V Rechargeable Flat Top 18650 Batteries for Headlamp, LED Flashlight, Electronic Devices etc (Green) [Image Taken From Reference]," Amazon, [Online]. Available: https://www.amazon.ca/Rechargeable-Batteries-Headlamp-Flashlight-Electronic/dp/B0BPFBG7Z3.

3.8 Limitations, Issues and Concerns

Due to time and budget constraints, an advanced prototype will be developed, and as such, certain aspects of the project will be limited in scope to ensure feasible progress. Below are the key limitations, issues, and concerns for the project:

1. **Dataset for AI Model Training**
   The dataset used to train the AI classification model will be limited in size due to the constraints of time and resources. The lack of time to create a sufficiently large dataset tailored to the project's use case means that the model will rely on a combination of publicly available datasets and images, and photos taken by the members of this group. This may affect the model's accuracy and generalization when deployed in real-world conditions.
2. **Deployment and Hardware Components**
   The final product may encounter challenges when deployed and tested in real-world scenarios, as factors such as road conditions and weather may hinder the final prototype's performance. Due to budget constraints, the hardware components selected will perform adequately for the project's needs but may not be optimized for maximum performance. Additionally, due to the wireless and autonomous nature of the project, power consumption as well as connectivity need to be heavily considered.
3. **Scalability**
   An important requirement for the project is the ability to support multiple rovers running concurrently while sending data in real-time back to the off-site user. Given that only one rover will be designed and built, this requirement can be validated by simulating multiple rovers. Synthetic data could be generated and sent to the user, creating the illusion that several rovers are connected and operating concurrently.

3.9 Change of information

The information in this document is for informational purposes and may change at the sole discretion of the authors of this project who are enrolled in the course CEG4912 and CEG4913 without notice.

3.10 Confidential Information

This document contains confidential information regarding the Autonomous Pothole Detection Rover design specification information and is purely intended for the professor and supervisors of the Capstone project (CEG4912 & CEG4913) and not for release/disclosure in whole or in part to any other party unless agreed upon in writing by the authors of the project.

3.11 Third Party Confidentiality Restriction

This project may incorporate software or libraries licensed under the GNU General Public License. While this license does not impose confidentiality restrictions, it requires that the source code be made publicly available when distributing the software. No Non-Disclosure Agreements (NDAs) apply to the use of these open-source components. The AI pothole pipeline developed and created by the members of this group utilizes the YOLOv5, DeepLabV3+, and Depth-Anything-V2 models. The source code and all credit for these models belong to their respective authors and organizations. As a result, this project is also subject to the licenses governing these models.

4 Roles & Responsibilities

4.1 Objective

The objective of assigning diverse roles and responsibilities to each team member is to efficiently distribute the workload and enable a more focused approach to the different aspects of the project. It is important to consider each member's prior experience and skills developed throughout their respective careers to ensure that the roles and tasks are assigned effectively.

4.2 Project Stages

The project was separated into five different stages, allowing a structured and measured approach to the overall progression of the work required for the project. The five stages which were used throughout the development cycle of the project are presented below, with a high-level description of the tasks required per stage.

- **Planning:** requirement gathering, procurement, architecture design
- **Development:** front-end and back-end design, AI pipeline construction and development, rover control systems, autonomous navigation and obstacle avoidance algorithms
- **Integration:** connection and setup of hardware to the rover, communication between software and hardware modules, 3D designs of hardware mounts for the cameras, integration of hardware components on the rover body
- **Testing:** validation of different aspects of the project (unit tests, AI model tests, integration tests)
- **Documentation**: recording the progress of work in reports and presentations to maintain visibility and ensure proper traceability

Additional details and a breakdown of the required tasks per project stage are presented in the Project Management section in this report.

4.3 Clients

Table 5: Clients associated with the project

Client	Background	Main Interest
University of Ottawa	Academic institution overseeing the development of the project. The authors of this project are also members of the University of Ottawa.	Focused on advancing research and practical applications for the Capstone project.
Transport Agencies	Government and private organizations focused on road maintenance and infrastructure management.	Seeking cost-effective solutions for automated road maintenance and pothole detection to optimize planning and repairs, and reduce time-consuming tasks.
Community and Public	The public, local residents, and communities who are concerned with the condition of the roads.	Interested in safer roads and better maintenance which leads to a reduced number of accidents.

4.4 Participants

Table 6: Project Participants

Person	Contact	Role	Contribution
Louis Marleau	lmarl090@ uottawa.ca	Hardware and Robotics Developer	Hardware design, power consumption breakdown, 3D printed designs, embedded development on rover, GPS module and autonomous navigation
Maxime Couture	mcout088@ uottawa.ca	Hardware and Robotics Developer	Hardware design, UE5 simulations, embedded development on rover, LiDAR configuration and integration, web application development
Nathan Gawargy	ngawa073@ uottawa.ca	Project Manager, AI and Software Developer	Project coordination and management, embedded development on rover, developing AI/ML pipeline, communication stack between rover and software, autonomous navigation, web application development
Syed Yead Zaman	szama023@ uottawa.ca	Software and DevOps Developer	Web application development, database and API access, setup CI/CD pipeline, containerize applications with Docker

4.5 Market Space and Industry Sector

The autonomous rover will operate within the smart city and urban infrastructure market. It aims to address the growing demand for AI-driven solutions in municipal road maintenance. It also serves the transportation sector by providing real-time road surface monitoring, assisting local governments, more specifically the City of Ottawa, and city planners in proactive infrastructure management.

5 Requirements

This section outlines the requirements and constraints for the project. These define the criteria that the product must meet and provides a framework for how the system should operate. The requirements also serve as a benchmark for evaluating the project's status and progress.

5.1 Functional Requirements

- **Autonomous Navigation:** The rover shall navigate autonomously, following a path predefined by the user.
- **Obstacle Detection:** Using the LiDAR, the rover shall detect obstacles along its path.
- **Obstacle Avoidance:** The rover shall use the LiDAR data to avoid obstacles.
- **Road Defect Detection**: The AI model shall detect potholes using the rover cameras.
- **Image Detection Interval:** The rover shall periodically scan the live image feeds at a minimum of 5-second intervals to allow for continuous pothole monitoring.
- **Speed Synchronization Requirement:** The rover will adjust its speed as necessary to ensure that its movement aligns with the processing capabilities of the onboard AI classification model.
- **Pothole Assessment:** Detected pothole images shall be analyzed by an AI/ML pipeline to assess their severity based on predefined criteria of the potholes.
- **User Authentication:** The system shall provide a user authentication mechanism.
- **Path Planning:** The user shall be able to define a rudimentary path for the rover to follow.
- **Metrics:** The web application shall display live metrics from the rover.
- **Map Progress:** The dashboard shall display the rover's current location on a map.
- **Map Pins:** The locations of the detected potholes shall be pinned on a map using the GPS location captured by the GPS sensor.

- **Database Explorer:** The dashboard shall provide a comprehensive table of the detected potholes for user interaction.
- **Camera Feeds:** The dashboard shall display live camera feeds from the cameras aboard the rover.
- **Manual Override:** The user shall be able to teleoperate the rover when required.
- **E-Stop**: The user shall be able to stop all operations of the rover, independent of the current rover state.
- **Data Storing:** The data captured by the rover shall be stored in a database to enable efficient tracking of metrics.

5.2 Non-Functional Requirements

- **Scalability:** The web application shall be able to support a minimum of two concurrent rover connections, enabling users to manage multiple rovers simultaneously.
- **Live Data Transmission:** The rover shall transmit captured data to the web application in real-time for monitoring with a latency of a maximum of 15 seconds.
- **Intuitive Controls:** The rover shall respond naturally to user inputs, ensuring that movement inputs correspond naturally to the rover's motion.
- **Battery Life:** The system shall be capable of operating at full load while following its designated path for a minimum duration of 30 minutes.
- **Network Communication:** The rover shall communicate over a stable wireless connection.
- **Usability:** The web application shall have an intuitive UI for ease of use.

5.3 Constraints

- **Vision Model License:** The AGPL 3.0 license used by the YoloV5 model requires the AI code repository to be open-source [2].
- **Depth Model License:** The CC-BY-NC-4.0 license used by the DepthAnythingV2 large model requires that the project is for non-commercial use only.
- **Budget:** The project in its entirety is limited to $500 in total.
- **Testing Conditions:** The rover will be tested in a controlled environment, under strict supervision, since harsh weather and terrain conditions such as snow and ice could affect the performance of the hardware.

5.4 Risk Analysis

The following table presents the potential risks which could arise and a corresponding solution to mitigate each risk. The priority and type of risk is also included.

Table 7: Potential risks with the project

Risk	Priority/ Type	Mitigation/Solution
Excessive vibrations during rover operation may impact the stability of the onboard camera and LiDAR sensor, leading to blurred images and problems with obstacle detection.	High/ Internal	Adding vibration dampers at different mounting points to lower the impact of the vibrations. When mounting, ensure that everything is tightly secured. Having tight screws at the mounting points can help reduce the vibrations. Using a very high shutter speed on the cameras to ignore the vibrations. Run the rover at a slower speed to decrease vibration.
Certain appliances are incompatible with ROS2.	Low/ Internal	Find alternative appliances and components supported in ROS2.
The onboard rover	High/	Move the AI detection model to the server side and

processor may struggle with real-time inference with the AI classification model.	Internal	transmit the video streams back to the server for inference.
The AI pothole detection dataset may not be optimized for the rover's camera height or specific pothole characteristics in the Ottawa region, potentially leading to false detections.	High/ Internal	Take pictures of local potholes and streets in Ottawa to train the AI detection model on a custom dataset tailored for the rover camera angle and height.
Insufficient pothole data for the AI classification model.	Medium/ Internal	Expand the dataset by incorporating a mix of publicly available pothole images, images captured by team members, and artificially augmented images to enhance diversity and size.
Environmental challenges (rain, snow) may affect the rover performance.	Medium/ External	Test the final prototype in controlled environments. Attempt to make the rover more robust by creating more stable and protected mounts for the sensors. The manual override option can be used when the rover is stuck. In cases where it remains stuck, allow for physical intervention to reposition the rover.
Bystander interference	Low/ External	Implement an alarm feature to sound when unauthorized individuals approach the rover, alerting bystanders and deterring interference.
One of the side cameras failing during operation	Low/ Internal	Use the side-mounted camera on the opposite side of the rover and reverse the rover's direction of travel.
Connectivity issues between the rover and the server in real-time	Low/ Internal	Test the rover connectivity in various environments and scenarios to ensure the connection is robust and does not drop out.
GPS inaccuracies or connectivity issues	Medium/ External	Test the GPS in different environments to ensure accuracy. Test different constellations to discover the most accurate.
Server failure or crash	Low/ External	Add mechanism for server monitoring and data backup.

6 Software Architecture

6.1 High Level Design

The diagram below illustrates the software architecture for the project (see Figure 1). The enterprise architecture modeling language ArchiMate was used, and a legend is included on the right side of the diagram. The software can be separated into three different main sections:

1. **Front-End:** GUI which allows the user to perform different actions and view diverse metrics (see Figure 2 for the user actions).
2. **Server Back-End:** Back-end functionalities and processes running on the server-side such as the AI pothole detection and analysis pipeline, the database and the API endpoints used to access and write to the database.
3. **Hardware Back-End:** Back-end software functions running on the Raspberry Pi connected to the rover. In most cases, ROS2 will be used as a central connection hub between the hardware sensors and the software nodes running on the Raspberry Pi.

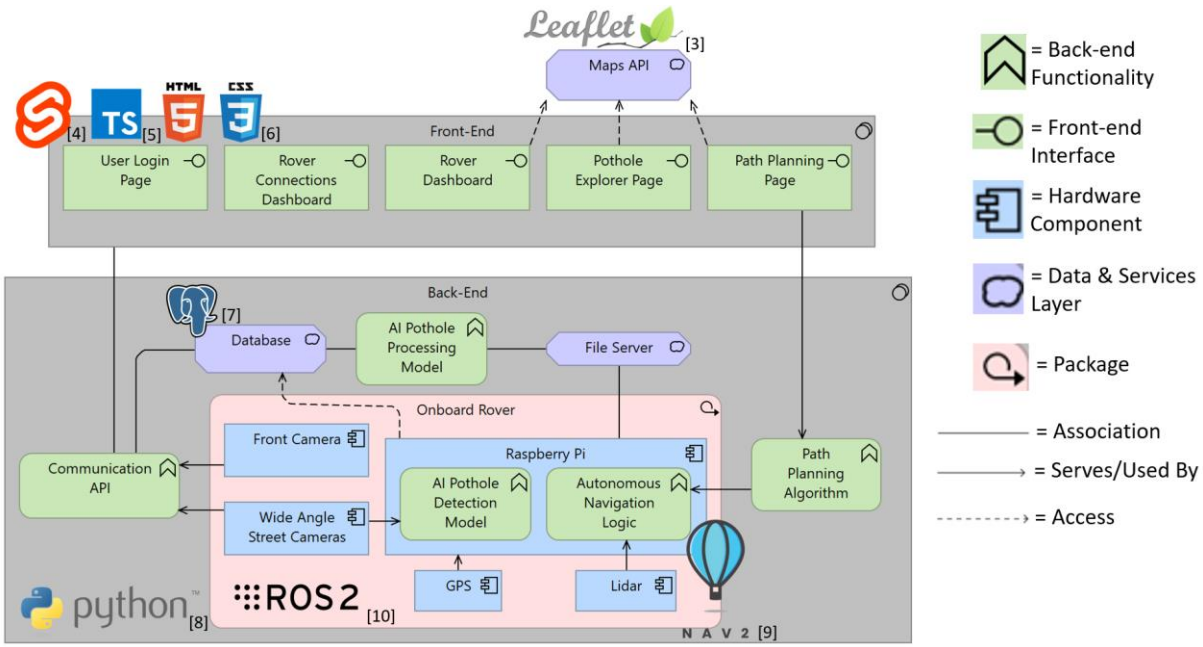

Figure 1: Software Architecture

The following use-case diagram presents the different actions the user can perform and the corresponding high-level responses the system must take for each action.

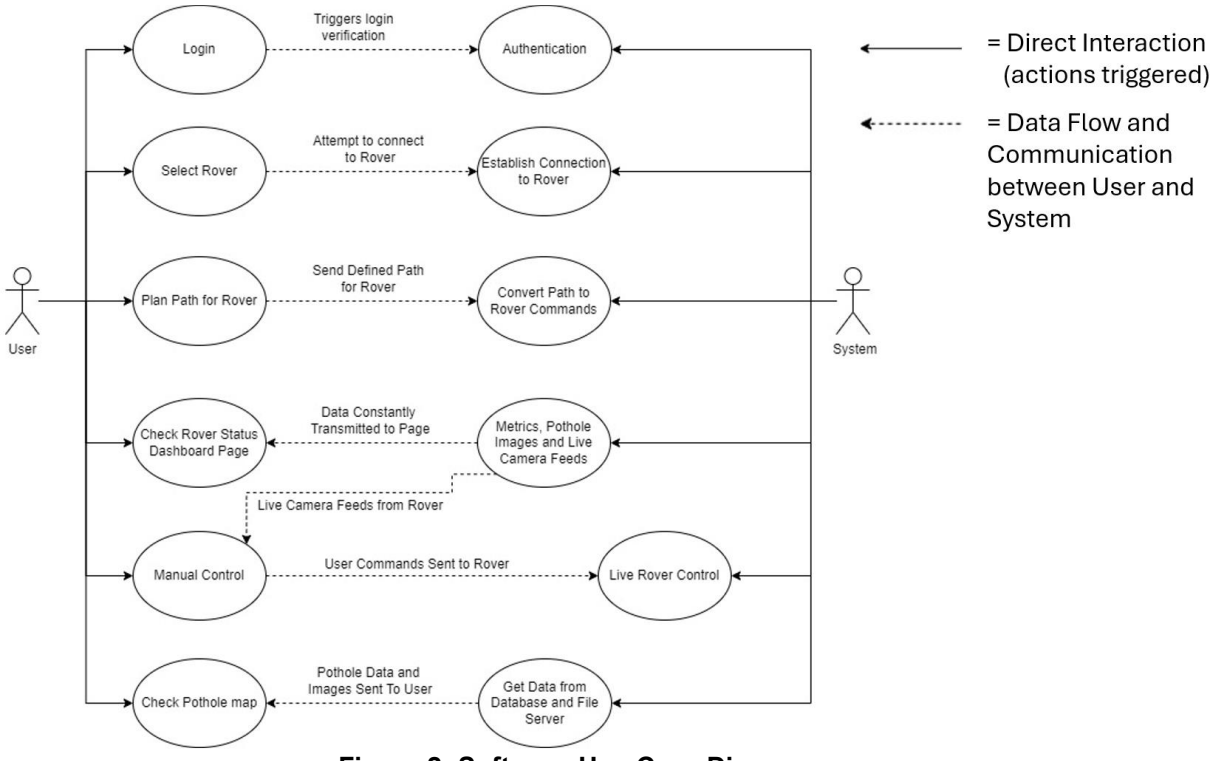

Figure 2: Software Use-Case Diagram

6.2 Front-End (Graphical User Interface)

6.2.1 Overview

While the front-end is built on the standard languages for web applications such as HTML, CSS and TypeScript, many libraries are also used to provide better services and features. Svelte is used as the web framework to compile code to JavaScript, enabling fast and reactive UI development. Specifically, SvelteKit, an application framework built on Svelte, simplifies API handling and server-side rendering [11]. This framework was chosen instead of React as it requires simpler and less code, while providing excellent performance [12]. TailwindCSS and DaisyUI were used to simplify the UI design and provide pre-designed UI components. This accelerates development while providing clean and attractive designs for the website. Leaflet was used to display an interactive map visualization on the front-end. This library is crucial for allowing the user to define the path for the rover to follow autonomously.

6.2.2 Figma UI Mockup

Before the implementation of the web application front-end, a UI mockup was created using Figma. The images below present the concept mockup pages for the web application.

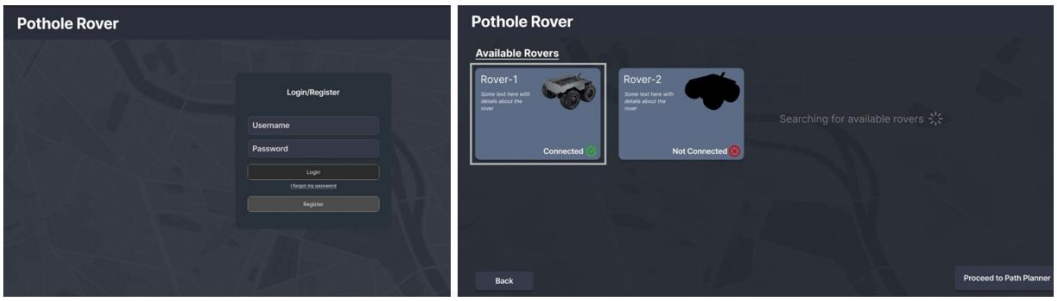

Figure 3: Login Page (left) & Rover Connections (right) Mockup Pages

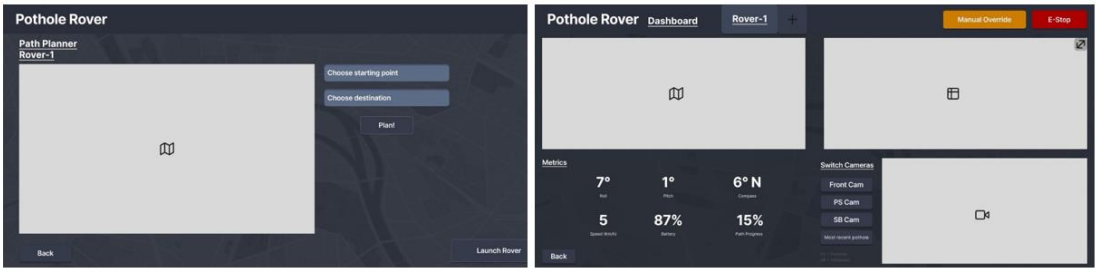

Figure 4: Path Planning (left) & Rover Dashboard (right) Mockup Pages

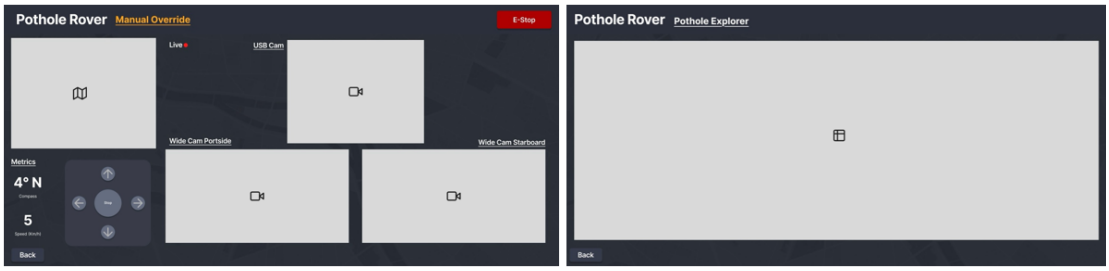

Figure 5: Manual Override (left) & Pothole Explorer (right) Mockup Pages

6.2.3 Implementation of UI Pages

The section presents the implemented pages for the web application. The home landing page is presented in Figure 6. It was important to pick a theme and color scheme which suited the application and which was consistent throughout the different pages.

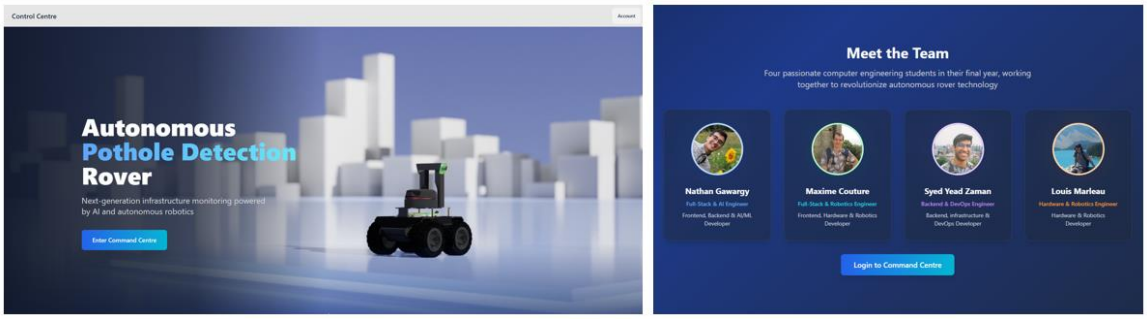

Figure 6: Home Landing Page

The Login/Registration page serves as the entry point for users to either sign in or create a new account (see left image in Figure 7). This page uses the data stored in the database to authenticate users and store user credentials. Once authenticated, the user is redirected to the Rover Control Centre page (see right image in Figure 7). From this main dashboard, users can access three key features:

- **Rovers**: View all available rovers, monitor live metrics of active and navigating rovers, or take manual control. Users can also plan and assign paths for inactive rovers to begin operation.
- **Path History**: Access logs and statistical data from previous rover missions. This feature supports future path planning and trend analysis of previously explored routes.
- **Pothole Explorer**: Review all detected potholes, including severity scores and detection locations. This feature assists users in prioritizing which potholes require inspection or repair.

Once logged in, the user is provided with three navigation buttons at the top of the page, allowing quick access to each of these main sections.

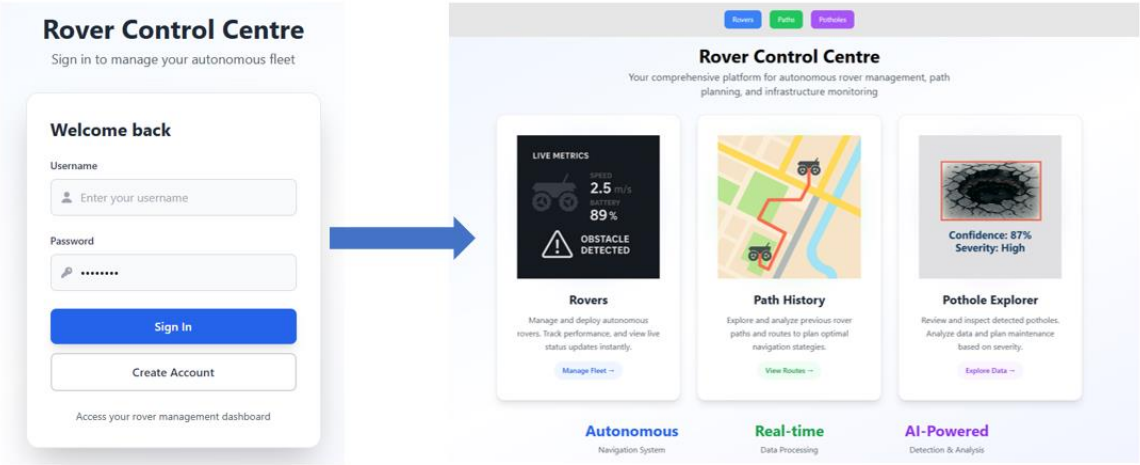

Figure 7: Login/Registration Page (left) & Rover Control Dashboard (right)

Upon entering the Rover section, the user is presented with a list of all rovers to which they have access (see Figure 8). The interface allows filtering between Active and Inactive rovers, as well as searching by rover name. A rover's active status is determined based on a timestamp associated with the latest heartbeat message received by the UI from the rover. Each time a heartbeat is received by the UI, the corresponding timestamp in the database is updated. If the timestamp indicates that more than one minute has passed since the last heartbeat, the rover is considered inactive, meaning it is no longer running or communicating with the rover system.

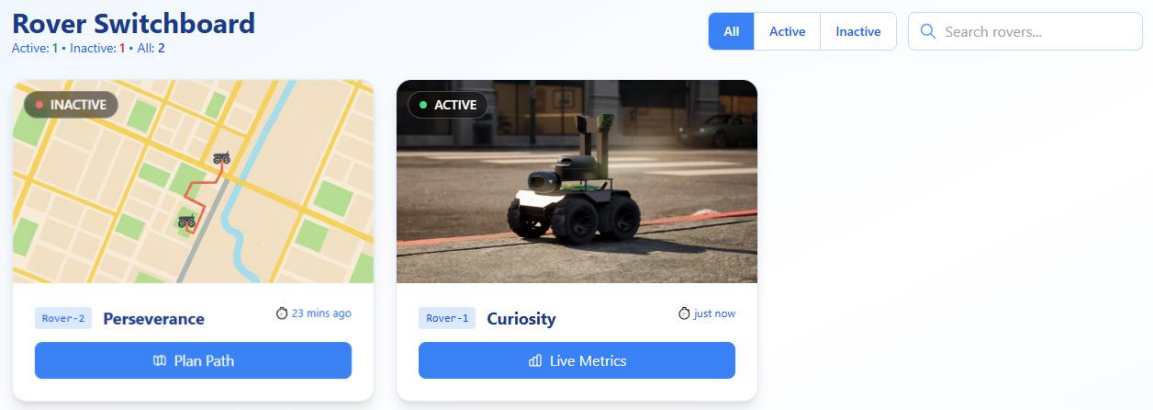

Figure 8: Rover Switchboard Page

If the selected rover is inactive, the user can plan a path by specifying a start and end destination (see left image in Figure 9). After defining the path, the user is redirected to the Launch Page, where they can confirm the launch to initiate the rover's operation (see right image in Figure 9). If the connection and launch are successful, the user is re-directed to the Live Metrics page for the rover.

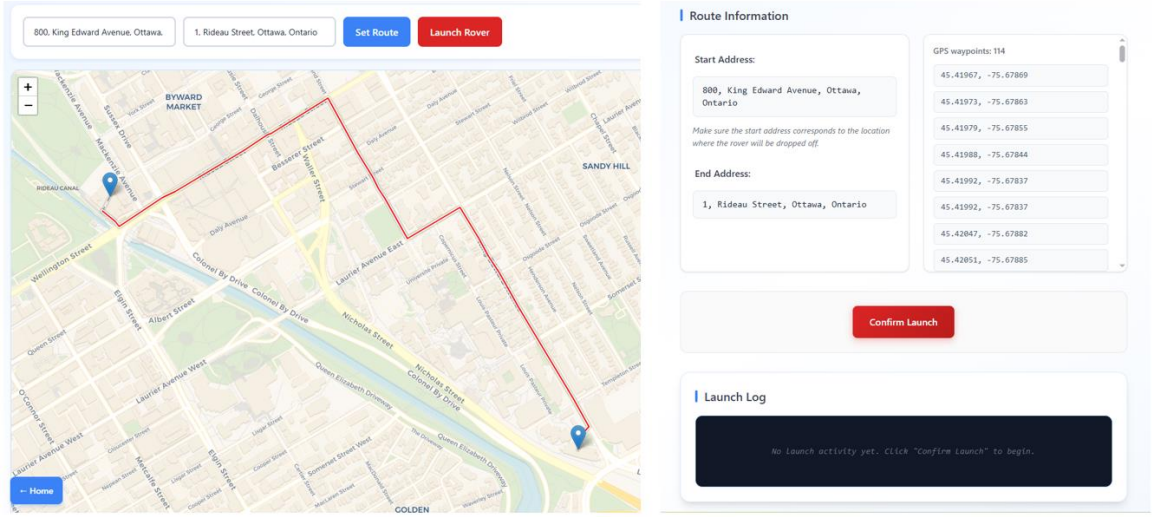

Figure 9: Path Planning Page (left) & Launch Rover Page (right)

If a rover is active, the user can access the Live Metrics Page, where real-time data from the rover is displayed (see Figure 10). From this page, the user can also manually control the rover using the Manual Control interface or trigger an emergency stop if necessary. The live data received from the rover includes its GPS location, battery voltage, IMU readings, LiDAR point cloud, camera streams, and a list of detected potholes. By selecting any detection from the table, a detailed view providing additional information about the specific pothole is provided.

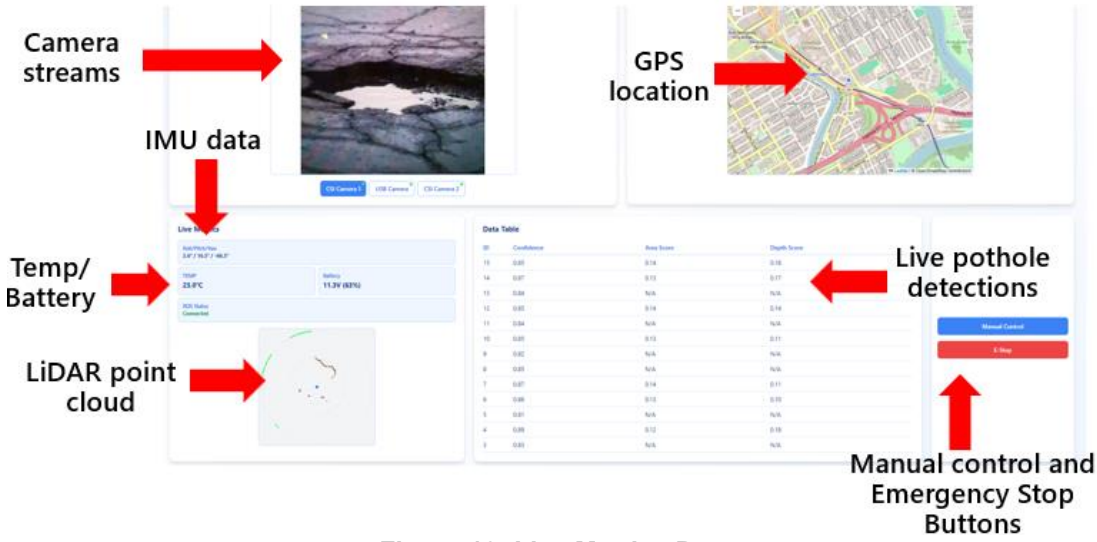

Figure 10: Live Metrics Page

The manual override page allows the user to view the live camera streams from the rover, monitor the LiDAR point cloud data, and manually control the rover using the arrow keys on the interface or the WASD keys on the keyboard (see Figure 11). Using this page, the user can teleoperate the rover remotely from their location using the live camera stream and the LiDAR detections which are updated in real-time. There is also an alert which notifies the user whenever there are obstacles detected in the forward corridor in front of the rover.

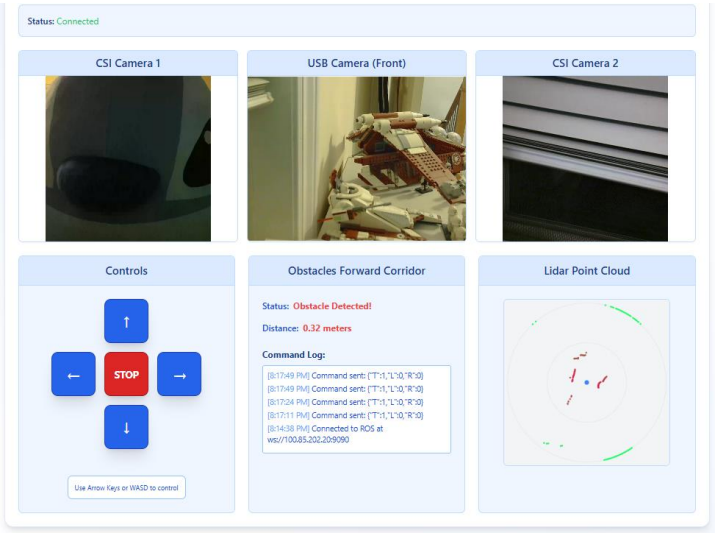

Figure 11: Manual Control Page

The user can also open the Path History page, which displays all previously traveled routes. Selecting any entry opens a detailed view of that route, including the logs the rover recorded and transmitted to the server during its journey (see Figure 12).

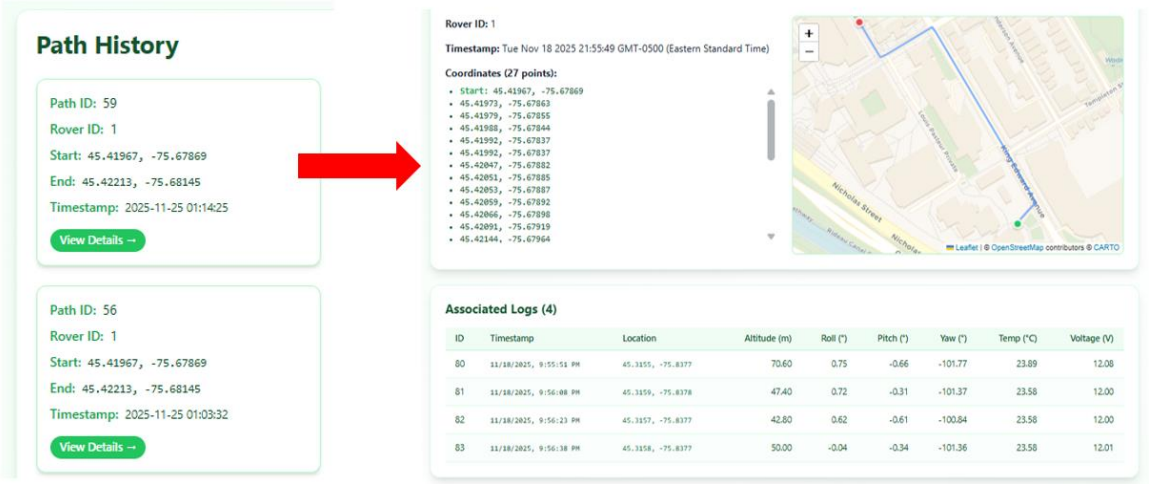

Figure 12: Live Metrics Page

On the Pothole Explorer page, all detected potholes are listed. The user can select any detection to view additional details, including confidence, as well as area and depth severity scores. Each pothole is also assigned a risk label (High, Moderate, or Low), determined by its area and depth estimation scores (see Figure 13).

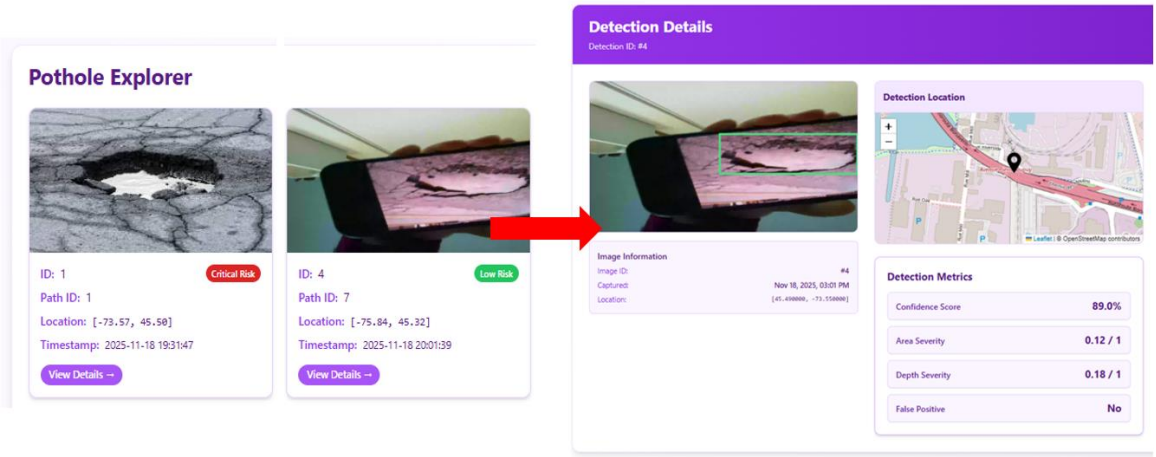

Figure 13: Live Metrics Page

6.3 Back-End

The primary programming language used for most back-end functionality on the rover is Python. Python was also utilized for the AI detection and analysis pipeline. It was chosen for its seamless integration with ROS2 for rover control, as well as its strong compatibility with AI training and deployment frameworks such as PyTorch.

For the web application, TypeScript was used to implement the back-end logic. This includes handling database operations for reading and writing data, establishing connections to the rover to retrieve live metrics, and defining API endpoints for interaction between the different parts of the application and the database.

6.3.1 Rover Operations

This section includes the details for the programs and code running directly on the Raspberry Pi which are used to control the rover and its sensors, as well as communicate directly with the UI.

The architecture diagram in Figure 14 illustrates the software components onboard the rover. The Raspberry Pi functions as the main processing unit, running all programs within individual ROS2 nodes. The system is built on ROS2 which serves as the core framework for managing these modular nodes, with each node responsible for a specific task. Acting as middleware, ROS2 integrates all software components, ensuring efficient communication and coordination between modules. The ESP32 serves as the primary motor controller for the rover, interpreting the commands and directly driving the motors to perform the requested movements. It acts as the main low-level interface, bridging high-level ROS2 commands with physical motor control on the rover.

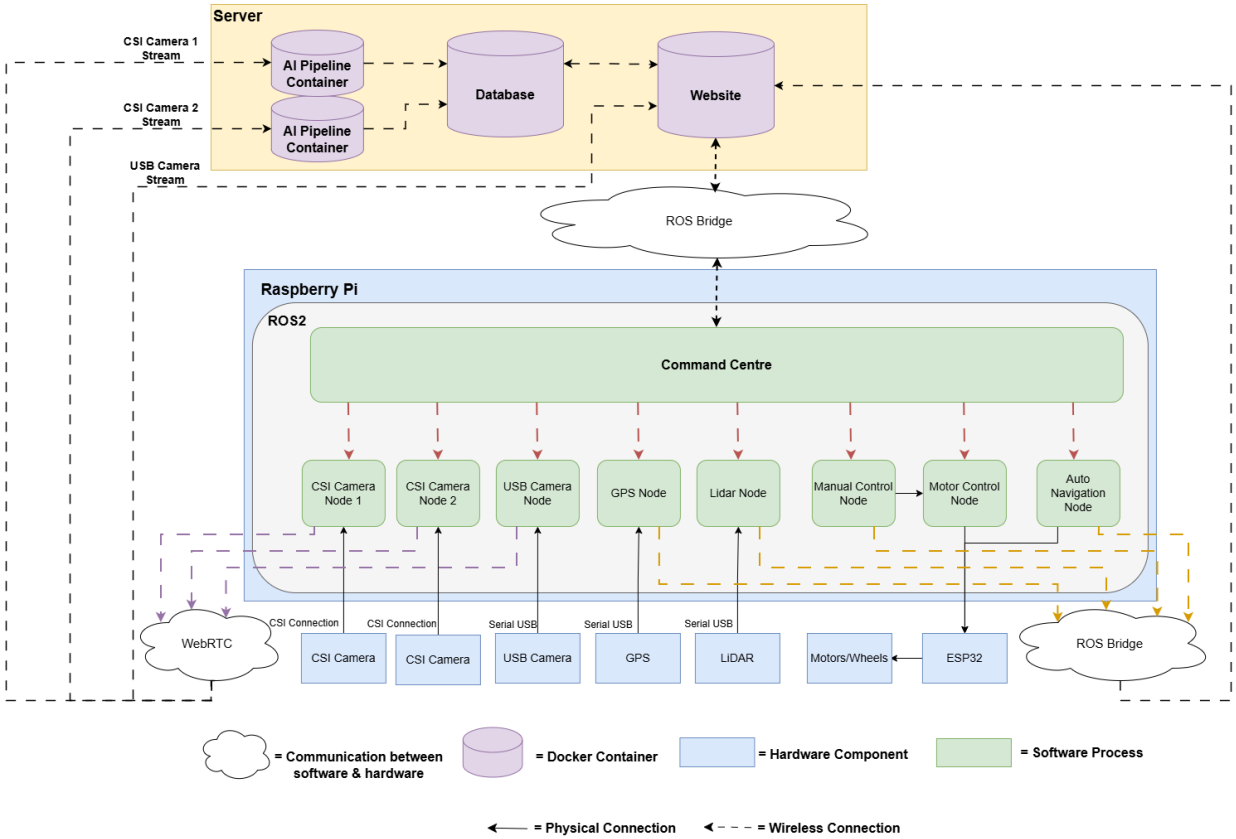

Figure 14: Rover Back-End Architecture

The following ROS2 nodes are utilized onboard the rover:

- **Command Centre Node:** Receives commands from the web application, performs software-side connection and validation, sends heartbeat messages, and ensures that the UI remains connected. Based on the commands received, it determines which nodes should be started or stopped.
- **CSI Camera Nodes (1 & 2):** Handles the side cameras on the rover, streaming live video feeds to the server where pothole detection and analysis are performed in real time on the incoming frames.
- **USB Camera Node:** Activates the USB camera and streams the video to the UI via WebRTC. The front-facing camera serves as the primary driving camera, allowing the user to view and control the rover.
- **GPS Node:** Provides real-time location data used in the autonomous navigation algorithm and sends the rover's current position back to the UI.
- **LiDAR Node:** Activates and manages the LiDAR sensor, providing distance and obstacle data for mapping and navigation.
- **Manual Control Node:** Receives input commands from the UI and transmits them to the Motor Control Node to drive the rover manually.
- **Motor Control Node:** Executes movement commands received from the Manual Control Node to control the rover's motors. This node is connected serially to the ESP32 to transmit motor movement commands. Note that the Manual Control and Motor Control nodes are combined into a single node called the Serial Rover Bridge.
- **Auto Navigation Node:** Utilizes GPS and LiDAR data to perform obstacle detection and execute autonomous navigation algorithms.

The ROS Bridge server is used to establish a WebSocket connection to the front-end, enabling a publish/subscribe model between the web application and the rover. Data is published to distinct ROS2 topics (e.g. lidar_data, gps_data), and the front-end subscribes to the relevant topics to update the UI in real time. The ROS bridge communication method is also used to setup direct communication channels between the Command Centre Node and the web application. For live video streaming, WebRTC is used to transmit the rover's camera feed directly to the UI with minimal latency.

A screenshot captured on the Raspberry Pi showing the active ROS2 nodes and the corresponding Manual Control page is presented in the figure below. This example demonstrates the system's capability to receive real-time data from the individual nodes running on the Raspberry Pi. In this configuration, the user can remotely control the rover while simultaneously monitoring its surroundings through the LiDAR data and camera stream in real time. Note that this image shows the Manual Control page before the latest UI enhancements.

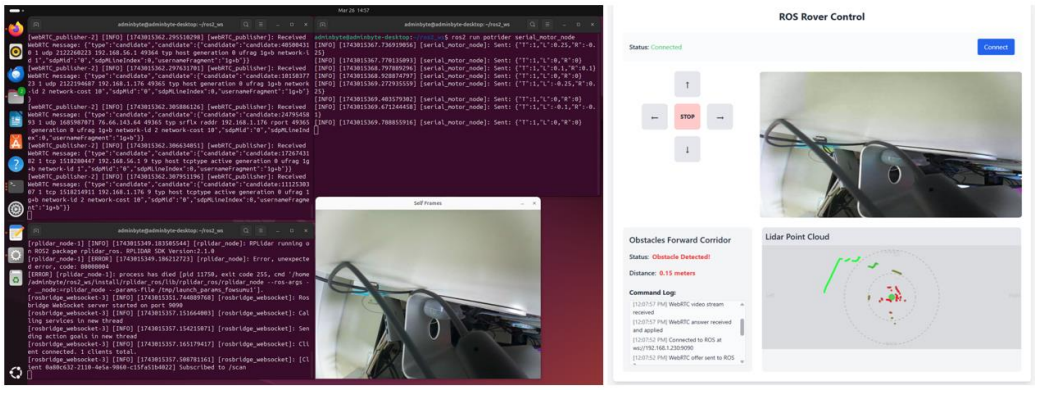

Figure 15: ROS2 nodes running (left) and the connected front-end page (right)

6.3.2 Communication Between Web Application & Rover

To simplify the management of individual ROS2 nodes, the Command Centre node was developed as the primary intermediary between the nodes and the web application (see Figure 16). It serves as the main communication hub, coordinating data exchange, node activation, and status monitoring across the system. Four main communication channels are used between the software-side and the rover:

1. **Rover Commands:** The web application sends operational commands to the rover, such as *Launch*, *Stop*, and *Manual Control*. The Command Centre node interprets these commands and determines which ROS2 nodes should be started or stopped accordingly.
2. **Software Heartbeat:** Once connected to the rover, the web application transmits a heartbeat message every three seconds. The rover uses this signal to verify that the software-side connection remains active.
3. **Rover Heartbeat:** Similar to the software heartbeat, this channel allows the rover to notify the web application that it is still connected. Each time a heartbeat is received from the rover, the corresponding timestamp in the database is updated. As described earlier, this timestamp is used to determine whether the rover is active or inactive.
4. **Node Status:** Reports the operational status of individual ROS2 nodes. For a command to execute successfully, all required nodes must be running. A launch sequence is initiated only when all necessary nodes are active and functioning correctly. An example of a successful *Launch* command is presented in Figure 17.

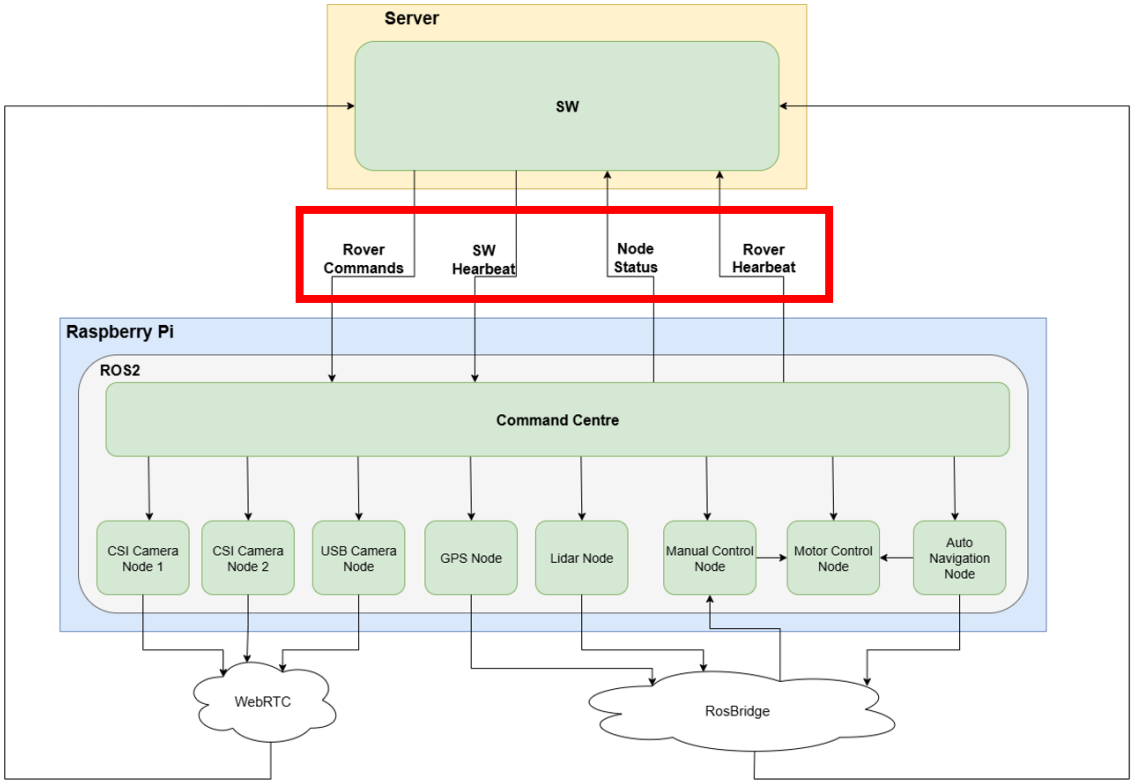

Figure 16: Communication Architecture Between the Rover and the Web Application

It is important to note that individual nodes, such as the LiDAR, GPS, and camera streams, transmit their respective data directly to the UI without passing through the Command Centre node. However, the Command Centre remains responsible for starting and stopping these nodes as part of the overall system management.

The screenshots below demonstrate the *Launch* command in action. When the user clicks the *Launch* button, the UI log gets updated (left image) and establishes a connection with the Command Centre node running on the Raspberry Pi (right image).

Figure 17: Launch Rover Command Connection

6.3.3 Autonomous Navigation

An algorithm was developed to enable autonomous navigation on the rover. GPS, IMU, and LiDAR data are used to orient and guide the rover, as well as to detect and avoid obstacles (see Figure 18). Each hardware component connected to the rover operates through its own dedicated ROS2 node, responsible for handling its specific task.

Autonomous navigation begins once the user launches the rover from the website. This action prompts the server to send GPS waypoints to the Raspberry Pi. These waypoints are longitude/latitude coordinates that define the rover's route from start to finish. Using its current GPS position along with IMU readings, the algorithm determines the appropriate heading. The IMU is used to identify the rover's orientation, while the GPS indicates its position relative to the next waypoint.

Based on this information, the algorithm continuously computes movement and steering commands in real time. These commands are sent through the Rover Serial Bridge to the ESP32, which then controls the rover's motors. LiDAR data is used for obstacle avoidance. When an obstacle is detected within a defined threshold (default 0.2 meters), the rover stops and steers toward a clear corridor before continuing. Note that when the user switches to the manual control override on the website, autonomous navigation is paused. This allows the user to directly control the rover, and autonomous navigation only resumes once the user returns to the Live Metrics page.

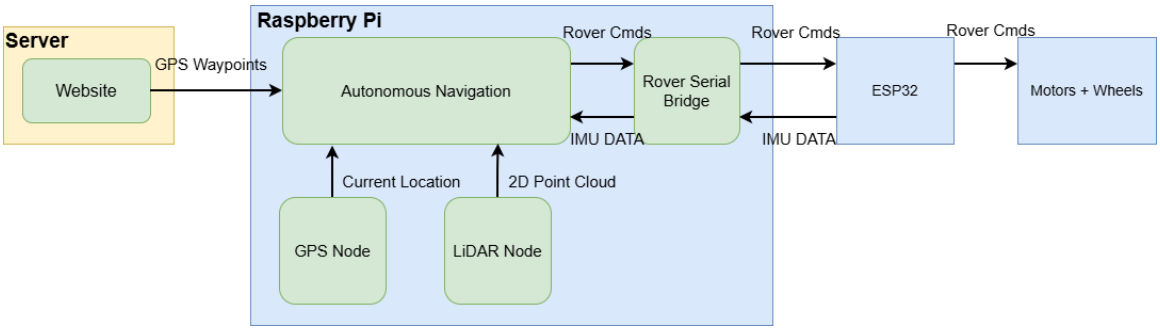

Figure 18: Autonomous Navigation Architecture

6.3.4 Database Architecture & Interface

As shown in the software architecture diagram (see Figure 1), PostgreSQL is used as the database management system to store data such as user credentials, current rover metrics, and detected pothole locations. It was chosen due to its strong performance and support for complex structures and queries. It also has an extension called PostGIS that allows for powerful data types and queries. Drizzle ORM is used by the web application to interact with the PostgreSQL database, assisting in the management of database queries and schema structures. The schema for the database is presented in the image below.

Figure 19: Database Schema Organization

Server-side operations handle database interactions, including retrieving stored information, writing new entries, and updating existing records. SvelteKit provides an easy-to-use API routing system and function-based data access. Using TypeScript within the SvelteKit framework, a set of API endpoints were developed to interface with the database. These endpoints are utilized by both the web application and the pothole detection pipeline application.

6.3.5 AI Detection & Analysis Pipeline

The diagram below demonstrates the full machine learning pipeline used to detect and process potholes in real-time. All stages are performed on the server-side. Once an image or frame is provided, it passes through 6 pipeline stages for comprehensive pothole analysis. The following sections provide further details on the purpose and design of each stage of the pipeline.

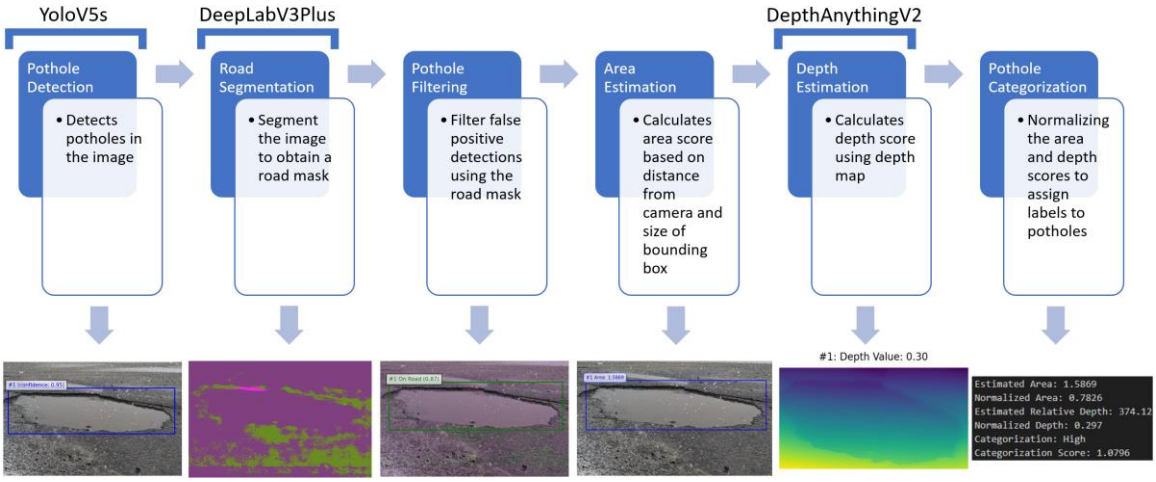

Figure 20: Full AI Detection & Analysis Pipeline Architecture

6.3.5.1 Stage 1: Pothole Detection

The AI pothole classification model was trained using the YOLOv5 model [13]. YOLO provides all the tools necessary to train and create a real-time image detection model. The small model was selected as it is lightweight while providing strong performance. A combination of different publicly available datasets was used to fine-tune the model and multiple training runs were completed to figure out the best weights for the model found during training. The training process involved experimenting with different datasets and hyperparameters, such as the number of epochs, batch size, and augmentation techniques.

To improve the pothole detection model for the project, a custom pothole dataset was created using images captured in the local Ottawa area. This allowed the model to learn and recognize pothole patterns specific to Ottawa's road conditions. Additionally, the images were taken at a similar camera angle and height as those on the rover, ensuring the model was trained on data that closely resembles the conditions in which the rover would be deployed. This helps improve detection accuracy and robustness in practical scenarios. This process involved creating annotations and structuring the labels and the images in the format that the YOLO model requires. Figure 21 provides an example of an image which was taken as part of the custom dataset, and which is being annotated with the open-source library labelImg. This library provides features to create a bounding box around the object in question with an associated target label (i.e. Pothole). The bounding box coordinates are then generated into a corresponding label file for that image.

Figure 21: Custom Image being Annotated with labelImg

The final dataset used to train the model was comprised of pothole images from online datasets, pothole images taken from different streets in the Ottawa area, and images of roads without potholes. It was important to provide images of streets without potholes to train the model to differentiate between potholes and normal roads without potholes. Additionally, images of objects such as cars and pedestrians were also included in the dataset to reduce the likelihood of having false pothole detections. The results below were obtained on 181 sample images as part of the validation split set.

- **Precision:** 0.867
- **Recall:** 0.758
- **Mean Average Precision (mAP with IoU 50):** 0.843

These metrics indicate that the final fine-tuned pothole detection model has high precision and a slightly lower recall. This means the model effectively minimizes false positives, ensuring that most detections are correct. However, the lower recall suggests that some pothole detections could be missed. Despite this trade-off, the results align with the project's objectives. It is preferable to detect fewer potholes with higher accuracy rather than produce more detections with potential false positives, ensuring reliable identification for further analysis. Additionally, the mAP at intersection over union (IoU) 50 shows that the model effectively identifies potholes and places reasonably accurate bounding boxes around them.

6.3.5.2 Stage 2 & 3: Road Segmentation & Pothole Filtering

During the evaluation of the trained pothole detection model, some cases were observed where potholes were incorrectly detected in the background of the image. Although the training dataset included road samples with background elements, the model occasionally struggled to differentiate potholes from background regions such as the sky or buildings (see Figure 22).

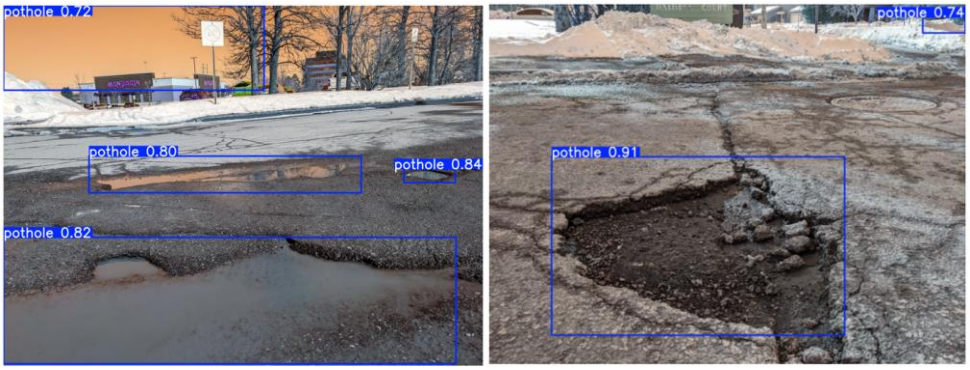

Figure 22: Examples of False Positive Detections

A segmentation stage was introduced to filter out false positives by ensuring that detected potholes were located on the road. The DeepLabV3+ segmentation model [14] was employed to segment the images effectively. The model assigns each pixel in the input image to a specific category. The ResNet-101 architecture was selected for its deeper convolutional network, which offers better accuracy compared to the MobileNet architecture. Additionally, the Cityscapes version of ResNet-101 was chosen, as it comes pretrained on a dataset with 19 classes, including roads, making it particularly suited for the project.

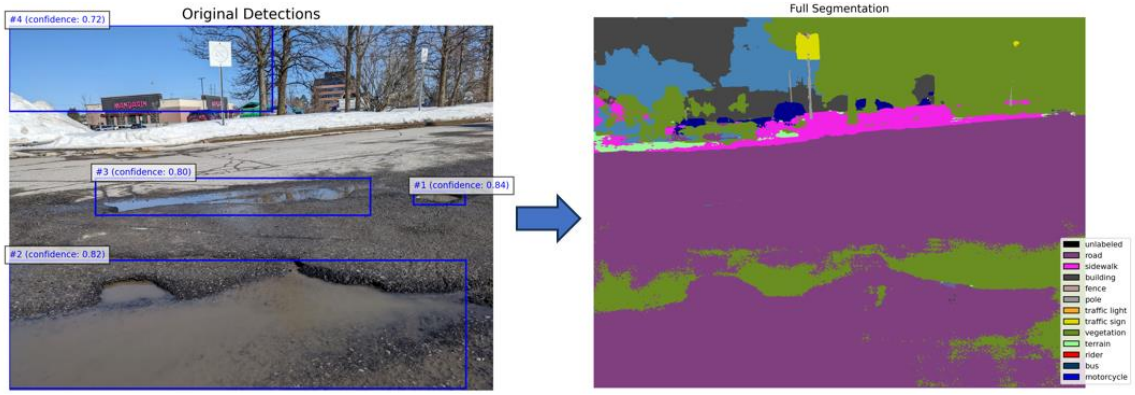

Figure 23: Full Segmentation Pipeline Output

The output from the segmentation model is used to estimate the percentage of pixels assigned to the road class within each detected bounding box (see Figure 23). If more than 60% of the pixels in a bounding box are classified as road, the detection is considered valid and classified as being on the road. This approach enables the filtering out of false pothole detections, ensuring that only the detections with the majority of pixels within their bounding box belonging to the road class are retained.

6.3.5.3 Stage 4: Area Estimation

The area of each pothole is estimated by calculating a relative score based on the size of its bounding box. This is done using the equation below. The scaling function helps refine the estimation, accounting for the fact that potholes appearing farther away in the image (distance in y-axis) or farther away from the center of the image (distance in x-axis) will have different bounding box areas.

$$Area\ Estimation = Bounding\ Box\ Area\ \times Distance\ from\ Camera \times Scaling\ Function$$

It is crucial to consider the location of the pothole in the image because potholes that are farther away from the camera will naturally appear smaller, resulting in smaller bounding box areas, even if their actual area is similar to that of closer potholes. Similarly, the x distance affects the bounding box size as potholes farther from the center of the image tend to have larger bounding boxes. This phenomenon is illustrated in Figure 24, where each white square has the same dimensions and area. It is possible to observe that the bounding boxes decrease in size as the y distance increases, and the bounding box size increases as the squares move farther from the image center.

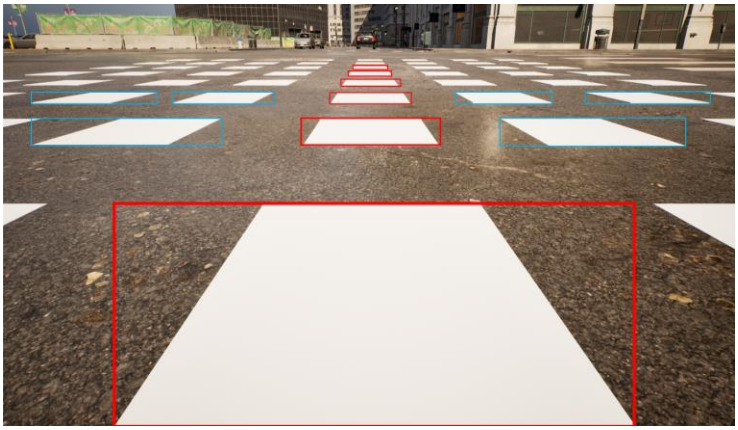

Figure 24: Unreal Engine simulation to estimate the scaling functions

Using Unreal Engine, the image presented above was created to estimate the scaling function needed to approximate the area score. By knowing the actual area of each square (0.25 m²), the corresponding scaling factor was computed by dividing the actual square area by the bounding box area. This scaling factor indicates the value required to multiply the bounding box area to obtain an estimation of the actual physical area of the pothole (see Figure 25 & 26). A scaling function was obtained for both the y and x distances in pixels.

Rectangle	Distance from camera to centre of rectangle (pixels)	y-axis distance from camera (pixels)	Bounding box coordinates (top left, bottom right)		Bounding box area (pixel^2)	Actual square area (m^2)	Scaling Factor Needed (m^2/pixel^2)
1	(1662, 1972)	1972	(508, 1484)	(2816, 2460)	2252608	0.25	0.000000110982470
2	(1642, 1158)	1158	(1340, 1096)	(1944, 1220)	74896	0.25	0.000003337961974
3	(1648, 1006)	1006	(1468, 980)	(1828, 1032)	18720	0.25	0.000013354700855
4	(1643, 941)	941	(1520, 928)	(1766, 954)	6396	0.25	0.000039086929331
5	(1643, 900)	900	(1548, 894)	(1738, 907)	2470	0.25	0.000101214574899
6	(1641, 873)	873	(1564, 869)	(1718, 878)	1386	0.25	0.000180375180375
7	(1641, 854)	854	(1576, 851)	(1706, 857)	780	0.25	0.000320512820513

Figure 25: Y Scaling Factor Data Points

Rectangle	Distance from camera to centre of rectangle (pixels)	x-axis distance from camera (pixels)	Bounding box coordinates (top left, bottom right)		Bounding box area (pixel^2)	Actual square area (m^2)	Scaling Factor Needed (m^2/pixel^2)
1	(417.0, 1010)	417	(138, 981)	(696, 1038)	31806	0.25	0.000007860152172546
2	(985.5, 1010)	986	(759, 981)	(1212, 1038)	25821	0.25	0.000009682041748964
3	(1644.0, 1010)	1644	(1467, 981)	(1821, 1038)	20178	0.25	0.000012389731390624
4	(2232.0, 1010)	2232	(2010, 981)	(2454, 1038)	25308	0.25	0.000009878299351984
5	(2874.0, 1010)	2874	(2586, 981)	(3162, 1038)	32832	0.25	0.000007614522417154

Figure 26: X Scaling Factor Fata Points

The scaling factors were plotted against the respective distances in pixels from the camera. These graphs illustrate how the scaling factors vary with distance. The y and x scaling functions were derived by fitting a suitable equation to fit the curves. The y scaling function is demonstrated in Figure 27, and the x scaling function is presented in Figure 28.

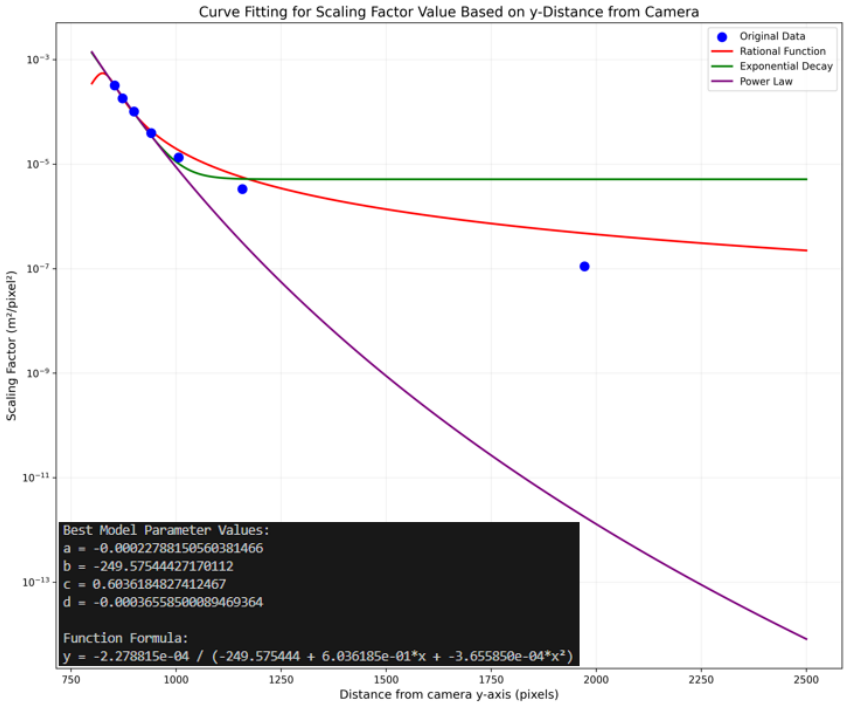

Figure 27: Y Scaling Function Graph and Equation

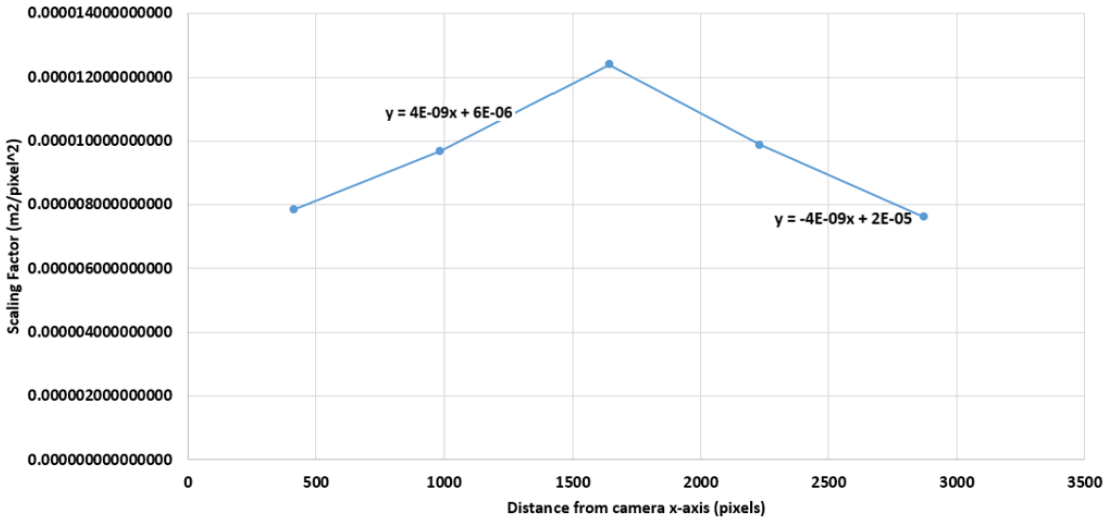

Figure 28: X Scaling Function Graph and Equation

6.3.5.4 Stage 5: Depth Estimation

The filtered potholes are passed into the depth estimation pipeline stage, where the DepthAnythingV2 model [15] is used to obtain a relative depth score for each pothole. The highest performing model (large) was selected for this stage, where accuracy is prioritized over speed. Since this model estimates depth for each pixel relative to the camera's position and given that the input images are at an angle, it is necessary to normalize the depth measurements to obtain a more accurate representation of pothole depth. Relying solely on raw maximum depth values would be problematic, as larger potholes with bigger bounding boxes inherently exhibit greater variation in depth measurements and would have larger depth values by default. The estimation is done by using the equation below:

$$Relative\ Depth\ Score = P_{95}(depth\ map) - P_{50}(depth\ map)$$

Using the 95th percentile ensures that the maximum depth value is not an outlier. Subtracting the 50th percentile from the 95th percentile prevents excessively high relative depth scores for larger potholes, as the variation between the maximum and minimum values would otherwise exaggerate their depth. This approach penalizes and balances the depth score for potholes with large areas. If we used the maximum minus the minimum depth value instead, larger potholes would inherently receive disproportionately high scores. However, to fully mitigate the effect of larger potholes receiving higher depth values, the normalized depth is calculated by dividing the square root of the estimated area score obtained from the area estimation pipeline (see equation below). Normalization ensures a fair comparison between potholes of different sizes and distances from the camera, improving the accuracy of the depth estimation calculation.

$$Normalized\ Depth\ Score = \frac{Relative\ Deph}{\sqrt{Estimated\ Area\ Score}}$$

6.3.5.5 Stage 6: Pothole Categorization & Scoring

The final pipeline stage aggregates the area and depth scores together to classify the potholes into different categories. The depth score is already normalized, and the area score is normalized between 0 and 1 by using the following equation:

$$Area\ Normalization\ Score = \frac{Area\ Score - Max\ Area\ Score}{Max\ Area\ Score - Min\ Area\ Score}$$

By adding the area and depth scores together, the following categories were assigned based on the total score.

- **Critical:** total score is between 1.6 and 2.0
- **High:** total score is between 1.0 and 1.5
- **Moderate:** total score is between 0.6 and 0.9
- **Low:** total score is between 0 and 0.5

Figure 29 provides an example output from the categorization stage where a category label is assigned to each pothole. Additionally, a summary is provided for the results after processing the image through the full pipeline (see text output on the right of Figure 29). Figure 30 provides a complete visual summary of the AI pipeline results for a given sample image.

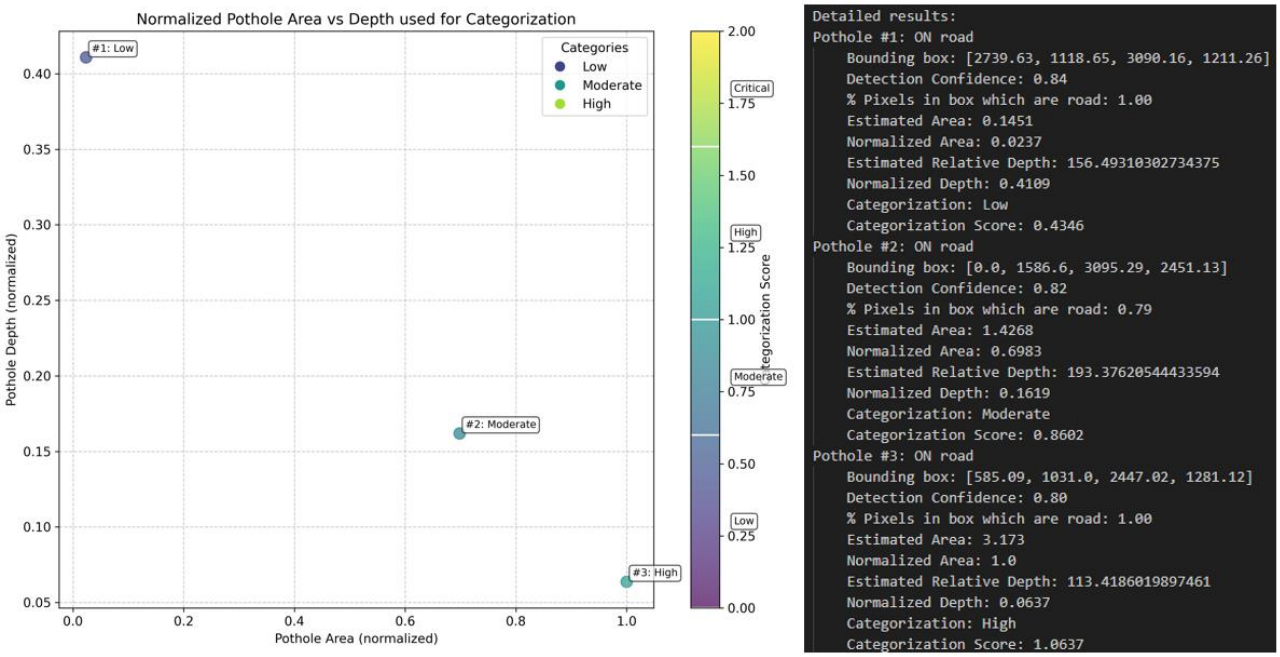

Figure 29: Example Pothole Categorization Results

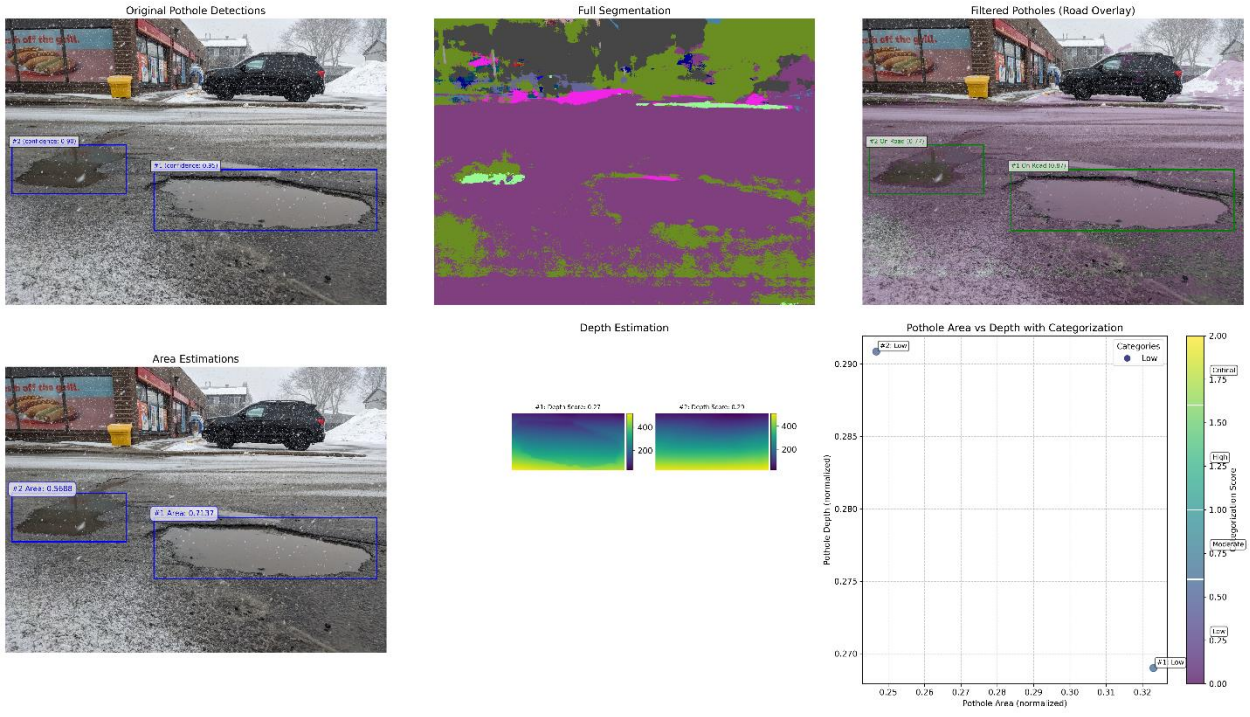

Figure 30: Full AI Pipeline Results for a Sample Image

6.3.5.6 AI Pipeline Deployment

The full AI pipeline was integrated into a modified program for optimized processing and concurrent multithreading (see Figure 31). The first stage of the pipeline (pothole detection) now connects directly to the live video stream from the rover and performs real-time inference. Since detections occur much faster than the subsequent analysis stages (segmentation and depth estimation), detected frames are added to a queue for later processing. The segmentation stage, which filters out false positives (see Section 6.3.5.2), and the area and depth estimation stages (see Sections 6.3.5.3 & 6.3.5.4) are executed in separate threads to improve overall throughput by using parallel processing. Whenever a stage has completed its processing, it writes directly its results to the database. This is done by utilizing the API endpoints to write new entries to the database (see Section 6.3.4). This allows real-time updates to be displayed directly on the UI as soon as a specific stage completes its processing.

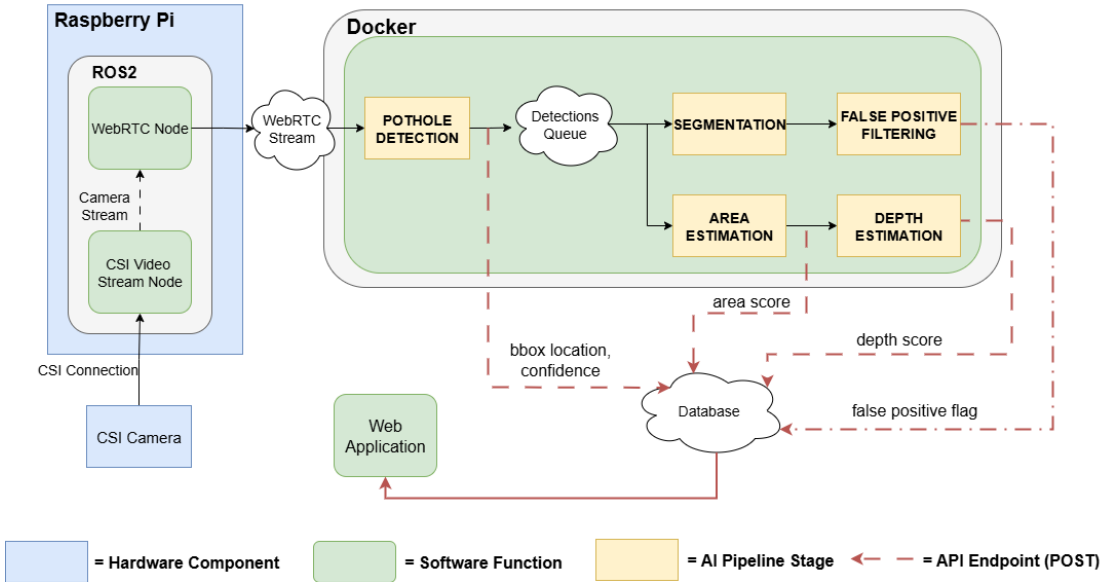

Figure 31: Deployed AI Pothole Pipeline Architecture

The figure below demonstrates the fine-tuned model performing live video inference using a connected camera. Once a pothole is detected, the image itself, the bounding box location, and the confidence score are sent to the database through an API request. The database table shows the newly added entries resulting from this detection. Although not shown here, the UI simultaneously updates to display the latest detected pothole frames.


```
API response, detection id: 11
Pothole detected! Saved frame as pothole_detection_6.jpg (Total detections: 6)
Added detection to queue. Queue size: 1
Sent image/frame to API, Status Code: 201
API response, image id: 10
Sent detection results to API, Status Code: 201
API response, detection id: 12
Pothole detected! Saved frame as pothole_detection_7.jpg (Total detections: 7)
Added detection to queue. Queue size: 2
Sent image/frame to API, Status Code: 201
```

id serial	path_id integer	image_url text	timestamp timestamp wi…	location geometry(point)
11	1	uploads\frame_10_8.jpg	2025-09-14 18:47:28.21…	0101000020E61000000000…
12	1	uploads\frame_11_9.jpg	2025-09-14 18:47:28.48…	0101000020E61000000000…

Figure 32: Example of Deployed AI Application

6.3.6 CI/CD Pipeline & Deployment with Docker

In the project, Docker was used to containerize the different services including the web application, the database, and the AI pothole pipeline. This enables a consistent and scalable deployment environment. By containerizing dependencies and configurations, Docker ensures that the applications run reliably across different systems. The following parts of the system were containerized:

- **Web Application:** Includes the UI, back-end logic, and server-side APIs.
- **Database:** PostgreSQL database used for storing data.
- **AI Pothole Application:** Connects to the live video stream from the rover to perform pothole detection and analysis. The results are written to the database in real time using the API endpoints. There is one AI pothole application container per CSI side camera

Docker Compose was used to orchestrate the various Docker services running in parallel (see Figure 33). As a result, regardless of the host machine, the entire server-side system can be started using simple Docker Compose commands.

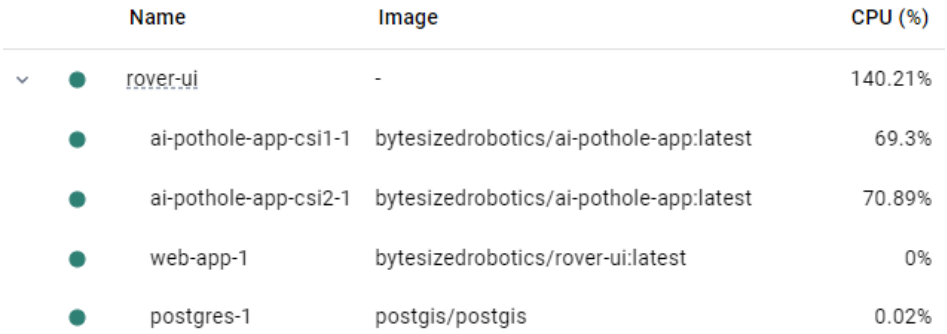

Figure 33: Docker Containers Running from Docker Compose Command

Additionally, an automated CI/CD pipeline was created with GitHub Actions to build and push a new Docker image to the online Docker Hub repository after every release of the front-end code (see left image in Figure 34). This ensures that the Docker images are up-to-date and refreshed after every release. The CI/CD pipeline also has unit tests which run on each commit to the repository, ensuring code quality (see right image in Figure 34).

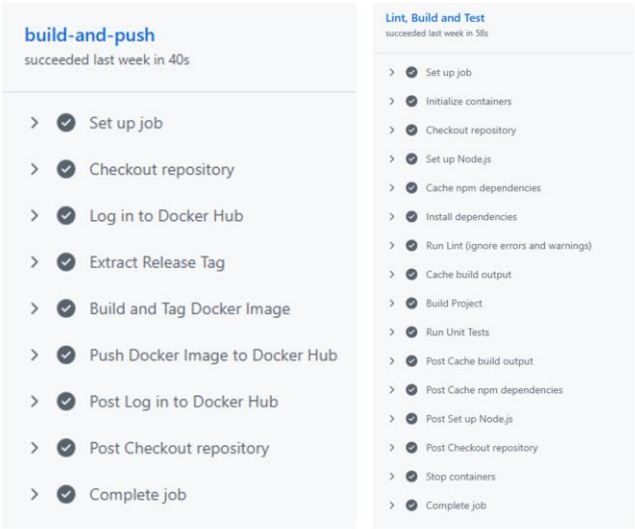

Figure 34: GitHub Actions: Build/Push Docker image upon new release (left) and Unit Tests running every commit (right)

7 Hardware Design

7.1 Hardware Components

The table below provides the list of components available and the justification for the selected hardware. The selected component is highlighted in green.

Table 8: Hardware Component Selection

Hardware Component Category	Options	Rationale
Processor	Raspberry Pi 4b [16] Raspberry Pi 5 [17] Jetson Nano [18]	- 8GB of RAM for better performance with many processes running concurrently (ROS2, AI inference, multiple camera feeds) - Raspberry Pi 5 instead of 4 for better processing performance - Sufficient IO for multiple cameras, sensors, and all accessories - Raspberry Pi 5 fits within the budget constraint, while keeping strong performance - Jetson has higher performance, specifically in AI applications, but it consumes more power, which would decrease the battery performance of the rover - Raspberry Pi 5 is sufficient and won't limit the capabilities of the rover
Microcontroller	ESP32 Driver Board [19] STM32F1 [20] Arduino Uno Rev3 [21]	- ESP32 and STM32 with higher processing capabilities than Arduino - ESP32 has built-in Wi-Fi and Bluetooth capabilities - While STM32 has the most peripheral support, ESP32 has sufficient support to control the rover - Arduino has the simplest ease of use and integration, followed by the ESP32
Rover Chassis	Waveshare Wave Rover [22] Freenove 4WD Car Kit [23] Elegoo Uno R3 [24] STM32 Smart Car [25]	- The Waveshare rover arrives fully built and assembled, whereas the other options come as kits that require assembly - Both Waveshare and Freenove include an ESP32, the preferred microcontroller - Elegoo uses a proprietary controller, which is less ideal due to limited documentation and integration support. - Waveshare rover provides an extra mounting plate for adding additional hardware components on the rover

Lidar Sensor	A2M8 Lidar [26] A1M8 Lidar [27]	- Both are 2D lidars - A2M8 has better range (18m vs 12m) - A2M8 has better angular resolution, which captures more detail from the surroundings - A1M8 is larger and would not fit on the rover - A2M8 costs significantly more ($400 vs $150) - Both sensors would've been provided by the lab technician
GPS Sensor	Beitian BN-220 [28] Beitian BN-880 [29] u-Blox NEO-6M [30]	- All 3 modules provide ~1m precision and are UART compatible - All 3 options provide multiple constellations (GPS, GLONASS, BeiDou, Galileio, etc.) and Latitude/Longitude coordinates in NMEA format - BN-220 provides the simplest pinout and lowest power consumption while having all requirements checked - Neo-6M has unnecessary pinouts and consumes more power
Cameras	IMX219-120 [31] Raspberry Pi Cam Module 3 Wide [32]	- Identical FOVs (120°) - Raspberry Pi Cam has a max video resolution of 2304×1296 at 56 FPS, or 1536×864 at a max framerate of 120 FPS - IMX219 has a max video resolution of 1920×1080 at 47 FPS, or 640×480 at a max frame rate of 206 FPS - IMX219 doesn't support autofocus, but this feature consumes more power, and is not needed for pothole detection - Raspberry Pi has a superior video mode with higher resolution and frame rate, however the IMX219 provides a sufficient resolution and frame rate for the purposes of this project - Not every frame will be processed by the vision model, so a high FPS is not required. - 1080p video is plenty for the vision model, so higher resolution is not needed. - IMX219 consumes about 0.5 watts at max usage - Raspberry Pi Cam consumes about 1.5 watts at max usage - IMX219 has a better price-to-performance ratio ($25 vs $55)

7.2 Hardware Specifications

The table below provides the specifications for the hardware components used in the project.

Table 9: Hardware Component Specifications

Hardware Component	Specifications
Raspberry Pi 5 [17]	- 8 GB RAM - SSD Storage 64 GB - Operating Voltage 5 V - 4 USB-A ports - 2 4Kp60 HDMI display outputs - DC Power provided via USB-C - 64-bit Arm Cortex-A76 CPU
ESP32 Microcontroller [19]	- Built-in Wi-Fi and Bluetooth - 240 MHz clock - SPI, I^2C, UART interfaces - Built-in ADC and DAC channels
Rover Chassis [22]	- Built-in ESP32 microcontroller - Dimensions: 194×168×100 mm - Motor Voltage: 12 V - Motor Power: 1.5W × 4 (per wheel) - Running speed: 1.25 m/s
RPLidar A2M8 2D LiDAR [26]	- Range: 18m - Scanning Rate: Default at 10Hz, Max. 15Hz - Angular Resolution: 0.45° - 1.35° - Max Sample Rate: 16,000 samples/sec - Operating Voltage: 5 V
Beitian BN-220 GPS Module [28]	- ~1m Precision - Constellations: GPS, GLONASS, BeiDou, QZSS, Galileio - Data format: NMEA Out - UART compatible
IMX219-120 [31]	- FOV of 120 degrees - 8 mega pixels - Resolution: 3280 x 2464 - Dimensions: 25mm × 24mm - Operating Voltage: 3.3 V

7.3 Power Consumption Breakdown

The total power consumption was calculated to approximate the battery life of the rover. The system is estimated to have a battery life of 70 minutes when operating at full load (max power usage), and a battery life of close to 8 hours when the system is idle (see Figure 37). The calculation was done by taking the total watt-hour (Wh) value of the power supply divided by the total number of watts required by the different hardware components.

Component	Running Voltage	Watts (Max load)	Watts (Idle)	Notes
Raspberry Pi 5	5	15	2.75	Will vary depending on load
RPLIDAR A2M8 LIDAR	5	3	1.1	Startup: 6-7.5
4x Motors	12	6	0	4x 1.5 Watts
ESP32-WROOM-32 Controller Board	3.3	0.132	0	Idle is in microWatts => negligible
AK09918C (Electronic Compass)	1.65 - 1.95	0.002145	0	mW range, negligible
QMI8658C (Motion sensor)	1.8 - 3.6	0.00142	0	mW range, negligible
TB6612FNG (Motor Control Chip)	3	1.36	0.78	Consumed wattage negligible, but heat dissipation high
Wifi module	5	1.25	0	Idle is in microWatts => negligible Also max load assumes constant comms
2x IMX219 8MP Cameras	2.8	1.65	0	Camera usage may vary a lot
TROPRO Tw8 Webcam	5	2.5	0	Power consumption unsure, max USB 2.0 wattage used (0.5 A)
BN-220 GPS Module	5	0.25	0	Active when capturing signal only
Total Wattage		31.145565	4.63	

Figure 35: Hardware Components Power Consumption

Power spread	Max (%)	Idle (%)
Raspberry Pi 5	48.16095004	59.39524838
Motors	19.26438002	0
Lidar	9.632190008	23.75809935
Control Board	8.81526792	16.84665227
Cameras	13.32452951	0
GPS	0.8026825007	0

Figure 36: Max vs Idle Power Consumption Spread

Source	Voltage	mAh	Wh
Battery Pack 3x	3.7	9900	36.63
Battery life Hours (Ideal case)	Battery life Hours (Worst Case)		
7.911447084	1.1760904		

Figure 37: Rover Battery Life Estimation

7.4 Hardware Block Diagram

Similarly to the software architecture diagram (see Figure 1), the hardware architecture was created using the ArchiMate modeling language. The architecture below illustrates the hardware components and their interconnections.

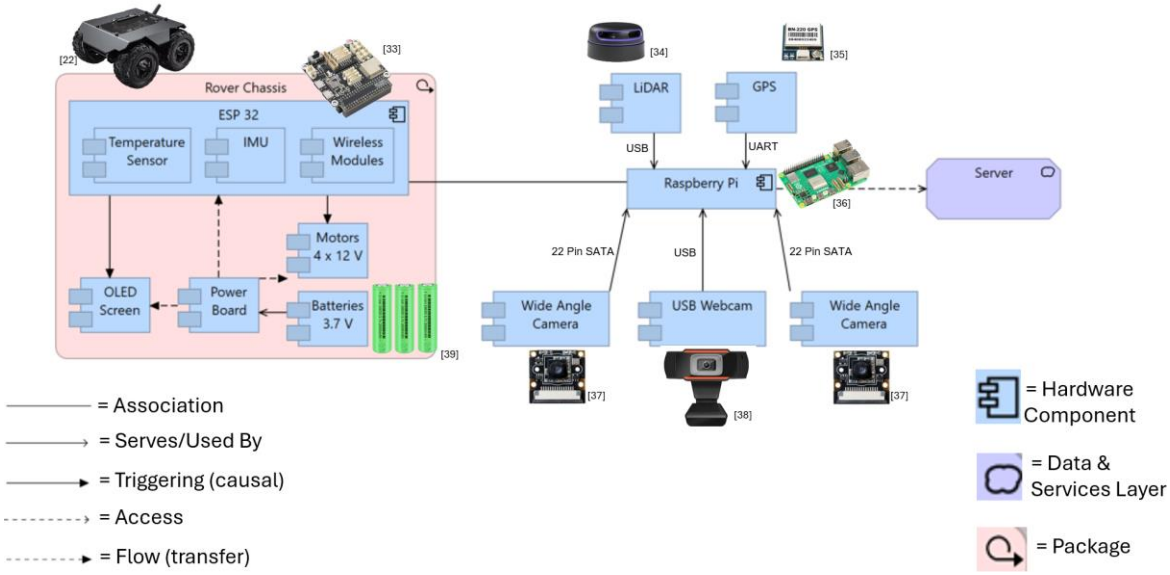

Figure 38: Hardware Architecture

The Raspberry Pi 5 is the main control for all onboard devices. It is physically attached to the ESP32 slave chip. The ESP32 has full control over the motor driver and the power distribution for all the components connected to the power board.

Rechargeable batteries are housed in a holder connected to the power board and provide the necessary power without being plugged in to a wall power source. As for sensors, the cameras are connected directly to the Raspberry Pi to not incur any noticeable latency in the live video streams.

The GPS and LiDAR sensors are also connected to the Raspberry Pi in order to run natively on the ROS2 library onboard the processor. The IMU, which features a gyroscope, electronic compass and pitch/yaw sensor is imbedded in the ESP32 for relative navigation. Finally, the Raspberry Pi transmits relevant data to the web application through Wi-Fi communication (ROSBridge and WebRTC.

7.5 Hardware Development

7.5.1 Design & Planning

The figure below provides the design for the completed rover assembly, simulated in Unreal Engine 5 (UE5). The main components visualized are the rover body, the LiDAR, the camera mount with the wide cameras, the front facing USB camera, and the Raspberry Pi placed under the mounting plate. This ensures that all components will fit together before attempting to assemble and integrate the hardware of the system. It was crucial when doing the simulated fit test that all components have the correct scale applied to them.

Figure 39: UE5 Simulated Complete Rover Assembly

Figure 40 illustrates the assembly schematics of the chassis and other layered parts. The Raspberry Pi is inserted upside down onto the control board just like in the 3D model above. The LiDAR fits on the mounting plate with the camera tower behind it, and the front camera attaches to the front lip of the camera tower.

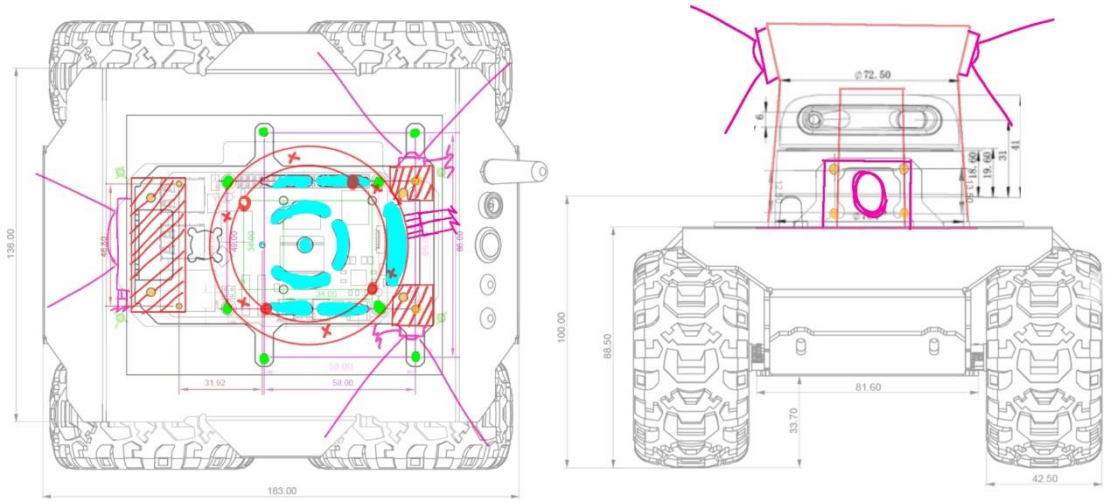

Figure 40: Chassis Assembly Schematics

Below is the 3D prototype of the camera mount tower with accurate dimensions. The tower features an arch structure with two slanted surfaces, each tilted at a 15-degree angle, designed for the side cameras. The arch pillars also feature a rounded front to accommodate the space taken by the LiDAR. The 3D model was designed in Blender and printed using a 3D printer (see Figure 45 for the physical printed design).

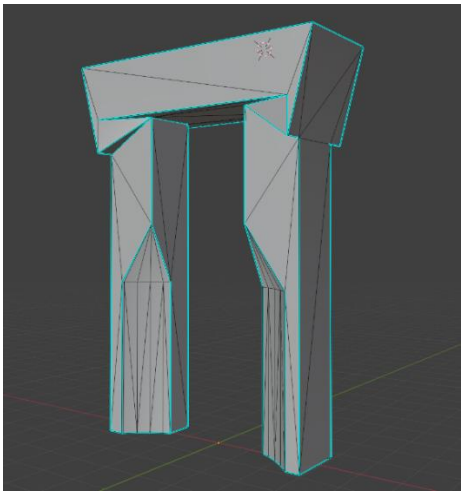

Figure 41: 3D Blender Model for the Camera Mount

7.5.2 Hardware Integration

The various hardware components were successfully integrated onto the rover (see Figure 42). The system operates under full load, running the cameras, GPS, and LiDAR simultaneously on the Raspberry Pi. At the same time, the rover can be remotely controlled via the front-end interface, receiving commands wirelessly and relaying them to the ESP32 to execute the corresponding actions.

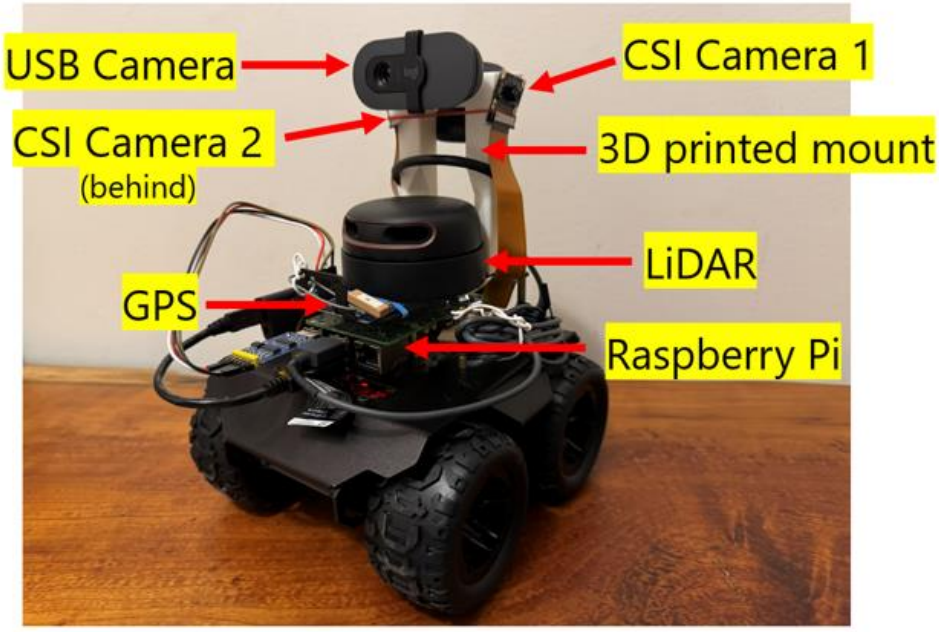

Figure 42: Final Hardware Integration and Prototype

8 Testing

8.1 Software

The table below presents the verification tests required to validate the functionality of the software components in the project. A description of each software component is also provided along with the process done to verify the completed tasks. Furthermore, unit tests have been created for testing the API endpoints using a simulated database. These tests are run automatically whenever there is a code commit pushed to the main branch.

Table 10: Software Tests and Results

Test Item	Test Type	Test Steps	Expected Result	Result
User Authentication	Security	Attempt to create an account and log in with valid and invalid credentials.	The system allows users to create an account and only accepts valid credentials stored in the database.	The system performs as expected.
Web GUI	Usability	Navigate the UI to test login, rover connection, path planning, live data, and map views.	Users can log in with correct credentials and are blocked with incorrect ones. User can connect to the corresponding rovers. User can plan a path and view live telemetry from the connected rover.	The system performs as expected and all pages have been implemented. The UI was also fully enhanced for a more attractive design.
Database Configuration	Functional	Create an account and verify credentials are stored, upload pothole images, and check data storage.	Data is correctly stored and retrievable.	The system performs as expected for the user credentials and database stores pothole related data, as well as logs for each given path.
AI Pothole Detection Model Accuracy Test	Performance	Run the detection model on test dataset and evaluate results.	The model detects potholes with high confidence and minimizes false positive detections.	Precision: 0.867 Recall: 0.758
AI Pothole Detection Model Performance Test	Performance	Run the AI model on a host machine and test performance when connecting to live camera stream	The model performs inference in real-time without significant lag.	No significant delay is noticed and potholes are detected correctly by the model running on the software side.

ML Pothole Processing Pipeline	Performance	Process detected pothole images to assign severity categories based on depth and relative area.	Potholes are categorized accurately into severity levels.	Potholes are assigned severity labels based on area and depth score as expected.
Path Planning	Functional	Define a path using the UI and verify that it is sent to the rover.	The planned path is displayed and sent to the rover.	Path planning allows the user to define a path for the rover to follow, and upon launch the GPS waypoints are sent to the rover.
Autonomous Navigation	Functional	Instruct the rover to follow a predefined path using a custom built autonomous navigation algorithm.	The rover successfully follows the defined path without manual intervention.	The rover follows the GPS waypoints in sequence. Because these waypoints represent positions along the road, the rover may drift slightly while still being considered within the acceptable range of the current waypoint.
Obstacle Detection	Functional	Place obstacles along the rover's path and check if the rover detects the obstacles and avoids them.	The rover correctly identifies and avoids obstacles.	Obstacle detection and avoidance are handled as follows: when the rover detects an obstacle within a 0.2 m threshold, it comes to a stop. It then rotates in place to search for an open corridor, allowing it to navigate around the obstacle.
Network Communication Between Software and Rover	Performance	Send control commands, retrieve pothole images, and monitor GPS data transmission when the rover is running.	Real-time data transmission occurs without significant delays between the software and the rover	Works as expected, but 1-2 second delay in the camera livestream feeds received by the front-end.
Manual Override	Functional	Attempt to manually control the rover using the front-end interface and the live-camera feeds.	The rover responds accurately to user commands and the live-camera feeds are displayed on the front-end.	Works as expected, able to teleoperate the rover and view the live camera streams.

8.2 Hardware Testing

The table below provides the tests which are done to verify the hardware modules.

Table 11: Hardware Tests and Results

Test Item	Test Type	Test Steps	Expected Result	Result
LiDAR Sensor Range	Functional	Place objects in front of the sensor at different distances.	Object detected between 0.15m and 8m (based on the datasheet)	Works as expected.
Camera Detection Range	Functional	Go outside with the hardware and test the maximum pothole detection range.	Potholes detected up to the middle of the street.	Pothole detection is functional, but potholes generally need to be relatively close to the camera for reliable identification due to the slow frame rate and lower resolution of the camera streams.
GPS Sensor Accuracy	Functional	Get the GPS location from the sensor and compare it with the actual location.	~1m difference between the sensed location and the actual location.	Works as expected, but the GPS module struggles to affix satellites in concrete buildings. Works well outside or close to a window.
Camera Vibration and Live Pothole Detection	Functional	Test the live camera detection and inference while the rover is in movement.	Detection is still possible and minimally impacted at low speeds.	When running the rover at a slower speed, the pothole detection with the camera works reasonably well.
Power Supply Maximum Load Test	Durability	Run the rover with all the components and test the rover under maximum component load.	Rover runs for a minimum of 30 minutes.	Tested the rover outdoors and lasted for more than 30 minutes.
Structural Durability test	Durability	Test the integrity of physical components and fasteners against vibrating and bouncing forces.	Fasteners hold and components stay seated in the intended configuration.	Integrated hardware modules onto the rover chassis, and the rover maintained its structural integrity and stability. Tested the rover under full load outside and rover remained secure and operational.

8.2.1 Camera Tests

The following figures demonstrate how UE5 was used to test and select the wide cameras for the real-time pothole detection on the rover. The first step was to set an initial camera height that ensured a clear field of view of the road. Because the rover is small, a custom camera mount needed to be designed. Based on the simulation, the camera height needed to be around 10cm above the top of the rover in order to clearly see the road (see Figure 43).

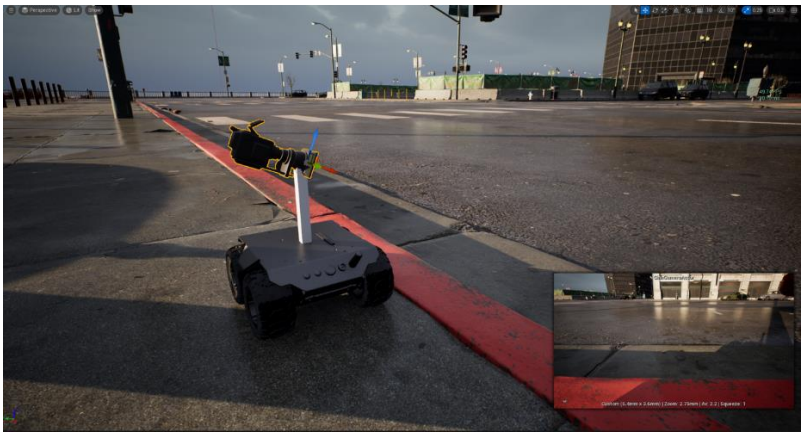

Figure 43: UE5 Camera Height Testing

Once the height of the camera was decided, the camera models were compared. There were two main camera models that were viable for the rover: the IMX-219 camera and the Raspberry Pi Camera Module 3 Wide. By using the camera datasheets, the exact specifications were applied to the virtual cameras in UE5, and the cameras were placed at the expected rover height and camera angle to face the surface of the road. Placing 1m x 1m white cubes along the road surface enables a better comparison of the cameras by providing visual references for scale and distance (see Figure 44). This simulation was very beneficial to verify the different camera options and help decide which camera to procure (see Table 8 for hardware component selection).

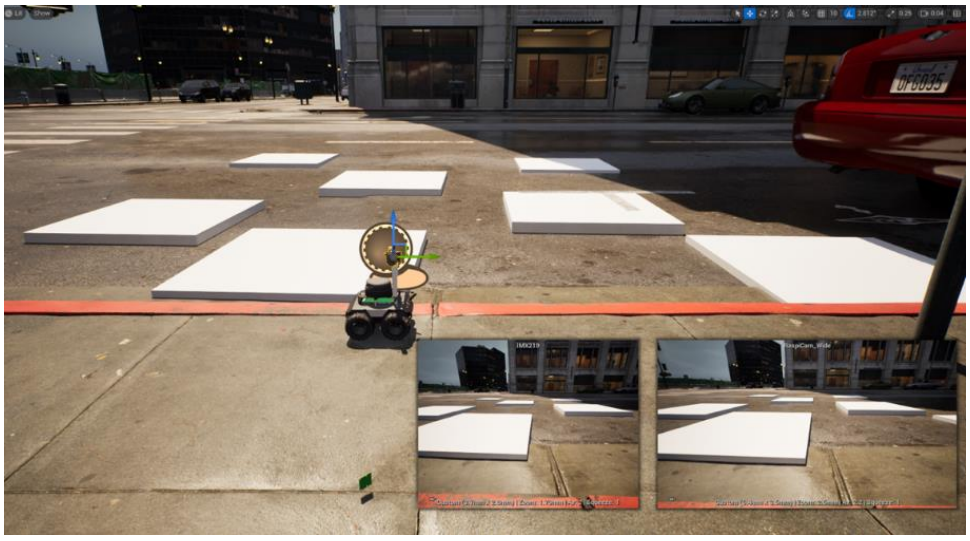

Figure 44: UE5 Camera Specifications Comparison

8.2.2 Hardware Mount and Fit Test

As discussed previously the 3D camera mount was designed and printed. The prototype was verified by doing a fit test of the LiDAR and the camera mount together on the rover mounting plate (see Figure 45). All the hardware components were then mounted and integrated onto the rover.

Figure 45: Physical Camera Mount and Lidar Fit Test

8.2.3 Outdoor Tests

The images below show the rover under full load being tested outdoors. The rover was able to fully operate and navigate in cold weather conditions.

Figure 46: Rover Images During Outdoor Testing

9 Project Management

9.1 Protocols and Procedures

The development process for this project began with the Work Breakdown Structure (WBS), where project requirements were divided into high-level goals (see Figure 51 & Figure 52 for the WBS). These goals are documented in an Agile management board using Trello (see Figure 53 for the Trello board), enabling their assignment to different sprints with an estimated completion date.

To ensure a more structured workflow, high-level tasks are further broken down into detailed subtasks within a GitHub Projects management board (see Figure 54). This approach provides a clearer division of work, making tasks more manageable and organized. Additionally, the GitHub Projects board facilitates seamless task tracking by linking specific work tickets directly to relevant repositories. This procedure allows for better monitoring of progress, ensuring that each task is associated with the corresponding commits and code changes while maintaining an Agile workflow.

When necessary, modifications to the Trello board are made to adjust the high-level strategy and sprint planning for the project. However, in most cases, task updates and refinements are carried out on the GitHub Projects board, as it provides a more detailed and structured approach to tracking progress and linking tasks directly to code changes.

9.2 Tasks and Timelines

The following table lists the milestones for CEG4912 and CEG4913 as well as the deliverables required for each milestone.

Table 12: Milestones and Deliverables for CEG4912

Date	Milestone	Deliverable
31/01/2025	Requirements Gathering	- Functional and non-functional requirements - Project constraints
03/02/2025	Architecture Design	- Software component block diagram - Software use-case diagram - Hardware component block diagram
07/02/2025	3D Simulation Design	- UE5 design for rover and fit test
14/02/2025	Basic Web Application	- User login page - Rover connection page - Database configured
18/02/2025	AI Model Pothole Detection Model	- Model trained using custom dataset - Live inference works running on the Raspberry Pi
21/02/2025	Procurement and Hardware Setup	- Survey and research into hardware components - Hardware selected and purchased - Hardware setup for components (Raspberry Pi, Rover, power supply, LiDAR, GPS module, cameras)
07/03/2025	Path Planning UI and Algorithm	- Leaflet integration with UI - User can plan a path for the rover and converts the path to a set of commands to control the rover
20/03/2025	ML Processing Pipeline	- Pipeline to process detected pothole images and assign severity scores and categories
28/03/2025	Basic System Integration	- Integration of working modules and components onto the rover
26/09/2025	Enhance UI Design & Integrate Database Access	- More complex, consistent and attractive UI design - Enhance existing pages to match theme

01/10/2025	Final Pothole AI App	- Complete AI pipeline integrated in a main program able to connect to the live camera streams on the rover
09/10/2025	Communication between Rover and Server	- The IMU readings, LiDAR point cloud, GPS location and camera streams sent back to the web application. - Updates on the UI are in real-time - The detections and analysis scores from the pothole AI app are written to the database
07/11/2025	Finalized Web App	- Live Metrics, Path History and Pothole Explorer pages are complete and display the information stored in the database
14/11/2025	Autonomous Navigation	- NAV2 working with LiDAR and GPS sensors to perform autonomous navigation and obstacle detection
21/11/2025	Final Prototype	- Full integration and modules working together

The high-level Gantt charts for CEG4912 and CEG4913 are presented below in Figure 47 and Figure 48 respectively. The Gantt chart used for task specific tracking is presented in Figure 49 and Figure 50.

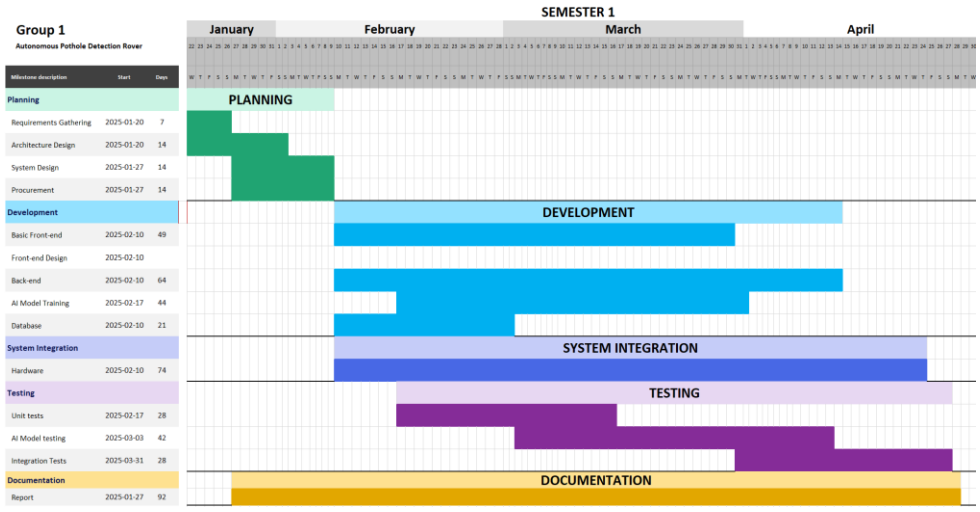

Figure 47: High-level Gantt Chart for CEG4912

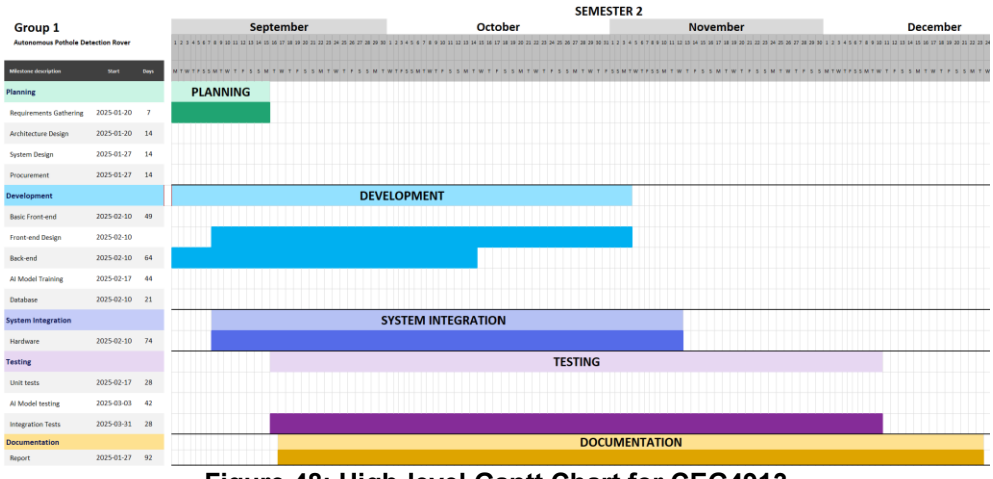

Figure 48: High-level Gantt Chart for CEG4913

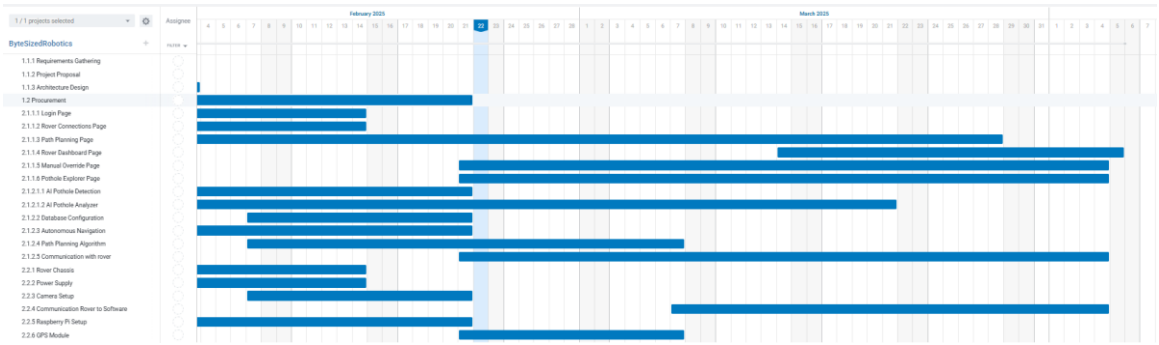

Figure 49: Detailed task tracking Gantt Chart (1)

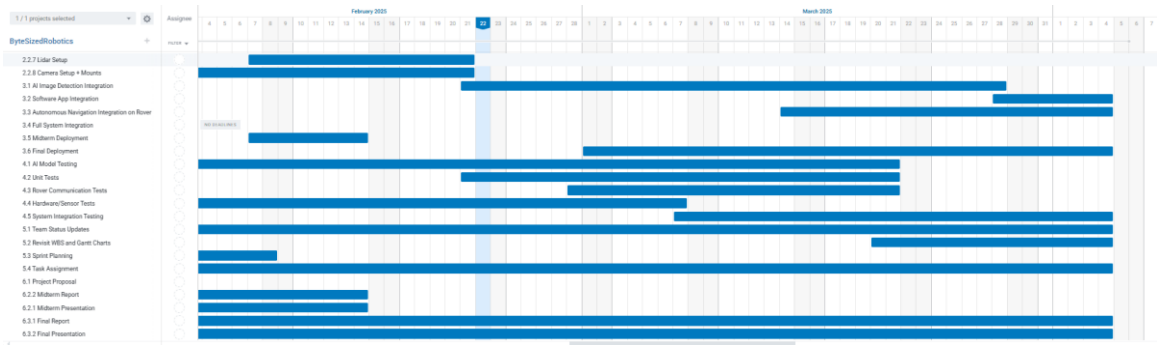

Figure 50: Detailed task tracking Gantt Chart (2)

9.3 Work Breakdown Structure

The WBS distributes the required workload for the project into six different categories. These categories are aligned with the project stages discussed previously (see section 4.2 on Project Stages). Additionally, the current status for the different tasks is included in the WBS.

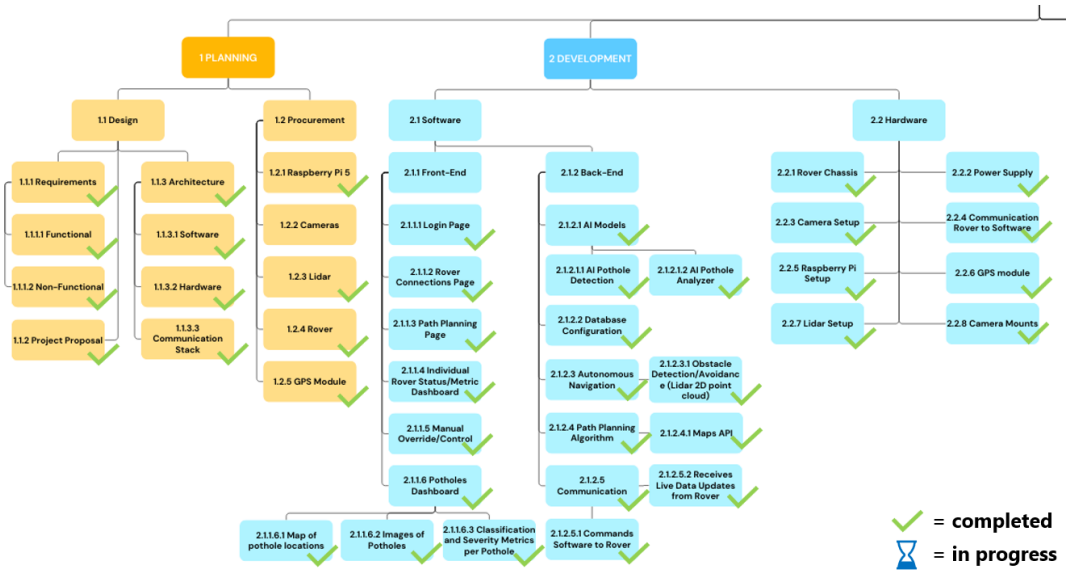

Figure 51: WBS Planning & Development Stages

Figure 52: WBS Integration, Testing, Management & Documentation Stages

9.4 Management Boards

Trello was used to track and manage high-level tasks outlined in the WBS. These tasks were assigned to different sprints, facilitating task distribution and coordination within the team. The high-level tasks are then divided into smaller sub-tasks on the GitHub Projects board (see Figure 54).

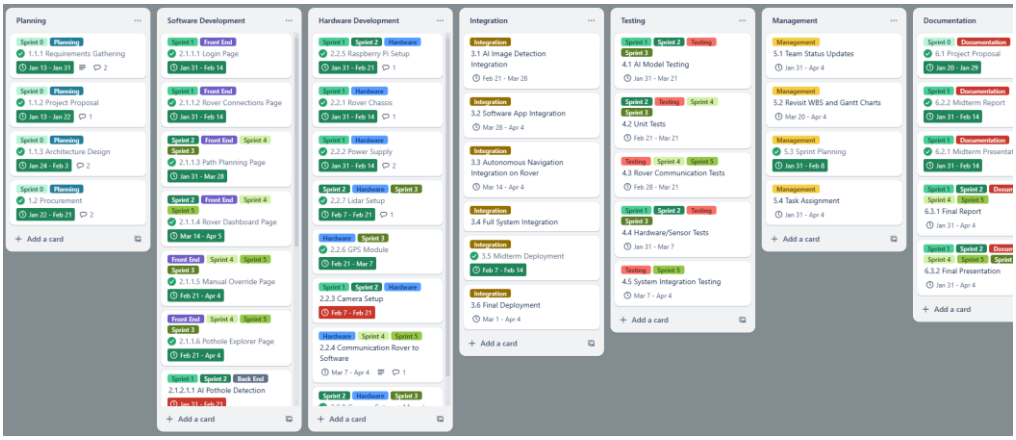

Figure 53: Trello Management Board

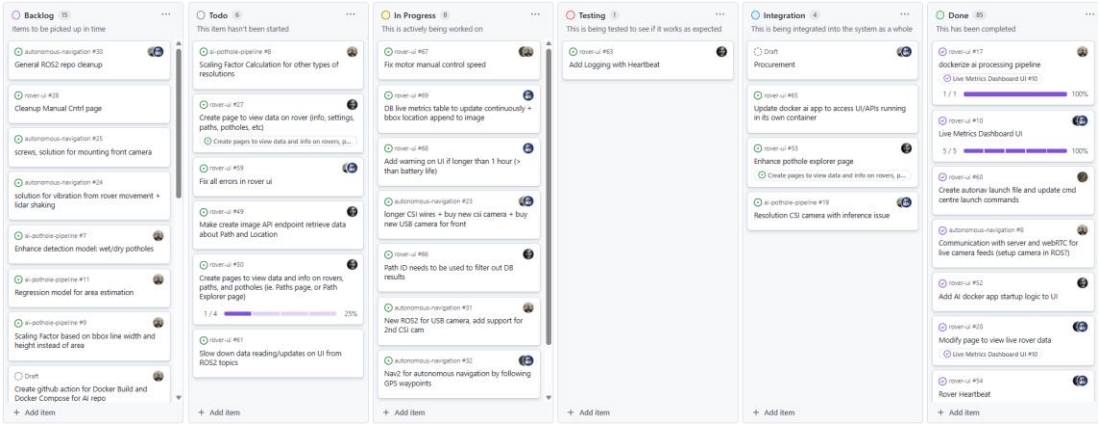

Figure 54: GitHub Project Management Board

10 Future Works & Closing Remarks

This project leverages AI and machine learning to enhance road maintenance by detecting and assessing potholes in real time. The final system is designed to navigate autonomously while performing real-time pothole detection. This solution aims to optimize road repairs and improve overall infrastructure safety. As development progresses, new features and modules could be integrated to enhance the rover's capabilities. If time permits, the following future enhancements could be implemented:

- **Solar Panels:** Improve energy efficiency and extend operational time by integrating solar charging for the battery.
- **3D Model Map:** Utilize 3D mapping to enable more precise navigation and a better view for the user when defining a path for the rover to follow.
- **Security Features:** Implement an alarm system that triggers when potential threats are detected. Notifications to alert the off-site user.
- **Automated Maintenance Alerts:** Detect wear and tear on components and notify users for maintenance.
- **Multi-Rover Coordination:** Enable communication between multiple rovers for collaborative pothole detection and cover larger areas of the city.
- **Rover History & Route Optimization:** Store and analyze past routes to improve navigation and prioritize high-risk areas for new pothole detections.

11 Project Code

The project's source code is hosted in a GitHub organization created by the team, structured into three different repositories to support a more modular design of the project and maintain clear separation of functionalities.

- **AI-Pothole-Detection-Analysis-Pipeline**: Contains the trained AI detection model and full AI processing pipeline.
- **rover-ui**: Includes the web application, database, and server-side APIs.
- **autonomous-navigation**: Houses all programs for robotic control with ROS, handling sensor integration and rover control. This code is deployed on the Raspberry Pi.

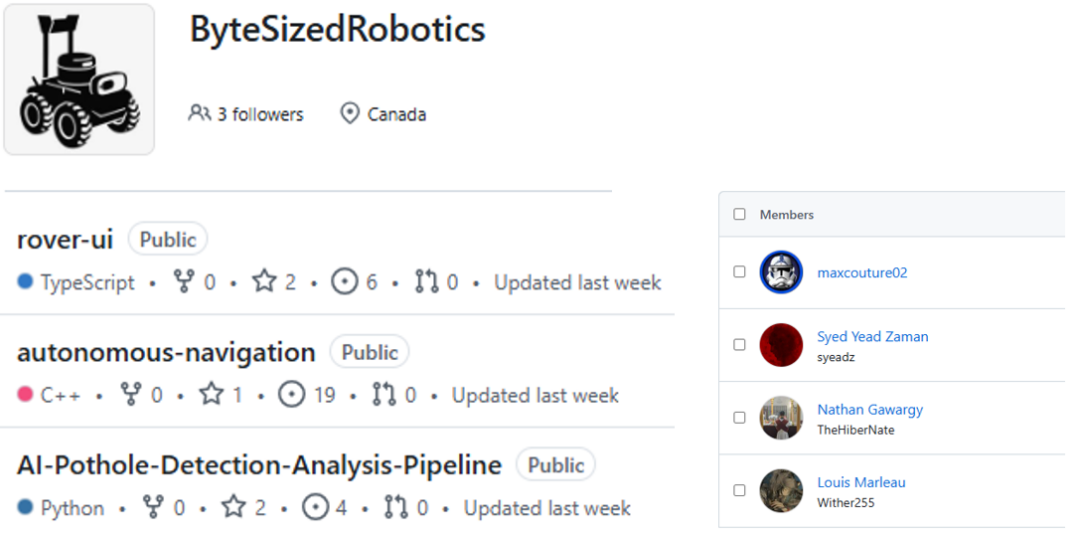

Figure 55: GitHub Organization and Repositories