

Systems and Computer Engineering

Final Report

Autonomous Vehicle

SYSC 4907 A

ENGINEERING PROJECT

Professor Richard Dansereau

Professor Chao Shen



April 8th, 2025

CARLETON UNIVERSITY

OTTAWA, ON

1. Abstract

This report presents the comprehensive development and integration of a modular Autonomous Vehicle (AV) system designed by students in the Systems and Computer Engineering department at Carleton University. The project's objective is to create a functional electric autonomous vehicle capable of operating without human intervention through seamless collaboration between power management, navigation, autonomous control, sensor perception, simulation, and safety subsystems. Using a ROS 2 based architecture, the vehicle incorporates real-time embedded systems, custom hardware like PCBs, sensor technologies and robust power distribution to ensure system wide communication, control, and safety. Each subsystem was built and tested independently, then integrated into a fully operational system. The report highlights the engineering challenges, design methodologies, software and hardware implementations, testing procedures, and future improvements necessary to enhance system performance and autonomy.

2. Team Structure

The project has been split into six subgroups. A list of the subgroups and the team leads and members is below

Team Leader: Abed Qubbaj 101205030

Power Management

Leader: Jacob Wilde #101188310

Members:

- Taran Basati #101161332

Autonomous control

Leader: Charlie Wadds # 101181414

Members:

- Mohammad Saud #101195172
- Abed Qubbaj #101205030
- Hussein Ghamlouch # 101089558

Navigation

Leader: Shreeyansh Gupta #101154480

Members:

- Harsh Patel #101183596
- Karthikeyan Bhavani Shankar #101214895
- Tauheed Alamgir #101194927
- Ibrahim Faisal #101209598

Sensors

Leader: Ryan Dash #101083052

Members:

- Daniel Godfrey #101156147
- Joshua Robson # 101195802
- Alexei Fetissov #100724437

Reliability and Safety

Leader: Ethan Bradley # 101158848

Members:

- Rahul Cheruku #101185639
- Ali Nadim #101192767
- Ali Zaid #101223823
- Arjun Pathak #101212581

Simulation

Leader: Abdurahman Jama #101162633

3.Table of Contents

1. Abstract	2
2. Team Structure	3
3. Table of Contents	4
4. List of Figures	14
5. Introduction	19
5.1. Background and Motivation	19
5.2. Problem Statement	19
5.3. Objectives and Scopes	19
6. Power Management	20
6.1. Introduction	20
6.1.1. Background & Motivation	20
6.1.2. Problem Statement	21
6.1.3. Objective & Scope	21
6.1.4. Methodology Overview	21
6.2. Engineering Requirements & Justification	22
6.2.1. Justification for Relevance to Degree Program	22
6.2.2. Engineering Principles Applied	22
6.2.3. Health and Safety Consideration	22
6.2.4. Ethical Considerations	22
6.2.5. Regulatory & Standards Compliance	23
6.3. Literature Review & Related work	23
6.3.1. Overview of Existing Technologies & Techniques	23
6.3.2. Comparison with Previous Work	23
6.4. System Design & Implementation	24
6.4.1. System Overview & Architecture	24
6.4.2. Requirements Definition	25
6.4.3. Hardware Design	28
6.4.4. Software Design	42
6.4.5. Integration & Testing Approach	43
6.4.6. Challenges & Troubleshooting	44
6.5. Project Management & Execution	45
6.5.1. Work Breakdown Structure	45
6.5.2. Gantt Chart & Milestone Tracking	45
6.5.3. Risk Analysis & Mitigation Strategies	46
6.5.4. Budget & Cost Analysis	46
6.6. Experimental Setup & Testing	46
6.6.1. Testing Environment & Procedures	46

6.6.2. Experimental Results and Findings	47
6.7. Discussion & Analysis	48
6.7.1. Interpretation of Test Results	48
6.7.2. Comparison with Expected outcomes	49
6.8. Conclusion & Future work	49
6.8.1. Summary of Key Achievements	49
6.8.2. Lessons Learned	50
6.8.3. Recommendations for Future Development	50
6.9. Reflections	51
6.9.1. Original Project Proposal & Changes Over Time	51
6.9.2. Team Reflection on Challenges & Successes	51
6.9.3. Hindsight Analysis: What would we do Differently?	52
7. Navigation	52
7.1. Introduction	52
7.1.1. Background & Motivation	52
a. Background:	53
b. Current State of the Art:	55
c. Motivation:	57
7.1.2. Problem Statement	57
7.1.3. Objective & Scope	58
7.1.4. Methodology Overview	59
7.1.5. Report Overview	61
7.2. Engineering Requirements & Justification	62
7.2.1. Justification for Relevance to Degree Program	62
7.2.2. Engineering Principles Applied	63
7.2.3. Health and Safety Consideration	64
7.2.4. Ethical Considerations	64
7.2.5. Regulatory & Standards Compliance	65
7.3. Literature Review & Related work	66
7.3.1. Overview of Existing Technologies & Techniques	66
7.3.2. Comparison with Previous Work	69
7.4. System Design & Implementation	70
7.4.1. System Overview & Architecture	70
7.4.2. Requirements Definition	80
7.4.3. Hardware Design	85
Touchscreen Display:	87
Raspberry Pi 4B	87
Custom 3D-Printed Mount	88
7.4.4. Software Design	89

7.4.5. Integration & Testing Approach	91
7.4.6. Challenges & Troubleshooting	95
7.5. Project Management & Execution	98
7.5.1. Work Breakdown Structure	98
7.5.2. Gantt Chart & Milestone Tracking	99
7.5.3. Risk Analysis & Mitigation Strategies	102
7.5.4. Budget & Cost Analysis	103
7.6. Experimental Setup & Testing	104
7.6.1. Testing Environment & Procedures	104
7.6.2. Experimental Results and Findings	106
7.7. Discussion & Analysis	107
7.8. Conclusion & Future work	107
7.8.1. Summary of Key Achievements	107
7.8.2. Lessons Learned	108
7.8.3. Recommendations for Future Development	109
GNSS and Localization	109
Camera System	109
Raspberry Pi Server and UI	109
Navigation Stack (Nav2)	110
7.9. Reflections	111
7.9.1. Original Project Proposal & Changes Over Time	111
7.9.2. Team Reflection on Challenges & Successes	112
7.9.3. Hindsight Analysis: What would we do Differently?	114
8. Autonomous Control	115
8.1. Introduction	115
8.1.1. Background & Motivation	115
8.1.2. Problem Statement	115
8.1.3. Objective & Scope	116
8.1.4. Methodology Overview	116
8.2. Engineering Requirements & Justification	117
8.2.1. Justification for Relevance to Degree Program	117
8.2.2. Engineering Principles Applied	117
8.2.3. Health and Safety Consideration	118
8.2.4. Ethical Considerations	119
8.2.5. Regulatory & Standards Compliance	119
8.3. Literature Review & Related work	119
8.3.1. Overview of Existing Technologies & Techniques	119
8.4. System Design & Implementation	120
8.4.1. System Overview & Architecture	120

8.4.2. Requirements Definition	121
8.4.3. Hardware Design	123
8.4.4. Software Design and Tools	128
8.4.5. Challenges & Troubleshooting	132
8.5. Project Management & Execution	133
8.5.1. Work Breakdown Structure	133
8.5.2. Gantt Chart	134
8.6. Conclusion & Future work	134
8.6.1. Summary of Key Achievements	134
8.6.2. Lessons Learned	134
8.6.3. Recommendations for Future Development	135
8.7. Reflections	135
8.7.1. Original Project Proposal & Changes Over Time	135
8.7.2. Team Reflection on Challenges & Successes	136
8.7.3. Hindsight Analysis: What would we do Differently?	136
9. Reliability and Safety	138
9.1. Introduction	138
9.1.1. Background & Motivation	138
9.1.2. Problem Statement	138
9.1.3. Objective & Scope	139
9.1.4. Methodology Overview	139
9.2. Engineering Requirements & Justification	140
9.2.1. Justification for Relevance to Degree Program	140
9.2.2. Engineering Principles Applied	141
9.2.3. Health and Safety Consideration	142
9.2.4. Ethical Considerations	142
9.2.5. Regulatory & Standards Compliance	143
9.3. Literature Review & Related work	143
9.3.1. Overview of Existing Technologies & Techniques	144
a) Manual Override Architectures in Autonomous Systems	144
b) E-Stop Architectures in Autonomous Systems	144
9.3.2. Comparison with Previous Work	145
a) Manual Override Architectures in Autonomous Systems	145
b) E-Stop Architectures in Autonomous Systems	146
c) Remote Monitoring Architectures in Autonomous Systems	146
9.4. System Design & Implementation	148
9.4.1. System Overview & Architecture	148
a) Manual Override Architectures in Autonomous Systems	148

b) E-Stop Architectures in Autonomous Systems	148
c) Remote Monitoring Architectures in Autonomous Systems	149
9.4.2. Requirements Definition	149
9.4.3. Hardware Design	150
9.4.4. Software Design	166
9.4.5. Integration & Testing Approach	168
9.4.6. Challenges & Troubleshooting	169
9.5. Project Management & Execution	171
9.5.1. Work Breakdown Structure	171
9.5.2. Gantt Chart & Milestone Tracking	172
Table X: Gantt chart	172
9.5.3. Risk Analysis & Mitigation Strategies	173
9.5.4. Budget & Cost Analysis	174
9.6. Experimental Setup & Testing	174
9.6.1. Testing Environment & Procedures	175
a) Manual Override Architectures in Autonomous Systems	175
b) E-Stop Architectures in Autonomous Systems	175
c) Remote Monitoring Architectures in Autonomous Systems	176
9.6.2. Experimental Results and Findings	177
a) Manual Override Architectures in Autonomous Systems	177
b) E-Stop Architectures in Autonomous Systems	178
c) Remote Monitoring Architectures in Autonomous Systems	180
9.7. Discussion & Analysis	181
9.8. Conclusion & Future work	182
9.8.1. Summary of Key Achievements	182
a) Manual Override Architectures in Autonomous Systems	182
b) E-Stop Architectures in Autonomous Systems	183
c) Remote Monitoring Architectures in Autonomous Systems	183
9.8.2. Lessons Learned	184
a) Manual Override Architectures in Autonomous Systems	184
b) E-Stop Architectures in Autonomous Systems	184
c) Remote Monitoring Architectures in Autonomous Systems	185
9.8.3. Recommendations for Future Development	185
a) Manual Override Architectures in Autonomous Systems	185
b) E-Stop Architectures in Autonomous Systems	186
c) Remote Monitoring Architectures in Autonomous Systems	187
9.9. Reflections	187
9.9.1. Original Project Proposal & Changes Over Time	187

a) Manual Override Architectures in Autonomous Systems	187
b) E-Stop Architectures in Autonomous Systems	188
c) Remote Monitoring Architectures in Autonomous Systems	188
9.9.2. Team Reflection on Challenges & Successes	189
9.9.3. Hindsight Analysis: What would we do Differently?	190
10. Sensors/Perception	191
10.1. Introduction	191
10.1.1. Background & Motivation	191
10.1.2. Problem Statement	192
10.1.3. Objective & Scope	192
10.1.4. Methodology Overview	193
10.1.5. Report Overview	193
10.2. Engineering Requirements & Justification	194
10.2.1. Justification for Relevance to Degree Program	194
10.2.2. Engineering Principles Applied	194
10.2.3. Health and Safety Consideration	195
10.2.4. Ethical Considerations	195
10.2.5. Regulatory & Standards Compliance	196
10.3. Literature Review & Related work	196
10.3.1. Overview of Existing Technologies & Techniques	196
10.3.2. Comparison with Previous Work	197
10.4. System Design & Implementation	199
10.4.1. System Overview & Architecture	199
10.4.2. Requirements Definition	200
10.4.3. Hardware Design	201
10.4.4. Software Design	207
10.4.5. Integration & Testing Approach	209
10.4.6. Challenges & Troubleshooting	210
10.5. Project Management & Execution	213
10.5.1. Work Breakdown Structure	213
10.5.2. Gantt Chart & Milestone Tracking	214
10.5.3. Risk Analysis & Mitigation Strategies	214
10.5.4. Budget & Cost Analysis	216
10.6. Experimental Setup & Testing	217
10.6.1. Testing Environment & Procedures	217
10.6.2. Experimental Results and Findings	218
Object Detection for single-camera applications	220
10.7. Discussion & Analysis	232

10.8. Conclusion & Future work	234
10.8.1. Summary of Key Achievements	234
10.8.2. Lessons Learned	235
10.8.3. Recommendations for Future Development	236
10.9. Reflections	237
10.9.1. Original Project Proposal & Changes Over Time	237
10.9.2. Team Reflection on Challenges & Successes	238
10.9.3. Hindsight Analysis: What would we do Differently?	239
11. Simulation	239
11.1. Introduction	239
11.1.1. Background & Motivation	239
11.1.2. Problem Statement	239
11.1.3. Objective & Scope	239
11.1.4. Methodology Overview	240
11.1.5. Report Overview	240
11.2. Engineering Requirements & Justification	240
11.2.1. Justification for Relevance to Degree Program	240
11.2.2. Engineering Principles Applied	240
11.2.3. Health and Safety Consideration	241
11.2.4. Ethical Considerations	241
11.2.5. Regulatory & Standards Compliance	241
11.3. Literature Review & Related work	241
11.3.1. Overview of Existing Technologies & Techniques	241
11.3.2. Comparison with Previous Work	241
11.4. System Design & Implementation	241
11.4.1. System Overview & Architecture	241
11.4.2. Requirements Definition	242
11.4.3. Hardware Design	243
11.4.4. Software Design	243
11.4.5. Integration & Testing Approach	244
11.4.6. Challenges & Troubleshooting	244
11.5. Project Management & Execution	245
11.5.1. Work Breakdown Structure	245
11.5.2. Gantt Chart & Milestone Tracking	245
11.5.3. Risk Analysis & Mitigation Strategies	246
11.5.4. Budget & Cost Analysis	246
11.6. Experimental Setup & Testing	247
11.6.1. Testing Environment & Procedures	247

11.6.2. Experimental Results and Findings	247
11.7. Discussion & Analysis	247
11.8. Conclusion & Future work	248
11.8.1. Summary of Key Achievements	248
11.8.2. Lessons Learned	249
11.8.3. Recommendations for Future Development	249
11.9. Reflections	250
11.9.1. Original Project Proposal & Changes Over Time	250
11.9.2. Team Reflection on Challenges & Successes	250
11.9.3. Hindsight Analysis: What would we do Differently?	250
12. References	250
13. Appendices - A	253
Figure A-1: GNSS UBX mode code part 1.	253
14. Appendix B - Proposal	256
Table of Contents	257
Team Structure	260
Introduction	261
History of the Project	261
5. Power Management	262
5.1 Background	262
5.2 Objectives	262
5.3 Plan	263
5.3.1 Battery Pack	263
5.3.2 Battery Pack Monitor	264
5.3.3 Wiring Harness	265
5.3.4 Custom Interface PCB	266
5.3.5 Documentation / User Manual	267
5.4 Relation to Degree	267
5.5 Group Skill	267
5.6 Methods	267
5.7 Timetable	268
5.8 Risk Assessment	270
6. Navigation	270
6.1 Background	270
6.1.1 Current State of the Art	271
6.1.2 Motivation	274
6.2 Objectives	274
6.3 Plan	275
6.3.1 Major Tasks	275
6.3.1.1 UI/Front-End	275

6.3.1.2 Hosting Server/Backend	281
6.3.1.3 Algorithm	284
6.3.1.4 Route Tracking	287
5.3.1.5 Object Detection	290
6.3.2 Functional Requirements	291
6.3.3 Non-Functional Requirements	293
6.3.4 Software Requirements	294
6.3.5 System Design	295
6.3.6 Milestones	297
6.3.7 Work Breakdown Structure (WBS)	299
6.3.8 Testing Plan	299
6.4 Relation to Degree	301
6.5 Group Skill	302
6.6 Methods	303
6.7 Timetable	304
6.8 Risk Assessment	306
7. Autonomous Control	307
7.1 Background	307
7.2 Objectives	307
7.3 Plan	308
7.3.1 Dynamics Controller	308
7.3.1.1 MPC	308
7.3.1.2 PID	309
7.3.1.3 Reinforcement Learning based control	309
7.3.1.4 Final Considerations	310
7.3.2 Braking System	310
7.3.2.1 Physical Braking System Interface	311
7.3.2.2 Linear Actuator Control and Feedback	311
7.3.2.3 Modes of operation and Safety	312
7.3.2.3 Testing Plan	313
7.3.3 Steering System	313
7.3.3.1 Linear Actuator Control and Feedback	313
7.3.3.2 Modes of operation and Safety	314
7.3.3.3 Testing Plan	314
7.3.4 Speed Sensors	314
7.3.4.1 Physical Design	315
7.3.4.2 Signal Processing and Speed Calculation	316
7.3.4.3 Speed sensor interface	316
7.3.4.4 Testing Plan	317
7.4 Relation to Degree	319
7.5 Group Skill	320

7.6 Methods	320
7.7 Timetable	321
7.7.1 Work Breakdown Structure	321
7.7.2 Phase Breakdown	321
7.7.3 Phase Deadlines	322
7.8 Risk Assessment	323
8. Reliability and Safety	324
8.1 Background	324
8.2 Objectives	324
8.2.1 Functional Requirements	324
8.2.2 Non-Functional Requirements	325
8.3 Plan	325
8.3.1 E-Stop Systems	326
8.3.2 Remote Monitoring System	327
8.3.3 Manual Override System	329
8.3.3.1 RC interface	330
8.3.3.2 GUI and joystick interface	330
8.3.4 Integration with Other Vehicle Systems	331
8.3.5 Heartbeats	332
8.4 Relation to Degree	332
8.5 Group Skill	334
8.6 Methods	337
8.7 Timetable	338
8.7.1 List of Activities	339
8.8 Risk Assessment	340
9. Sensors/Perception	342
9.1 Background	342
9.2 Objectives	342
9.2.1 Functional Requirements:	342
9.2.2 Non-Functional Requirements:	343
9.2.3 Measuring Progress:	343
9.3 Plan	343
9.4 Relation to Degree	345
9.5 Group Skill	346
9.6 Methods	347
9.7 Timetable	348
9.8 Risk Assessment	349
10. Simulation	349
10.1 Background	349
10.2 Objectives	350
10.3 Plan	350

Test Plan	351
10.4 Relation to Degree	352
10.5 Group Skill	353
10.6 Methods	353
10.7 Timetable	354
10.8 Risk Assessment	354
11. References	355

4. List of Figures

Figure 1: Power Audit	26
Figure 2: The new batteries mounted inside the existing enclosure using the foam spacers	28
Figure 3: The 3D model of the new battery cover with mockups for the distribution components	29
Figure 4: The battery cover after being cut out with mounting holes drilled	29
Figure 5: The completed battery cover installed onto the vehicle with distribution components mounted	29
Figure 6: The simplified wiring scheme for the power distribution	30
Figure 7: The shunt and bus bars installed in the ‘HV Junction Box’	30
Figure 8: The motor relay and motor relay control box	31
Figure 9: The regulator installed onto the battery cover	32
Figure 10: The two 12V fuse boxes mounted to the battery cover and connected to the wiring harness	33
Figure 11: The charging port on the rear of the car. To the left is the black charge breaker and below is the blue	33
Figure 12: The inside of the battery monitoring enclosure showing the interface board mounted in the rear	34
Figure 13: The inside of the battery monitor enclosure showing the raspberry pi mounted inside with the usb cable	35
Figure 14: The battery monitor enclosure installed on the vehicle with the BMV showing for manual monitoring	35
Figure 15: The end of a subsystem power harness connector	36

Figure 16: The ethernet switch power adapter	37
Figure 17: The Jetson and USB hub adapter (only one barrel jack shown)	37
Figure 18: The end of one of the subsystem ethernet cables	38
Figure 19: The networking panel with both the switch and modem attached	39
Figure 20: The two emergency stop wires connected to an interface board to form the daisy chain	40
Figure 21: The cut out pattern to mount an interface board to the side of an enclosure	41
Figure 22: An interface board mounted on the vehicle with power and emergency stop harness attached	41
Figure 23: The power management subteam gantt chart	44
Figure 24: Pack Voltage Sag Test	46
Figure 25: Oscilloscope reading at 12V regulator	47
Figure 26: logical Flow of execution for camera stream processing	70
Figure27: Utilizing overpass api to generate a data occupancy grid map	72
Figure 28: a screenshot showing homescreen of the designed UI	73
Figure 29: a screenshot showing search results	73
Figure 30: a screenshot showing edit option for stored locations	75
Figure 31: Data flow of the Navigation Dashboard	77
Figure 32: Flowchart of URDF transform	79
Figure 33: components of the camera sensors.	83
Figure 34: lcd mounted Figure 35: Mount	86
Figure 35: Sensor output data	89
Figure 2: .sh file that starts all the nodes	91
Figure 38: Milestones	92
Figure 39: Timeline part 1	93
Figure 40: Timeline part 2	93
Figure 41: Timeline part 3	93
Figure 42: Timeline part 4	93

Figure 43: Timeline part 5	94
Figure 44: Wheel encoder odometry plus inertial measurement unit (IMU) sensor fusion [NAV24]	103
Figure 45: System Architecture for the Autonomous Control System	112
Figure 46: Autonomous Control full circuit mounted on vehicle	115
Figure 47: PCB Schematic	116
Figure 48: Final PCB 3D Design	117
Figure 49: ROS Bag Vehicle State from test run	122
Figure 50: PID Tuning Page	123
Figure 51: Graph tool for monitoring inputs, outputs and calculated values.	123
Figure 52: Work Breakdown Structure for Autonomous Control	125
Figure 53: Gantt Chart	126
Figure 54: Previous Setup for Override System	138
Figure 55: Conceptual Open Drain Circuit	142
Figure 56: Equivalent circuit when inputs are turned off (output = 12V)	142
Figure 57: Equivalent circuit when an input is turned on (output = 0V)	143
Figure 58: Physical E-Stop circuit	143
Figure 59: Kar-Tech Wireless E-Stop Circuit	144
Figure 60: Interface Board E-Stop Circuit	145
Figure 61: Interface Board Connectors and Signal Pull-Up resistors	145
Figure 62: Interface Board Connectors and Signal Pull-Up resistors	146
Figure 63: E-Stop PCB Layout	146
Figure 64: eStop signal net highlighted in PCB editor	147
Figure 65: 3D Model Front and Back	147
Figure 66: Fully assembled E-Stop board being powered.	147
Figure 67: Power cut-off circuit with relay protection	148
Figure 68: MOSFET and Flyback Diode and Power Cut-Off Relay	149
Figure 69: Power Input of the Interface Board	149
Figure 70: 5V Buck Converter Circuit	150

Figure 71: 3.3V Regulator Circuit	150
Figure 72: Voltage Rail Outputs with 12V Bypass.	151
Figure 73: E-Stop Status and Trigger Interface	152
Figure 74: VDD Selection	152
Figure 75: E-Stop Harness Distribution and Breakout	153
Figure 76: Interface Board Layout	153
Figure 77: Interface Board 3D Model Front and Back	154
Figure 78: Interface Board Powered On	154
Figure 79: Interface Boards Connected Together	155
Figure 80: Interface Board Used by Autonomous Control Team	155
Figure 81: E-Stop System Housing	156
Figure 82: Render of E-Stop System Housing	156
Figure 83: E-Stop Housing with Kar-Tech Wireless E-Stop Mounted	157
Figure 84: Controller Mapping	158
Figure 85: E-Stop Web User Interface	159
Figure 86: E-Stop With Client not Triggering E-Stop	160
Figure 87: Aggressive Timeout Parameters	160
Figure 88: MOSFET circuit without pull-down resistor	162
Figure 89: Gantt Chart for Safety and Reliability team	165
Figure 90: Multiple Clients Controlling Web UI E-Stop	171
Figure 91: Camera Enclosure	194
Figure 92: Camera Mounting	195
Figure 93: GNSS Enclosure	195
Figure 94: Key component locations	197
Figure 95: Wiring	198
Figure 96: Sensors Gantt Chart	204
Figure 97: rosbag plot of Carleton University campus around academic buildings in comparison to Google Maps	208
Figure 98: EDC comparison between Google Maps, an iPhone, and the GNSS with RTK on and off.	208
Figure 99: Azrieli comparison between Google Maps, an iPhone, and	229

the GNSS with RTK on and off.

Figure 100: Billings Bridge Shopping Centre comparison between Google Maps, an iPhone, and the GNSS with	229
Figure 101: Traffic Stop-sign image.	231
Figure 102: Detected objects within corresponding Bounding Boxes.	231
Figure 103: The background scheme for Distance and Direction estimation.	232
Figure 104: Combining images from 2 cameras into disparity map [97]	234
Figure 105: Disparity map generated using stereo_image_proc and synchronized camera feeds.	235
Figure 106: Overview of Simulation Subsystems and interactions with other subsystems.	251
Figure 107: Work Breakdown Structure of the Simulation Subgroup.	253
Figure 108: Gantt Chart of the Simulation Subgroup.	254
Figure 109: Top portion of shows placement of front left and side left cameras. Left bottom portion image displays the Lidar costmap. Right bottom shows long (right) and short range (left) radar.	257
Figure 110: Model of P7 Parking Lot at Carleton University using RoadRunner	257

5. Introduction

5.1. Background and Motivation

The Autonomous Vehicle, AV, project represents a multi-year effort aimed at developing a fully autonomous electric vehicle platform capable of navigating complex environments without any human interaction. Originally started in 2020, the Mechanical

and Aerospace Engineering, MAE, initiated the project with a focus on building the foundations of the project. The project has since evolved through collaboration with the Systems and Computer Engineering, SYSC, and Department of Electrical Engineering, DOE. With the growing interest in autonomous systems and real-time intelligent solutions, this project aligns with global efforts towards safer, smarter and sustainable means of transportation.

In 2024, the project was handed over exclusively to the SYSC department to handle the computer and systems side of the project. Twenty-two students and two supervising professors are tasked with pushing the AV project towards a long-term goal: achieving a robust vehicle autonomy through seamless integration of navigation, control, perception, safety systems and simulations. The motivation for the project is rooted in its real-world applications, from smart transportation to autonomous delivery systems, offering broad societal and technological benefits.

5.2. Problem Statement

The Autonomous Vehicle project aims to address the complex challenge of developing a fully electric, self-driving vehicle capable of navigating and responding to its environment without any human intervention. With autonomous systems becoming increasingly essential in modern transportation, there is a growing need for an integrated platform that encompasses navigation, control, perception, safety, and power management. The core issue lies in unifying these subsystems under a robust, scalable, and modular software and hardware architecture that ensures real-time performance, reliability, and safety.

5.3. Objectives and Scopes

The objective of this capstone project is to develop a modular and scalable autonomous vehicle. This system must support:

- Seamless navigation in a real-world environment.
- Robust control systems for safe throttle, steering, and braking operations.
- A reliable power management subsystem for efficient and safe energy distribution.
- An advanced sensor/perception system to interpret environmental data.
- Integrated safety and reliability mechanisms such as emergency stops and manual overrides.
- A simulation environment for validating software components before deployment.

The Scope of the AV subgroups includes:

- **Power Management:** Designing and implementing a safe, modular system to distribute 12V power, support Ethernet and E-stop connections, and monitor battery

performance.

- **Navigation:** Developing algorithms and integrating hardware for GNSS-based and visual-based positioning to enable autonomous route planning and execution.
- **Autonomous Control:** Creating a robust control system for drive, steering, and braking using actuators and motor controllers interfaced through ROS 2.
- **Sensor/Perception:** Utilizing LiDAR, cameras, and other sensors to detect and interpret the environment for navigation and obstacle avoidance.
- **Reliability and Safety:** Incorporating manual override, remote monitoring, and emergency stop mechanisms to ensure operational safety during autonomous driving.
- **Simulation:** Creating a virtual testing environment to validate vehicle behaviors and subsystems before hardware deployment.

6. Power Management

6.1. Introduction

6.1.1. Background & Motivation

This project aims to build a functional autonomous vehicle with motors, actuators, sensors, a LiDAR, and many microcontrollers. These devices require various different voltages and are placed throughout the vehicle. Due to the fact that other teams are unable to test their hardware on the vehicle without our wiring harness, the power system within this vehicle is very important. Its role is to safely distribute power to all of the components that make up the vehicle. This subgroup, called power management, is in charge of ensuring this. As this vehicle is autonomous, it is crucial that the users have confidence in the power distribution so that all of the components are always working to their full capabilities.

6.1.2. Problem Statement

The power management subgroup is tasked with ripping out the current system, and creating a new power distribution system from scratch. The new system needs to be capable of powering the drive motor from the battery pack and distribute regulated 12V power through a fuse box for all other sub-components. Further, the subgroup is tasked with creating a new charging solution and one that preferably incorporates safety features such as limiting the amount of current able to be drawn from the charger when it is connected to the vehicle.

6.1.3. Objective & Scope

The key deliverables of the subgroup can be broken down into 5 main goals. The first deliverable is creating a new battery pack. The goal of the new battery pack is to both get it to a functional state and to improve its operational safety. The next deliverable is to monitor the battery pack. The goal of battery pack monitoring is to ensure that the battery is operating within safe margins and to allow the software that is operating the vehicle (ROS), to be able to access the live status of the battery. Another deliverable is the wiring harness. The goal here is to provide 12V power, ethernet connectivity, and direct connections to the various subsystems throughout the vehicle. We will create internal specifications for each power subsystem, and create custom cables using a standardized set of wires and connectors. Further, another deliverable is to create a custom interface PCB for each subsystem. The goal of this deliverable is to make a standardized voltage regulator and estop interface in order to make connecting to the wiring harness easier for other subgroups. This will be in collaboration with the reliability and control subgroup to include the E-stop interface components. Finally, the last deliverable is the overall documentation and user manual. This is to share knowledge of the modular 12V system and proper practices for operating, charging, and maintaining the new batteries.

6.1.4. Methodology Overview

The waterfall methodology was utilized for working on the power management tasks. This was accomplished by creating the battery pack to start and ensuring that the entire pack was generating at least 60V total as advertised. We then moved on to the charging circuit and completed this before moving on. Finishing the battery portion of the system first, allowed us to start the wiring harness with peace of mind knowing that if the 12V system was not working later on, we would not have to debug the batteries as this was already done. This is why the waterfall method worked well for our subgroup. After the battery pack and wiring harness, which were the two most important tasks because they directly impacted the other subgroups for the project, we moved on to the voltage regulating interface PCBs and the battery monitoring system.

6.2. Engineering Requirements & Justification

6.2.1. Justification for Relevance to Degree Program

This project relates back to computer systems engineering and software engineering in many ways. The task of implementing the interface PCBs for voltage regulation requires printing circuit boards. ELEC 2501 and ELEC 2507 (circuit design) are courses that will help with the design of this task as it taught us the basics of how

circuits work. Also, the task of battery monitoring requires a lot of programming which we have had countless courses in throughout software engineering and computer systems. SYSC 3010 (project development) taught us how to handle databases and the battery monitoring results will need to be uploaded to the ROS network and then put into online tables. Also, the project development is essentially a smaller version of the capstone project as it taught us report writing and working with a team to develop a project.

6.2.2. Engineering Principles Applied

Some of the engineering principles we used are modular design and fail-safe design. The wiring harness is entirely modular due to the connectors as well as the voltage regulation. This allows future groups to upgrade the vehicle easily as well as unplug sub-systems when debugging. The charging solution we designed has a breaker that will turn itself off if the circuit is overloaded. This can occur if someone tries to charge the vehicle while also operating it, in this case the breaker will protect the system from damage.

6.2.3. Health and Safety Consideration

Laboratory safety was highly valued during the project. The vehicle was housed in the engineering design center (EDC) Bay 4 for the duration of the year. This space had all the necessary safety measures. While working with batteries up to 60V and 55Ah, as well as many wires carrying 12V at a time, the area provided the isolation we required. Bay 4 had a fire extinguisher as well as a first-aid kit. Also, before being given access to this space, we had to read and sign the health and safety forms highlighting how to prevent any risks.

6.2.4. Ethical Considerations

Some of the ethical aspects taken into consideration are passing down all data sheets to next year's capstone group. The development of this project will continue next year and so it is important that future students are made aware of the components they are working on. For example, the batteries and charger are both major potential hazards and so it is crucial to share all information about them and how to operate them.

6.2.5. Regulatory & Standards Compliance

Industry standard regulation was followed for many aspects of the power management system. The wiring harness has standard colour coded wires for positive and negative. Red wires are always carrying positive charge and the black

wires are being grounded. This is important for safety when handling high voltage and amperage.

6.3. Literature Review & Related work

6.3.1. Overview of Existing Technologies & Techniques

The current state of the art for the power distribution in electric vehicles can be split into many sub categories. To begin, the battery monitoring shows the user key information regarding the status and health of the batteries. The users should know how much power is being drawn as well as battery life remaining. Next, modern vehicles have step down converters to supply the smaller systems such as lights or display screens. There are many electrical components in vehicles other than the motors themselves that require much less power to operate and so voltage regulation is required. Further, CAN and ethernet are two of the methods that the controllers in the vehicle can communicate with one another [93]. These are examples of modern technologies that we have incorporated into our power management as well.

6.3.2. Comparison with Previous Work

We used an off the shelf BMV battery monitor that will inform users of current power draw and battery capacity. This is in line with modern solutions as far as keeping the users up to date with what is happening under the hood. Also, our power system uses an ethernet cable to communicate with the emergency stop line and this is in line with modern solutions. Further, we have created interface PCBs that allow the 12V lines to step down to either 5V or 3.3V, as we see in modern vehicles on a larger scale.

6.4. System Design & Implementation

6.4.1. System Overview & Architecture

The power management system was divided into five main parts to be worked on throughout this year. The battery pack, the battery monitoring system, the wiring harness, the interface PCB, and the combined documentation and user manual.

Battery Pack

The battery pack is completely replaced from the previous year. It consists of five 12V 55Ah SLA batteries wired together in series. These batteries are mounted inside the factory battery compartment using foam as a vibration dampening material. This compartment is covered with a newly made plastic cover with built in vents to allow for passive cooling. On top of this cover mounts most of the power

distribution components. This includes the pack voltage bus bars, the shunt for the battery monitoring, the motor relay, the 12V regulator, and the 12V fuse boxes. The battery pack also encompasses the rear charging door. This was modified from previous years with the addition of a new charging breaker to increase the safety of the charging system.

Battery Monitoring

The battery monitoring system is completely new. It consists of the off the shelf Victron BMV712 battery monitor connected to a raspberry pi using a Victron VEDirect USB cable. The BMV uses the shunt mounted on the battery cover to measure the battery pack's current and voltage as well as the output voltage of the 12V regulator. The raspberry pi will read the battery information being transmitted over serial from the BMV. It then converts these values into the standard Battery Status ROS message. Notably calculating the SOC from voltage instead of using the BMV's readings due to unreliability. These messages are then sent over ROS to the rest of the vehicle to be used by other subsystems. The BMV, the usb cable, and the raspberry pi are then mounted into the vehicle inside the battery monitor enclosure.

Wiring Harness

The wiring harness is completely new. It consists of the following parts: the power wiring, the emergency stop wiring, the ethernet wiring, and the ethernet hub and switch. The goal of the wiring harness is to provide a standardized consistent set of power, ethernet, and emergency stop wiring for all subsystems present on the vehicle. The power connects the fuse boxes in the battery pack with the interface boards in the subsystem. This provides each subsystem with an independently fused power circuit. The ethernet is run between every subsystem and a switch which is also connected to a LTE router. The emergency stop wiring daisy chains all interface boards together and terminates in the motor relay enclosure through the E-Stop board. This allows any subsystem to read or trigger an emergency stop through hardware allowing the whole ROS network to be bypassed. More information can be found in the safety subsection of the report.

Interface PCB

The interface PCB was a collaboration between the power management and safety subteams. The interface PCB is responsible for interfacing a subsystem to the power and emergency stop harnesses. It provides a passthrough of the power harnesses 12V power and also provides regulated 5V and 3.3V power. It also houses the interfacing circuitry to drive and level shift the emergency stop lines

which are at 12V. A Standard Interface panel and corresponding template for the enclosure opening was also designed in CAD and 3D printed to make the interface panel easier to incorporate into enclosures.

Documentation / User Manual

A set of documentation and a user manual was made to ensure good knowledge transfer between this year's power management subteam and the future group of students who will work on this vehicle. The user manual portion includes a set of operating procedures for common tasks relating to the vehicle. The hope is to improve how informed all students are to prevent the destruction of the battery or charger which happened in previous years. The technical documentation section provides details similar to the report on how certain systems were designed, implemented, and ideas on how to improve them in the future. The documentation is available on GitHub [94].

6.4.2. Requirements Definition

Some universal internal requirements below were decided in order to narrow down and assist the design of the individual parts.

- **Hardware**
 - M4 socket head hardware was chosen due to its availability and suitable size
- **Stud sizing**
 - The stud and ring terminal sizing outside of the battery terminals and motor relay was standardized to #10 to reduce the bill of materials. This size was chosen as the ring terminals would fit in the regulator and fuse boxes which we would reuse from last year
- **Voltage tolerances**
 - We aimed for a maximum voltage rating of at least 80V for anything that would connect to the battery pack directly as the system can reach upwards of 75V when charging.

The requirements we set out for our design of the five parts are listed below

Battery Pack

The requirements for the battery pack as a whole is to be able to supply both 60V for the main drive motor as well as 12V for all the vehicle's subsystems. The individual components that make up the battery pack had more in depth requirements that are listed below.

- **Batteries**

- Must be of similar chemistry to the old batteries: Sealed Lead Acid (SLA)
- Capacity comparable to the old batteries: 50Ah
- Must fit in the existing battery enclosure
- Cover
 - Must be made out of a suitable material: UV resistant and able to handle direct sunlight
 - Must be easily machinable: Preferably we would be able to machine this part ourselves without relying on the machine shop
- Pack Voltage Bus Bars
 - Must have #10 size compatible studs to be used with the ring terminals.
 - Must be able to handle at least 60V
 - Must be able to handle at least 50A
- 12V Regulation
 - Based off the power audit conducted (see the following figure) we needed at least 32.3A of total current capacity with preferably a large overhead for future expansion
- Charging Breaker
 - Must be able to handle pack voltages up to 80V
 - Rated for current above the charger's current rating of 3A. Preferably around 4-5A

Subsystem Name	Subteam Responsible	Voltage Level (V)	Wattage (W)	12V Amperage (A)	Regulator	Ethernet	Estop	Fuse (A)	Notes
Dashboard	Navigation	5 V	15 W	1.3 A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2.0 A	Navigation + Dashboard Display
Autonomous Controller	Autonomous Control	12 V	15 W	1.3 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2.0 A	Pi + ESP32 + Any other control circuitry
Steering	Autonomous Control	12 V	180 W	15.0 A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	15.0 A	
Breaking	Autonomous Control	12 V	60 W	5.0 A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	7.5 A	
Safety	Reliability & Safety	5 V	15 W	1.3 A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2.0 A	Pi + Estop circuitry
Motor Relay	Reliability & Safety	12 V	14 W	1.2 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2.0 A	Power for the motor relay coil
Jetson Xavier + USB Hub	Sensors	12 V	30 W	2.5 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3.0 A	
Lidar	Sensors	12 V	8 W	0.7 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2.0 A	
Radar	Sensors	12 V	12 W	1.0 A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2.0 A	
Battery Monitor	Power Management	5 V	15 W	1.3 A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2.0 A	
Ethernet Switch	Power Management	5 V	6 W	1.0 A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2.0 A	
Wireless Modem	Power Management	12 V	12 W	1.0 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2.0 A	
Total Amperage :					32.3 A				

Figure 1: Power Audit

Battery Monitoring

The battery monitoring system had the following general requirements. We wanted to be able to measure at least current, pack voltage, and state of charge. We needed to be able to also post this information onto the ROS network to be accessible by other parts of the vehicle. We

wanted these messages at a usable frequency (≥ 1 Hz) for monitoring changes between sudden transitions in motor current for testing.

Wiring Harness

The three harnesses are split up below with their requirements

Power Harness

- Must be able to handle 12V
- Must be suitable for a vehicular environment
- Must be able to carry at least 15A
- Must have a set of latching connectors with a pcb receptacle offered

Ethernet Harness

- Must have at least 1Gb speeds
- Must be relatively cheap (no shielded or expensive “tough” cabling)
- Must be able to handle at least all devices planned (6)
- Must have ability to connect to LTE for remote internet connection
- Must be able to host a wifi network for remote monitoring

E-Stop Harness

- Preferably re-use ethernet wiring to reduce number of components

Interface PCB

- Provide connection to the 12V from the power harness over more reasonable sized connectors for small systems
- Provide connection to the emergency stop harness through its interface circuitry
- Provide a standard interface panel and cut out template for others to use
- More requirements and information relating to the interface PCB can be found in the safety section of the report

Documentation / User Manual

- Provide step-by-step instructions on how to perform common tasks
- Include information to explain why things are done to ensure steps are followed
- Include technical information about the systems added this year such that they can be quickly referred to by future years

6.4.3. Hardware Design

A lot of hardware design work went into this sub team's efforts due to the very physically focused aspect of the power management team.

Specific part numbers and links can be found in the appended KEFC funding requests.

The hardware design is split up among the various main parts

Battery Pack

The batteries chosen for the battery pack were the BW12500 12V 55Ah SLA batteries. These were chosen as a compromise as we were unable to find a direct replacement for the vehicles old batteries. These batteries are of the same chemistry and a higher capacity than the old ones. They are a slightly different shape and size compared to the old ones but after using the datasheet and some measurements of the battery enclosure we were able to ensure that an arrangement of the new five new batteries would fit in the existing enclosure. These batteries were not the first choice we had but we ended up having to change plans due to the supplier's stock running out.

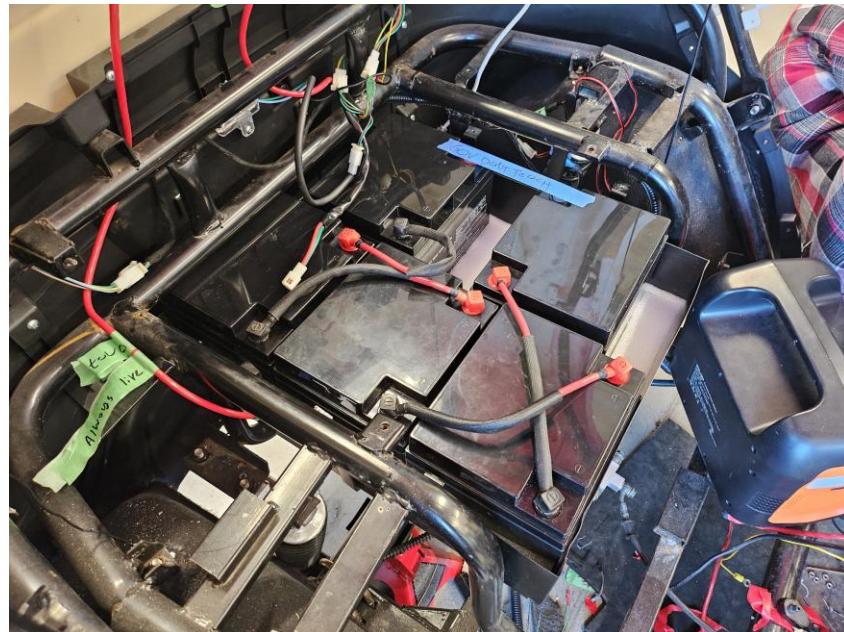


Figure 2: The new batteries mounted inside the existing enclosure using the foam spacers

The material we decided on for the battery cover was a 1/8th sheet of HDPE plastic. This plastic is UV resistant and thus suitable for outside use in direct sunlight while also being soft and easily machinable. The sheet was designed in Fusion360 then manufactured and installed onto the vehicle. The vents which attach to the bottom of the sheet to allow for battery cooling and off-gassing are 3d printed and attached with M4 Hardware. This cover has various holes to allow the mounting of the following power distribution components.

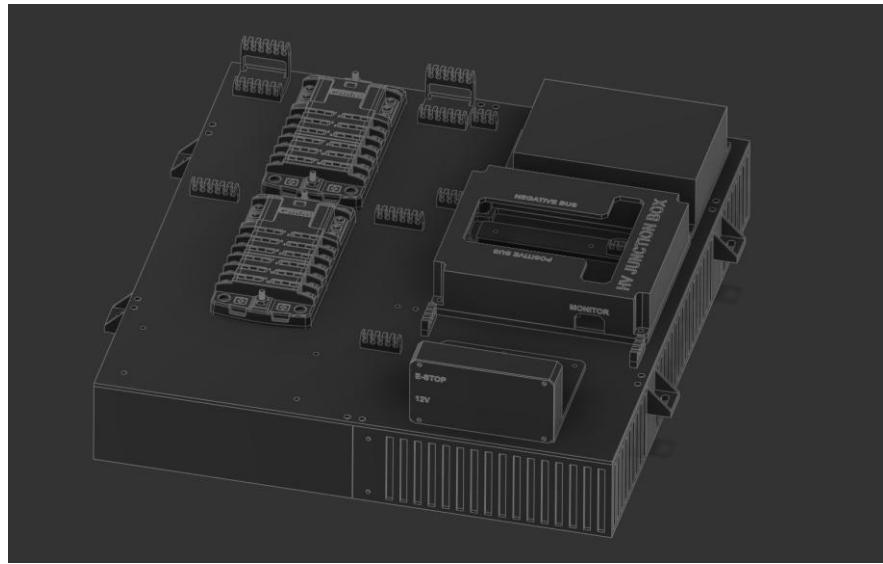


Figure 3: The 3D model of the new battery cover with mockups for the distribution components

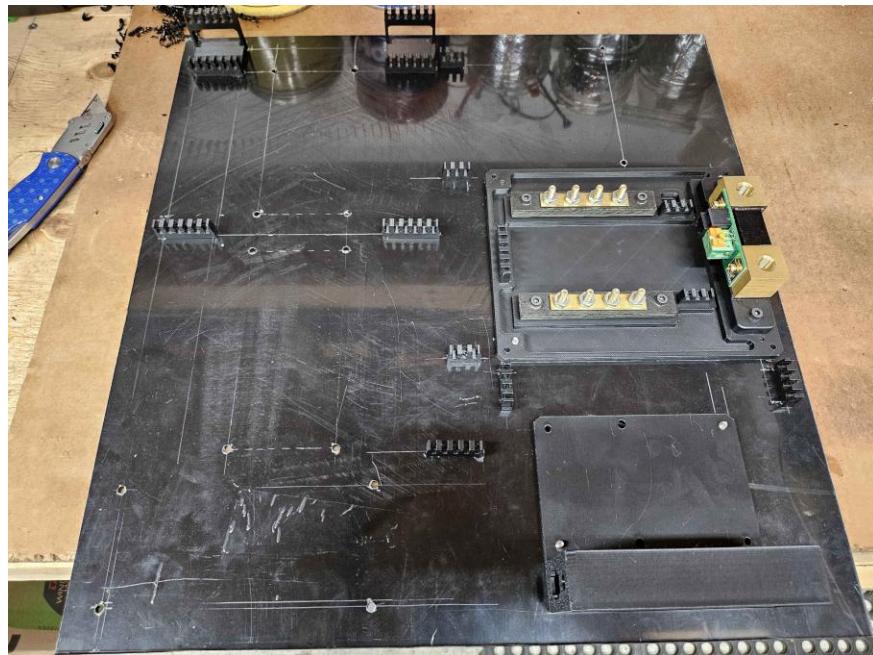


Figure 4: The battery cover after being cut out with mounting holes drilled

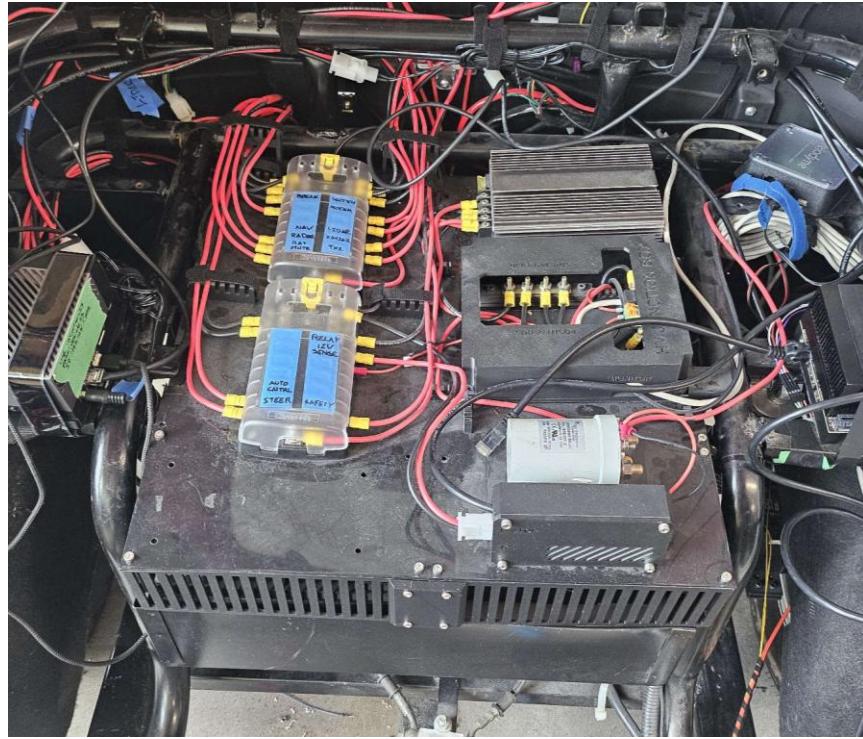


Figure 5: The completed battery cover installed onto the vehicle with distribution components mounted

The power distribution system consists of the following parts: the shunt, the bus-bars, the motor relay, the 12V regulator, and the 12V fuse boxes. The shunt was included with the Victron BMV712 battery monitor we are using and is connected in between the battery pack's ground and the system's ground so that we can monitor the total current strain being put on the batteries. It is a 500A/50mV shunt (meaning that at 500A of load it will generate a 50mV voltage differential across its two terminals). This is extremely overkill for our use case, this led to some issues in regards to accuracy at low current loads. The bus bars selected were 2x 46206-04 bus bars from Littelfuse. These parts were selected as they include size 10 studs and are rated for currents up to 60A. These are used to break out the positive (post battery pack breaker) and negative (post shunt) connections from the batteries. The shunt and bus-bars are mounted in a 3D printed enclosure called the "HV Junction Box" which is then attached to the battery cover with M4 hardware.

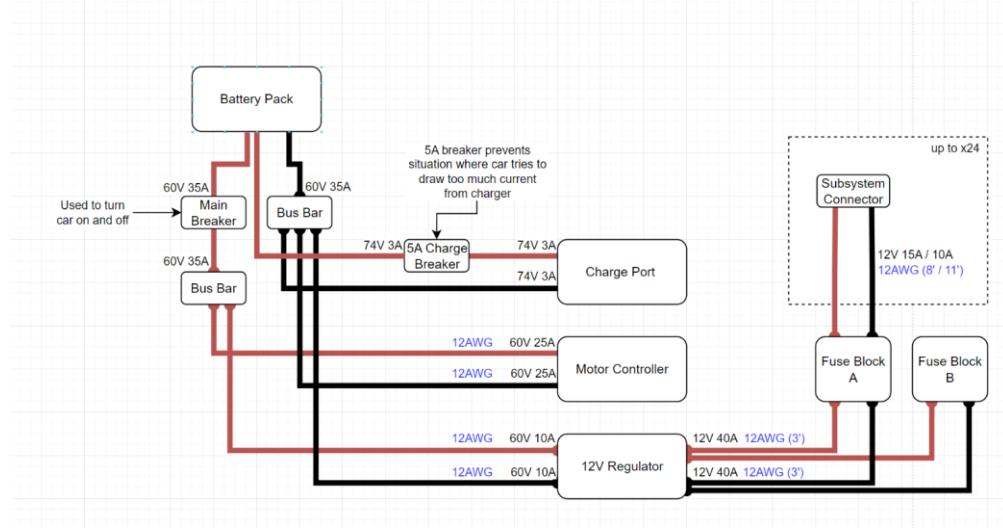


Figure 6: The simplified wiring scheme for the power distribution

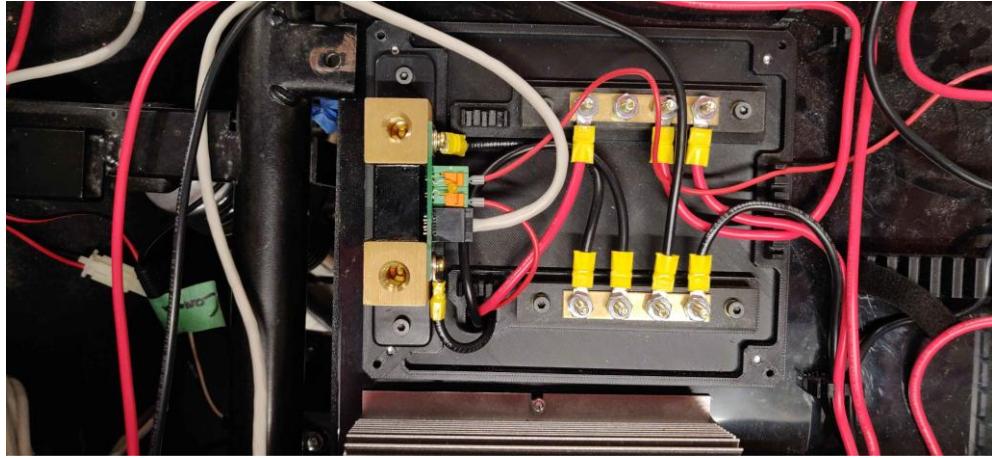


Figure 7: The shunt and bus bars installed in the “HV Junction Box”

The motor relay was added to the vehicle to allow the emergency stop system of the safety subteam to be able to control the power supplied to the vehicle's main drive motor. This feature was included to allow the vehicle to still retain power to all other systems while an emergency stop is triggered. This allows the vehicle to perform additional functions such as applying brakes, and steering out of the way of obstacles while allowing for the vehicle's motor to be disengaged in the case of an emergency. The motor relay mount attaches to the battery cover using M4 hardware.

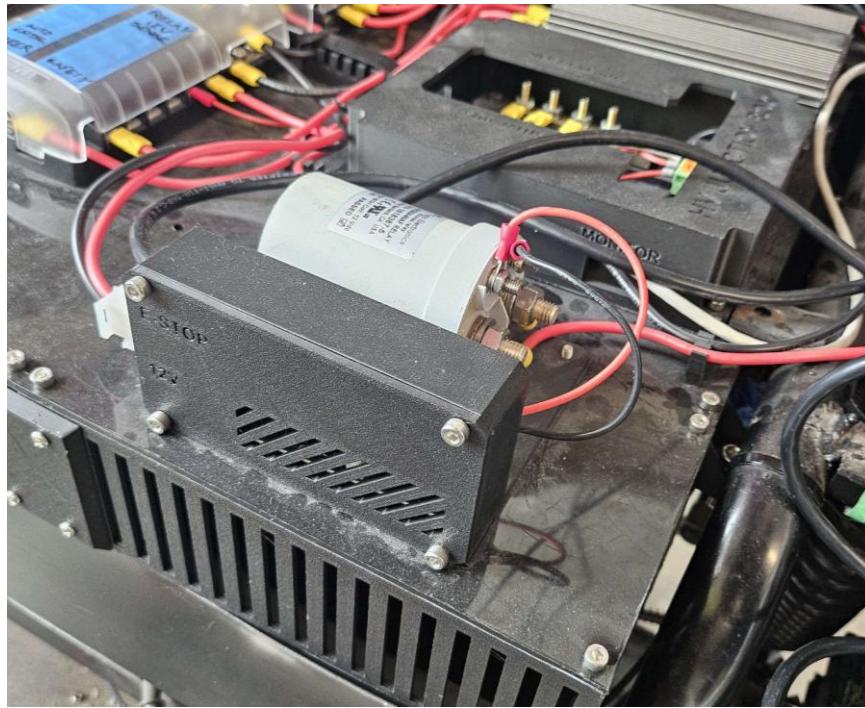


Figure 8: The motor relay and motor relay control box

The 12V Regulator used by previous years was retained as it seemed to be capable enough according to the power audit performed at the beginning of the year. It is a 12V 600W (50A) regulator which allows us to have plenty of overhead on top of the overhead provided in the power audit amperage estimations. It is mounted to the battery pack cover using M4 hardware.



Figure 9: The regulator installed onto the battery cover

The 12V fuse boxes are another carry over from previous years. They were integrated into the system to split up the 12V power coming out of the regulator. From there they provide up to 48x fused subsystem circuits that can then be attached to the wiring harness. These fuse boxes in combination with the reused 12V regulator were used to decide on the ring terminals that we standardized across the vehicle (Except for the battery terminal connectors and motor relay). The fuse boxes mount to the battery cover using M4 hardware.

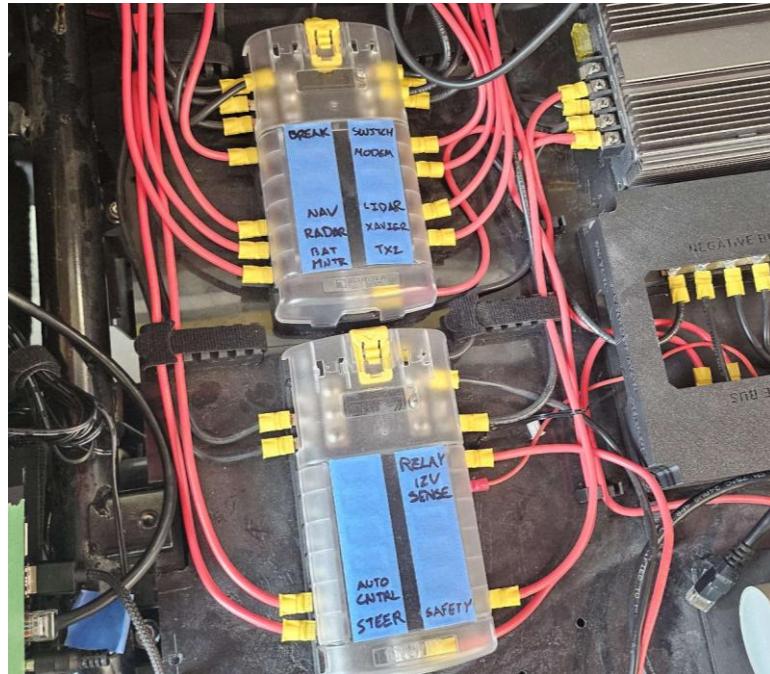


Figure 10: The two 12V fuse boxes mounted to the battery cover and connected to the wiring harness

The charging circuitry was modified this year to increase the safety and protection it offered. A charging breaker in between the charging port was added to prevent the car attempting to pull too much current from the charger when it is attached. It also allows us to turn off the power to the receptacle when there is no charger plugged in, on the old design the prongs in the receptacle would have exposed 60V across them. A new charging port panel was printed and installed onto the vehicle with M4 hardware.



Figure 11: The charging port on the rear of the car. To the left is the black charge breaker and below is the blue battery pack breaker

Battery Monitoring

The hardware design portion of the battery monitoring part includes the CAD design of an enclosure to house the BMV, usb cable, raspberry pi, and interface board. This part was 3D printed and attached to the vehicle's dash using a 3d printed clamp.



Figure 12: The inside of the battery monitoring enclosure showing the interface board mounted in the rear

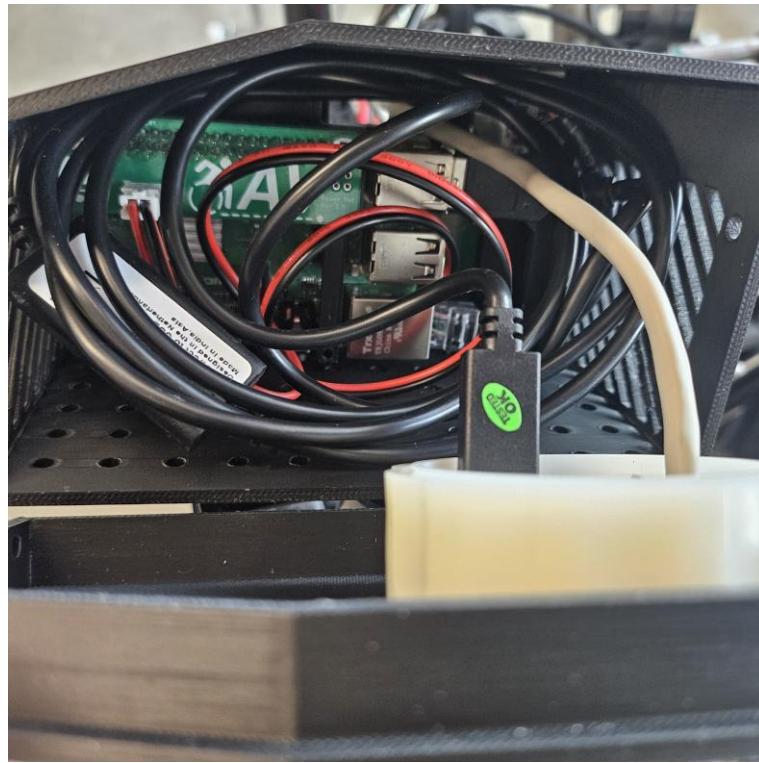


Figure 13: The inside of the battery monitor enclosure showing the raspberry pi mounted inside with the usb cable and ethernet pass through attached



Figure 14: The battery monitor enclosure installed on the vehicle with the BMV showing for manual monitoring

Wiring Harness

The power harness consists of three parts: the wiring, the connectors, and the subsystem adapters.

For the power wiring we went with 12 AWG TEW wire in both black and red (for positive and negative). 12 AWG was chosen as it should be plenty able to handle the 15A load a subsystem would place on it. TEW was specifically chosen due to its high voltage rating (600V) allowing us to use it on both the pack voltage runs and the lower voltage subsystem runs. TEW rating also means it is good for a wide temperature range of -25C to 105C and has resistance against oil and water which is needed in an automotive environment. For the connectors we chose to go with TE Mate-N-Lok connectors due to their good price, locking latches, and ability to operate with a wide range of wire gauges from the 12 AWG wire we use for the power harness down to the 18AWG wire used on some of the subsystem adapters.

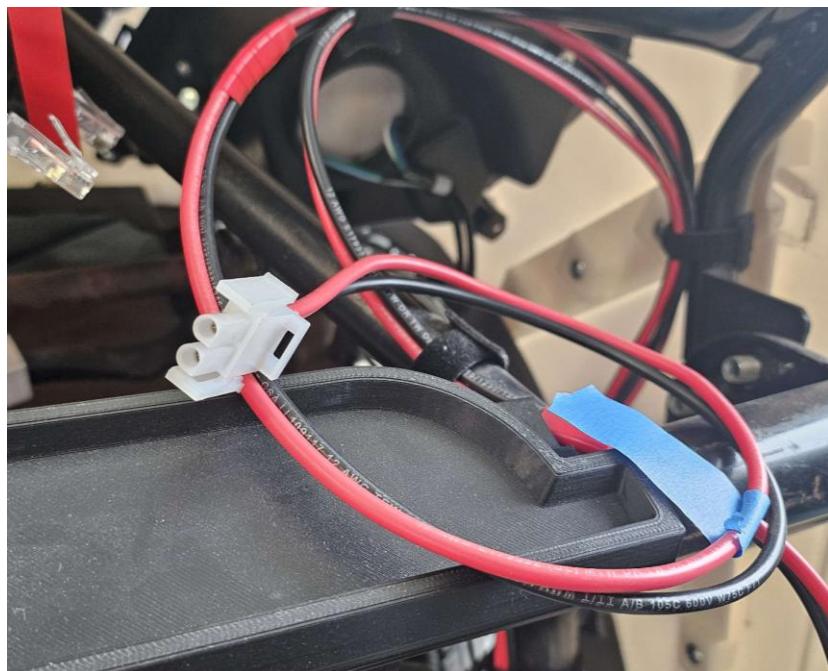


Figure 15: The end of a subsystem power harness connector

Some subsystems require a different voltage such as in the case of the ethernet switch, or a special connector such as with the jetsons' barrel jack. These necessitate the need for adapters. In the case of the ethernet switch which takes a 5V input rather than 12V an adapter was designed housing an interface board in a 3d printed enclosure with the subsystem Mate-N-Lok connector on one end and a 3.5x2.1mm barrel jack on the other.

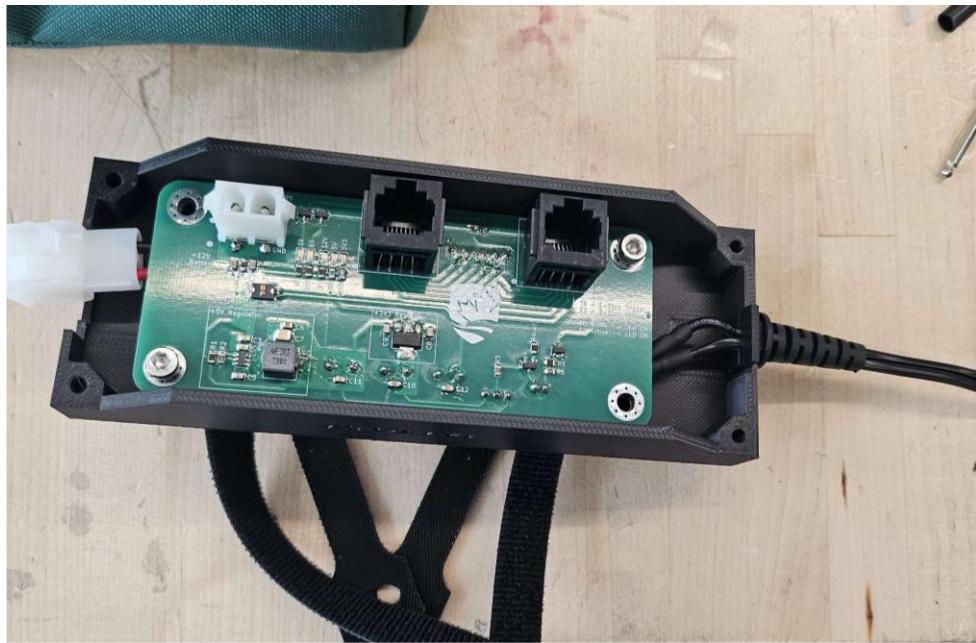


Figure 16: The ethernet switch power adapter

For the jetsons, USB hub, lidar, and LTE modem a simpler adapter could be made as these devices all need 12V inputs. A simple adapter that converts the Mate-N-Lok connector to a barrel jack was made for these.

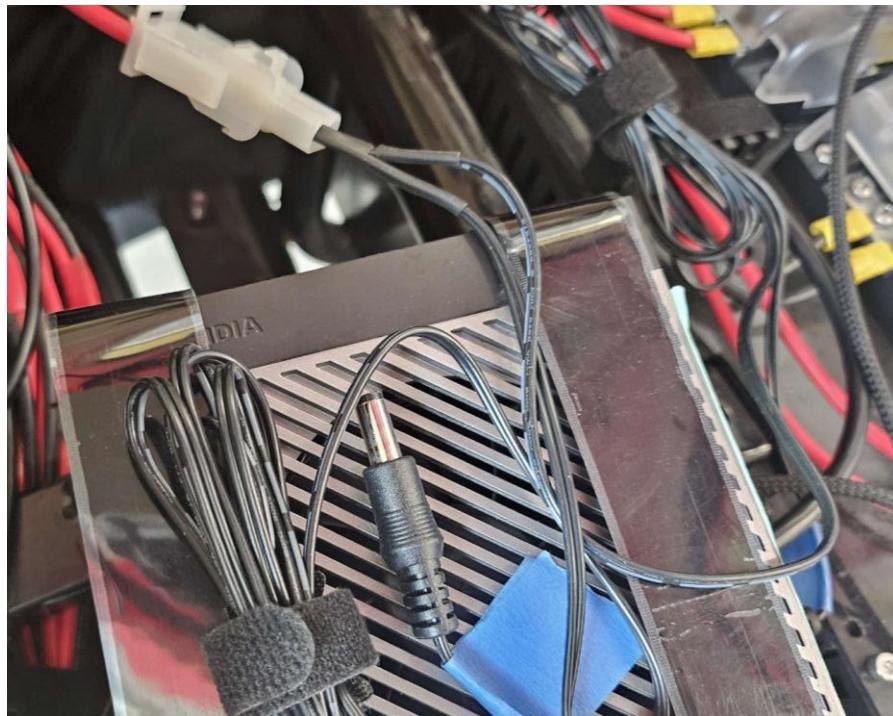
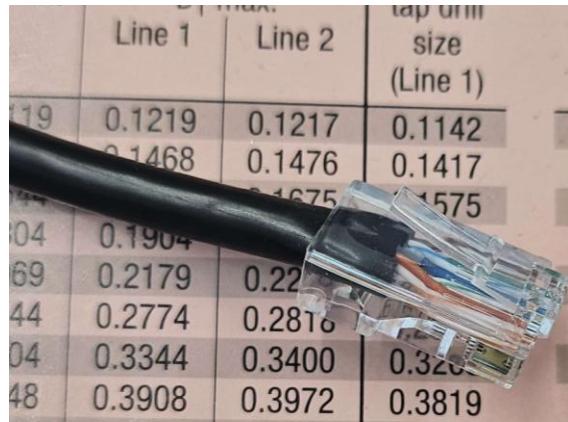


Figure 17: The Jetson and USB hub adapter (only one barrel jack shown)

The ethernet harness consists of three parts: the wiring, the ethernet switch, and the LTE modem.

For the ethernet wiring we decided to go with CAT6 grade cabling and connectors as it is plenty overkill for the 1Gb speeds we are targeting and could allow for upgrading to 10Gb in the future if needed. The cables were cut to length and crimped with added service loops at the subsystem end to provide for an exact length cable but with enough flexibility to allow for designs to change where they are mounted and where exactly the harness plugs in.



	Line 1	Line 2	tap off size (Line 1)	size D	P [mm]	D ₁ min. [mm]
19	0.1219	0.1217	0.1142	LK-M 3	0.5	2.673
468	0.1468	0.1476	0.1417	LK-M 4	0.7	3.549
1675	0.1675	0.1575		LK-M 5	0.8	4.324
04	0.1904			LK-M 6	1	5.152
69	0.2179	0.22		LK-M 8	1.25	6.931
44	0.2774	0.2816		LK-M 10	1.5	8.700
04	0.3344	0.3400	0.320	LK-M 12	1.75	10.477
48	0.3908	0.3972	0.3819	LK-M 14	2	12.237
23	0.4493	0.4564	0.4375	LK-M 16	2	14.237
37	0.5067	0.5148	0.5000	LK-M 18	2.5	15.787
43	0.5633	0.5728	0.5512	LK-M 20	2.5	17.787

Figure 18: The end of one of the subsystem ethernet cables

For the ethernet switch we went with an 8 port unmanaged 1Gb switch as it would allow for the speed targets we were aiming for while also being affordable. The 8 port model was chosen as it would accommodate the 6 wired devices on the vehicle as well as the connection to the mode. This leaves one free port which was used to connect the vehicle to Carleton's LAN when testing to give the vehicle internet access without needing the LTE. The modem allows for the vehicle to connect to a LTE network to receive internet access, this was done to allow the remote monitoring to sync their data with a Google Firebase backup. The modem also hosts a WiFi network allowing the remote monitoring to connect to the vehicle wirelessly and display realtime information about the vehicle when it is in action. It is also extremely helpful to be able to connect the WiFi to SSH into the vehicle's various systems while testing. The switch and modem are mounted onto the networking panel which was designed in CAD and 3D printed. It is attached to the vehicle on the back right.

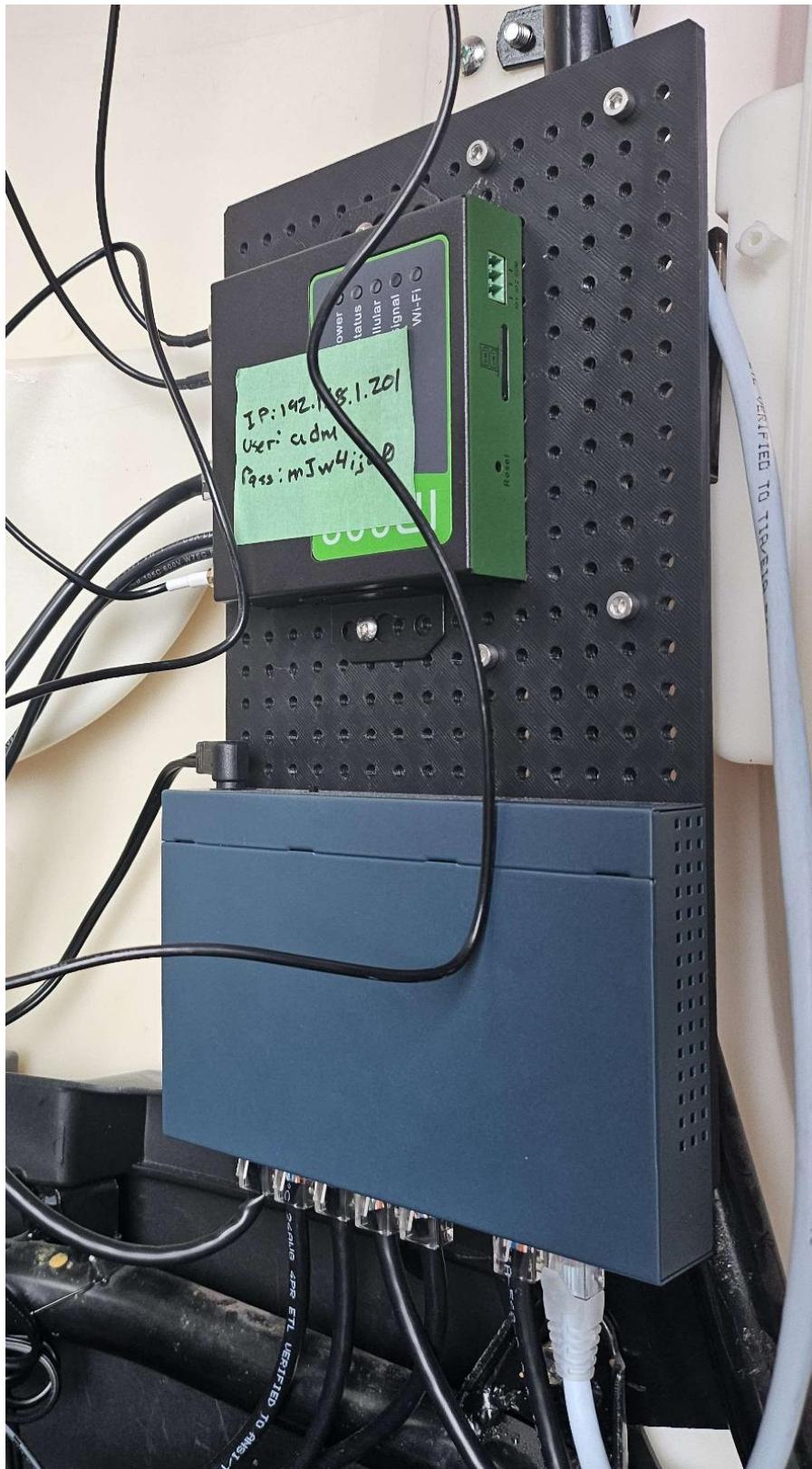


Figure 19: The networking panel with both the switch and modem attached

The emergency stop harness consists of three parts: the wiring, the motor relay, and the interface boards.

The wiring used was the same CAT6 we used for the ethernet harness. This allowed us to reduce the amount of materials needed. The emergency stop wires are differentiated from the normal ethernet wires using a piece of red heat shrink tubing. If a mistake were made and these wires are swapped it should be okay as the emergency stop signals operate at 12V which is less than what ethernet is equipped to handle. This wiring is used to daisy chain all the interface modules together which serve as the interface between the power and emergency stop harnesses and a submodule. This daisy chain circuit is terminated on one end by the motor relay which then reads the signal on the emergency stop line and controls the relay to cut off power to the vehicle's motor when an emergency stop is triggered. Two of the CAT6's eight conductors are used for this functionality with the other six being reserved for future use.



Figure 20: The two emergency stop wires connected to an interface board to form the daisy chain

Interface PCB

As this was a collaboration the majority of the credit should be attributed to the safety team. Power management's role was mainly to pick out the connectors required for the power and emergency stop harnesses to connect to the interface PCB. The power is connected using the matching Mate-N-Lok PCB receptacle part. The emergency stop uses normal RJ45 connectors without magnetics. Magnetics are not needed as we are not following the ethernet spec for the emergency stop harness and just using the cabling for our own purposes. The power management

subteam did design the interface board panel and cutout. This reusable part takes into account the complex geometry around the connectors and pcb components and instead uses a simple square cutout with mounting holes to make it simple for others to include the interface board into their design. The panel is then mounted using M4 hardware.

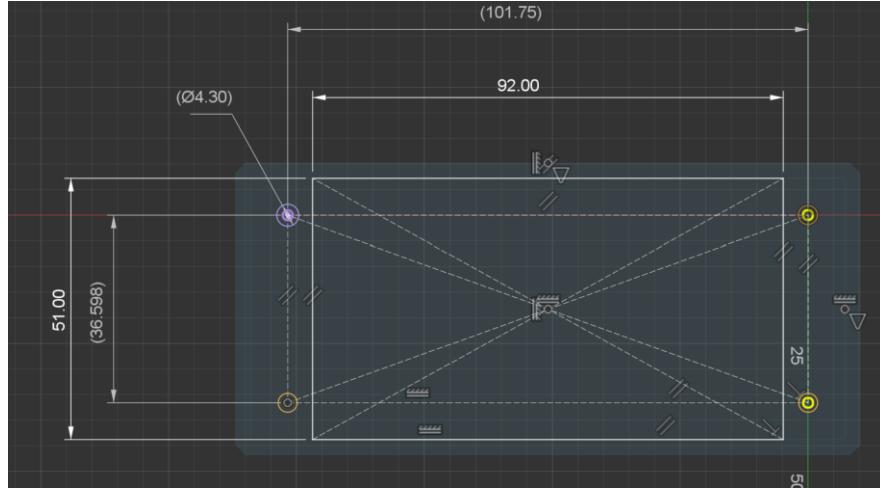


Figure 21: The cut out pattern to mount an interface board to the side of an enclosure



Figure 22: An interface board mounted on the vehicle with power and emergency stop harness attached

6.4.4. Software Design

Battery Monitoring

The battery monitoring system needed a custom ROS2 module to be created to interface the Victron battery monitor's serial output with the vehicle's ROS network. This module consists of the following parts: VEDirect class responsible for handling the serial connection, the BatteryStatePublisher responsible for sending BatteryState messages to the ROS network, and the main battery_monitor file which creates the BatteryState message from the VEDirect data, calculates the SOC from voltage and then sends it to the publisher.

VEDirect.py

- State machine used to change how the program will parse a byte being passed to it from the serial connection depending on what portion of the message block the system is currently reading
- Calculates and checks block checksums to ensure no corrupt packets are recorded on the battery monitor
 - This also handles a lot of edge cases such as when the serial interface is connected midway through a block being sent. Due to only a portion of the block arriving the checksum will not match meaning the block will be discarded
- Stores valid block data in an internal dictionary and issues a callback to let the system know that a new valid block has arrived

BatteryStatePublisher.py

- Publishes a passed BatteryState message to the ROS network

battery_monitor.py

- Initializes the serial connection and passes all incoming bytes to the VEDirect object
- When a callback is called from the VEDirect it then checks if both blocks have been read (the information is passed in two blocks so we check if a value has been set in both sections to ensure both blocks have arrived).
- If so it will convert the VEDirect dictionary into a BatteryState ROS message
- As part of this it will convert voltage into SOC rather than using the SOC from the battery monitor as it has found to be very inaccurate without calibration
- Then send the BatteryState to the BatteryStatePublisher to publish the message
- This is all deployed onto the raspberry pi as a docker container which is hosted on DockerHub. This docker is built automatically when the GitHub has a push on main allowing the raspberry pi to be easily updated to new code.

6.4.5. Integration & Testing Approach

Throughout the year as progress was made on the battery pack, wiring harness, and interface PCBs they were tested iteratively. We ensured that the battery pack was regulating voltage correctly, we tested the continuity of the wiring and fuses by checking that we were getting 12V on the end of the power harness. The custom ethernet cables of the ethernet and emergency

stop harnesses were tested with a RJ45 network cable tested to ensure the wires were crimped correctly.

The integration between the power management team, the safety team, and the autonomous control team was regularly tested and improved as our implementations were gradually realized.

The battery monitoring system was tested by echoing the output of the battery_status ROS topic and comparing the ROS values with the values displayed on the BMV inside the vehicle.

To test the power management system's full capabilities and performance in real world action we drove the car around as a test in a closed off parking lot. During this test we set up a ROS bag to record all the ROS activity while this test was being performed. This allowed us to see the readings of the pack voltage, current drawn, and 12V regulated output in relation to what the rest of the vehicle was doing.

6.4.6. Challenges & Troubleshooting

One challenge we encountered was with the purchasing of new batteries. After we were approved for the KEFC funding, we had an issue as the original supplier we got the quote from had run out of stock. This meant that we had to find alternative batteries of similar price. Unfortunately the only suitable choice was batteries that were physically bigger than the original choice.

When the supplier notified us that they were out of stock, we had to look for alternatives. Thankfully, we were able to gather multiple quotes from different suppliers, allowing us to find an alternative battery very quickly. To deal with the differently shaped and bigger batteries, we measured the existing enclosure and modeled it with the new batteries inside using the dimensions from the specification sheet. This allowed us to confirm that the new batteries would be able to fit even though they were bigger. We then remade the foam inserts to fit the new batteries into the existing controller.

Another challenge was when the car stopped moving during our inaugural full system test run. Upon checking the battery voltage it was dangerously low. After looking into it we discovered that the batteries were totally dead but the Victron battery monitor was reporting them at 70% SOC. This led to the realization that the Victron battery monitor does not calculate its SOC based on the battery pack's voltage but instead the amount of amp hours that have passed. In my opinion this is a flaw in our use case. In the battery monitoring code I replaced the Victron's battery SOC with our own calculated SOC based on the battery's voltage that provides a much more accurate state of charge for the vehicle. It does fluctuate based on the load on the batteries as the voltage will sag through.

6.5. Project Management & Execution

6.5.1. Work Breakdown Structure

Work was split between the team members fairly evenly with all members contributing and assisting on every part. We decided that the tasks should have someone responsible as the main driver though for each part. These main drivers are as follows

Battery - Jacob Wilde

Battery Monitoring - Jacob Wilde

Wiring Harness - Taran Basati

Custom Interface PCB connection - Taran Basati

Documentation / User Manual - Jacob Wilde

6.5.2. Gantt Chart & Milestone Tracking

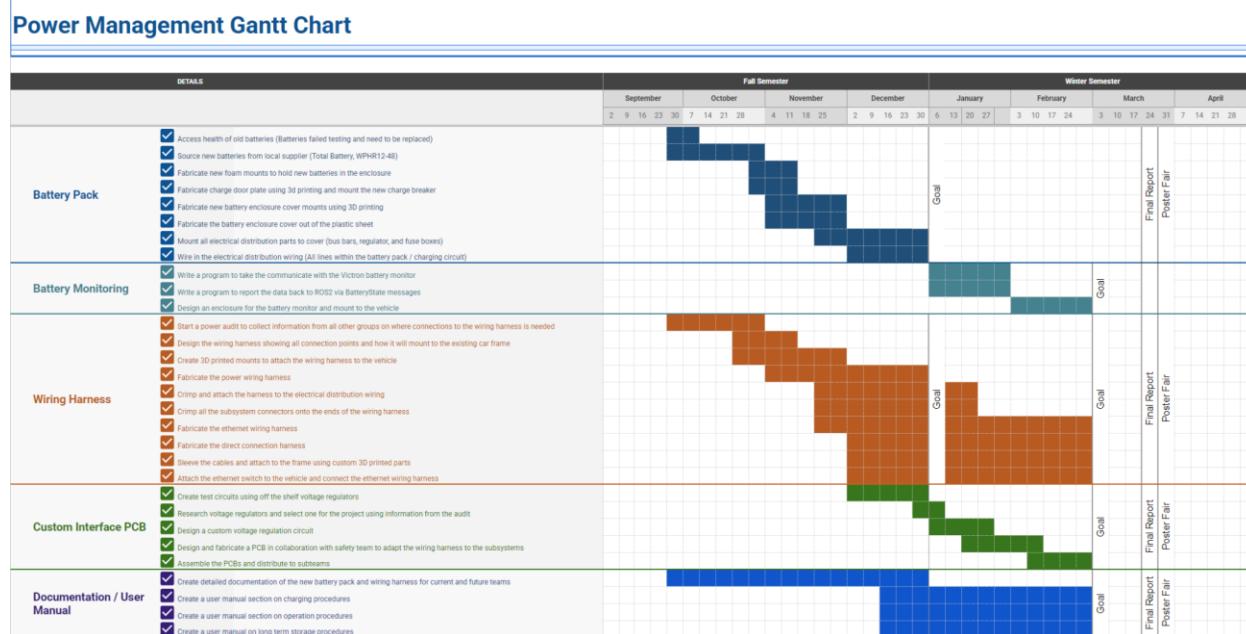


Figure 23: The power management subteam gantt chart

Battery Pack

Goal was to have this finished before the winter break. This goal was achieved.

Battery Monitoring

Goal was to have this finished before the poster fair. This goal was achieved.

Wiring Harness

Goal was to have this finished before the winter break. This goal was missed and moved to before the poster fair. This revised goal was achieved.

Custom Interface PCB

Goal was to have this finished before the poster fair. This goal was achieved.

Documentation / User Manual

Goal was to have this finished before the poster fair. This goal was achieved.

6.5.3. Risk Analysis & Mitigation Strategies

Based on the state that the vehicle was left in last year, we observed that the biggest risk for the power side of the vehicle is the charging aspect. This can be split into two categories, the first is how to safely charge batteries and the second is how often to charge the batteries. To start, it is suspected that the group from last year had fried the charger, by trying to operate the vehicle itself, while the batteries were being recharged. This is something we really wanted to avoid happening again as buying a new charger is not something that needs to be done every year and more importantly, it is a massive safety risk to have a piece of equipment blow up on you like that. To mitigate this, we designed a new circuit for the charging sequence that includes a breaker that will flip if too much current is drawn while it is on. This also adds a switch specifically for charging, which will help future users in knowing when it is safe to turn on the vehicle itself and also if there is 60V exposed in the charging port.

Next, the group from previous years also killed the batteries. This is suspected to have happened because the batteries were left at a very low charge for far too long. It is important for the long term health of the battery to keep them close to full charge and not allow them to be discharged too low for too long. In order to prevent this from happening to the new batteries, we have documented it in the user manual for future groups to remember to charge the batteries every so often.

6.5.4. Budget & Cost Analysis

Our budget was determined by how much KEFC funding we were able to get based on our plans and designs. We managed to get approved for everything we needed except for a supplemental crimp kit we failed to be approved for in the winter semester KEFC. We managed to stay on budget and get a full reimbursement from KEFC without using departmental funding.

6.6. Experimental Setup & Testing

6.6.1. Testing Environment & Procedures

There were not a lot of quantitative testing opportunities for the analysis of the power management subteam's contributions but we found two tests that we could see being useful. The two tests are: the pack voltage sag test to test how much the batteries sag under load from the car

being driven in regular use and a test of the 12V regulator's accuracy and stability by measuring the output voltage's mean and standard deviation.

Pack Voltage Sag Test

For testing we monitored and recorded the ROS data coming out of the battery monitoring system while the car was being driven around an empty parking lot. Our main goal was to monitor the pack voltage in relation to the current being drawn. This would allow us to calculate the sag on the battery.

From our research we are hoping to achieve a under 10% sag when maximum load is applied to the pack. A great result would be to keep the sag below 5%.

12V Regulator Test

For testing the performance of the 12V regulator installed on the vehicle we set up the following test. Attached the oscilloscope to the fuse box positive and negative terminals. Turned the car on with all the systems running in idle to simulate the expected use case. This has us drawing around 100W of power from the 12V regulator. We then measure the mean and std deviation of the 12V regulator output using the oscilloscope.

From our research we are hoping to achieve a nominal voltage that is within 1% to 5% of the 12V output with a std. dev. of 0.2% to 2% of the nominal voltage measured.

6.6.2. Experimental Results and Findings

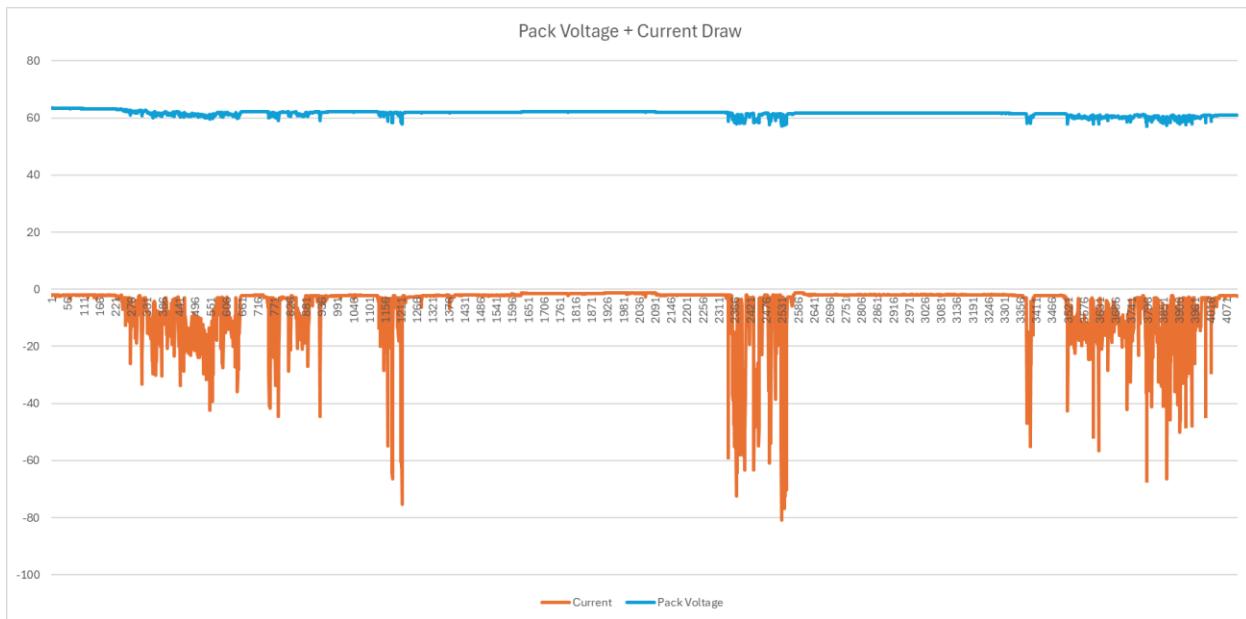


Figure 24: Pack Voltage Sag Test

The graph above shows the measurements from the three hour test. In blue is the pack voltage and in orange is the current being drawn from the pack. The large negative spikes on the orange line indicate when the vehicle is in motion and drawing its most power. The corresponding dips in the blue line are the voltage sag on the pack that we want to measure.

Looking at the section in the middle between approximately 2200 and 2600 I gathered the following information.

The maximum pack voltage (The voltage when current draw is low)

61.91V

The minimum pack voltage (The voltage when current draw is high)

57.13V

12V Regulator Test

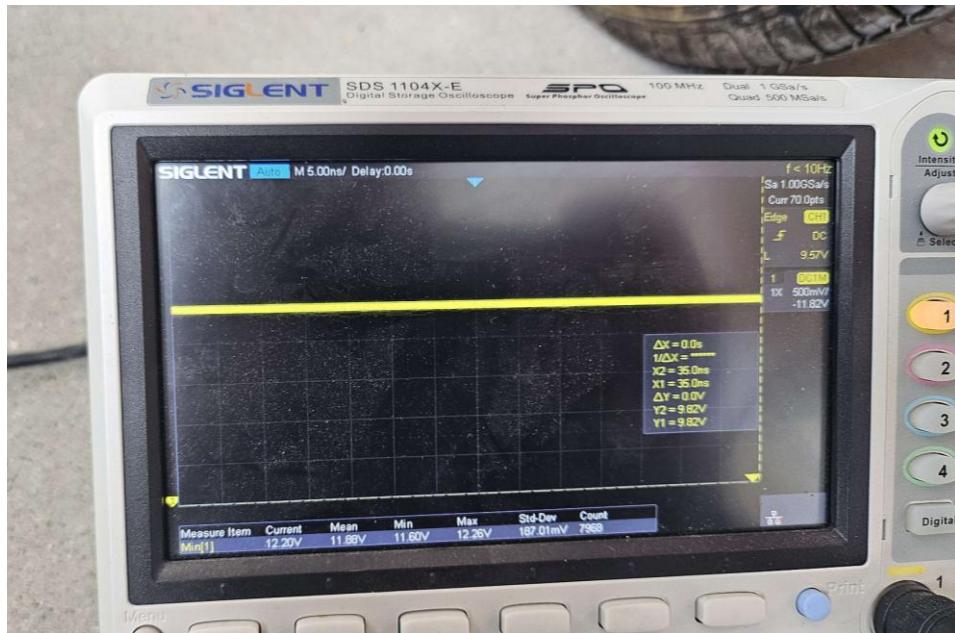


Figure 25: Oscilloscope reading at 12V regulator

The output above is taken from the oscilloscope measuring the output of the 12V regulator at the fuse box. At the bottom of the screen we are measuring the Mean/Min/Max/Std-Dev of the voltage level.

From this we can use the Mean Voltage and Std-Dev to compare to our expected values.

12 Regulator Output: $\mu = 11.88V$, $\sigma=0.18701V$

6.7. Discussion & Analysis

6.7.1. Interpretation of Test Results

Pack Voltage Sag Test

The delta between the maximum and minimum pack voltage will result in our absolute worst voltage sag over the analyzed area.

$$61.91V - 57.13V = 4.78V \text{ sag}$$
$$4.78V / 61.91V = 7.72\% \text{ voltage sag}$$

12V Regulator Test

The accuracy of the 12V regulator can be determined by taking the delta of our mean over the target voltage of 12V

$$(12V - 11.88V) / 12V = 1\% \text{ accuracy}$$

The standard deviation can be calculated as a percentage of the mean voltage measured using the equation below

$$(0.18701V / 11.88V) = 1.57\% \text{ std.dev vs mean}$$

6.7.2. Comparison with Expected outcomes

Pack Voltage Sag Test

This falls right into between an expected voltage sag of up to 10% and a great result of 5%. Therefore the system is operating properly and the motor's current draw is handled well by the new battery pack.

12V Regulator Test

The values of 1% accuracy on the voltage with a std.dev of approximately 1.6% falls right into the ranges we were expecting.

6.8. Conclusion & Future work

6.8.1. Summary of Key Achievements

Battery Pack:

The new battery pack is assembled and working. The cover with all the distribution circuitry is mounted in the vehicle. This allows the vehicle to be tested and driven under its own power completely untethered.

Battery Monitoring

The battery monitoring system's connection to ROS over the raspberry pi is reliable and consistent. The information from the Victron BMV is received, processed, and published to ROS in a timely manner. The system is able to monitor the battery's pack voltage, current draw, and the 12V regulator's output and display them in ROS for other systems on the vehicle to use.

Wiring Harness

The wiring harness provides a reliable well protected way for subsystems to receive 12V power, ethernet connection, and emergency stop connection. It has individual fused circuits for each subsystem. Allows a full 1Gb connection between all the vehicles computers. And allows any subsystem to read or trigger an emergency stop physically bypassing the ROS network.

Interface PCB

The interface PCB allows the power and emergency stop harness to integrate with a subsystem. It provides 12V, regulated 5V, and 3.3V as well as the circuitry to allow the emergency stop signals to be interfaced with a microcontroller. The interface PCB panel allows an enclosure to be easily designed for the interface board without having to worry about the tight tolerances and complex shapes of the connectors and PCB components.

6.8.2. Lessons Learned

The main lesson learned for this project is the importance of integration testing. For the power management aspect of the vehicle, it is not really possible to test the system unless there are actual components to supply with power. The earlier and more frequently that other subgroups connect their work to the power system, the more confidence the power system can be tested with. This would have also helped the other subgroups see how their systems performed under actual in-vehicle conditions, rather than a perfectly controlled environment outside of the vehicle.

6.8.3. Recommendations for Future Development

Battery Pack:

While the battery pack is functional lots of improvements could be made to improve it. The primary improvement that me and the supervisor recommend is the addition of per-cell voltage monitoring and pack balancing. While this is not necessarily required for lead acid batteries it would allow for far better monitoring, charging, and would ensure the batteries longevity.

Battery Monitoring

In combination with the switch to a different battery monitoring system would be beneficial as the BMV712 seems to be inaccurate in displaying current and dangerously designed for battery state of charge monitoring without an external battery protection circuit. A smaller shunt to allow for better accuracy at currents experienced by the vehicle (roughly 0-50A) would help improve the accuracy compared to the 500A range of the BMV.

Wiring Harness

The wiring harness is good. It can be further expanded with new fused circuits or more ethernet / emergency stop connections easily.

Possibly move to a more advanced switch with more ports and better features for routing to allow for more complicated subnet setups. Currently we have all the ports on the 8 port switch saturated when the vehicle is fully connected.

Interface PCB

From the perspective of the power management team the interface pcb and its interface panel is good. More of them can be made and printed for expansion to new subsystems. See the safety subteam for more recommendations.

6.9. Reflections

6.9.1. Original Project Proposal & Changes Over Time

While the original project proposal is fairly true to what has been achieved there are a few minor changes that were decided upon throughout the year. This mainly has to do with the battery monitoring system. As this was scheduled for the later half of the year not much research had been done into how specifically it would be interfaced with ROS. The original assumption was that the off the shelf battery monitor from Victron was a closed system and we would have to design our own parallel battery monitor which would function independently to report battery status to the rest of the vehicle. This assumption was false however, upon further research we discovered that the Victron unit could communicate with a computer over serial using the VEDirect port on the back (with a proprietary cable of course). This allowed us to vastly simplify the plan as we could simply interface the existing battery monitor with a raspberry pi to relay the battery information from the serial connection to the rest of the vehicle.

For the wiring harness we had to extend its time frame from the original proposal document as the construction and installation took far longer than was originally expected.

6.9.2. Team Reflection on Challenges & Successes

As a whole, this subgroup was a massive success. We managed to create a fully functional power system from scratch. New batteries were connected to make a 60V system and further regulated along with fuse boxes to create a stable 12V line. This allowed every other component in the vehicle to be connected to one standardized power line and still function as expected. A new charger was purchased with improved circuitry to help aid future safety with both the users and the battery itself. The wiring harness is professional looking and also functionally durable. It also allows for future upgrades with its modular design. Finally, the battery monitoring with its bluetooth capabilities allows the system to be easily surveilled from a smartphone.

The biggest challenge is documentation. We quickly learned the importance of documenting as you go along. Leaving it all to the end would have caused us to forget details and with the large amount of power we are dealing with, every detail in the documentation can be crucial.

6.9.3. Hindsight Analysis: What would we do Differently?

In one word, more. Throughout the process we sought to reuse components from previous years in order to reduce the cost of the overall project. This, in hindsight was foolish as if we are ripping everything out we should have gone further in what we were replacing. If we had done more research into the BMV's functionality and capabilities we would have discovered it's inadequately for our purpose sooner. That would allow us to purchase an alternative and replace it. Some other idea's came up throughout the year that might have changed our choice of implementation for the vehicle and could be considered in future years. PoE for powering subsystems to reduce cabling? Switching from ethernet to automotive ethernet or ditching the ethernet altogether and moving to CAN?

7. Navigation

7.1. Introduction

7.1.1. Background & Motivation

The goal of this project was to convert an Ecolo-cycle ET4-cruise spark, a 60V 1200W high-torque electric vehicle, into a fully autonomous driving system. This project remains a part of a multi-year initiative that began under the Aero and Mechanical Engineering departments and is now managed by the Systems and Computers department. This project builds on previous work but introduces the navigation system as a new module that was not part of the original scope.

The vehicle was completely stripped of its original components, and new systems were introduced and integrated from scratch. The project for the year 2024 - 2025 was divided into six submodules: Navigation, Control Systems, Simulation, Power Management, Safety and Reliability, and Perception, each handled by separate teams.

Our team focused on the **Navigation module**, which aimed to implement route tracking, mapping, and user interface functions. The module aimed to implement the following items:

- **Route Plotting:** Generate a route and forward waypoints and directions to the Control Systems team for execution.
- **Dynamic Map Display:** Provide a real-time map on the user dashboard, showing current location, destination, and upcoming actions such as turns. This map interface shall also include useful information such as speed, battery charge level, weather, and a shortcut to view real-time diagnostic data on a separate web app/website.
- **Obstacle Detection and Avoidance:** Ensure safe navigation by identifying and avoiding obstacles.

While these objectives defined the intended scope of the Navigation module, the project encountered practical constraints typical in multidisciplinary time-limited capstone environments. Nonetheless, the team successfully made foundational progress across all three pillars:

- **For route plotting**, we configured and integrated GPS data acquisition into our system, laying the groundwork for real-time localization. Waypoint management and communication with the ROS2 ecosystem were partially implemented through simulated environments.
- **In the UI domain**, a core framework was developed using React, capable of displaying a map centered on the vehicle's current location. Essential features such as search functionality, recent location memory, and integration with diagnostic data endpoints were prototyped. The UI was hosted on a Raspberry Pi and designed for real-time updates.
- **For obstacle detection and avoidance**, we successfully implemented a **local costmap generation system** using LiDAR data and ROS2's navigation stack. This allowed the vehicle to dynamically update its understanding of surrounding obstacles, a critical step toward autonomous path replanning and safe maneuvering.

While full end-to-end integration and advanced features such as live waypoint forwarding, route instructions, and complete obstacle avoidance behavior were not achieved within the project timeline, the technical infrastructure has been solidly established. Future teams can build on this work with confidence, as the core architectural components, sensor integrations, and user interface systems are now in place and tested in simulated environments.

a. Background:

The development of autonomous vehicle navigation systems has evolved significantly since the early 1980s. Initial research was led by institutions like Carnegie Mellon University (CMU) and the University of the Bundeswehr Munich, with a focus on simpler tasks such as highway driving, lane detection, and adaptive cruise control [NAV1]. These projects laid the foundation for more advanced autonomous systems by exploring how vehicles could detect their environment and react to basic driving scenarios.

A pivotal moment in the development of autonomous vehicles came with the DARPA Grand Challenge (2004-2005) and the Urban Challenge (2007), which pushed autonomous ground vehicle (AGV) technology to new levels [NAV1]. These challenges forced autonomous vehicles to move beyond controlled environments like highways and into more complex, dynamic urban settings, where vehicles had to navigate unpredictable traffic, pedestrians, and obstacles.

Autonomous vehicle navigation today relies on a variety of sensors and technologies to provide accurate and reliable environmental perception. Lidar, radar, cameras, and ultrasonic sensors are commonly used to detect objects, track their movement, and help the vehicle navigate safely through its surroundings. For precise and up-to-date route information, HD maps provide highly detailed data on road geometry, lane markings, and traffic signs [NAV2]. These maps, in combination with sophisticated path-planning algorithms, allow autonomous vehicles to safely determine optimal routes and adapt to changing environments.

An important component of modern autonomous navigation is visual odometry (VO), which estimates the vehicle's position and orientation using sequential images. VO is especially useful when traditional positioning systems, such as GPS or wheel odometry, are either unavailable or unreliable, such as in urban environments where GNSS signals can suffer from blockage or multipath effects [NAV3]. To help mitigate these challenges, Inertial Navigation Systems (INS) and LiDAR are used in combination to provide accurate short-term navigation and detailed environmental maps.

For obstacle detection and avoidance, vision-based approaches are the most popular due to their cost-effectiveness and their ability to analyze the environment's appearance and shape. However, challenges such as illumination variability (The amount of light cast on an object.) and environmental factors complicate object detection [NAV4]. Recent advancements in machine vision and deep learning have enhanced autonomous vehicles' abilities to navigate complex scenarios, making these systems more robust in detecting and avoiding obstacles [NAV5].

This project aims to take inspiration from these advanced navigation and obstacle detection technologies and implement a solution into the existing electric vehicle platform. The addition of the **navigation module** will address a critical gap in the autonomous vehicle's ability to safely and reliably navigate urban environments, forwarding real-time route information and obstacle data to the vehicle's control systems.

b. Current State of the Art:

Recent advancements in autonomous vehicle navigation have focused on improving sensor fusion, path planning, and obstacle detection, utilizing bleeding-edge algorithms and hardware solutions. These developments aim to increase the reliability and efficiency of autonomous vehicles (AVs) in complex environments.

Note: For clarity, the current state of the art is spread across three sections:

Autonomous Vehicles Navigation Systems:

One of many significant innovations is the integration of pulsed LED technology with machine vision algorithms (**Patent US 20180364730A1**). This approach enhances object detection by combining infrared illumination¹ and frequency-based object recognition², offering a redundant safety mechanism in low-light conditions [**NAV6**]. This development addresses challenges like poor visibility and improves overall detection reliability for Autonomous Vehicles in complex driving environments.

Another notable trend is the use of sparse mapping (**Patent EP 4180768 A1**), a lightweight alternative to traditional high-density maps [**NAV7**]. This method enables efficient data storage and processing, making it ideal for large-scale deployment in autonomous fleets. The integration of AI-driven mapping solutions and sensor fusion allows AVs to aggregate data from multiple sources, such as cameras, LiDAR, radar, and GPS. Crowdsourcing techniques for real-time map updates have also become more prevalent, enabling AVs to continuously adapt to changing road conditions.

Lidar-based SLAM (Simultaneous Localization and Mapping) is another bleeding-edge advancement. It is particularly useful in environments with poor GPS connection, where Autonomous Vehicles rely on Lidar sensors to create high-definition maps. This method utilizes Lidar odometry, which estimates the vehicle's movement by comparing consecutive Lidar frames. It allows Autonomous Vehicles to ground themselves into the local environment in real-time by aligning Lidar scans with prebuilt 3D maps, making it highly effective for autonomous parking and indoor navigation [**NAV2**].

To enhance precision in navigation, modern systems integrate GNSS³, LiDAR, and INS⁴ data to help mitigate signal blockages in urban environments. This approach, which uses the strengths of these technologies in combination, provides accurate and continuous navigation even in challenging conditions [**NAV8**].

Route Mapping:

Advanced pathfinding algorithms such as A* and RRT are central to modern AV navigation [**NAV9**]. The A* algorithm employs methods to reduce unnecessary exploration during route mapping, ensuring the shortest and most efficient path is chosen. RRT (Rapidly-exploring Random Trees), known for handling high-dimensional spaces and complex constraints, is frequently used

¹ **pulsed LED illumination** emitted from objects like traffic signs and vehicles, allowing the vehicle to detect objects and decode their identity based on frequency, intensity, and wavelength of the pulsed illumination.

² The technique of identifying objects by analyzing the frequency patterns of signals, such as infrared or other electromagnetic waves, reflected off the objects.

³ Global Navigation Satellite System

⁴ Inertial Navigation System

in path planning. These algorithms, however, have limitations such as path discontinuity and suboptimal routes.

An innovation in adaptive route planning (**Patent US 11,022,449 B2**) combines real-time environmental data with historical performance records and simulated vehicle data, enabling Autonomous Vehicles to find the safest and most efficient route based on prior experiences [10]. This aligns with trends in machine learning and predictive analytics, where past performance affects future decisions.

Recent methods, such as Perception-Based Local Route (PBLR - It uses vehicle sensors like cameras, radars, and lidar to detect road lanes, boundaries, and obstacles.) and Map-Based Local Route (MBLR - It relies on pre-built, highly detailed maps with centimeter-level accuracy combined with precise vehicle localization to map a route), provide flexible route generation depending on environmental complexity. The Hybrid Local Route (HLR) approach intelligently switches between PBLR and MBLR based on real-time conditions, optimizing costs and reliability in both structured and complex environments [NAV11].

Obstacle Detection:

Obstacle detection systems have seen major advancements with the use of deep learning and sensor fusion technologies. Probabilistic occupancy maps, digital elevation maps (DEM), and geometry-based clustering are widely used to detect and classify obstacles in real-time. Modern systems such as Faster R-CNN (Region-based Convolutional Neural Network) enable the accurate classification of dynamic obstacles like vehicles, pedestrians, and animals [NAV4].

Additionally, innovations in Markov Random Field (MRF - A set of random variables having a Markov property⁵ described by an undirected graph) models, combined with deep neural networks, have improved the segmentation of obstacles, enhancing AV safety and navigation in dynamic environments [NAV5].

A key development in this field is the fusion of sensor data from LiDAR, cameras, and radar with advanced path-planning algorithms. This allows Autonomous Vehicles to make rapid decisions, adjusting for both stationary and moving obstacles in real-time (**Patent US 11,796,673**). The system's focus on adaptive control and mode-switching reflects best practices in modern autonomous navigation, ensuring flexibility and responsiveness in challenging environments [NAV12].

⁵ Markov Property refers to the memoryless property of a stochastic process, which means that its future evolution is independent of its history.

c. Motivation:

Advancing autonomous navigation systems plays an important role in improving safety, efficiency, and reliability across several industries. In precision farming, for example, autonomous tractors may optimize resource usage, reduce costs, and enhance sustainability by guiding machines with high precision to eliminate wasteful movements, improving agricultural efficiency [NAV13].

Similarly, in military applications, unmanned ground vehicles (UGVs) rely on autonomous navigation to operate safely in dangerous terrains, reducing human risk in defense and disaster relief [NAV13].

Visual odometry is widely used not only in vehicles but also in space exploration (e.g., NASA's Mars rovers), underwater vehicles, and robotics, allowing machines to navigate challenging environments autonomously [NAV3]. The rapid development of route planning algorithms has drastically improved navigation performance, making these systems faster and more reliable in real-time applications such as transportation and logistics [NAV14].

In urban settings, autonomous vehicles have the potential to reduce traffic, lower emissions, and enhance road safety by optimizing routes and reacting to complex dynamic traffic conditions. Additionally, these systems contribute to environmental sustainability by minimizing fuel consumption, reducing greenhouse gas emissions, and enabling more efficient, green transportation.

7.1.2. Problem Statement

The original Ecolo-cycle ET4-cruise spark electric vehicle lacked any form of autonomous navigation, rendering it incapable of performing route planning, localization, or dynamic obstacle avoidance. For the vehicle to function autonomously in real-world environments, it must be able to determine its position, map its surroundings, plan a path to a destination, and adjust its course in response to environmental changes. These capabilities are essential not only for vehicle autonomy but also for ensuring passenger safety, energy efficiency, and functional collaboration with other vehicle subsystems such as control and perception.

The problem faced by the Navigation submodule team was the design and implementation of a system that can:

1. Accurately track the vehicle's position using real-time sensor data.
2. Compute an optimal route to a user-defined destination using path-planning algorithms.
3. Detect and respond to obstacles in dynamic environments.

This needed to be achieved under resource constraints, within a modular architecture compatible with ROS2, and deployed on embedded hardware such as the Raspberry Pi. The absence of these capabilities in the existing platform represented a critical gap in enabling full vehicle autonomy.

7.1.3. Objective & Scope

The primary objective of this project was to design and implement a robust navigation system for a retrofitted autonomous electric vehicle capable of dynamically planning and re-planning routes through various environments. This system is intended to guide the vehicle from a user-defined starting point to a desired destination while responding to real-time environmental factors such as traffic conditions, road obstructions, and surface variations. The navigation system must enable the vehicle to make rapid, autonomous decisions that meet high standards for safety, reliability, and performance.

A critical aspect of this system is the integration of real-time data from onboard sensors—including LiDAR, cameras, GPS, and radar—to enhance environmental awareness. Through sensor fusion and intelligent processing, the vehicle should be able to preemptively detect and avoid hazards, re-route in response to unexpected obstacles or closures, and maintain optimal path planning even in uncertain conditions. The final system must demonstrate its reliability and performance both in simulated testbeds and real-world driving scenarios.

To achieve this overarching goal, the Navigation submodule was assigned four key deliverables:

1. Obstacle Detection and Avoidance System:

- Integrating multiple sensor inputs (e.g., LiDAR, cameras) using ROS2 to enable real-time object detection and environmental mapping.
- Supplies dynamic obstacle data to support safe path planning and responsive maneuvering.

2. Route Mapping and Waypoint Generation:

- Utilizing OpenStreetMaps (OSM) data, converted to binary occupancy grids, as the foundation for route generation.
- Employing the use of the A* pathfinding algorithm through the ROS2 Nav2 stack to generate optimized waypoint sequences.
- Publishing waypoints to other ROS2 nodes to support collaborative navigation execution across subsystems.

3. User Interface (UI):

- Development of a front-end dashboard hosted on a Raspberry Pi, providing a map-based visualization of the current vehicle location, active route, and upcoming navigational instructions.
- Implementing user input functionality for destination selection and integrating diagnostic telemetry such as brake pressure, steering angle, and emergency stop status in coordination with the Safety and Reliability team.

4. ROS2 Node Integration:

- Construction of a ROS2-compatible navigation node responsible for publishing and subscribing to telemetry data, including position, heading, and waypoint updates.
- Ensuring seamless communication with other submodules such as Perception and Autonomous Control System, for synchronized vehicle behavior.

Together, these objectives were designed to enable the autonomous vehicle to navigate safely, efficiently, and independently in dynamic, real-world conditions. The Navigation module serves as a critical bridge between high-level planning and low-level control, forming the backbone of the vehicle's autonomous behavior.

7.1.4. Methodology Overview

Our approach to solving the navigation challenges and implementing a functional navigation system for the autonomous vehicle involved using a range of methods that ensure the system operates effectively in dynamic and unpredictable environments. A combination of advanced algorithms, real-time data integration, pre-existing software frameworks, packages, libraries, and system-wide communication is representative of what was used to implement the navigation system. These methods are all rooted in the knowledge we have acquired through our degree program, in addition to finding and learning new technologies on our own.

- **PathFinding and Optimization:** To enable the vehicle to autonomously navigate from a starting point to a user-defined destination, we attempted route planning using the A* algorithm —an established pathfinding method based on graph traversal and heuristic optimization. This algorithm comes integrated as a part of the ROS2 Nav2 package to generate efficient and realistic navigation paths while accounting for static obstacles. The implementation drew from concepts covered in our Data Structures and Algorithms coursework, particularly related to graph theory, cost functions, and priority queues.

- **Real-Time Data Processing:** To ensure adaptability in dynamic environments, we attempted to integrate real-time sensor data streams from the Perception team, including GPS, LiDAR, and camera inputs. This helps the navigation system in continuously adjusting its route based on changing conditions such as obstacle appearance or GPS drift. Using ROS2 topics and subscriptions, we processed incoming data and dynamically updated the local costmap, which is essential for safe maneuvering. This method leveraged principles from our Real-Time Systems coursework, including concurrency, low-latency data handling, and system responsiveness.
- **Dynamic mapping:** As the vehicle requires a global map for its long-distance route planning, we ended up utilizing an open-sourced database called **OpenStreetMap (OSM)** for our geographic data. **Overpass API** was another tool that we had to use to process and simplify these large datasets to only include relevant information. After getting the OSM files, we had to convert these files into **data occupancy gridmaps**, which would allow them to be utilized by our Nav2 stack and ROS2 pipeline. Converting these OSM files requires using tools such as **QGIS** to visualize and prepare these maps for ROS2. This is in turn improved our understanding of **Geographic Information Systems (GIS)** and data processing software such as **QGIS**, and also enables the dynamic route planning for our vehicle.
- **Modular System Design and Integration:** The navigation system was built to function as a modular component within the larger autonomous vehicle architecture. This required consistent communication and data exchange with other submodules, such as Perception (for sensor data). We adhered to Software Engineering principles including modularity, scalability, and separation of concerns, and implemented inter-module communication via ROS2 nodes and topics. This approach allowed each subsystem to be tested independently while ensuring smooth integration during full-system operation.
- **User Interaction and Interface:** An intuitive User Interface (UI) was designed to bridge the user and the vehicle. Built using React and hosted on a Raspberry Pi 4B, the UI displayed a dynamically updating map, current vehicle location, and route. The back-end was developed using Flask and configured as a ROS2 node, allowing it to fetch and relay telemetry data such as GPS location, heading, and diagnostic information. Although our background in UI/UX design was limited, we studied modern Full-Stack development practices to create a responsive, accessible, and informative dashboard.
- **Collaborative and Iterative Problem-Solving:** Throughout the project, we followed an iterative problem-solving approach by regularly testing and refining each component of the system. This approach ensured that we can identify and resolve issues early in the

development cycle.

This closely aligns with the Agile Development Methodologies commonly used in the technology industry during our co-op internships, where collaboration, feedback, and continuous improvement are key to success.

7.1.5. Report Overview

This report documents our journey developing the Navigation module for an autonomous electric vehicle as part of our capstone project. It covers our objectives, key design decisions (including some painful pivots), implementation approaches, testing methods, and what we ultimately achieved.

We have organized the report into these main sections:

- **Engineering Requirements and Justification:** This connects our work to the Computer Systems Engineering program. We explain the engineering principles we applied and wrestle with the health, safety, and ethical questions asked throughout the term of this project's development.
- **Literature Review and Related Work:** It dives into the existing tech landscape. A good portion of our time was pored over research papers, industry implementations, online tutorials — some brilliant, others less helpful than advertised. This section compares our approach with what's come before.
- **System Design and Implementation:** This section breaks down our system architecture and the hardware/software design decisions. This is where we share the real technical hurdles. You'll see our testing approaches and how we adapted when things didn't go as planned.
- **Project Management and Execution** covers how we operated as a team. From our initial optimistic timeline to the reality of missed deadlines and late nights, we document our planning process, milestone tracking, and how we managed risks. The budget section shows where we had to get creative when parts costs exceeded our estimates.
- **Experimental Setup and Testing:** This section details our makeshift testing environment, our procedures, and the results—both encouraging and disappointing.

- **Discussion and Analysis:** This section is our honest assessment of where we succeeded and where we fell short.
- **Conclusion and Future Work:** It summarizes what we accomplished and what we'd do differently if starting over. We've included specific recommendations for next year's team who might continue this work.
- **Reflections:** This is perhaps the most personal section. Here, we share how our vision evolved, the moments of frustration and breakthrough, and the invaluable lessons that weren't in any textbook.

This report tells the story of our navigation module from ‘on-paper’ to a working prototype.

7.2. Engineering Requirements & Justification

7.2.1. Justification for Relevance to Degree Program

Our Navigation subgroup comprises students from Computer Systems Engineering and Software Engineering programs, creating an ideal alignment between our academic backgrounds and the technical requirements of this module.

For those of us in Computer Systems Engineering, this project has provided valuable opportunities to apply theoretical concepts from our coursework. We learned about different types of route selection algorithms, out of which the A* algorithm was selected in particular for route planning. The real-time computing aspects have proven particularly demanding, as we've needed to process and respond to data streams from LIDAR, GPS, and GNSS sensors with minimal latency, drawing directly on principles learned in our embedded systems coursework.

The Software Engineering team members have focused their expertise on system architecture and integration challenges. ROS2 implementation has formed the backbone of our communication framework, requiring careful consideration of software architecture principles to ensure reliable data transfer between the vehicle's perception and control modules. This work has reinforced and extended concepts covered in our systems design and networking courses.

The user interface development, while initially underestimated in complexity, evolved into a significant component requiring full-stack development approaches. This aspect of the project challenged both Computer and Software Engineering students to apply human-computer interaction principles while maintaining system performance requirements.

Beyond technical skills, this interdisciplinary project has highlighted the practical importance of effective teamwork and project management, particularly during integration phases when competing technical priorities needed resolution under tight deadlines. These experiences have

complemented our formal education by providing contextual understanding of engineering practices that will prove valuable in our professional careers.

7.2.2. Engineering Principles Applied

- **Modular Design and System Integration:**

In Theory: The navigation module was developed as a standalone ROS2 node, interfacing with Perception and Control modules.

In Practice: During deployment, we distributed ROS2 nodes across both Jetson and Pi platforms, implementing automatic startup scripts to ensure integration remained seamless when the vehicle powers up.

- **Use of Established Algorithms and Theoretical Foundations:**

In Theory: We selected A* and EKF algorithms for path planning and localization based on their proven reliability.

In Practice: A* implementation is progressing through Nav2 integration, while our GNSS data is successfully visualized in our real-time map interface. EKF integration with sensor fusion remains scheduled for implementation once we resolve current hardware stability issues.

- **Real-Time Systems and Responsiveness:**

In Theory: Our system architecture was designed to ingest real-time sensor data and respond accordingly.

In Practice: We've achieved real-time GNSS data processing and map display. Our Flask server reliably initializes at boot and pushes updates automatically, demonstrating the responsive behavior required.

- **User-Centered Design**

In Theory: We built the dashboard using React and Leaflet to maximize clarity and responsiveness.

In Practice: Our UI is now operational on the vehicle's touchscreen, allowing users to input or select destinations with real-time location tracking.

- **Prototyping and Iterative Development:**

In Theory: Initial testing and iteration relied on Gazebo and RViz simulation environments.

In Practice: Our transition to real hardware has been partially successful, with functional GNSS and UI components now operational, while SLAM integration and A* implementation remain as future implementation goals.

- **Sensor Fusion and Localization:**

In Theory: We designed sensor fusion through EKF to combine data from GNSS, LiDAR, and IMU sources.

In Practice: The GNSS subsystem is fully functional, and LiDAR-based costmap generation operates as expected. EKF integration shall be handled by the next year cohort.

- **Data Management and Abstraction:**

In Theory: Our approach processes OSM data using osmnx to generate binary occupancy grids suitable for navigation.

In Practice: These grid maps have been successfully visualized in RViz and are ready for integration with our route generation system.

- **Software Engineering Best Practices**

In Theory: We selected Flask over Django specifically for its lightweight footprint and compatibility with ROS2.

In Practice: Our Flask-based servers now run reliably on the Pi, successfully bridging the ROS2 node and frontend dashboard through APIs, handling GNSS location and search functionality.

7.2.3. Health and Safety Consideration

(Common set of precautions and guidelines across all 6 subteams)

Safety protocols were followed throughout development, especially when working with high-current batteries and moving vehicles.

Key measures included:

- Testing in isolated environments, such as empty parking lots, to minimize risk to people and property.
- Strict adherence to lab safety procedures when working with electrical components.
- Clear communication and designated safety roles during vehicle testing.
- Emergency stop systems were tested and made accessible to all team members.

Outside of these general guidelines, since Navigation development is mostly software-based, there is not much that can harm a person's safety.

7.2.4. Ethical Considerations

When developing the navigation module, privacy protection and transparency are two of the biggest ethical considerations. Since the system involves user locations and travel routes, we've laid out ethical guidelines to ensure that our development stays on the right track as the system evolves.

Protecting user privacy is a top priority for us. That said, we're still early in the development cycle, so most of the safeguards we've envisioned are not fully in place yet. For example, we hope for the future cohorts to adopt a data minimization approach where location data is only

stored temporarily during active navigation and automatically deleted afterward—unless, of course, the user chooses to save it.

To further enhance privacy, we aim to use local encrypted storage within the vehicle itself rather than cloud-based solutions. These features are part of our vision and will be implemented as the Autonomous vehicles project progresses in the coming years.

Transparency is another core value we’re focusing on. We wish for the realization of user-friendly features like notifications to let users know when location tracking is active, along with visual indicators showing what data is being collected. Currently, all the necessary information is displayed on the system terminal, which is not the best experience outside of development. These tools are designed to empower users and ensure they remain in control, though they’re not yet part of the current version of the system.

Looking ahead, we also foresee integration with traffic networks or other vehicles. This will bring new challenges like data sharing and anonymization, which we aim to address with modular privacy controls that can adapt as needs evolve.

While many of these measures are still in the works, this ethical framework lays the foundation for future development. By prioritizing both privacy and functionality from the start, we were committed to building a navigation system that aligns with academic standards, industry best practices, and most importantly, user trust.

7.2.5. Regulatory & Standards Compliance

We made a conscious effort to keep ethics and regulations in mind throughout the development of the navigation module and web app. On the software side, we followed **GDPR**⁶ [NAV15] principles to respect user privacy and aimed for **WCAG**⁷ accessibility so the interface could be used by as many people as possible.

For the ROS2 navigation system, we followed **REP-105**⁸ [NAV16] to keep our coordinate frames consistent, and we used a modular design so that the system stays safe, flexible, and easier to manage. We also tried to follow industry best practices wherever possible, especially around safety and documentation.

⁶ **GDPR (General Data Protection Regulation):** A European Union regulation focused on protecting personal data and privacy. It mandates organizations to obtain user consent for data collection, provide transparency, and allow users to access, modify, or delete their data.

⁷ **WCAG (Web Content Accessibility Guidelines):** International standards for web accessibility, ensuring websites are usable by individuals with disabilities. It includes principles like perceivable, operable, understandable, and robust design.

⁸ **REP-105 (Coordinate Frames for Mobile Platforms):** A ROS standard defining naming conventions and semantic meanings for coordinate frames in mobile platforms. It ensures consistency in localization and navigation systems.

Beyond that, we considered things like protecting intellectual property, building privacy into the system from the start, and making sure we tested thoroughly. While some features like anonymized data handling and secure communication for future connected systems haven't been added yet, they're on our radar. Overall, we tried to build responsibly and with a focus on transparency and user safety.

7.3. Literature Review & Related work

7.3.1. Overview of Existing Technologies & Techniques

Aside from the already discussed technologies above in the introduction section, let's do a deeper dive on the technologies used and their influence on the project.

1. Cameras and URDF Modeling:

In autonomous systems, cameras play a central role in perception, providing critical data for tasks such as object detection, visual odometry, and SLAM. While high-end autonomous vehicles rely on stereo vision and AI-powered object recognition, our system focused on building foundational functionality using a Raspberry Pi camera. This decision was driven by the need for simplicity, affordability, and ease of integration during early prototyping.

To integrate the camera within the robotic framework, we used **URDF (Unified Robot Description Format)**. URDF allowed us to define the physical structure of the vehicle, including the placement of sensors about the base_link frame. Precise modeling was essential for ensuring correct TF (transform) broadcasts across both simulation and real-world environments. By aligning the simulated setup with the real vehicle, we avoided inconsistencies that could disrupt localization or mapping.

2. Simulation in Gazebo:

Before deploying on hardware, we validated our system in **Gazebo**, a widely used robotics simulator. In this environment, we tested camera field of view, frame alignment, and robot behavior in a controlled, virtual world. Simulating perception and navigation components in Gazebo allowed us to catch configuration errors early, reducing the risk of hardware damage or integration issues during real-world testing.

3. Mapping with OSM and QGIS

For global navigation and environmental awareness, we used **OpenStreetMap (OSM)** as our primary mapping source. OSM's open, editable nature made it well-suited for academic use, offering highly detailed information on roads, intersections, and other infrastructure.

To extract relevant segments of the map, we used the **Overpass API**, which allowed selective querying of features like roads, omitting unnecessary data. The raw data was then processed in **QGIS**, a GIS platform used to clean, simplify, and convert the information into occupancy grid maps compatible with ROS2's navigation stack. These maps were saved in .pgm and .yaml formats, which could be loaded directly into RViz and used for route planning and SLAM.

While earlier versions of the project relied on external routing services like the OpenRouteService API, we shifted towards generating in-house occupancy maps to work directly with Nav2, giving us better control and flexibility in testing.

4. GNSS and Localization:

Localization is fundamental in any autonomous system, and while commercial platforms use dense sensor suites and AI-driven localization, we aimed to achieve reliable positioning using cost-effective hardware and proven software tools.

Our system used the u-blox ZED-F9R GNSS receiver, which includes a built-in IMU. This provided both global position and motion data. Since GNSS signals can suffer from noise, dropouts, or interference, we implemented sensor fusion using the Extended Kalman Filter (EKF) via the `robot_localization` package in ROS2. EKF helped blend the GNSS and IMU data to produce a more accurate and continuous position estimate.

Unlike commercial solutions that rely on automated calibration and expensive toolkits, we manually tuned the EKF parameters, coordinate frame relationships (map, odom, and base_link), and sensor configurations. This hands-on process deepened our understanding of real-time localization challenges and mirrored real-world system integration practices—albeit at a smaller scale. The modular setup also allowed individual components, such as the GNSS node or IMU publisher, to be updated independently if needed.

5. Path Planning with Navigation 2:

For route planning and obstacle avoidance, we utilized the **ROS2 Navigation 2 (Nav2)** stack. Nav2 is a robust, server-based framework that processes the robot's current pose, destination goal, and surrounding costmap to generate an optimal, collision-free path.

We used the **smac hybrid planner**, which implements **Hybrid A***—an ideal choice for Ackermann steering vehicles like ours. This planner was paired with the **costmap_2D plugin**, which used LiDAR data and occupancy grids to dynamically map obstacles in the environment. The stack's plugin-based design allowed for parameter tuning and feature substitution without having to rewrite core components.

6. User Interface (Frontend):

For the user-facing component of the system, we developed a web-based UI using **React**. It was built with responsiveness and modularity in mind, and ran on a Raspberry Pi 4B connected to a 6" touchscreen display mounted in the vehicle.

To render real-time navigation and route information, we used **Leaflet.js**, a lightweight JavaScript library for map visualizations. The map was powered by OpenStreetMap, chosen over alternatives like Google Maps and MapBox due to licensing restrictions and cost considerations. Leaflet supported essential map interactions such as zooming, panning, and layering waypoints or routes, meeting all of our requirements without the need for commercial APIs.

The web interface allowed users to input destinations, visualize the vehicle's current location, and view the planned route, all in real time.

7. Integration with Robotics Systems (Backend):

The backend system was developed using **Flask**, a Python-based micro web framework well-suited for embedded systems and robotics integration. Flask's simplicity and low overhead made it ideal for running on the same Raspberry Pi as the UI, without consuming excessive resources.

We used Flask to expose a set of RESTful APIs, which allowed seamless communication between the frontend and the vehicle's internal ROS2 environment. Real-time data—such as GPS location, heading, and waypoint updates was handled using the REST API's with hopes of transitioning over to WebSockets, avoiding the need for constant polling and reducing latency.

To integrate with ROS2, we used **rclpy**, the official Python client library, enabling the backend to operate as a native ROS2 node. Through this setup, Flask could subscribe to sensor topics or publish navigation goals directly, maintaining tight integration with the rest of the autonomous stack.

Alternative frameworks like FastAPI and Node.js were considered, but Flask offered the best balance of performance, compatibility with ROS2, and ease of development in a Python-based ecosystem.

7.3.2. Comparison with Previous Work

Our system continues building on the foundation of existing work in both academic and industrial settings. High-end autonomous vehicle platforms like Tesla, Waymo, and Cruise employ complex sensor suites like stereo cameras, high-resolution LiDARs, and radars, combined with custom deep learning models and advanced sensor fusion algorithms. These systems rely on features like global shutter sensors, HDR imaging, AI accelerators, and extensive fleet data to enable real-time environmental perception, localization, and decision-making. While they offer state-of-the-art performance, they also come with enormous financial, computational, and data infrastructure requirements.

In contrast, our project was rooted in accessibility, modularity, and educational value. Rather than aiming to replicate commercial autonomous systems, our focus was on building a practical, research-oriented prototype suitable for testing and development in an academic environment.

Perception and Sensor Strategy:

The system operated with one Raspberry Pi camera instead of stereo cameras at an affordable price. The system concentrated its efforts on developing accurate TF frame detection while integrating sensors and publishing data. The system generated fundamental RGB and depth streams, which did not include complete AI-based object detection functionality. The system used modular ROS 2 integration, which enabled straightforward future upgrades, such as replacing Pi cameras with Intel RealSense or ZED cameras. The project established an educational base for URDF modeling and ROS 2 node development, TF tree management, and preliminary depth imaging instead of striving for real-world autonomous driving performance at first.

GNSS and Localization:

While major AV companies invest heavily in multi-sensor fusion systems with sub-meter precision, our GNSS-based localization was built around **affordability and modular design**:

- We used a **u-blox ZED-F9R GNSS receiver** with a built-in IMU, avoiding the need for expensive LiDARs or AI-based localization.
- Sensor fusion was achieved using an **Extended Kalman Filter (EKF)** through the open-source **robot_localization** package in ROS2.
- Unlike commercial systems with automated calibration pipelines, we **manually tuned** EKF parameters, coordinate frames (map, odom, base_link), and transformations to optimize real-world performance.
- Our modular ROS2 setup allowed each localization component—GNSS node, IMU node, and EKF node—to be independently upgraded, swapped, or debugged.

This approach provided us with direct hands-on experience in fusing noisy sensor data, understanding real-time constraints, and refining system configurations—skills highly relevant to industry, albeit developed on a smaller, more accessible platform.

Mapping and Path Planning:

Previous iterations of the capstone project explored routing using services like the OpenRouteService API, which allowed route generation based on real-world roads. In contrast,

our work focused on transforming OpenStreetMap (OSM) data into occupancy grid maps compatible with ROS2's Nav2 stack. We utilized Overpass API and QGIS to extract and preprocess road data, filtering out unnecessary features and converting map tiles into binary grid representations for SLAM and local planning.

While a hybrid system combining global and local maps was initially envisioned, we eventually chose to focus on local LiDAR-based SLAM for practical and time-based reasons. This allowed us to maintain autonomy features like obstacle avoidance, loop closure, and map updating without overextending the project's scope.

User Interface and route plotting:

A major new addition in this iteration of the project was the integration of a touchscreen display running a web-based user interface, a feature not present in prior years. Built using React (frontend) and Flask [NAV18] (backend), the interface follows modern full-stack development practices and was designed to run efficiently on a Raspberry Pi 4B with a 6" touchscreen.

The UI displays the vehicle's live location on a map (via Leaflet.js [NAV17]), allows users to input destinations, and visualizes the computed route in real time using data from the Open street routing machine [NAV19]. Unlike previous versions, which focused solely on backend navigation, this iteration introduces human-machine interaction, making testing, debugging, and user engagement more seamless.

This shift toward a more interactive system not only improves usability but also sets the foundation for future features like diagnostics, remote control, and cloud integration.

7.4. System Design & Implementation

7.4.1. System Overview & Architecture

The navigation module was built as a modular system using ROS2, with each of its subsystems, GNSS/Localization, Camera, User Interface, and Backend Server, designed to handle a specific part of the overall navigation workflow. Together, these parts support key functionality such as real-time location tracking, basic perception, user destination input, and data handling. The architecture emphasizes practical integration between simulation and hardware, making it easier to test and deploy features on the vehicle. ROS2 messaging ensures each subsystem can share information reliably, while a Flask backend running on a Raspberry Pi connects the UI to the rest of the system. This section outlines the structure of each component, how they interact, and the tools used to make the system work, covering everything from camera placement and GPS filtering to how the user interface responds to live data.

Camera sensor:

The Camera Navigation subsystem combined simulated and real-world camera data streams into the ROS 2 framework of the autonomous vehicle to support future perception-based navigation

tasks. The design was structured to ensure modularity, real-time performance, and compatibility with other subsystems (such as Control and Navigation Planning).

The system architecture can be logically divided into two main phases:

- Simulation Phase: Validation of camera placement, TF tree consistency, and image stream correctness using URDF modeling and Gazebo simulation.
- Hardware Phase: Physical integration of Raspberry Pi cameras with the Jetson AGX Orin, live data streaming, and TF tree verification in real-world conditions.

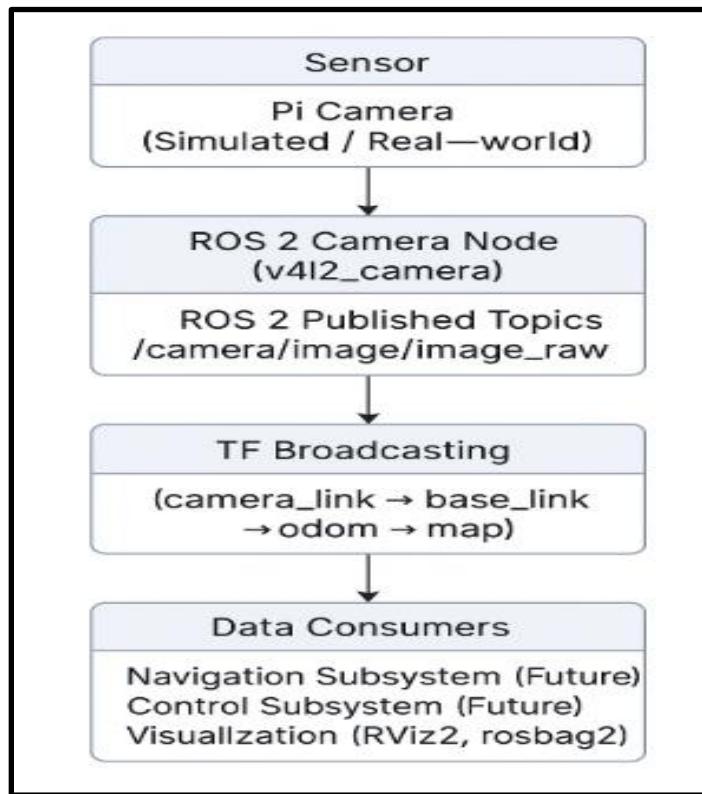


Figure 26: logical Flow of execution for camera stream processing

GNSS/Localization:

The GNSS/Localization subsystem was designed to provide accurate and reliable real-time positioning information to the Navigation and Autonomous Control teams. Our approach followed a modular architecture based on ROS 2 (Humble), where each component operated independently but communicated through standard ROS topics.

At the core of the system, the u-blox ZED-F9R GNSS receiver gathered global positioning data, while its built-in IMU collected acceleration and angular velocity measurements. These two streams of raw data were fed into custom ROS 2 nodes we developed. The GNSS node published

position data using the sensor_msgs/NavSatFix message type, while the IMU node published orientation and acceleration data through sensor_msgs/Imu.

Both data streams were then fused by an Extended Kalman Filter (EKF) node using the robot_localization package. The EKF provided a filtered pose estimate, outputting smoother, more reliable localization even during temporary GNSS signal dropouts. This filtered odometry data was published on the /odometry/filtered topic for use by the Navigation and Control subsystems.

The system maintained a consistent TF tree (map → odom → base_link) to ensure that all components could reference the robot's position and orientation correctly. The entire setup was hosted on the Jetson AGX Orin onboard computer, with remote monitoring capabilities enabled through VPN and SSH.

With the mapping subsystem, it was designed for global navigation so that the vehicle could use it for route planning and localization. This map provides contextual information to the vehicle for long-distance, along with localization inputs through GNSS and IMU, and with local mapping using the LiDAR. The mapping system was intended to work with the localization system, as interfacing the both of them together would allow the vehicle to localize itself within the map using the GNSS data. This would then pair with the navigation stack, which would use the static path to generate the routes and stay within the roads. The visualization of the map was handled through Rviz, which could be used alongside with the LiDAR system. While we had not been able to integrate it into the final system, we believe the development will be valuable for the future navigation teams implementing the hybrid mapping system. The image below shows what the map looks like after the QGIS conversion, keeping only the relevant information.



Figure27: Utilizing overpass api to generate a data occupancy grid map

User Interface (WebAPP):

The navigation application, integrated into a vehicle's system, is designed to enhance the user experience by combining practical navigation functionalities with vehicle diagnostics. This integration is achieved through a sophisticated setup involving a 6-inch touchscreen display, strategically positioned on the vehicle's dashboard, serving as the hub for interaction and information dissemination.

Interface Design and Interaction:

The applications interface is rendered on the touch screen display, which is meant to fit within the vehicle's dashboard, offering drivers a convenient means of accessing navigational data and vehicle telemetrics. This display showcases an interactive map, optimized for responsiveness and low latency, while being user-friendly and intuitive to use even when the vehicle is in motion. The main feature of the interface is the destination search functionality which is located on the central bottom of the screen. The search bar becomes active upon user interaction, prompting the user with a visual keyboard for entering a destination address. It is as straight-forward and seamless as possible, in order to complete its functionality without being a distraction. Upon entering a destination, the application suggests several possible matches baked on the input, allowing the user to select from the displayed results.

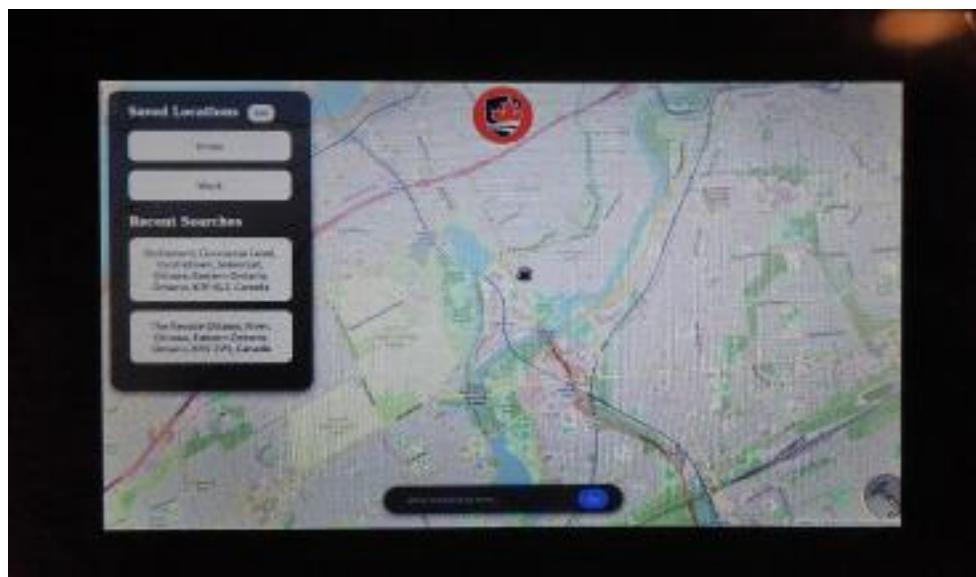


Figure 28: a screenshot showing homescreen of the designed UI

Route Navigation and Waypoint Generation:

Once a destination is selected, the system is designed to calculate an optimal route and set waypoints, although this feature is still under development and has not been fully implemented yet. When operational, this will allow the application to guide the driver towards the selected destination with turn-by-turn directions and live updates. Currently, upon selecting a destination, the application drops a pin at the exact coordinates and recenters the map at the destination, and zooms in to provide a detailed view.

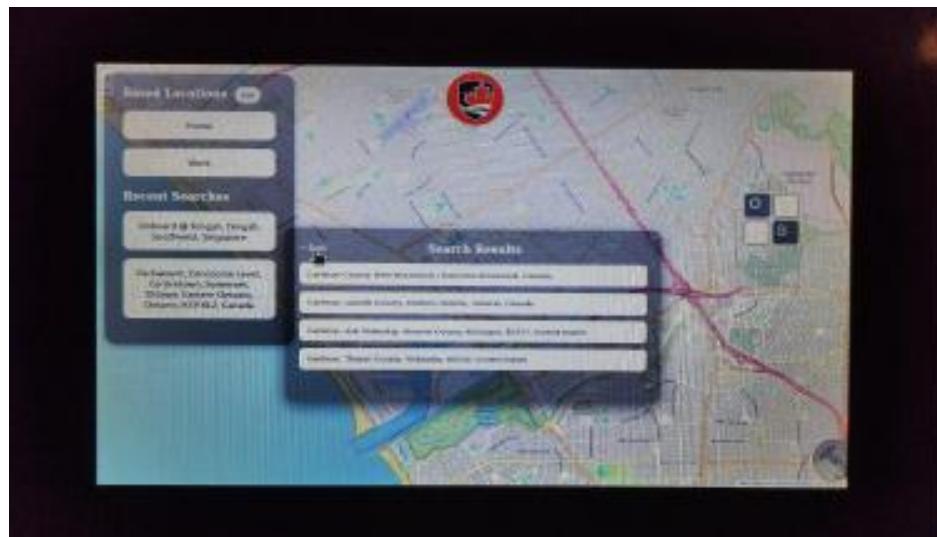


Figure 29: a screenshot showing search results

Integration with Vehicle Diagnostics

Another idea we had is to integrate this navigation portal with the vehicle diagnostics page created by the safety and reliability team to display crucial vehicle statistics such as battery utilisation, emergency stop mechanisms to ensure the driver is informed on all aspects of the vehicles health and status. A hammer icon is presented at the bottom right hand of the navigation user interface

Real-Time Position Tracking

The application pulls real-time positional data from the GPS data received by ROS2, vehicle, refreshing every second to update a blue marker on the map representing the vehicle's current location. As the vehicle moves, this marker moves correspondingly on the interface, providing a dynamic and accurate depiction of the vehicle's journey.

Recent Searches and Location Management

The application enhances usability by managing and storing location data effectively. Once a destination is searched, it is automatically added to the recent locations section within the menu bar at the top left corner. This feature allows users to quickly navigate to frequently visited addresses. Moreover, users have the option to save specific locations as "home" or "work," with these preferences being stored in the system's database for easy retrieval. This personalization within the navigation system helps make the dashboard more user-centric. As can be seen from the screenshot below, a new dropdown menu appears from the menu bar allowing the user to save a location based on its name , lat and long , along with selecting whether its Home or Work.

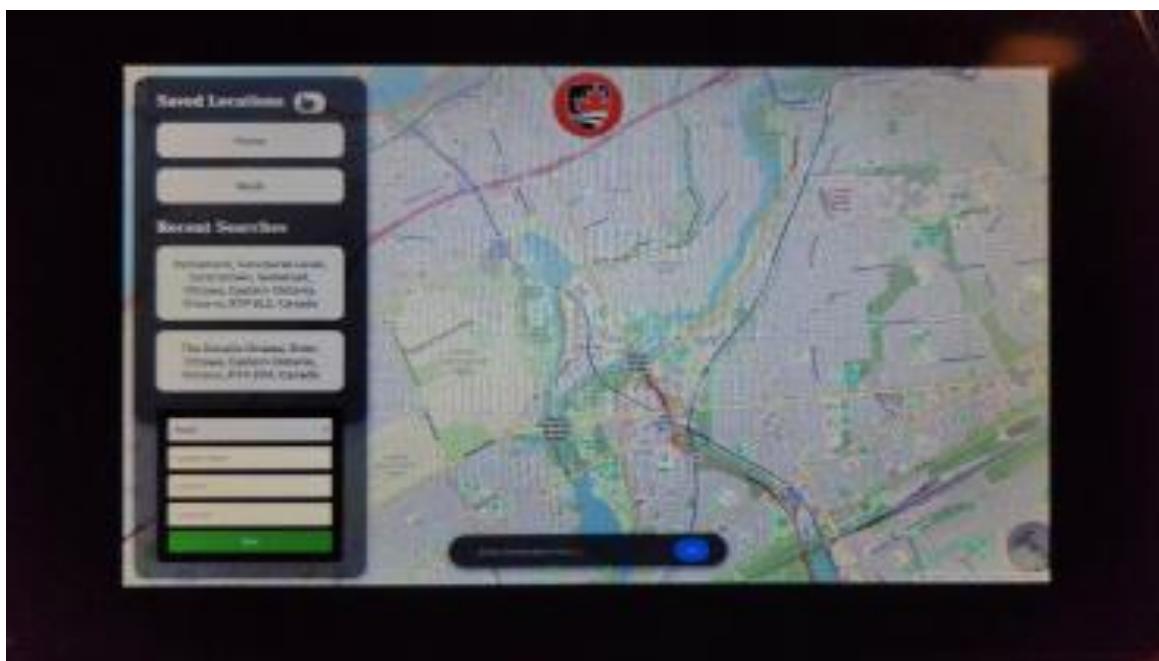


Figure 30 : a screenshot showing edit option for stored locations

Flask Backend Server:

The Flask backend server forms the backbone of the navigation module's communication layer. It connects the touchscreen-based user interface to the rest of the vehicle's ROS2-driven systems. Running on a Raspberry Pi that's mounted inside the vehicle and wired directly to the power system via GPIO, the server handles data flow for real-time positioning, user interaction, and location management. All data requests from the frontend are routed through a set of RESTful API endpoints. For now, these are all implemented using HTTP GET methods but a few of them should really be HTTP POST requests instead..

System Role and Deployment:
The backend is hosted on a Raspberry Pi 4B and is configured to launch automatically on boot using startup scripts. This ensures that the system is always active when the vehicle powers up. The server is also registered as a ROS2 node using rclpy, allowing it to subscribe to GPS data topics and provide live location updates to the frontend.

Backend Architecture:

The Flask server offers a range of API endpoints that handle tasks like searching for locations, storing user-defined points of interest, and managing recent destinations. Each endpoint returns data in JSON format, keeping it easy to parse from the React-based frontend.

S.No	Type	Endpoint	Method	Function
1	GNSS and Positional Data	/position	GET	Returns the latest GPS coordinates published by the ROS2 node. If the node isn't running, an error is returned.
2	Geocoding and Search	/search	GET	Accepts a keyword and returns suggested location matches.

3	Geocoding and Search	/geocode	GET	Converts an address into geographic coordinates.
4	Geocoding and Search	/reverse_geocode	GET	Converts latitude and longitude into a readable place name.
5	Recent and Saved Location Management	/add_location/<location_name>/<latitude>/<longitude>	GET	Adds a recent location entry.
6	Recent and Saved Location Management	/recent_locations	GET	Lists all recent searches.
7	Recent and Saved Location Management	/clear_locations	GET	Clears all stored recent locations.
8	Saved Location Storage and Editing	/save_new_location/<location_name>/<latitude>/<longitude>	GET	Saves a custom location like “Home” or “Work.”
9	Saved Location Storage and Editing	/edit_location/<id>/<location_name>/<latitude>/<longitude>	GET	Edits a previously saved location by ID.
10	Saved Location Storage and Editing	/get_location/<id>	GET	Retrieves data for a saved location.
11	Saved Location Storage and Editing	/clear_saved_locations	GET	Clears all saved entries.

NAV-Table1: API endpoints with their descriptions served by the webserver.

Data Flow

1. From Frontend to Flask: When the user interacts with the UI—searching for a destination or selecting a saved location—the request is sent to Flask via HTTP.
2. From Flask to ROS2: The server subscribes to GNSS data topics and can eventually be extended to publish navigation goals or diagnostics.
3. From Flask to Local Storage: Recent and saved locations are written to local memory (e.g., dictionaries or flat files) for persistence and quick access.

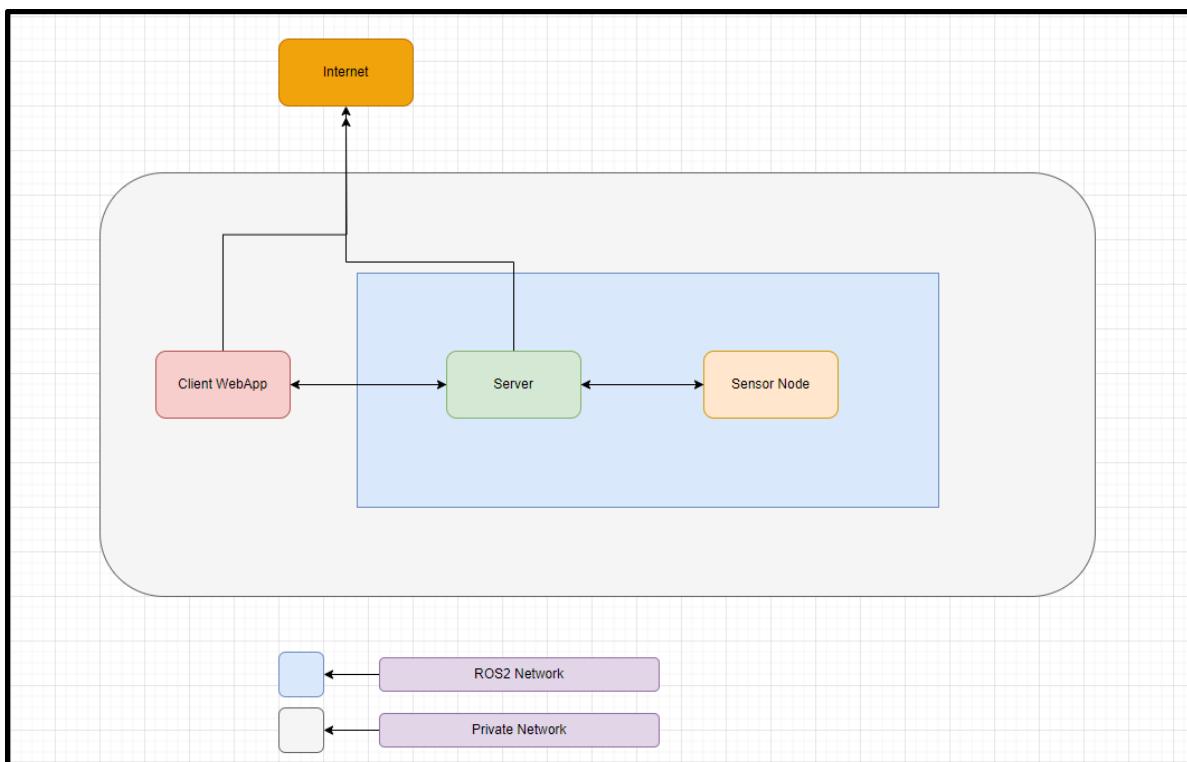


Figure 31: Data flow of the Navigation Dashboard

Using only GET requests made early development simpler, but future updates should convert certain endpoints (like those that modify data) to POST methods to keep in line with RESTful best practices. Right now, the frontend polls the backend for updates like live vehicle location, but real-time update mechanisms—such as WebSockets or AJAX—are under consideration for smoother and more efficient interaction.

The backend was built to be easy to maintain, with clear separation between API routes, ROS2 integration, and local data management. Its modularity means future features—like route publishing or diagnostics integration—can be added without needing to overhaul the system.

7.4.2. Requirements Definition

Camera Sensor (in context of navigation):

ID	Functional Requirement
F1	Define URDF link and transform for the Pi camera.
F2	Stream RGB and depth images from real cameras.
F3	Visualize camera frames and topics in RViz2.
F4	Record camera topics with rosbag2 for later analysis.

NAV-Table 2: Camera Functional Requirements

ID	Non-Functional Requirement
NF1	Maintain consistent and accurate transforms.
NF2	Ensure camera streams are reliable under outdoor testing conditions.
NF3	Lightweight enough to operate alongside other navigation nodes.

NAV-Table 3: Camera non-Functional Requirements

Requirement Validation Method:

- TF frames verified visually in RViz2.

- Live camera feeds confirmed working in RViz2.
- rosbag2 recordings validated topic publishing and synchronization.

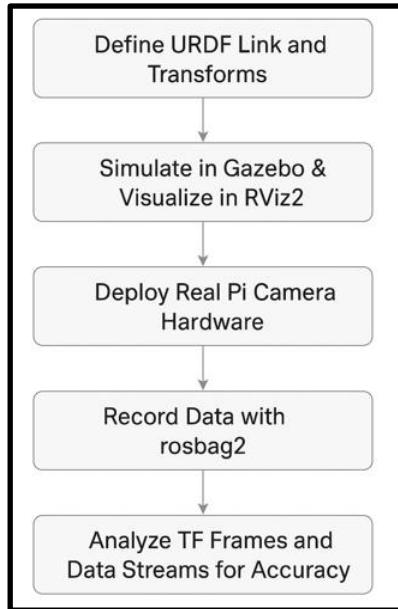


Figure 32: Flowchart of URDF transform

GNSS:

The GNSS subsystem was built around clear functional and non-functional requirements to ensure it met the needs of the autonomous vehicle platform.

ID	Functional Requirement
F1	The system must acquire GNSS and IMU data in real-time.
F2	It must fuse the data streams using an EKF to provide a filtered pose estimate.
F3	It must publish localization data on ROS 2 topics in a standard format usable by Navigation and Control subsystems.
F4	It must maintain consistent coordinate frames across the system.

NAV-Table 4: GNSS Functional Requirements

ID	Non-Functional Requirement
NF1	The system should maintain reliable performance even under partial GNSS signal loss.
NF2	Ensure camera streams are reliable under outdoor testing conditions. The localization system should operate with low latency, updating at a minimum of 10–20 Hz.
NF3	The system must allow remote access for monitoring, debugging, and updates during outdoor testing.

NAV-Table 5: GNSS non-Functional Requirements

User Interface (WebApp):

ID	Functional Requirement
F1	The system must compute and display routes in real-time, allowing the user to navigate from one point to another based on live geographical data.
F2	Continuously update and display the vehicle's current position on the map using GPS data received from ROS2.
F3	Users must be able to search for specific addresses or landmarks, with the system calculating and displaying viable routes using OSM data.
F4	Also users to save their frequently visiting destinations within the system for easy access

NAV-Table 6: UI Functional Requirements

ID	Non-Functional Requirement
NF1	The interface should be intuitive and easy to use for individuals without technical expertise, featuring clear icons, buttons and touch responsiveness.
NF2	The system should be operational and capable of recovering from common errors or issues without crashing, ensuring maximum uptime %.
NF3	The system should have a minimal response time for updating the map with new data under normal usage conditions.
NF4	The system should be able to handle overloading to some extent gracefully, in terms of longer routes or further destinations being processed.

NAV-Table 7: UI non-Functional Requirements

User Interface (Flask WebServer):

ID	Functional Requirement
F1	The backend must subscribe to ROS2 GPS topics and provide current position data through an API endpoint.
F2	The backend must accept user queries and return location suggestions using geocoding services (e.g., OpenStreetMap).
F3	Given coordinates, the backend must return a human-readable location.
F4	The backend must allow users to save, retrieve, and delete labeled locations like "Home" or "Work."
F5	The server must track recently searched locations and support clearing them.
F6	The server must respond to frontend requests using JSON-formatted data over HTTP.
F7	The backend should launch automatically when the Raspberry Pi powers on.

F8	The Flask server must be able to initialize and operate as a ROS2 node for future extensibility.
----	--

NAV-Table 8 : UI Functional Requirements

ID	Non-Functional Requirement
NF1	Response time for API endpoints should remain under 500 ms for basic queries.
NF2	The server should handle intermittent ROS2 topic interruptions gracefully and return fallback responses where applicable.
NF3	The endpoints must return structured JSON suitable for frontend parsing, with clear error messages when needed.
NF4	The server should be built in a modular way to allow future extensions like POST support or diagnostics integration.
NF5	The backend must be lightweight enough to run on a Raspberry Pi with limited resources.
NF6	The code should be clear, documented, and separated into logical components (e.g., routes, ROS2 integration, storage).

NAV-Table 9 : UI non-Functional Requirements

Requirement Validation Method:

- Manually test calling APIs to verify connection and data transmission.
- Manually test API calls by providing parameters and testing correct functionality by visual observation.
- Reboot test on Raspberry Pi and confirm Flask server autostarts using logs and endpoint response.
- ROS2 introspection (ros2 node list) and topic echoing to confirm node registration and activity.

- Peer code review for organization, readability, and inline documentation.

7.4.3. Hardware Design

Cameras:

The main sensor for this subsystem was the Raspberry Pi Camera Module V2, chosen for its low cost, availability, and ease of use with Linux-based systems such as the Jetson AGX Orin. Key specifications include:

- 8-megapixel Sony IMX219 sensor.
- Fixed focus lens.
- 3280×2464 maximum image resolution.
- Video capture capabilities at 1080p30, 720p60, and 640x480p90.

Mount: A custom 3D-printed mount was designed and fabricated specifically for our robot platform to physically attach the Pi camera to the vehicle chassis.(make by the sensor team)

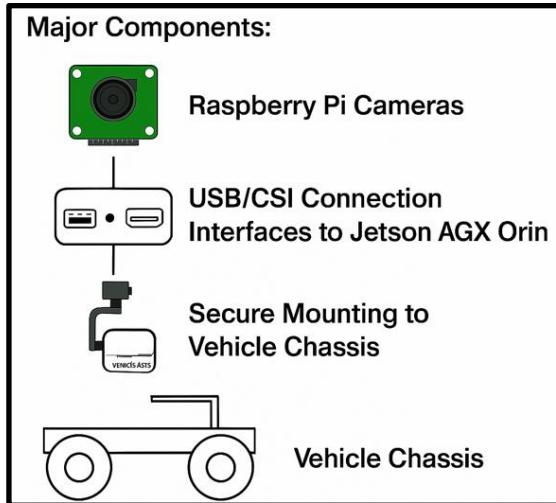


Figure 33: components of the camera sensors.

GNSS/Localization:

The hardware setup for the GNSS subsystem focused on balancing accuracy, reliability, and practical deployment on the vehicle.

The main sensing unit was the **u-blox ZED-F9R GNSS receiver**, which combines GNSS positioning and an onboard IMU to provide raw sensor data for fusion. An external GNSS antenna was mounted on the vehicle to ensure optimal satellite reception, minimizing interference from the vehicle body.

All sensor data was processed on the **NVIDIA Jetson AGX Orin**, a powerful embedded computing platform capable of handling real-time localization tasks and remote monitoring simultaneously. The Jetson was housed inside a custom-made, weather-resistant enclosure with internal mounts for vibration dampening and cable management.

UI Dashboard:

The UI dashboard was designed to provide an intuitive and reliable interface for human interaction with the autonomous navigation system. It integrates a 7-inch Raspberry Pi touchscreen display and a Raspberry Pi 4B, both mounted within the vehicle's cockpit on a custom-designed 3D-printed bracket. The hardware setup ensures accessibility, visibility, and secure attachment in a moving vehicle.

Touchscreen Display:

- **Model:** 6.0" TFT Capacitive Touchscreen Display (compatible with Raspberry Pi GPIO and HDMI)
- **Resolution:** 800x480 pixels
- **Interface:** Connected via GPIO for power and touch data, and HDMI for display output
- **Mounting:** Screen is secured in a recessed slot on the printed bracket with clearance for cables and ventilation
- **Power Supply:** Powered via the Raspberry Pi GPIO pins, drawing current from the vehicle's electrical system through regulated converters

Raspberry Pi 4B

- **Processor:** Quad-core ARM Cortex-A72 (1.5GHz)
- **Memory:** 8GB RAM
- **Storage:** 64GB microSD card for OS, data storage, and Flask server
- **Connectivity:** Connected to internal ROS2 network via Wi-Fi; accessible remotely via SSH or VPN
- **Functionality:** Hosts the Flask backend server, interfaces with the ROS2 system, and serves the frontend UI to the touchscreen browser.

- **Startup:** Configured with a startup script to automatically launch Flask and the UI on boot

Custom 3D-Printed Mount

To ensure a clean and vehicle-appropriate installation, a custom bracket was designed and 3D printed using PLA material. The mount was tailored to:

- Fit the curvature and space constraints of the vehicle dashboard
- Hold both the Raspberry Pi and the touchscreen at a readable viewing angle.
- Allow easy access to ports for power, HDMI, USB, and SD card.
- Enable passive airflow to avoid overheating during extended use

The design process involved multiple prototype iterations in **Fusion 360**, ensuring both functionality and form. The final print was installed using screws and vibration-dampening washers to account for movement during operation.



Figure 34: lcd mounted

Mount

7.4.4. Software Design

User Interface (Flask Backend Server):

The backend server for the UI dashboard was built using a modular, service-style architecture centered around Flask, ROS2, and Python. Its purpose is to link the touchscreen-based user interface with the ROS2 systems onboard the vehicle. It handles live positional data, manages saved and recent locations, and acts as a lightweight processor for map-related tasks.

It runs on Flask, a lightweight Python web framework that suited our needs for fast development and low system overhead, important given the Raspberry Pi's hardware limitations. It exposes a set of REST API endpoints, currently using only GET methods, to handle the following:

- Providing current GPS coordinates via ROS2.
- Searching for destination addresses.
- Saving, retrieving, and clearing locations.
- Converting between addresses and coordinates (geocoding and reverse geocoding).

The server also acts as a ROS2 node using rclpy, allowing it to subscribe to GNSS topics. The data is then served to the frontend using standard HTTP requests, currently polled on a time interval.

S.No	Software stack/platform	Description
1	Python 3.10 +	-
2	Flask	Web framework for serving REST APIs
3	ROS 2 Humble	Middleware layer for robotic communication
4	rclpy	Python client for creating ROS2 nodes
5	Geopy/Nominatim API	Used for location name-to-

		coordinate conversion
6	Local SQL storage database	For storing user-saved and recent locations
7	Linux Startup Service	Ensures Flask starts automatically on Raspberry Pi boot

NAV-Table 10 :Backend software stack

To ensure continuous availability, the server is configured to launch automatically using a Linux startup script whenever the Raspberry Pi is powered on. The addition of SQLite3 offers a more durable and structured way to manage data. It allows us to store user preferences like “Home” and “Work” locations with greater reliability, and makes data management simpler across restarts and updates.

API Logic and Usage:

The backend keeps things simple and efficient. Each API endpoint as described in a previous section of this document is tied to a specific task, with minimal logic overhead.

These APIs are consumed by the frontend dashboard, which calls them using HTTP requests triggered by user interaction.

User Interface (Front End):

Software Framework:

- React.js framework used{NAV21}, which is a open-source Javascript Library used to effectively build complex user interfaces by splitting code into reusable containers called components.
- Vite [NAV22] used as a build tool for deployment server, npm used as a package manager for loading dependencies for a javascript application.
- Leaflet.js framework used for displaying MapContainers onto the webpage, its an open source javascript library with functional tools for interacting and mapping Map Tiles onto the DOM and is fast and low latency, helps making the map output instant feedback due to high responsiveness.

Cameras (in context of navigation):

Software Framework:

- URDF modeling for camera placement and TF management.
- ROS 2 nodes for live image and depth image publishing.
- Use of robot_state_publisher to manage frame transforms.
- Visualization and debugging using RViz2.

GNSS/Localization:

Software Framework:

- ROS 2 nodes for GNSS and IMU data acquisition.
- robot_localization package for EKF implementation.
- Standard ROS 2 message types (sensor_msgs/NavSatFix, sensor_msgs/Imu).
- SSH/VPN setup for remote access and monitoring.

OpenStreetMap (OSM) was chosen as the database to source our geographic data as it is open-sourced. As we wanted only relevant features such as roads, intersections, and pathways we used Overpass API which allowed us to filter out the data that the navigation system does not require. After the data was collected, it was processed through QGIS and then converted into data occupancy grid maps. This involved configuring the map resolution, which would determine how big each pixel in the map represents in the real-world space. These values were set up in the “.yaml” file of the map. The final output was a “.pgm” file which represents a grayscale image of the map and the “.yaml” file consisting of the resolution and other metadata. These could then be utilized on Rviz for visualization and used by the ROS2 Nav2 stack. The figure below shows some of the information that is provided through OSM data files.

```

707   {
708     "type": "Feature",
709     "properties": {
710       "@id": "way/29570254",
711       "bicycle": "yes",
712       "foot": "yes",
713       "highway": "path",
714       "segregated": "no",
715       "surface": "asphalt",
716       "width": "3",
717       "winter_service": "yes",
718       "winter_service:quality": "good"
719     },
720     "geometry": {
721       "type": "LineString",
722       "coordinates": [
723         [
724           -75.6954854,
725           45.384962
726         ],
727         [
728           -75.6954472,
729           45.3849727
730         ],
731       ]
732     }
733   }
734 }
```

Figure 35: Sensor output data

7.4.5. Integration & Testing Approach

Camera (in the context of navigation) :

URDF Deployment and Simulation Testing

The implementation of the Camera Navigation subsystem followed a systematic approach through phased deployment to validate simulation models against real hardware before moving to complete system deployment. The first step involved modifying the robot's URDF model by adding a new camera_link which joined directly to base_link through an immovable joint. The robot was placed into a specialized Gazebo simulation environment after finishing the updated URDF file. The initial Gazebo tests verified how the camera behaved both when stationary and moving by confirming its predictable and reasonable positioning in relation to the robot chassis.

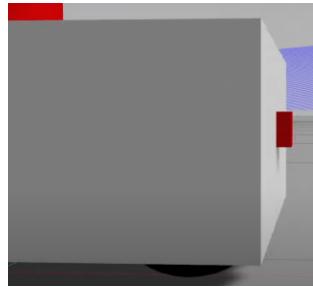


Figure 36: Camera on the chassy in gazebo

The transform (TF) tree was continuously monitored by RViz2 to verify that the map → odom → base_link → camera_link relationships were correctly maintained over time. The optical axis of the simulated camera was carefully aligned to the actual physical mounting position on the real vehicle. The origin parameters of the URDF were adjusted during simulation testing to get an exact match between the simulated and real world expected alignments.

After simulation validation, the real Raspberry Pi cameras were attached to the robot using custom-designed 3D-printed brackets. The cameras were connected to the Jetson AGX Orin via USB and the v4l2_camera ROS 2 nodes were launched to start live video streaming. The topics of images, including the RGB streams /camera/image_raw and the depth data /camera/depth/image_raw, were checked in RViz2 to ensure that they were being published actively without any noticeable latency or packet loss.

GNSS:

Integration of the GNSS subsystem involved connecting hardware and software components seamlessly through ROS 2 interfaces. ROS topics served as the communication backbone between different nodes.

The system's TF tree was carefully managed to ensure the map, odom, and base_link frames were aligned correctly, which is crucial for downstream subsystems like Navigation and Control. Any inconsistencies in frame orientation or timing were quickly detected through RViz visualization.

Testing was conducted in outdoor environments on campus. Field tests involved driving the vehicle along known paths and observing the GNSS and fused odometry outputs in real-time. We used RViz to visualize the vehicle's estimated position and compared it with actual visual landmarks to qualitatively assess accuracy.

ROS bag files were recorded during all tests, allowing offline analysis to detect anomalies like GNSS dropouts, IMU spikes, or EKF divergence. Data from multiple runs helped us refine the EKF settings for better fusion stability under real-world conditions.

We also made sure that when the jetson is powered on, all the nodes that are built will run automatically

```
avjetson23@ubuntuav:~/ros2_ws$ journalctl -u gnss.service -f
Mar 24 19:02:01 ubuntuav systemd[1]: Started Start GNSS + RTK Stream.
Mar 24 19:02:11 ubuntuav start_gnss.sh[10692]: stream server start
Mar 24 19:02:11 ubuntuav start_gnss.sh[10692]: 2025/03/24 23:02:30 [-C---]          0 B      0 bps
Mar 24 19:02:16 ubuntuav start_gnss.sh[10692]: 2025/03/24 23:02:35 [CC---]      1504 B    2898 bps (
0) rtk2go.com/CanalTerris
Mar 24 19:02:16 ubuntuav start_gnss.sh[10692]: stream server stop
Mar 24 19:02:16 ubuntuav systemd[1]: gnss.service: Deactivated successfully.
```

Figure 37: .sh file that starts all the nodes

In the figure above, you can see that all the nodes are running successfully and the GNSS data stream is running without errors.

User Interface (Front End):

To ensure the hardware, which was the 7 inch digital display used to view and interact with the software and the Raspberry Pi 4b, which was used to host the React Application on, integrated and worked as intended with the software, several types of testing. Initially, it was a lot of manual testing, we tested the touch display and used it to interact with the navigation platform, by dragging the map to different positions, zooming in and out etc. We tested the typing functionality of the search bar by entering a destination and seeing if we were prompted with the digital keyboard..We hosted the web server on the raspberry pi and ran it entirely off that locally to ensure all functionality was present. We had to optimize the web application initially and incorporate responsive web design principles in order for the navigation page to render appropriately for that display size, as it was initially coded for a desktop platform. We ran several tests to see if all parts of the display were able to interact with the platform and whether any functionality was being cut off due to size.

User Interface (Backend Flask Server):

The backend server and UI dashboard were tested incrementally, reflecting the modular way the navigation system was developed. Since the system depends on ROS2 data, embedded hardware, and API-based communication, the testing process focused on isolating each component early on, confirming that they worked individually, and then gradually bringing everything together for end-to-end validation.

We began by testing the Flask backend on its own. Each API endpoint was manually tested using curl, Postman, and short Python scripts to confirm they returned valid JSON responses and handled edge cases like incomplete requests or missing ROS2 data without crashing. The SQLite3 database was also tested for basic insert, update, and lookup operations, with particular attention paid to whether saved data persisted after rebooting the Raspberry Pi.

After that, we moved on to checking the backend's integration with ROS2. We set up mock GPS publisher nodes and verified that the backend was successfully subscribing to the data using ros2 topic echo. We also used 'ros2 node list' to make sure our server was properly registered as a ROS2 node. We tested how the system behaved when GPS messages were dropped or delayed to check for stability in low-signal conditions.

Once the backend was stable, we integrated the frontend dashboard and ran it against the server on the Raspberry Pi. The map's real-time updates were observed closely, and we tested how the UI responded to destination searches and saved location selections. We also tested the system's startup scripts by power cycling the Pi several times to make sure both the server and dashboard loaded on boot without needing manual input. journalctl and systemctl status helped us confirm the services were running smoothly.

The final round of testing took place inside the actual vehicle. Here, we looked at how the system performed with live GNSS data. We paid attention to GPS update speed, UI lag, and general responsiveness (most of the things broke when it came to real life testing as expected). We checked that saved locations appeared as expected, that the UI remained usable while the vehicle was moving, and that the system could place pins on the map accurately. We logged and debugged any issues like touchscreen misalignment, delayed API responses, or minor GPS drift.

This staged approach starting with individual components, then integrating them into a full system helped us catch problems early and made the final product more stable. It also allowed us to understand how each part behaved in isolation, which made troubleshooting easier when things didn't work as expected.

7.4.6. Challenges & Troubleshooting

Cameras (in the context of navigation):

The integration and testing phases of the Camera Navigation subsystem presented multiple challenges. The first major problem emerged when camera frames became misaligned thus the camera view displayed in RViz2 and Gazebo showed an incorrect orientation compared to the robot's expected forward direction. The initial URDF camera link modeling produced small positional and rotational inaccuracies that became visible only after visualization began. The URDF frame origins and joint orientations required multiple iterations of adjustment to set correct XYZ positions and RPY (roll, pitch, yaw) angles which resulted in proper camera view alignment with the vehicle's physical structure and motion direction.

System-level software problems together with technical difficulties caused additional delays in integration. The simulation testing encountered compatibility issues with Gazebo when it underwent a major version update because this update established new requirements for world file formatting and plugin configuration and sensor modeling. The robot model failed to spawn correctly while the camera plugins exhibited unpredictable behavior. The solution needed thorough examination of new Gazebo documentation followed by file updates to match new standards and complete simulation pipeline retesting.

GNSS:

- ‘rosbag’ recording was slow
- Some of the applications in the jetson automatically got uninstalled and we had to install it manually every time.
- GNSS signal loss near buildings and foliage.
- IMU noise during sudden vehicle acceleration or vibration.
- Initial TF frame misalignment causing pose drift.

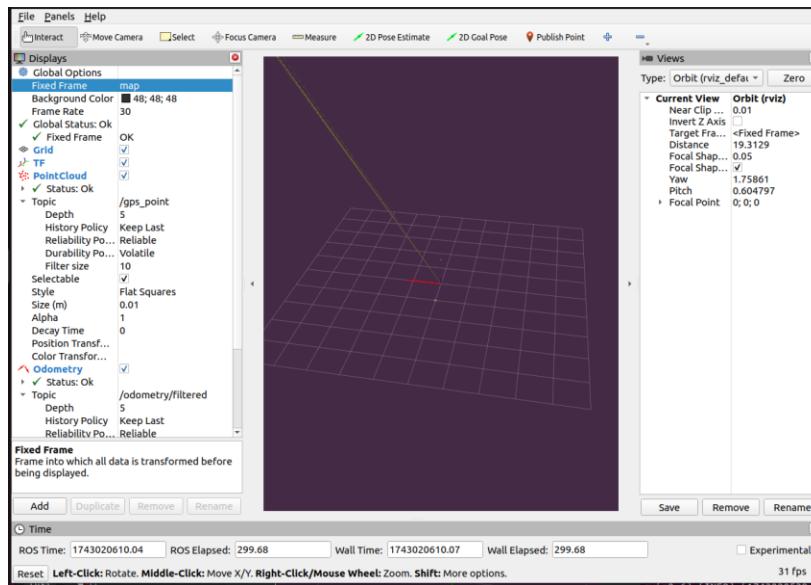


Figure 38: Rviz2 screenshot

In this figure above, as you can see that the yellow points (GNSS) is pointing outwards indicating that the rosbag recording was slow, resulting in an incorrect simulation result

Troubleshooting Strategies:

- Implemented fallback strategies in EKF to rely more on IMU during GNSS dropouts.
- Tuned EKF noise parameters to better handle IMU inaccuracies.
- Verified and corrected TF frame relationships visually in RViz during testing

User Dashboard (WebApp and WebServer):

When connecting the back end flask server with the front end react application we encountered a connectivity issue. The flask APIs, which include the ip address of the server were hardcoding into the POST requests to pull data into the fetch queries. But everytime the flask server was turned on, it would generate another ip address, rendering the hardcoded api to not work. This problem arose as the flask server was located on Carleton Campus using campus wifi, this way due to Carleton University's network policy[NAV20], which does not allow static IP address assignment. This policy meant that our device would receive a different IP address upon each connection. To solve this issue, We had to establish a private VPN service and establish our own authentication key. This setup allows our system components to maintain a fixed IP address within the VPN, irrespective of changes in the external network. This issue only remained a problem during development, since when the raspberry pi was mounted on the vehicle, it was assigned a static IP in the local environment.

Another issue we faced was inside the react application , we were importing external libraries from icons like markers/pins to place onto the map. This worked fine while connected to the network, as these libraries needed to be imported in real time. When the webapp was run with limited network access , a scenario that could present itself during the vehicle's travel, the pin would not load properly into the webapp and display a large corrupted file in place. We solved this issue by locally storing a svg image (Scalar Vector Image), which consists of long XML code in order to generate a 2d image of the desired pin in your webapp, hence importing that would not need any sort of network help.

Another recent problem we encountered when implementing the real-time position tracking of the vehicle on the map, was that each time we received a new position coordinates from the GPS ROS2 node , we would re-render(Javascript has built-in methods for this) the application in order to adjust the position. The issue was that the flask server would update the position every $\frac{1}{3}$ of a second, and the react application wouldn't be able to load the MapContainer within that time frame, and would constantly keep buffering. In order to solve this, we added several conditions into the react scripts that will only rerender the application if the current Containers had fully loaded in. From this, we discovered the application takes slightly less than 1 second to render. This wasn't an issue as the marker would still move smoothly across the page due to some transition effects and due to the slow speed of the vehicle itself.

7.5. Project Management & Execution

7.5.1. Work Breakdown Structure

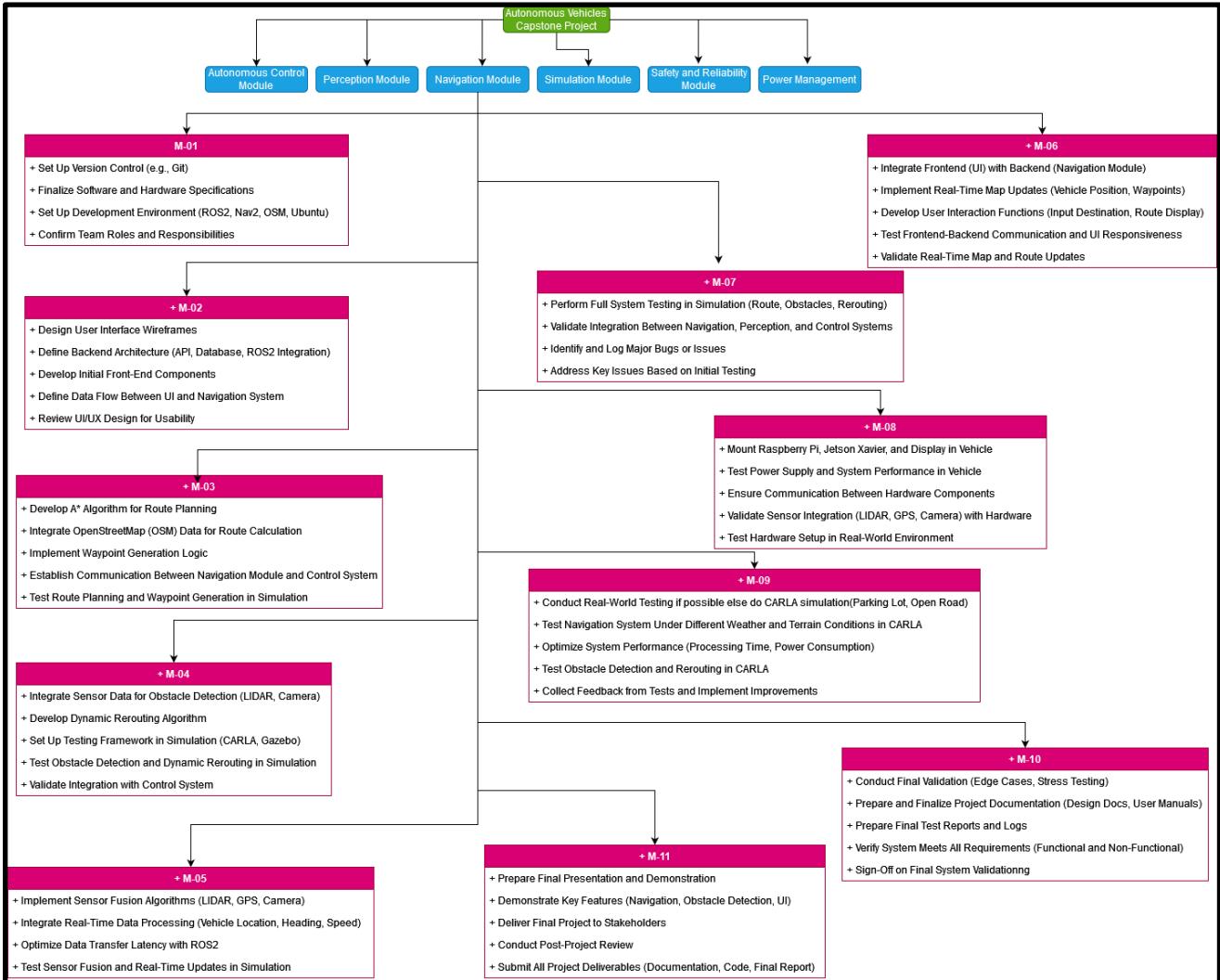


Figure 38: Milestones

7.5.2. Gantt Chart & Milestone Tracking

S.No	Milestone	Assigned To	Oct	Nov				
			WEEK1	WEEK2	WEEK3	WEEK4	WEEK5	
1	Project Setup	Everyone	28-Oct-24					25-Nov-24
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif		04-Nov-24	11-Nov-24	18-Nov-24		
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed						
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik						
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik						
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif			04-Nov-24	11-Nov-24	18-Nov-24	25-Nov-24
7	Early System Testing & Validation	Everyone						
8	Hardware Integration	Everyone						
9	Simulation and optimization	Everyone						
10	Final System Validation & Documentation	Everyone						
11	Final Demonstration and Delivery	Everyone						

Figure 39: Timeline part 1

S.No	Milestone	Assigned To	Dec				
			WEEK6	WEEK7	WEEK8	WEEK9	WEEK10
1	Project Setup	Everyone		02-Dec-24	09-Dec-24	16-Dec-24	23-Dec-24
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif					
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed					
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik	04-Dec-24				
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik			16-Dec-24	23-Dec-24	30-Dec-24
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif	04-Dec-24		16-Dec-24	23-Dec-24	30-Dec-24
7	Early System Testing & Validation	Everyone					
8	Hardware Integration	Everyone					
9	Simulation and optimization	Everyone					
10	Final System Validation & Documentation	Everyone		02-Dec-24	09-Dec-24	16-Dec-24	23-Dec-24
11	Final Demonstration and Delivery	Everyone					

Figure 40: Timeline part 2

S.No	Milestone	Assigned To	Jan				
			WEEK11	WEEK12	WEEK13	WEEK14	WEEK15
1	Project Setup	Everyone	06-Jan-25	13-Jan-25	20-Jan-25	27-Jan-25	03-Feb-25
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif					
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed					
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik					
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik	06-Jan-25	13-Jan-25	20-Jan-25	27-Jan-25	03-Feb-25
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif	06-Jan-25	13-Jan-25	20-Jan-25	27-Jan-25	03-Feb-25
7	Early System Testing & Validation	Everyone				06-Feb-25	13-Feb-25
8	Hardware Integration	Everyone					
9	Simulation and optimization	Everyone					
10	Final System Validation & Documentation	Everyone					
11	Final Demonstration and Delivery	Everyone					

Figure 41: Timeline part 3

S.No	Milestone	Assigned To	Feb					
			WEEK15	WEEK16	WEEK17	WEEK18	WEEK19	WEEK20
1	Project Setup	Everyone	03-Feb-25	10-Feb-25	17-Feb-25	24-Feb-25	03-Mar-25	10-Mar-25
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif						
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed						
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik						
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik						
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif						
7	Early System Testing & Validation	Everyone	03-Feb-25	10-Feb-25	17-Feb-25	24-Feb-25	03-Mar-25	10-Mar-25
8	Hardware Integration	Everyone	03-Feb-25	10-Feb-25	17-Feb-25	24-Feb-25	03-Mar-25	10-Mar-25
9	Simulation and optimization	Everyone						
10	Final System Validation & Documentation	Everyone						
11	Final Demonstration and Delivery	Everyone						

Figure 42: Timeline part 4

S.No	Milestone	Assigned To	Mar				Apr	
			WEEK20 10-Mar-25	WEEK21 17-Mar-25	WEEK22 24-Mar-25	WEEK23 31-Mar-25	WEEK24 07-Apr-25	WEEK25 14-Apr-25
1	Project Setup	Everyone						
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif						
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed						
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik						
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik						
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif						
7	Early System Testing & Validation	Everyone						
8	Hardware Integration	Everyone						
9	Simulation and optimization	Everyone						
10	Final System Validation & Documentation	Everyone						
11	Final Demonstration and Delivery	Everyone						

Figure 43: Timeline part 5

Milestone ID	Milestone	Deliverables	Completion Criteria
M-01	Project Setup	<ul style="list-style-type: none"> Set up development environment Finalize team roles 	<ul style="list-style-type: none"> development environment configured.
M-02	Initial UI/UX Design and Backend Architecture	<ul style="list-style-type: none"> Complete UI wireframes. Define backend architecture. Start front-end development. 	<ul style="list-style-type: none"> Initial UI design completed. Backend architecture defined. Basic UI functions operational.
M-03	Route Planning and Waypoint Generation Development	<ul style="list-style-type: none"> Develop route planning using A*. implement waypoint generation. Integrate with control system. 	<ul style="list-style-type: none"> Navigation module generates routes successfully. Navigation module generates waypoints successfully.

M-04	Obstacle Detection and Dynamic Rerouting	<ul style="list-style-type: none"> Implement obstacle detection using sensors. Develop dynamic rerouting. 	<ul style="list-style-type: none"> System successfully reroutes based on detected obstacles in a simulated environment.
M-05	Sensor Fusion and Real-Time Data Integration	<ul style="list-style-type: none"> Implement sensor fusion. Integrate real-time data (location, speed, heading) processing. 	<ul style="list-style-type: none"> Accurate sensor fusion. Real-time updates integrated.
M-06	Frontend-Backend Integration and Map Updates	<ul style="list-style-type: none"> Complete frontend-backend integration. Real-time map updates on the UI. Test user interaction. 	User can set destinations and view real-time route updates on UI.
M-07	Early System Testing and Validation	<ul style="list-style-type: none"> Perform full system tests in a simulated environment. Validate navigation, perception, and control system integration. 	All major functions operational in simulation with stable performance.
M-08	Hardware Integration	<ul style="list-style-type: none"> Mount Raspberry Pi, Jetson Xavier, display in the vehicle. Test hardware components, and communication. 	Hardware mounted and communication between components established.

M-09	Simulation and optimization	Conduct deeper tests in a simulated environment (CARLA), optimize performance.	Complex simulations completed, performance optimized for navigation, obstacle detection, and rerouting.
M-010	Final System Validation and Documentation	<ul style="list-style-type: none"> • Conduct final validation. • Complete project documentation. • Test edge cases. 	System validated and fully documented with a user manual, design docs, and test reports.
M-011	Final Demonstration and Delivery	Prepare final presentation, demonstration, and deliver final project to supervisors and Professors.	Successful final demonstration, project delivered to stakeholders.

NAV-Table1: Milestones Description

7.5.3. Risk Analysis & Mitigation Strategies

Risk analysis for this project module involved identifying technical, hardware, software, and integration-related issues that could impact development timelines or system performance. For each risk, mitigation strategies were either pre-planned or developed in response to emerging problems during implementation.

GNSS:

One of the earliest concerns was the reliability of GNSS signals, especially in areas with obstructed sky views such as near tall buildings or indoors where the car was kept during development. Anticipating this, we attempted sensor fusion with IMU data using an Extended Kalman Filter (EKF) to reduce reliance on GNSS during temporary outages. Additional testing was done to capture data in varying environments and verify localization consistency.

Another hardware-specific issue occurred when the GNSS antenna became physically disconnected due to its fragile attachment. To mitigate this, we physically secured the antenna using tape to the board and monitored its connection status more regularly, preventing further disruptions.

Cameras and Visual System:

Camera integration brought its own set of challenges, primarily in the form of TF tree inconsistencies. Improperly defined transforms could lead to major navigation errors if the camera's coordinate frame didn't align correctly with other subsystems. To address this, we enforced a validation loop using RViz2 and live field testing. During simulation and hardware deployment, we tracked the robot's transforms and adjusted URDF parameters until consistent and accurate alignment was achieved.

On a secondary note, we worried deeply about bandwidth limitations, particularly when streaming high-resolution RGB and depth data simultaneously. This led to frame drops and latency spikes, threatening the responsiveness of the navigation system. It was somewhat mitigated by dynamically adjusting camera resolution, frame rates, and using selective image compression techniques. Additionally, only critical topics were recorded during testing to reduce I/O strain.

Software Compatibility:

Software updates presented unexpected issues, especially when Gazebo simulator dependencies changed. A version update broke compatibility with certain world files and plugins, disrupting simulation. To manage this risk, backup versions of the simulation environment were created and version control mechanisms were introduced to lock compatible versions of the simulator and its plugins. We also documented plugin and model parameters for quick recovery and easier reconfiguration when updates occurred.

Web Application and Backend Server:

While the UI and Flask backend introduced less mechanical risk, they carried challenges common to web-integrated robotic systems. One key risk was loss of synchronization between the UI and real-time ROS2 data. In cases where the server lost connection the UI could display stale or incorrect information causing unnecessary panic in the user. To mitigate this, we attempted to create error-handling mechanisms to notify users of connectivity issues.

Another risk involved the Flask server running on limited hardware (Raspberry Pi). High-frequency data processing or multiple concurrent requests could lead to resource exhaustion. We addressed this by limiting polling rates, offloading some logic to the frontend, and testing system performance under load to catch bottlenecks early.

Costmap and Nav2 Configuration:

Costmap generation relied on Nav2's well-tested internal plugins, meaning risk was minimal from a software stability standpoint. However, if the URDF description or frame configuration was incorrect, the costmap could produce invalid spatial data, leading to unsafe paths. Recognizing this, we took extra care when populating the robot's URDF file and verified transform consistency through simulation and field testing.

7.5.4. Budget & Cost Analysis

The budgeting process for the Navigation module was coordinated alongside the other five subgroups involved in the autonomous vehicle project. Early in development, each team identified essential hardware and components required for their systems. These were compiled into a shared procurement list and submitted for approval as part of a collective funding request to KEFC (Kostiuk Engineering Funding Collective), the capstone program's designated funding body.

For the Navigation team specifically, our approved purchases included a Raspberry Pi 4B, an accompanying microSD card, a touchscreen display and a housing for them to support the deployment of our in-vehicle user interface. All four items were successfully funded through KEFC, allowing us to move forward with hardware integration early in the term.

A separate, smaller round of spending was required later in the semester when we designed and 3D printed a custom mount for the Raspberry Pi and display unit. This expense, though modest, was not covered by external funding and was therefore shared evenly among team members.

On the software side, the entire Navigation system was developed using open-source or freely licensed tools. This included ROS 2 (Humble), Gazebo, RViz2, and rosbag2, all of which were critical to development but carried no licensing costs. Additional tools such as QGIS, Flask, and React were also open-source, and any services used were selected to avoid recurring fees or commercial licensing restrictions.

Regarding the camera system, the required Raspberry Pi Camera Modules were purchased and provisioned by the Sensors team, as part of their responsibilities. While not directly managed by the Navigation team, our group contributed to camera-related tasks and integration where needed, particularly in URDF configuration, TF frame alignment, and simulation validation. The cameras cost approximately \$50 CAD each, totaling \$100 CAD, and were the only major expense attributed to camera-based perception. Other supporting items such as mounting brackets, vibration dampeners, USB extensions, and related hardware were supplied by the university's shared laboratory inventory at no extra cost.

Overall, careful planning and collaboration allowed us to meet all functional requirements without exceeding budget constraints, while leveraging institutional resources and community-supported software wherever possible.

7.6. Experimental Setup & Testing

7.6.1. Testing Environment & Procedures

GNSS:

The testing environment was carefully selected to closely represent the real-world scenarios the

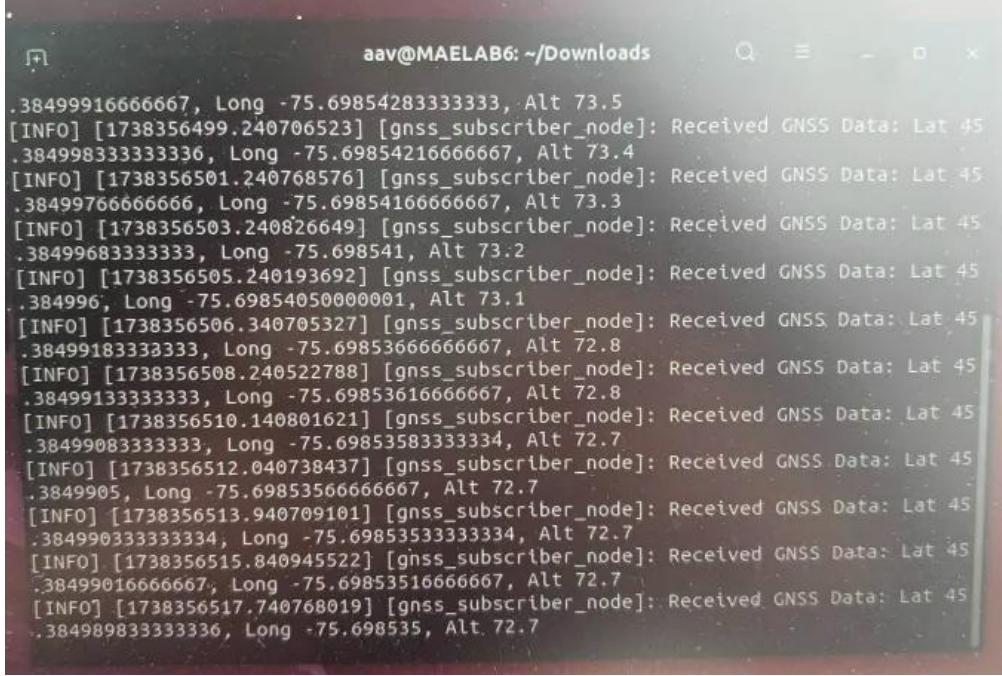
autonomous vehicle might encounter on Carleton University's campus. The primary goal was to verify and evaluate the GNSS and IMU localization system's ability to provide accurate localization whether it be outdoors or indoors.

We chose specific routes for testing where we tested the car in the parking lot. The variety of these testing locations allowed us to examine our system's robustness comprehensively, providing valuable insights into how environmental conditions impact GNSS signals. Tests were typically conducted on a clear weather to ensure consistent conditions and more accurate readings of the sensors.

Before each test session, the team went through a standardized preparation checklist, which involved verifying all hardware components such as the GNSS receiver (u-blox ZED-F9R), IMU sensors, antennas, and ensuring the Jetson AGX Orin computing platform was securely mounted and fully operational. Batteries and power supplies were double-checked to guarantee consistent voltage levels throughout the tests. Additionally, we made sure that we put a tape on the antenna so that it is stuck to the GNSS board at all times.

To automate the GNSS data collection processes, we created a custom Linux systemd service named `gnss.service`. This automated system handled the initiation and management of GNSS data streams. Real-time logs from this service were continuously monitored using standard Linux tools such as `journalctl`. This approach significantly streamlined our fieldwork by reducing manual configurations, ensuring consistency, and enabling rapid identification and resolution of potential issues during tests.

Visualization and monitoring were conducted in real-time through RViz, an integral ROS 2 visualization tool. RViz allowed the team to observe the localization data in the context of the vehicle's position relative to a global frame (map). Visualizing GNSS raw data points and EKF-filtered odometry provided immediate insight into sensor fusion effectiveness and data quality.

A screenshot of a terminal window titled "aav@MAELAB6: ~/Downloads". The window contains a log of GNSS data received by a subscriber node. The log consists of multiple lines of text, each starting with "[INFO] [timestamp] [gnss_subscriber_node]: Received GNSS Data: Lat <value>, Long <value>, Alt <value>". The values are static, indicating indoor testing conditions.

```
.38499916666667, Long -75.69854283333333, Alt 73.5
[INFO] [1738356499.240706523] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.384998333333336, Long -75.69854216666667, Alt 73.4
[INFO] [1738356501.240768576] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.384997666666666, Long -75.69854166666667, Alt 73.3
[INFO] [1738356503.240826649] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38499683333333, Long -75.698541, Alt 73.2
[INFO] [1738356505.240193692] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.384996, Long -75.69854050000001, Alt 73.1
[INFO] [1738356506.340705327] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38499183333333, Long -75.69853666666667, Alt 72.8
[INFO] [1738356508.240522788] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38499133333333, Long -75.69853616666667, Alt 72.8
[INFO] [1738356510.140801621] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38499083333333, Long -75.69853583333334, Alt 72.7
[INFO] [1738356512.040738437] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.3849905, Long -75.69853566666667, Alt 72.7
[INFO] [1738356513.940709101] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.384990333333334, Long -75.69853533333334, Alt 72.7
[INFO] [1738356515.840945522] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38499016666667, Long -75.69853516666667, Alt 72.7
[INFO] [1738356517.740768019] [gnss_subscriber_node]: Received GNSS Data: Lat 45
.38498983333336, Long -75.698535, Alt 72.7
```

Figure 38: GNSS Subscriber node

In this figure, as you can see the GNSS is publishing data in NMEA mode and the data is static since it was recorded indoors. In summary, the testing environment and structured procedures provided a solid foundation for validating the GNSS localization subsystem under realistic, dynamic outdoor conditions.

User Dashboard (WebApp and WebServer):

We tested the navigation dashboard and backend server using a mix of simulation tools, mocked ROS2 data, and real-world hardware trials. Everything was built and run on a Raspberry Pi 4B, which hosted both the Flask backend and the React frontend. This setup matched the in-vehicle deployment closely and made testing easier to manage early on.

During development, we used Leaflet to simulate different types of map activity like tile loading, zooming, and marker placement. To check accuracy, we manually dropped pins and compared their coordinates to Google Maps results. It wasn't a perfect method, but it gave us enough confidence in the map rendering and coordinate handling.

We also used Puppeteer[NAV22] to run some automated frontend tests. These weren't very complex, but they were useful for catching visual bugs and logic issues. We scripted basic tasks

like searching for a location or loading a saved one, and checked that the map responded the way we expected.

For backend testing, we built a simple ROS2 mock node that published fake GPS coordinates on a timer. This let us see how both the frontend and Flask server responded to real-time data. The Flask server subscribed to GNSS data and passed it to the UI through REST endpoints. We used Postman and some quick Python scripts to test these APIs and make sure responses were formatted correctly.

We also tested location storage using SQLite3. We saved places like “Home” and “Work,” rebooted the Pi, and checked that the data persisted. Nothing fancy, but it worked consistently.

Overall, this approach starting simple, testing frequently, and gradually moving into real-world conditions helped us catch issues before they became bigger problems.

7.6.2. Experimental Results and Findings

Our extensive testing gave us valuable insights regarding the GNSS-IMU localization subsystem's strengths and limitations. The findings confirmed that the system reliably provided accurate positioning data, particularly under optimal GNSS signal conditions in open environments.

In open spaces such as campus parking spot, the system demonstrated remarkable accuracy, typically maintaining localization within ± 0.5 meters of actual positions. This was validated visually through real-time RViz monitoring, where the EKF-filtered odometry closely matched our physical paths. These scenarios confirmed that our EKF parameter tuning effectively managed GNSS and IMU sensor data, creating a reliable and consistent pose estimate.

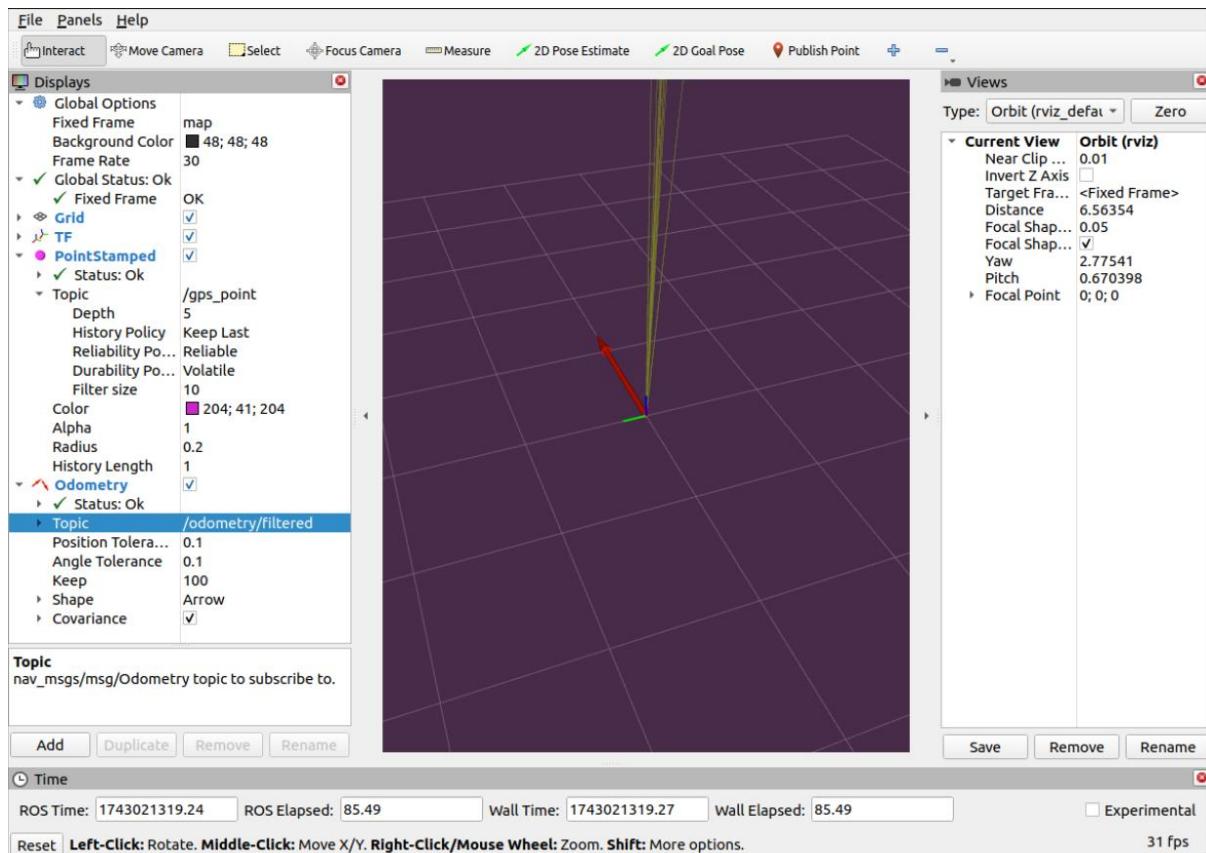


Figure 39: implementation in rviz

In the above figure, you can see that the GNSS yellow points are pointing towards the vehicle and the topic is subscribed to ‘/odometry/filtered’ which tells us that the odometry can be published through ros2 and can be viewed using the command ‘ros2 topic list’ and can be used by other subgroups if required.

Additionally, initial misalignment issues in TF frames were identified during early testing sessions, causing positional offsets. This misalignment was primarily due to incorrect static transform definitions. Upon identifying this through real-time visualization in RViz, we promptly recalibrated the transforms, dramatically improving the consistency and accuracy of the localization data.

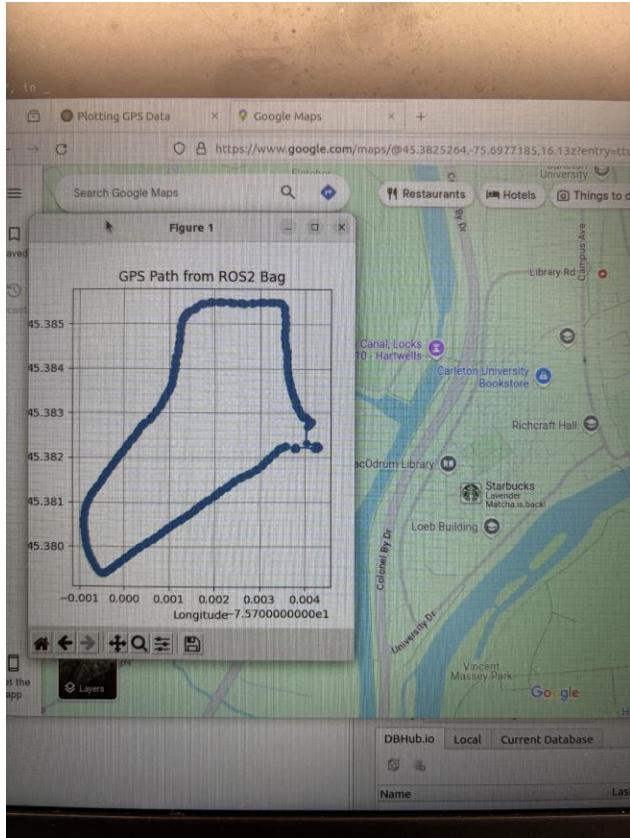


Figure 40: .GNSS data

The figure above is the recording of the rosbag of the GNSS data around Carleton University and this concludes that the GNSS board is working and can handle GNSS data.

```
avjetson23@ubuntuav:~/ros2_ws$ nano ~/ros2_ws/start_ekf.sh
avjetson23@ubuntuav:~/ros2_ws$ chmod +x ~/ros2_ws/start_ekf.sh
avjetson23@ubuntuav:~/ros2_ws$ sudo nano /etc/systemd/system/ekf_localization.service
avjetson23@ubuntuav:~/ros2_ws$ sudo systemctl daemon-reload
sudo systemctl enable ekf_localization.service
sudo systemctl start ekf_localization.service
Created symlink /etc/systemd/system/multi-user.target.wants/ekf_localization.service → /etc/systemd/system/ekf_localization.service.
avjetson23@ubuntuav:~/ros2_ws$ journalctl -u ekf_localization.service -f
Mar 24 18:59:02 ubuntuav systemd[1]: Started Start EKF Localization at Boot.
Mar 24 18:59:03 ubuntuav start_ekf.sh[10221]: [INFO] [launch]: All log files can be found below /home/avjetson23/.ros/log/2025-03-24-18-59-03-634716-ubuntuav-10221
Mar 24 18:59:03 ubuntuav start_ekf.sh[10221]: [INFO] [launch]: Default logging verbosity is set to INFO
Mar 24 18:59:03 ubuntuav start_ekf.sh[10221]: [INFO] [ekf_node-1]: process started with pid [10222]
```

Figure 41: EKF node

In this figure, we can conclude that the EKF localization node is working and we created a startup script that also runs the EKF localization node at boot. This helps us to come with a conclusion that if the GNSS and IMU rosbag recordings data are approximately working

properly, then the EKF node should also work correspondingly and it should give us a result like this if we plot it using pyplot.

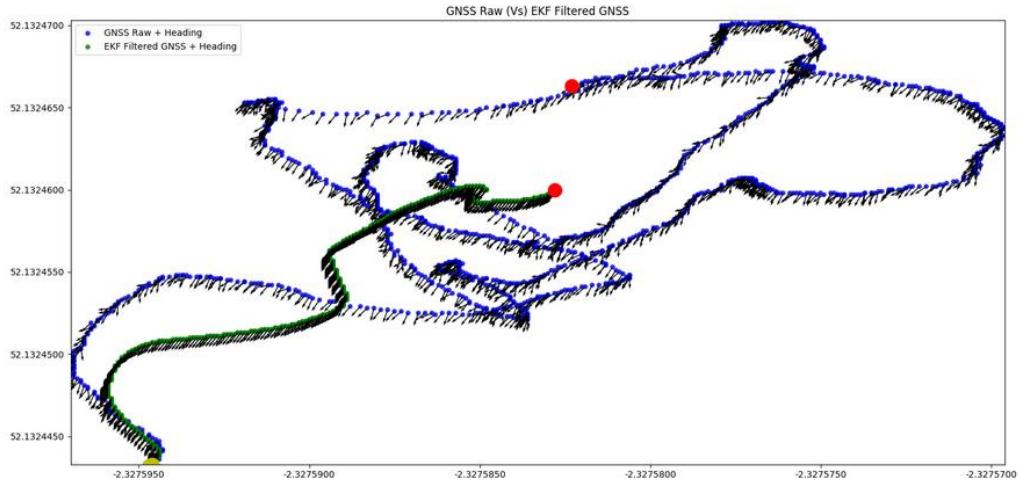


Figure 42 : raw vs EKF filtered

Overall, the results affirmed the effectiveness of our approach in realistic conditions and provided clarity on future development paths. Our findings underscored the necessity of multi-sensor integration for getting accurate localization, particularly when dealing with real-world environmental challenges. The insights gained from detailed data analysis and iterative testing helped our team understand precisely how different environmental factors and sensor behaviors impacted localization, ultimately guiding us toward significant system improvements.

Cameras

The updated robot URDF containing the new camera_link was implemented in simulation within a custom Gazebo world. A basic obstacle course existed within the simulation space which included basic walls and barriers arranged along a set path. The robot underwent manual movement through the course to confirm that the simulated camera displayed obstacles correctly within its field of view. RViz2 provided continuous monitoring of the TF tree structure and live camera feeds throughout simulated robot movements.

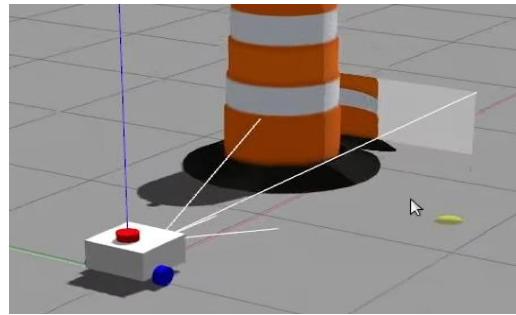


Figure 43a : .camera simulation

The Raspberry Pi cameras operated without robot chassis attachment by directly connecting to the Jetson AGX Orin hardware system. The main objective was to confirm fundamental camera operations through live video feed transmission and ROS 2 topic message publication. The v4l2_camera nodes published RGB and depth image topics which users monitored through RViz2. The cameras underwent static and small manual movements to replicate real-world camera movements while testing the stability of image streams through different environmental conditions. The hardware testing did not include any dynamic vehicle driving operations.

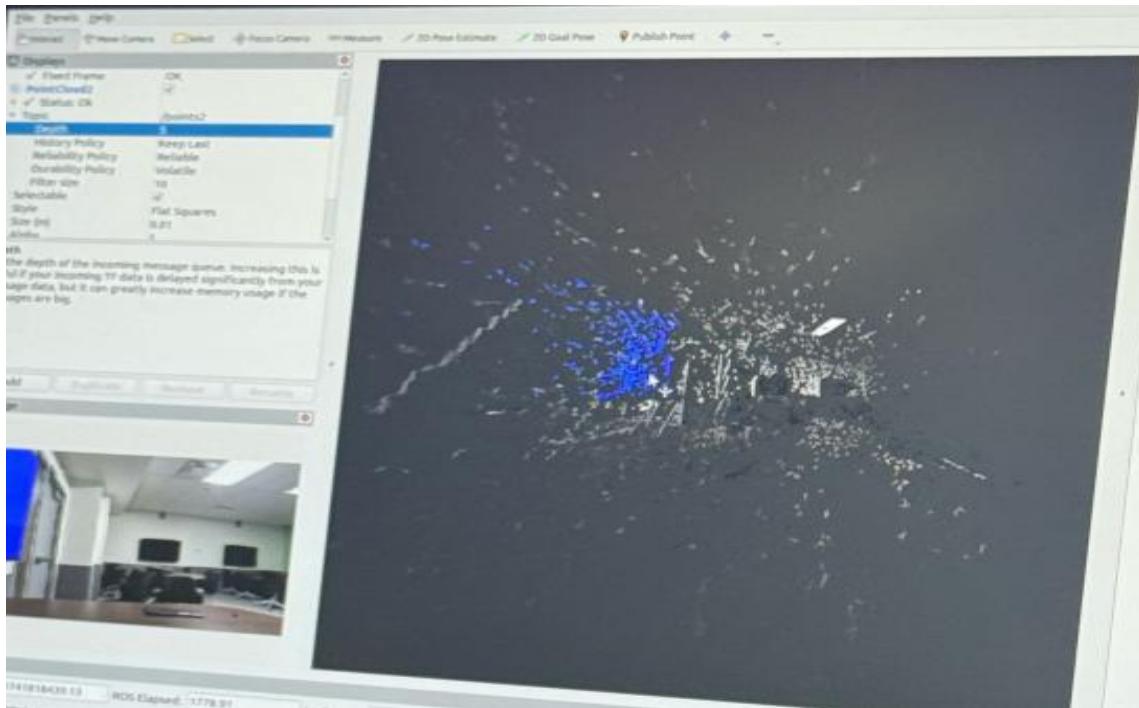


Figure 43b: rviz livefeed with depth

User Interface (Front End):

The results from our testing of the navigation system, both through simulations and real-world conditions, provided insights into the navigation's functionality and performance. These insights are listed below.

Simulated Environment Results:

- Leaflet Map Simulations: The simulations confirmed that our frontend accurately transformed GPS coordinates into visual markers on the map. The markers were placed with high precision, as verified with other open source geo-placing tools. This precision was crucial for ensuring that the navigation system would provide reliable and accurate guidance to users.
- Puppeteer Testing: Utilizing Puppeteer, a Node library, allowed us to automate interactions with the map and perform end-to-end tests effectively. The tests included automated clicks, drags, and zooms on the map to mimic user interactions.
- Mock ROS2 Data Testing: By creating a mock ROS2 node that emitted sample GPS data, we were able to test the web application's capability to handle live data updates. The React Leaflet map component successfully re-rendered in real-time whenever new data was received. This test was essential in demonstrating that our system could manage continuous data streams without high latency or disruption, maintaining smooth visuals .

User Interface (Backend Server):

During testing, the backend server running on the Raspberry Pi 4B demonstrated reliable performance across a variety of expected operating conditions. API response times were measured under both idle and active load conditions. When GPS data was actively being published via ROS2, the /position endpoint consistently responded within 45–70 ms. Under idle conditions (with no new GNSS data), the same endpoint returned cached or error responses in under 30 ms. These numbers were measured using Postman and time curl command-line testing over localhost.

The SQLite3 storage, used for recent and saved locations, performed without issue during normal read/write cycles. Insertion times for new location entries averaged around 4–6 ms, and lookup queries typically returned results in under 10 ms. No data corruption was observed during repeated Pi shutdowns and reboots, which confirmed that our local persistence setup was stable.

In terms of uptime, the Flask server successfully launched on boot using a systemd service. Over 5 power cycles were performed across different days, and the Flask service came online automatically in all instances without a manual restart. The average boot-to-ready time (from power-on to API being accessible) was approximately 65 seconds, largely dependent on the Pi's startup process and ROS2 node initialization.

When exposed to mock ROS2 GPS data published at 1 Hz, the backend was able to process and deliver updates to the frontend without delay or noticeable data loss. Even when update frequency was increased to 5 Hz, the Flask server remained stable, though minor lag was observed on the UI due to rendering limits, not server throughput.

Overall, the backend proved to be lightweight and dependable within the constraints of the Pi's hardware. The system handled API calls consistently, managed location data effectively, and maintained its role as a bridge between ROS2 and the user-facing interface without crashing or freezing—even during longer testing periods of 2–3 hours.

7.7. Discussion & Analysis

Camera:

The results of the Camera Navigation subsystem integration and testing were very close to the original project objectives. Simulation testing confirmed that the camera could be correctly modeled, positioned and visualized within the ROS 2 framework. The URDF modifications were successfully validated in Gazebo, with the simulated camera maintained a consistent pose relative to the robot's base link as the vehicle moved through the custom obstacle course. TF trees remained stable, and obstacle visualization in RViz2 matched expected camera perspectives. The Raspberry Pi cameras demonstrated reliable real-world hardware testing by publishing both RGB and depth image topics but in real time we had a lot of issue with depth calibration. The cameras showed good performance in basic testing scenarios but the frame rate and resolution needed to be carefully managed to avoid overloading the system's bandwidth and compute resources. The trade-offs highlighted the need for performance balancing when integrating multiple real-time sensor streams into a constrained embedded platform like the Jetson AGX Orin.

LiDAR:

Interpretation of test results:

Costmap testing with the VLP-16 lidar showed successful generation and live updating of the global costmap based on a single layer from the lidar's point cloud, however due to issues with

the transforms and with mapping and localization integration the static map and the local costmap updated only once when the navigation stack was initialized and remained static.

Comparison with expected outcomes.

The expected result was for both the local and global costmaps to update in real time, however this ended up not being the case.

Strengths and limitations of the system.

The main strength of using navigation 2 is that it is widely supported and easily configurable, though the limitations in this specific case are that using it with an ackerman steering vehicle requires a number of adjustments and custom code to interpret directions.

GNSS:

Interpretation of Test Results

Our field-testing of the GNSS-IMU localization subsystem provided significant insights of the system's performance under real-world conditions. During these tests, several critical observations were made. Under conditions—such as open areas with clear views of the sky—the GNSS and IMU data fusion achieved excellent accuracy, typically within half a meter of the vehicle's actual position. This success can largely be attributed to our careful implementation and fine-tuning of the Extended Kalman Filter (EKF), which effectively combined GNSS and IMU sensor readings into a coherent and smooth positional estimate.

However, the tests also highlighted limitations of the system under less ideal conditions such as indoors etc. Despite these GNSS disruptions, the EKF came into play by using IMU data to bridge short-term GNSS outages, ensuring smoother positional estimation than raw GNSS alone could provide.

Analysis of ROS bag files recorded during each testing session allowed a deeper examination of these phenomena. Through offline processing and plotting of GNSS and fused localization data, we confirmed that even small changes in the EKF parameters had significant impacts on the system's accuracy and stability. For example, increasing confidence in IMU measurements during periods of GNSS interference significantly reduced positional drift, providing critical evidence for the effectiveness of our sensor fusion approach.

Comparison with Expected Outcomes

The system's real-world performance aligned closely with our initial expectations and design goals. We anticipated that GNSS alone would be insufficient for reliable localization in challenging conditions and that the integrated IMU would play a crucial role in maintaining consistent localization accuracy. These assumptions were validated through our testing outcomes. The EKF effectively utilized IMU data to maintain stability during GNSS outages, thereby meeting our primary objective of reliable real-time localization.

However, certain outcomes differed slightly from initial expectations. We underestimated the degree to which prolonged GNSS outages (over several seconds) would result in IMU-driven drift. This drift became increasingly pronounced over extended periods without GNSS corrections. Initially, we expected the IMU to bridge these gaps more effectively, but tests showed that sensor biases and inherent noise significantly impacted accuracy over prolonged intervals.

Strengths of the System

One clear strength of our localization subsystem was its excellent real-time performance and modular design. The integration of GNSS and IMU data through ROS 2 provided seamless, real-time position updates. The Extended Kalman Filter reliably fused sensor streams, offering robust performance even under short-term GNSS disruptions. Furthermore, the modular architecture allowed each sensor and processing node to operate independently yet cohesively, simplifying debugging and future component upgrades.

Another significant advantage was the real-time monitoring and visualization capability provided by RViz. Visualizing GNSS raw data points alongside the EKF-filtered odometry allowed immediate identification of anomalies such as sensor drift, misalignments, or GNSS dropouts. Coupled with remote monitoring via VPN and SSH, these tools considerably enhanced the efficiency and effectiveness of our field-testing sessions.

Additionally, the automated GNSS data streaming setup through our custom Linux systemd service (`gnss.service`) proved exceptionally reliable, reducing setup errors and providing consistently repeatable test conditions. This automation greatly simplified test procedures, allowed quicker turnaround between tests, and increased overall reliability during the project.

Limitations of the System

Regardless of the clear strengths, our system had notable limitations primarily related to sensor characteristics and environmental interference. The principal limitation observed was the system's ability to GNSS multipath data reading errors and signal interruptions when we use it indoors. Under such conditions, the GNSS module frequently lost accuracy, and although the

IMU integration was used right after it , waiting for a longer amount of time resulted outages inevitably led to cumulative IMU-based drift.

Another limitation is from the IMU sensor itself. Consumer-grade IMUs, such as the one integrated with the u-blox ZED-F9R, inherently suffer from biases and drift over extended operation periods. This became evident during extended GNSS interruptions, causing gradual positional inaccuracies. Thus, we concluded that higher-quality IMUs or additional sensors (such as wheel encoders or visual odometry) would likely improve long-term localization performance.

Additionally, the tuning of the Extended Kalman Filter proved more sensitive than initially anticipated. Slight changes in EKF parameters significantly impacted the accuracy and responsiveness of localization estimates, requiring substantial iterative tuning efforts. This sensitivity demonstrated the importance of a carefully managed testing and validation process.

User Interface (Front End)

Interpretation of Test Results:

The testing phases for our navigation system, both simulations and real-world scenarios, have demonstrated that the system performs as optimal under varied conditions. The high accuracy in transforming GPS data into visual markers and the system's responsiveness in updating these markers in real-time validates our design decisions and technical approach. The use of web testing tools further helped us solidify our understanding and ensure the system reacts as expected.

Comparison with Expected Outcomes:

The experimental results are closely aligned with our expected outcomes. We anticipated a high degree of accuracy in GPS tracking and an interface that could refresh in real-time without any noticeable lag to the user. Although we had intended to rerender the application at a faster rate and were not aware of the rendering time limitations, the end result surpassed our expectations. The simulated environment tests confirmed the system's capability to handle live data efficiently, and the real-world tests on Carleton's campus further reinforced its operational reliability and effectiveness in a practical setting.

Strengths of the system:

- High accuracy and fluid like responsiveness
- User-centric and intuitive interface design
- Robust performance in a variety of conditions
- Limitations of the system
- Scalability concerns regarding longer routes and higher storage demands

- Dependency on stable network conditions for some features
- Limited testing in some adverse conditions and scenarios

User Interface (Back End) :

The backend server demonstrated solid and predictable behavior under a range of real and simulated conditions. Its ability to maintain stable API performance especially with active GPS data streaming at 1 Hz and even up to 2 Hz, showed that the Flask framework was a good fit for the system's lightweight data-handling needs. The SQLite3 database also held up well, with quick access times and no signs of data loss, even across repeated power cycles. What stood out most was the server's reliability during startup: it consistently launched on boot without needing any manual input, which is critical for an embedded system like this. While we observed a slight delay in map updates at higher data rates, the bottleneck was clearly on the frontend rendering side and not the server itself (It's a raspberry pi so the performance gets choked in high traffic situations). These results confirmed that the backend was doing its job quietly and efficiently in the background, just like we designed it to.

7.8. Conclusion & Future work

7.8.1. Summary of Key Achievements

The Navigation module achieved a range of milestones across both hardware and software systems, all aimed at advancing the vehicle toward semi-autonomous operation. While the team encountered technical hurdles and setbacks in coordination, substantial progress was made in localization, user interface development, system integration, and perception.

The Raspberry Pi 4B served as the onboard computing platform for the interface and backend server, drawing power directly from the vehicle via GPIO pins. The team designed and 3D-printed a dashboard mount to hold both the Pi and its touchscreen display, keeping the setup compact and accessible. Startup scripts were added so that the Flask server and React web application would launch automatically on boot, removing the need for manual intervention. The Pi subscribed to GNSS data published over the ROS2 network, feeding live position updates to the UI. A lightweight storage system was also added to the Flask server for managing simple data. Several REST APIs were developed to enable interaction between the frontend and backend, and the touchscreen's input coordinate mapping was resolved during testing. The Flask server was also made to function as a ROS2 node, communicating with the rest of the robotic system entirely through REST.

In terms of GNSS and localization, the team developed a real-time pipeline that fused GNSS and IMU data using an Extended Kalman Filter (EKF) within ROS2. A custom service handled data publishing, and subscribers were validated to ensure consistent reception. Launch files were used to automate node execution, reducing manual setup during field tests. Using QGIS and ROS tools, the team built and refined data occupancy grid maps, which were then loaded in RViz2 and scaled to match real-world distances.

For the camera subsystem, both simulation and physical setup were completed. The camera's placement was defined in a URDF model and verified through Gazebo and RViz2. Two Raspberry Pi cameras were mounted using custom brackets, allowing image data to be collected during tests. These cameras streamed RGB and depth feeds through ROS2 and were also recorded with rosbag2 for later review. The setup maintained a modular structure that can be expanded on in future development.

The web application loaded a live map on boot using Leaflet.js and OpenStreetMap. A location marker was displayed based on real-time GNSS data. Most of the backend REST APIs were integrated into the frontend, enabling users to input destinations, view paths, and access diagnostics. The web interface also allowed interaction with the server's storage, supporting basic read/write operations.

Finally, the team implemented a partial 2D costmap using LiDAR data and the Nav2 stack. While path planning and motion control integration were not completed, the costmap accurately displayed obstacles and was visualized in RViz2, validating its functionality in static conditions. Although the final system had limitations, the project produced a functional and extensible foundation. The core services, integrations, and testing infrastructure developed this year provide a reliable starting point for future work on the autonomous platform.

7.8.2. Lessons Learned

This project drove home a number of points that extend well beyond the scope of our coursework. Working with physical hardware and integrating various software systems, particularly ROS2, exposed us to challenges we hadn't fully anticipated, many of which could only be learned through hands-on trial and error.

An interesting observation made showed the incredible sensitivity of autonomous systems to seemingly minor details. Small errors in our URDF files or TF tree configuration led to noticeable navigation issues, especially when dealing with localization. Tuning the Extended Kalman Filter (EKF) required a deeper understanding of sensor noise and timing discrepancies than we expected. We learned that even small misalignments between sensor data and frames can compound into larger system-level problems.

We developed a more grounded understanding of how vehicles interpret their surroundings. Processing raw LiDAR and GNSS data into usable forms like occupancy grids or costmaps wasn't just about technical correctness, it was about ensuring the system could respond predictably to real-world inputs. The work forced us to think critically about how data flows through a navigation system and what assumptions we could safely make.

Our experience with the Linux environment, especially Ubuntu was also a rather unique learning experience. It is amazing how many open source softwares are available for the Linux environment and how straightforward it can be to work with them. We became comfortable working with the ROS2 toolchain, managing launch files, debugging nodes, and monitoring live data through RViz and CLI tools.

During testing and integration, system resource issues surfaced, particularly while streaming camera data or running multiple high-bandwidth topics. These bottlenecks pushed us to monitor resource usage more actively and adjust frame rates or message sizes where needed. We also ran into version compatibility problems with Gazebo, which emphasized the importance of locking versions and maintaining backups for simulation environments.

One of the more useful strategies we adopted was designing our software in a modular fashion. Breaking the system into separate, testable components—such as the Flask server, ROS2 nodes, and UI, helped us isolate issues quickly and move forward even when one part of the system was misbehaving. This saved us significant time during debugging and system updates.

From a team standpoint, we learned how crucial communication and accountability are in group work. While things started well, as the project progressed, a lack of consistent follow-up and ownership led to delays. We eventually adapted by redistributing tasks and running tighter check-ins, but we recognized how much smoother things could have gone with better structure early on.

Power inconsistencies, startup sequences, and inter-device communication delays all introduced bugs that we never saw in Gazebo. These challenges taught us to be more patient, flexible, and prepared for surprises during real-world testing.

Overall, this project gave us a clear sense of the complexity involved in deploying even a partially autonomous system. It also gave us tools and habits, like modular development, rigorous testing, and better team coordination, that we'll carry forward into future work.

7.8.3. Recommendations for Future Development

To carry this project forward, we've identified several areas that could benefit from continued development across the navigation module.

GNSS and Localization

- Add support for wheel encoders and consider integrating an RTK base station to help maintain localization accuracy when GNSS signals are weak or unreliable.
- Experiment with more advanced fusion strategies that can better combine data from GNSS, IMU, and potentially other sensors.
- Plan field testing in a wider variety of environments—including areas with limited sky visibility or high interference—to test the system's limits.
- Explore combining global map data with LiDAR-based SLAM to build a hybrid mapping setup that adapts to real-world navigation scenarios.

Camera System

- Expand the use of camera data to include visual odometry, obstacle detection, and object tracking.
- Move beyond the basic Raspberry Pi cameras and test stereo or depth-sensing systems like Intel RealSense or ZED 2 to improve how the vehicle understands its surroundings.

Raspberry Pi Server and UI

- Adjust the current API structure to use POST requests where appropriate, instead of relying only on GET methods.
- Add WebSocket or AJAX functionality to make live data—like speed and location—update more smoothly in the browser.
- Embed the diagnostics dashboard (created by the Safety and Reliability team) directly into the interface.
- Include a software-controlled emergency stop (e-stop) within the web interface for immediate safety control.
- Implement full global route generation, and make sure it's published over ROS2 so the local navigation system can use it.
- Finish the remaining parts of the user interface, including better controls for destination input and clearer system feedback.
- Refine the on-screen keyboard for a smoother touchscreen experience.
- Consider upgrading to a larger display and a newer Raspberry Pi model if performance or screen space becomes a bottleneck.
- Update the dashboard mount to fit additional components like the power supply PCB.

Navigation Stack (Nav2)

- Add support for sending movement commands in the form of `Twist` messages to the control system, allowing the vehicle to act on planned paths.
- Improve path planning behavior so the system can react in real time to obstacles or changes in its surroundings.

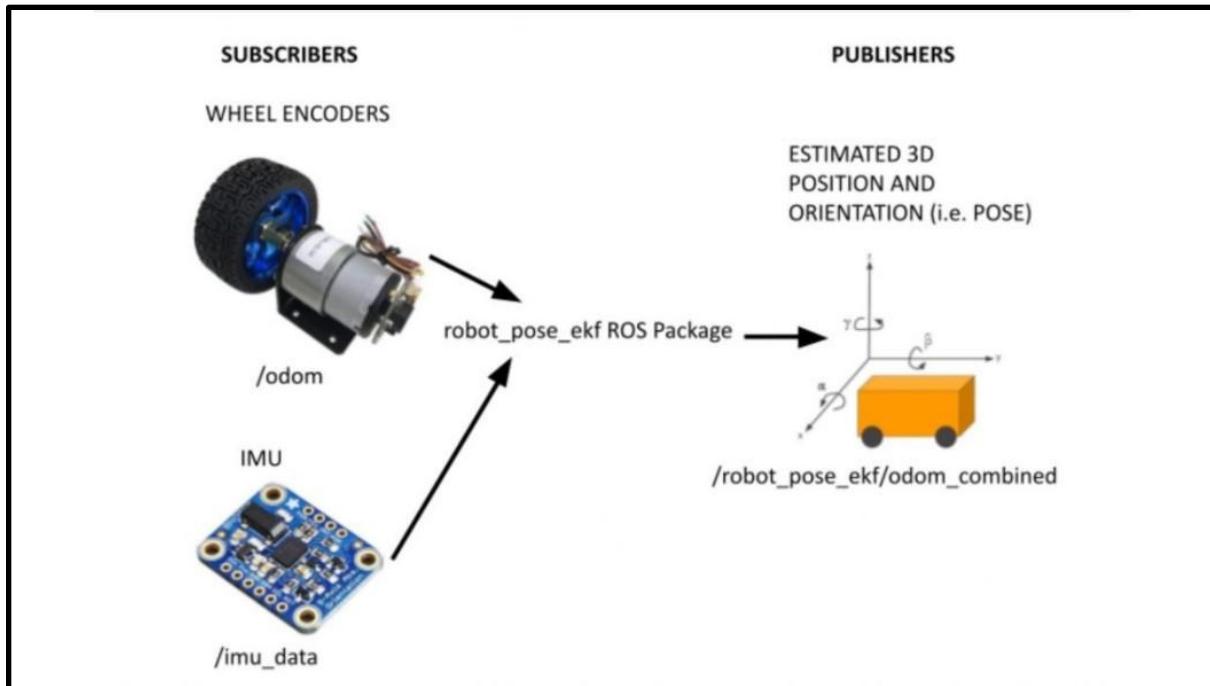


Figure 44: Wheel encoder odometry plus inertial measurement unit (IMU) sensor fusion [NAV24]

7.9. Reflections

7.9.1. Original Project Proposal & Changes Over Time

The original project proposal laid out a clear goals for building a navigation module that would integrate route plotting, localization, and real-time mapping through a combination of ROS 2 tools and hardware interfaces. While many of the original goals were met, several changes and course corrections occurred along the way, influenced by technical constraints, evolving priorities, and available resources.

GNSS and Localization:

Initially, the system was expected to rely solely on GNSS for localization. Early testing, however, made it clear that signal instability and temporary outages, especially around buildings or in lab environments, would compromise reliability. This led to a shift in approach, with the integration of IMU data and the use of an Extended Kalman Filter (EKF) for sensor fusion. This change required a deeper understanding of ROS 2's robot_localization package and led to improved pose estimation under inconsistent signal conditions. The GNSS node was successfully publishing data, the EKF was properly configured, and both were integrated into the system as expected.

Path Planning and Mapping:

Originally, Nvidia Isaac's Nvblox was selected for local mapping and path planning. After further evaluation, the team opted to use slam_toolbox in combination with a 2D costmap setup, which was better supported and more accessible for the project's timeline. The proposal also called for a hybrid approach involving both global mapping and local SLAM, but due to time limitations and added complexity, efforts were focused on generating a reliable local costmap using LiDAR. A partial costmap was successfully produced and visualized in RViz; however, conversion of Twist messages into Ackermann steering commands, required for true hardware compatibility could not be completed.

Cameras:

The original scope for camera integration was limited to adding a single camera for visual input in simulation. As we progressed further into the project, camera integration expanded to include URDF configuration, TF frame alignment, and Gazebo-based simulation validation. Hardware testing became part of the scope after initial simulation success. ROS 2 node configuration difficulties and Gazebo compatibility issues slowed this transition, but steady progress was made in stabilizing camera inputs. Hardware-related efforts were primarily supported by the Sensors team, with Navigation assisting in simulation and transform tuning.

Display and User Interface:

The proposal outlined five interface wireframes for the vehicle's UI. Of those, three were implemented, including a live map interface with vehicle positioning, a destination search bar, and a basic UI display. The GNSS feed was successfully integrated into the frontend using RESTful APIs through Flask, allowing the user to view real-time location updates. However, the proposed functionality for global route generation was not achieved. Despite this, the completed UI components provided a working foundation for real-time human-vehicle interaction.

7.9.2. Team Reflection on Challenges & Successes

Throughout this project, the team faced a number of technical and administrative challenges that tested our problem-solving skills, adaptability, and communication. Signal interruptions and

IMU noise were persistent issues during the localization phase. These were addressed through collaborative troubleshooting and careful tuning of the Extended Kalman Filter parameters, ultimately leading to consistent and reliable localization in outdoor environments. We were also able to establish a semi-functional remote monitoring or observation setup that supported real-time feedback from the vehicle during tests, a milestone that increased our confidence in real-world system deployment.

One of the major technical challenges was attempting to integrate both global and local mapping systems into a unified hybrid navigation pipeline. While we saw some success in having built and tested a local costmap of the environment, integrating it with the local LiDAR-based SLAM pipeline proved too complex within the available timeframe. The generation of the global route failed completely due to a threading issue, which likely arose because of a weaker compute on the Raspberry Pi. This challenge highlighted the need to make informed scope decisions, and we eventually chose to focus on refining the webapp and backend aspect of the dashboard system rather than route generation. Learning when to pivot was a critical part of keeping the project on track.

We also faced issues with URDF misalignments, exposure control for cameras, and ROS 2 communication bugs, particularly during multi-node testing. These complications often slowed progress, but they also taught us to be more methodical in debugging and to value consistent documentation practices. The Gazebo simulator update that dropped out of nowhere mid-semester caused further great chaos by breaking compatibility with our world files and plugins. In response, we developed a habit of version-locking dependencies and backing up simulation environments regularly, habits that proved invaluable as the project grew in complexity.

Challenges also arose during the attempt to implement Navigation 2 for an Ackermann steering vehicle, which is natively designed for differential drive systems. Despite extensive experimentation, we were unable to generate a traversable path using our costmap setup. While we did manage to generate a local costmap from LiDAR data, the final step of converting Twist messages into Ackermann-compatible motion commands was not completed. This outcome reflected the practical limitations we encountered with time and resource constraints.

Beyond technical hurdles, the team grew significantly in terms of collaboration and workflow. Early in the semester, unclear task divisions and miscommunications led to delays. However, we gradually adopted more structured practices, including weekly sync-ups and shared documentation, which improved our coordination. As the project progressed, we became more effective at balancing individual contributions.

Finally, frequent testing in both simulation and real hardware taught us how configuration mismatches and minor inconsistencies, such as coordinate frame offsets, could result in system-

level failures. This reinforced the importance of rigorous testing, tight version control, and validating every configuration change across all layers of the stack.

In the end, while not every goal was fully realized, the project provided a rich learning environment. It sharpened our ability to navigate uncertainty, work as a team, and apply engineering principles under real-world constraints, lessons that extend far beyond the scope of this capstone.

7.9.3. Hindsight Analysis: What would we do Differently?

Looking back, there are a number of things we would approach differently if we had the chance to do this project again.

On the technical side, we should have started field testing much earlier. GNSS signal dropouts and localization drift were issues we only fully understood late in the project. If we had caught them sooner, we could have planned around them and tuned the system more effectively. We also didn't give enough time to properly dial in the EKF or look into RTK corrections, both of which would've improved the consistency of our position estimates. Although we managed to generate global maps, integrating them into the rest of the system ended up being more complicated than we expected, especially with limited time and coordination.

Camera integration also came with its own problems. When Gazebo pushed a major update, it broke compatibility with some of our simulation setups. At that point, we realized we should've locked versions earlier and kept backups of working environments. We also didn't budget time for performance testing—things like CPU usage and bandwidth monitoring—which left us guessing about some of the system's limits until they became bottlenecks.

But the biggest challenges weren't technical—they were internal. Things started off fairly well, but about halfway through the semester, team coordination started to break down. Some members stayed active, but others pulled back, and that imbalance started to wear on the group. We had a Gantt chart and timelines on paper, but they weren't followed. A lot of the planning, communication, and documentation ended up on just a couple of people, and that uneven workload led to burnout.

We tried to course-correct in the final month—redistributing tasks, running tighter meetings, and trying to get back on track—but by then, there just wasn't enough time left to recover fully.

If we could do it over, we'd focus more on building better team habits from the start: clearer ownership of tasks, regular check-ins that actually meant something, and better documentation so that everyone could contribute without feeling lost. The technical challenges were tough, but it was the coordination issues that really held us back in the end.

8. Autonomous Control

8.1. Introduction

8.1.1. Background & Motivation

As autonomous vehicles advance and their capabilities improve, the need for their operation to be safe, precise, and reliable becomes an essential prerequisite—especially when the goal is to aid those with physical disabilities. The autonomous control system acts as the underlying component of vehicle operation, being responsible for the interpretation of high-level commands and the translation of these instructions into low-level actuator commands for throttle, steering, and braking.

In the view of our research team, the significance of this project is the development of a control system that is not only technically sound but also versatile enough to be incorporated in a number of other subsystems such as path planning, localization, perception, and safety protocols. The vehicle should be capable of adapting to road environments, safe movement, and carrying out movement instructions autonomously, without any manual intervention from the driver. This requires robust software architecture that supports modularity, real-time performance, and system integration.

We leverage the Robot Operating System (ROS) to create a standardized communication system that bridges the vehicle's actuators and higher-level vehicle control commands. Our control system is developed to facilitate simulation-based development in addition to deployment on physical hardware, enabling ongoing testing, validation, and refinement.

Additionally, the inclusion of safety features, such as emergency stop integration and manual override function, are key responsibilities of our team to ensure that the vehicle can be safely stopped or controlled under critical situations.

8.1.2. Problem Statement

The problem of the Autonomous Control Team is to design a control interface whereby autonomous and remote systems can reliably control the movement of the vehicle. This involves managing low-level control of actuators responsible for throttle, braking, and steering, and being consistent with a modular, ROS-based software architecture.

The system must enable real-time communication between subsystems, provide deterministic and safe command execution, and be equipped with fail-safes such as emergency stop (E-stop) and manual override control. The control system must also provide its state to simulation and real-world environments to enable iterative development and testing.

In addition, our team will offer wireless connectivity through the use of a cellular network interface, which will enable remote access to the vehicle. This will enable real-time monitoring, remote debugging, and system control.

The challenge is to design and implement a control layer that is robust, responsive, and tightly coupled to the entire vehicle system, forming a dependable platform for autonomous operation.

8.1.3. Objective & Scope

The primary objective of the Autonomous Control Team is to implement the vehicle's basic motion control system and provide standardized ROS interfaces for integration with other subsystems. Our teams objectives are to:

- Controlling the vehicle motion using a brushless DC motor and motor controller to propel forward and backward. This requires motor interfacing, tuning, and velocity control at a stable level.
- Steering and braking control through linear actuators. This involves translating steering and brake commands into actuator signals with accurate positioning and safety assurances.
- Subscription to ROS control commands, i.e., Ackermann-style messages from external path planning modules, and their real-time execution.
- Translation of high-level commands into low-level hardware signals, i.e., PWM or digital outputs, for direct control of motors and actuators.
- Publishing the vehicle information (e.g., speed, steering angle, brake status) to ROS topics for use by remote monitoring, simulation and navigation subsystems.
- E-stop safety management through integration, where it responds to emergency stop requests from another subsystem by commanding the vehicle to halt immediately and engaging the brakes.
- Manual override using an RC controller interfaced via GPIO. This allows a user to take direct control when needed, with safe mode switching.
- Implementing speed sensing using the appropriate sensors to measure and report vehicle speed for closed-loop control and system feedback.
- Implementing cellular connectivity to enable remote monitoring, diagnostics, and limited control over the vehicle network.
- Designing support components, such as PCBs for control electronics and 3D-printed hardware mounts for integration.

8.1.4. Methodology Overview

Our group follows a methodology based on the foundations of solid system design from the beginning of the project. By getting well-defined goals early on, we aim to build a solid and stable foundation that supports long-term evolution. To achieve this, we put great

importance on modular design philosophy, which allows separate subsystems to be developed, tested, and improved independently. This modularity permits flexibility in software and hardware and incorporates new components and features without adding to the overall system. Our methodology is closely aligned with Agile principles, especially in its focus on adaptability, iterative development, and continuous improvement over the project life cycle.

8.2. Engineering Requirements & Justification

8.2.1. Justification for Relevance to Degree Program

This project aligns directly with the objectives of the Computer Systems Engineering degree, allowing us the opportunity to deploy and integrate our knowledge from various areas, such as system design, electronics, software development, and networking. Control system design for an autonomous car involves a thorough understanding of real-time embedded systems, hardware-software co-design, and communication protocols; skills central to the program.

We participate in system-level design with the architecture of modular control subsystems, safety, and ROS-based communication interfaces. The project requires the development of electronic hardware such as microcontrollers, motor controllers, speed sensors, custom printed circuit boards (PCBs), and manual override integration, reinforcing concepts from digital systems and electronics classes.

From a networking perspective, the addition of a cellular communication module for remote vehicle monitoring and control provides hands-on experience with wireless connectivity, IP networking, and secure system access subjects growing in significance in embedded and IoT designs today.

The project replicates a real engineering development cycle that entails collaborative working, requirements analysis, cyclical testing, documentation, and project management. It promotes interdisciplinary thinking and provides students with hands-on experience in managing complicated real-world engineering problems bridging the gap between academic theory and professional practice successfully.

8.2.2. Engineering Principles Applied

This project encompasses a full complement of engineering principles that are developed over computer systems engineering, such as embedded development, electronics, communication protocols, and system integration. The control framework is developed around an ESP32 microcontroller operating on FreeRTOS, facilitating precise real-time operation and concurrency, which are fundamental to motor control and sensor interface. Higher-level communication to the network are managed on a Raspberry Pi providing

modularity with the use of Docker containers, thus allowing for a modular and isolated deployment of software components, including ROS-based communication nodes. Device-to-device communication is achieved using I²C and UART protocols, requiring the team to manage timing, synchronization, and bus-level contention. The creation of custom printed circuit boards (PCBs) plays a key role in the integration of sensors, actuators, and supporting circuitry, involving principles of circuit layout, power distribution, and signal routing. A modular ROS architecture is followed by the team to define clean interfaces among subsystems, thus making it scalable and ensuring separation of concerns. Networking concepts are realized through the incorporation of a cellular module, which provides secure remote access, monitoring, and debugging via IP. Safety features are integrated into the system as a whole, such as the processing of an external emergency stop signal and also fault-tolerant transitions between autonomous and manual modes of control. Together, they represent the practical application of embedded systems, electronics design, signal processing, networking, and real-time software engineering in a single, real-world engineering project.

8.2.3. Health and Safety Consideration

Safety and health were given high priority at all times during the project, particularly due to the hands-on exposure of the activities with electronics, mechanical equipment, and physical testing of a powered vehicle. All the team members successfully underwent the required Basic EDC Bay 4 Safety Training, which qualified them to work independently in Bay 4. Training included essential procedures on working tools, electrical system handling, and maintaining a safe working environment for mechanical tools.

Work at Bay 4 was done in accordance with institutional safety procedures, including the use of appropriate personal protective gear (PPE), secure storage and management of batteries and electrical units, and maintaining a clean and organized work space. When performing activities such as soldering, drilling, or operating mechanical components, team members put on safety glasses, heat gloves, and other PPE when appropriate.

During the vehicle testing, strict safety protocols were followed. The vehicle was operated only in test beds with a clearly defined safety boundary. Emergency stop capability and manual override features were tested and verified before any real movement tests. Additionally, power supplies were turned off while updating the software or performing hardware modifications to prevent accidental motor or actuator activation.

By following these protocols and working within approved laboratory facilities, the team ensured that all development and test work was performed safely and ethically.

8.2.4. Ethical Considerations

Though ethical consideration was not the target of the project, basic issues such as safety, accessibility, and environmental impact were considered. No personal or sensitive information was collected and utilized for development. Reusable and modular components were utilized to develop the system in order to reduce material loss, and electric propulsion favors smaller environmental prints. In addition, the goal of the project to improve mobility for the disabled is aligned with broader values of accessibility and social benefit.

8.2.5. Regulatory & Standards Compliance

Although the vehicle is not made for public use, relevant standards were considered in development. Software design took inspiration from the principles of ISO 26262 to guide safety-critical aspects like emergency stop and manual override. Industry standard protocols such as I²C, UART, and Ethernet between controllers were used for robust communication. ROS was chosen in order to provide modularity and adhere to conventional robotics frameworks. Institutional lab safety practices were followed in all the work.

8.3. Literature Review & Related work

8.3.1. Overview of Existing Technologies & Techniques

The previous iteration of the project was primarily developed by a Mechanical Engineering department team, wherein the main focus was on the mechanical design and construction of the autonomous vehicle platform. Their work yielded a specially designed chassis customized for actuator integration, also providing structural support for steering, braking, and propulsion units. The team devised a complete mechanical steering system through the application of a linear actuator to provide motion to the front wheels and an equally embedded linear actuator to supply braking force. The existing brushless DC motor was adapted to be digitally controlled for acceleration, complemented by a mounting system and drivetrain that guaranteed stability and long-term endurance on motion.

At the systems level, the group introduced key electrical and embedded components to enable basic motor and actuators control. They developed a basic embedded system that provided manual and semi-automatic control features, allowing users to send commands through direct hardware interfaces. The control system functioned but was primitive in nature and was primarily designed to test the mechanical systems and basic movement and not completely autonomous operation. Communication software and modularity were kept low, with much of the logic being implemented on a single microcontroller.

The team also conducted a series of test validations to establish the performance of the mechanical systems under load. Testing included linear actuator response times, steering angle range, braking, and motor acceleration. These tests confirmed that the central mechanical systems were working and were capable of responding to control inputs in a reliable manner. Though accuracy and control smoothness were restricted, the findings demonstrated that the platform was structurally and functionally able—waiting to be outfitted with more advanced electronics, control logic, and autonomous software.

Overall, the earlier team did a good job of establishing a good mechanical and electromechanical basis for development. Their approach emphasized modularity and accessibility so that subsystems could be replaced or upgraded without extensive redesign. While full autonomy and system integration were outside their mandate, what they did was a solid base on which later teams could build more advanced control systems, autonomous navigation, and safety-critical functionality. They provided a solid foundation for this year's group to improve upon.

8.4. System Design & Implementation

8.4.1. System Overview & Architecture

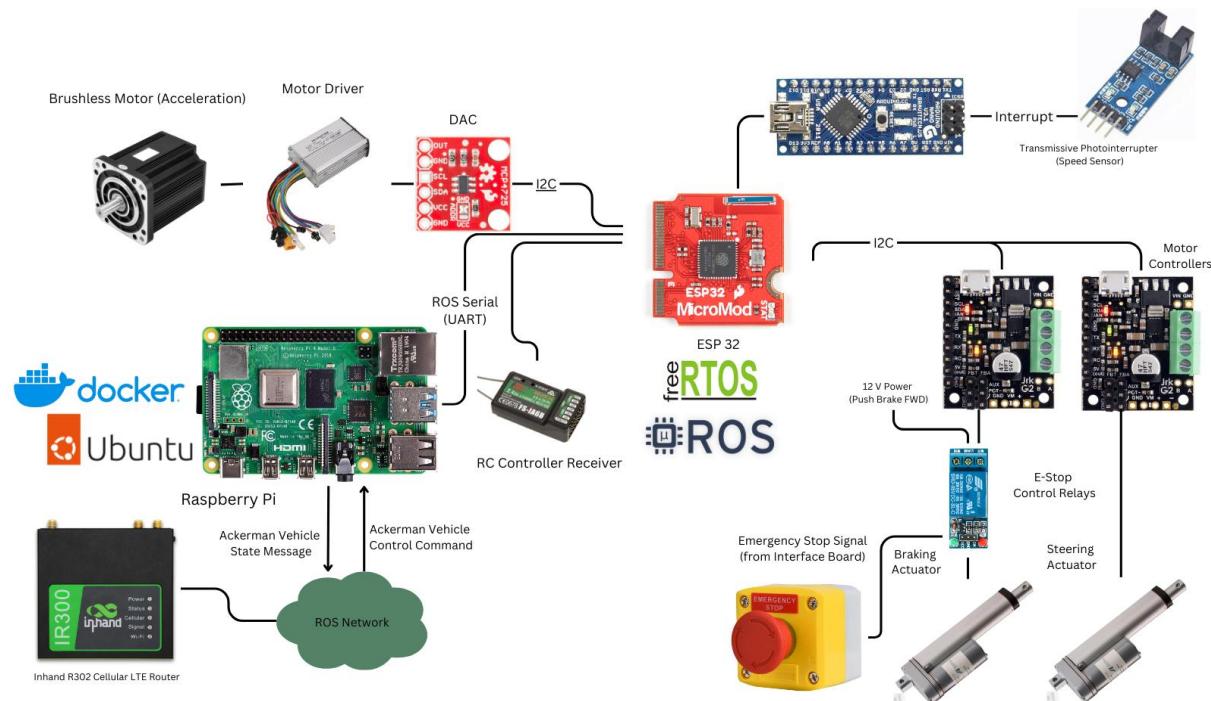


Figure 45: System Architecture for the Autonomous Control System

The Autonomous control system was designed with modularity in mind. Like other groups, ros2 was used in order to integrate the system with the rest of the car. The system is built around a custom-designed PCB that integrates all core electrical components required for low-level autonomous vehicle control. At its center is an ESP32 microcontroller running

FreeRTOS, responsible for real-time management of actuator commands and sensor feedback. It communicates with a Raspberry Pi 4 over UART via the MicroROS implementation of Micro XRCE-DDS. The Pi runs ROS 2 within Docker containers to handle high-level decision-making and networking and provide significant computational headroom for future implementations of dynamics controllers or safety systems. The ESP32 interfaces with motor controllers for steering and braking via I2C and controls the main drive motor through a DAC that converts digital throttle signals into analog voltages. CAN bus connectivity enables communication with additional subsystems and sensors, while an SPI link between the CAN controller and Raspberry Pi supports high-speed data exchange and CAN access via the linux kernel. This architecture ensures deterministic control, modularity, and seamless integration between embedded hardware and ROS-based software.

8.4.2. Requirements Definition

Functional Requirements

Vehicle Movement Control

Vehicle movement must be enabled using existing brushless motor and steering/braking using linear actuators.

ROS2 Integration

The system shall subscribe to ROS2 topics (e.g., Ackermann messages) to receive vehicle control commands.

Feedback-Based Control

System should provide feedback based control to ensure accurate control of braking and steering

Vehicle State Publishing

The system shall publish real-time vehicle status data (e.g., speed, actuator positions) to ROS2 topics for use by other subsystems.

Emergency Stop Handling

The system shall respond to a digital E-stop signal by immediately halting the vehicle and entering a safe state.

Manual Override Control

The system shall accept manual control input from an RC controller or other reliable non network based solution.

Speed Sensing

The system shall measure vehicle speed using onboard sensors and make this data available to the control loop and ROS topics.

Wireless Connectivity

The system shall enable remote monitoring and diagnostics via a cellular-connected router.

Non Functional Requirements

Real-Time Performance

The system shall respond to incoming ROS messages and sensor inputs with minimal latency to support safe vehicle operation.

Modularity

The control architecture shall be modular to allow easy replacement or upgrade of individual components (e.g., sensors, actuators).

Reliability

The system shall operate reliably under extended runtime conditions and recover safely from transient faults.

Safety

All control logic shall prioritize safe operation, especially in the presence of conflicting or invalid inputs (e.g., both E-stop and manual override triggered).

Scalability

The control system shall support future expansion, such as adding more actuators or sensors with minimal redesign.

Maintainability

The software and hardware design shall be well-documented and organized to support future debugging, updates, or handover.

Compatibility

All software components shall be compatible with ROS2 and the FreeRTOS task scheduler running on the ESP32.

Startup Behavior

On startup, the system shall initialize in a known safe state and not engage any actuators until valid commands are received.

8.4.3. Hardware Design

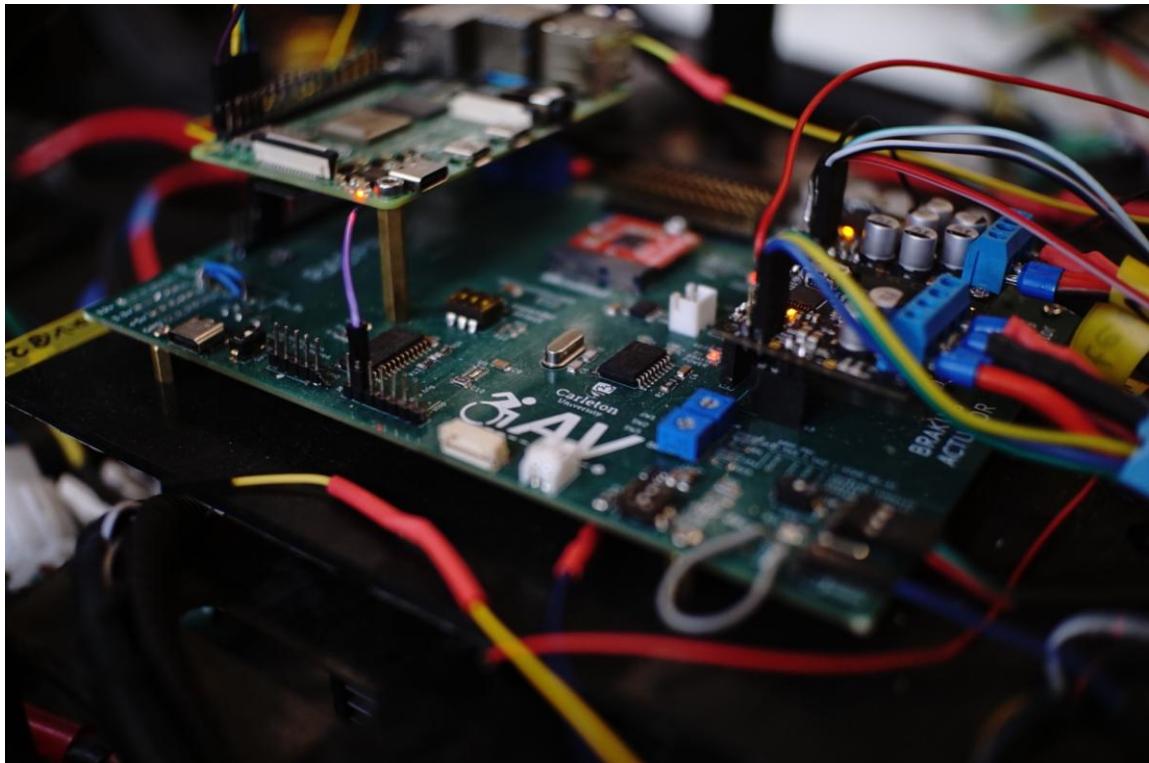


Figure 46: Autonomous Control full circuit mounted on vehicle

In order to design the hardware at the heart of the Car a variety of different techniques were used. The following list outlines each hardware component and its integration with the rest of the system and any specific techniques we used to design or implement them.

1. PCB

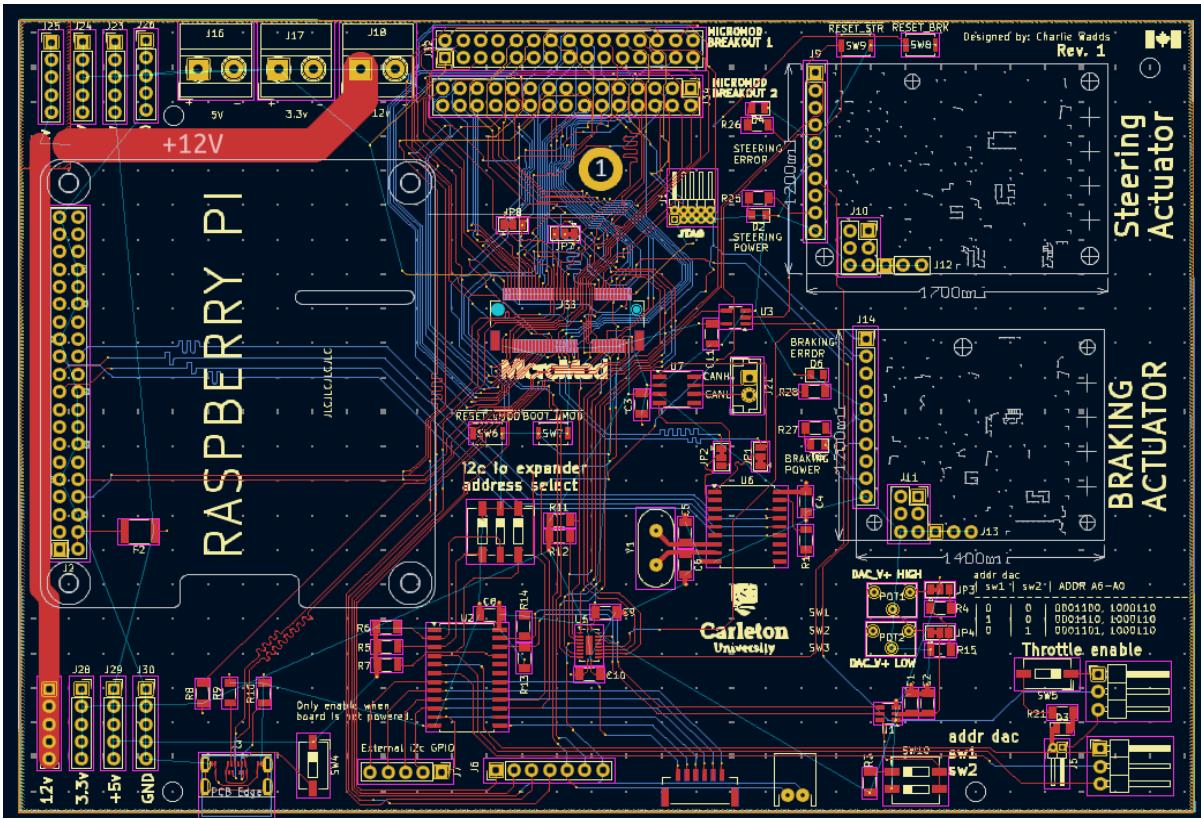


Figure 47: PCB Schematic

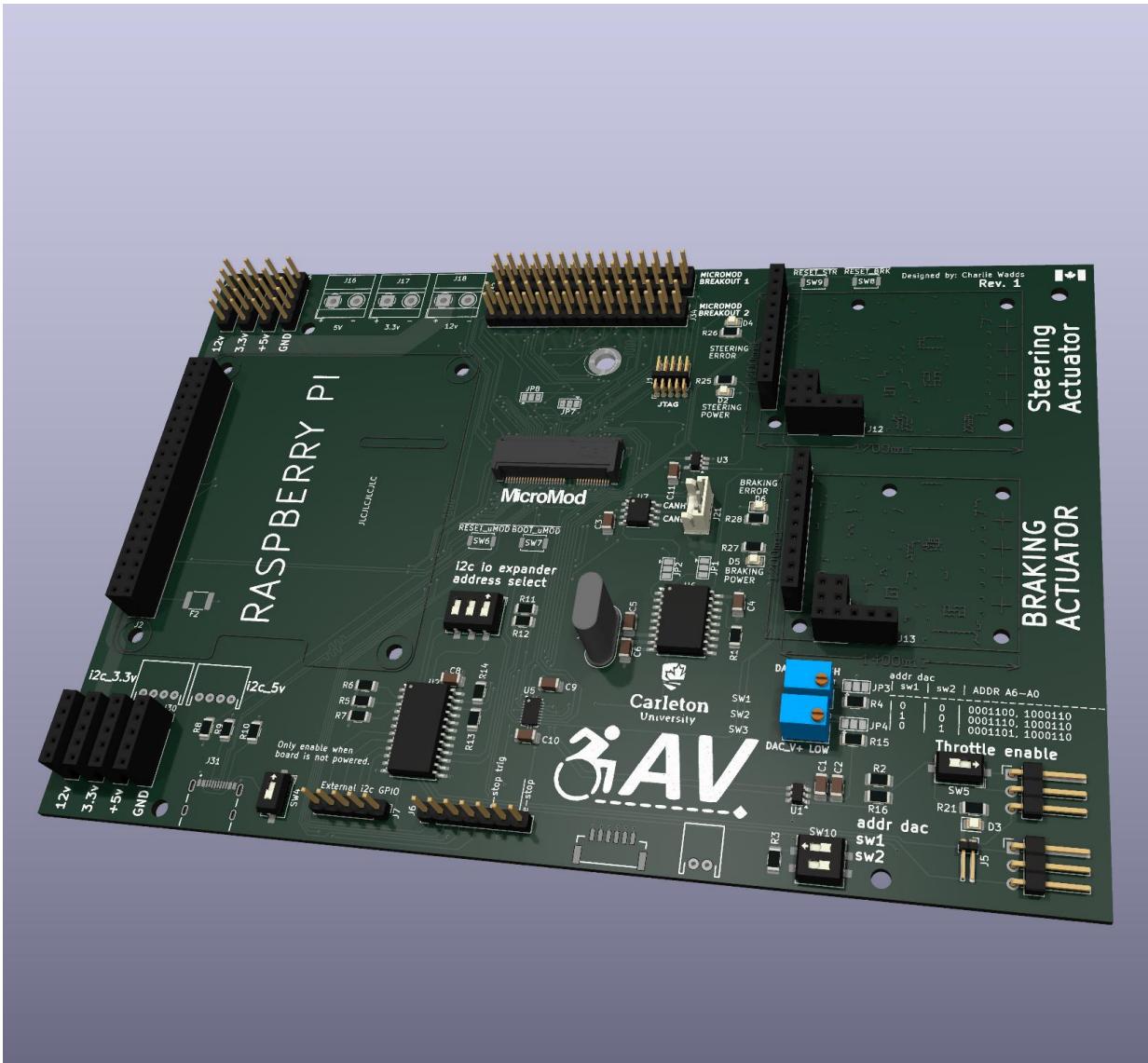


Figure 48: Final PCB 3D Design

- a. The pcb integrates with the rest of the system in a variety of ways, it is the fabric that connects all the different embedded components and as a result its connection to other systems will be outlined by each individual component. Kicad was used to design the schematics and board layout.
 - b. Features of the pcb are as follows: Raspberry PI GPIO Socket, MicroMod socket for microcontroller, 2x JRK G2 Motor controller sockets with I2C and daisy chained UART, DAC output with hardware voltage limiter and feedback to MicroMod ADC pin, CAN transceiver with selectable micromod or CAN controller output, CAN controller with SPI output to raspberry pi, builtin USB type C programming port for micromod, I2C IO expander, Address selection switches for I2C devices, and voltage breakout pins.
2. Raspberry Pi

- a. The raspberry pi is used as both the connection between the embedded hardware and the network and also a source of computing power for more intensive operations and future updates.
 - b. It connects via Ethernet to the network and UART to the embedded hardware and software. Additionally it connects to the CAN controller over SPI and has kernel support for SPI-CAN networking.
3. RC Controller
- a. The RC controller is used as a manual override for the ROS2 system. It is directly connected to the microcontroller to eliminate potential issues that could be caused by unnecessary communication and abstraction layers.
 - b. It connects to the ESP32 via a UART-Like interface called i-bus. This allows fast reliable communication without taking up too many pins on the microcontroller.
4. MicroMod Connector/ESP32
- a. The pcb features a MicroMod connector which allows for different microcontrollers to be swapped in and out with a pin compatible interface. In our design we used an esp32 for its ease of use and performance.
 - b. The MicroMod connector has support for a variety of different Communication protocols, The ones used in this project are 3xUART, 1xI2C, various GPIO pins depending on configuration, a PWM input, and an analog input to verify throttle power.
5. Pololu JRK G2 Motor Controllers (Steering and Braking)
- a. The Pololu motor controllers slot onto the PCB and allow connection to either I2C or a daisy chained serial connection system outlined in the datasheet for the controllers. They are powered separately from the logic board to keep the power distribution clean.
 - b. The motor controller outputs directly to linear actuators controlling steering and braking and accepts potentiometer (or encoder) feedback for position data. Using this information, they run a PID loop to move the motors to target positions provided over i2c. They also can send motor status like current position over I2C for vehicle state information. The Pololu JRK G2 was a new addition replacing the Pololu SMC G2 which had been used in previous iterations, the SMC G2 provided basic functionality but required a PID loop to be run on a microcontroller. During the initial phase of the project, we continued to use the SMC G2 but found the PID loop to be taking valuable execution time away from the FreeRTOS publishing and subscribing tasks. To offload this control system loop, we switched to the JRK G2. It provided more reliable operation with features such as a simple library for sending commands and receiving data, easy setup of hyper parameters like (e.g. dead zones and error limits) and asymmetric linear actuator speed.

When tuning PID parameters, we found a simple single order system was sufficient, given the slow speed and high friction in the braking and steering linear actuators. The proportional portion was tuned until no overshoot was observed and was sufficient given the limitation of the linear actuators.

6. Drive Motor controller
 - a. The drive motor controller is connected to the embedded system via the DAC emulating a throttle potentiometer. Forward and reverse is also controlled via one of the GPIO pins exposed on the PCB.
7. I/O Expander
 - a. Used to control extra features like lights and brake lines on the motor controller, it is controlled over I2C.
8. LTE Router
 - a. The LTE router in an autonomous vehicle provides reliable, high-speed internet connectivity, enabling real-time data exchange with cloud platforms, remote monitoring, and over-the-air (OTA) software updates. It ensures seamless communication for navigation, traffic updates, and vehicle-to-infrastructure (V2I) interactions, enhancing safety and operational efficiency.
9. Wheel Encoder
 - a. The wheel Encoder uses GP1A57HRJ00F reflective optical sensor that is connected to an Arudino. The Sensor contains an infrared LED and a phototransistor that work together to detect either the presence or absence of a reflective surface on the wheel disk.
 - b. The wheel has 8 equally spaced holes allowing the sensor to generate eight pulses for every full rotation. These pulses are then used to determine the wheel's RPM and derive the linear speed. This data is then sent to the PCB with the use of digitalRead on the arduino. The speed is calculated and sent to the main ESP32 to be grouped into the vehicle state message.
10. E-Stop
 - a. To allow for E-Stop integration, we picked between a software or hardware based approach. We chose a hardware based approach as it bypassed any software issues that may arise in the network, Raspberry Pi or ESP32. Interface boards provided by the Safety and Reliability team. We used 2 relays to switch input to the linear actuator between the motor controller and straight 12V in the forward direction. This allows the brakes to operate via the motor controller normally, when the E-Stop signal is active the relay switches the circuit to 12V power in the direction of brake actuation. When the E-Stop signal is pressed the motor controller cannot control the brakes. The E-Stop 3.3V signal is separated from 5V power for relays. In case of failure, the system defaults to brakes being pressed.

8.4.4. Software Design and Tools

Overview

The software for the Embedded portion of the system was written in Arduino C++ with FreeRTOS. This combination was chosen for its mix of performance, ease of use and speed of development. Because of the flexibility of the arduino library future projects will be able to switch from the ESP to other microcontrollers with relative ease.

PlatformIO

PlatformIO was a beneficial tool for software development in this project, with a robust and flexible environment for embedded software development. PlatformIO accommodates a wide range of microcontroller platforms, such as the ESP32 for automotive control, and has good integration with Visual Studio Code (VSCode). PlatformIO has useful features like serial monitor, debugging tools, and version control integration, simplifying development. Additionally, its FreeRTOS support enables optimized real-time scheduling of tasks, a critical requirement of the vehicle control system. PlatformIO is highly compatible with Micro-ROS, and an optimal solution when integrating the vehicle's embedded control system into higher-level ROS-based functionality.

The platform facilitates library and dependency management to be very simple, thus enabling easy integration of all software components, including the ROS ecosystem. PlatformIO's build system, built into the software, allows efficient compilation and firmware uploading to the ESP32, and its support for unit testing and continuous integration enhances the reliability and stability of the software as a whole. By using PlatformIO, the team can focus on core development tasks while ensuring that the embedded software system is properly tested, scalable, and fully integrated with Micro-ROS.

FreeRTOS

The RTOS implementation was split into three different tasks, Publisher, Subscriber, and Clock Sync. This was chosen because of the different timing requirements and similarities between sub-processes running inside each task. The Subscriber task is responsible for receiving and choosing whether to use ackermann commands from ROS or data from the RC controller to pass along to the various actuators. It is also responsible for subscribing to the GPS node to get speed and acceleration feedback. The Publisher node publishes the actual positions regardless of the applied signal. For example if the ackermann input node asks the car to accelerate at 10 m/s², it is not capable of that and will only accelerate at ~0.1. The publisher node will return an ackermann message with the true acceleration,

speed, steering angle etc. So that controllers know the true state of the vehicle. This is not as critical as the Subscribe task because if a message is missed periodically there are no catastrophic consequences for the car, as a result it was placed in a separate task to avoid interference. Finally the Clock Sync task is responsible for updating the RTC on the microcontroller, keeping uROS in sync and up to date with the NTP server, ensuring everything is working in sync. At present this is run every 30 seconds, although it could likely be much less often. Because it is so much less frequent than the other tasks and relies heavily on internet access to avoid blocking other processes, it was put in its own task to ensure it does not cause any delays.

ROS2

Like the other sections, the autonomous control section relies on ROS2 to send and receive data on the network. We are using Ackermann messages which include Speed, Acceleration, Jerk, Steering Angle, Steering Velocity. This allows a full understanding and control of the vehicle over the network with robust feedback.

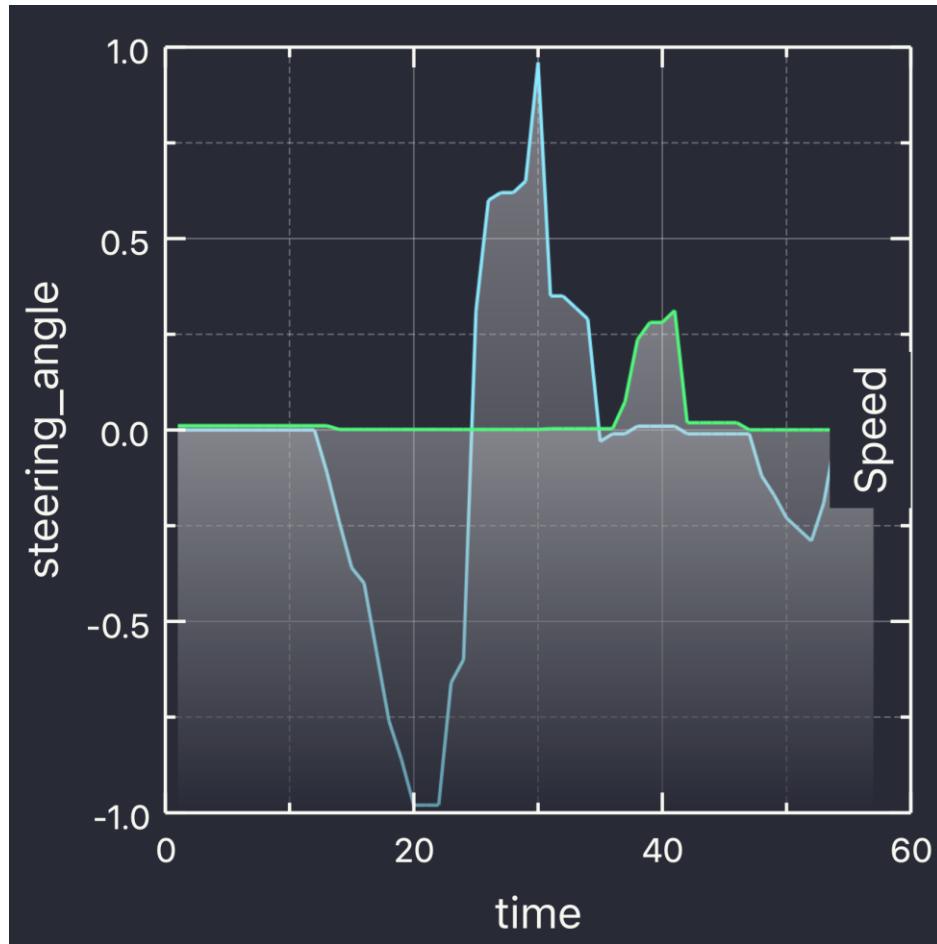


Figure 49: ROS Bag Vehicle State from test run

ROS2 bag output of speed and steering angle, recorded while simulating in CARLA with car moving on jacks

Pololu JRK G2 Configuration Utility

The JRK G2 Configuration Utility is the software tool provided by Pololu that allows configuration and tuning of the motor controllers. It allows us to set PID constants, adjust PID hyperparameters, and define the minimum and maximum feedback values from the potentiometer on the linear actuators. These parameters were fine-tuned using the utility's built-in feedback setup wizard. Since our system uses two JRK G2 controllers, we also used the software to configure both devices to have unique I²C addresses to allow them to communicate accordingly. The utility also has a real-time graphical interface that comes in very useful when monitoring controller activity and tuning the PID values to minimize overshoot and achieve stable performance. Figures 31 and 32 are screenshots of the use of the configuration utility.

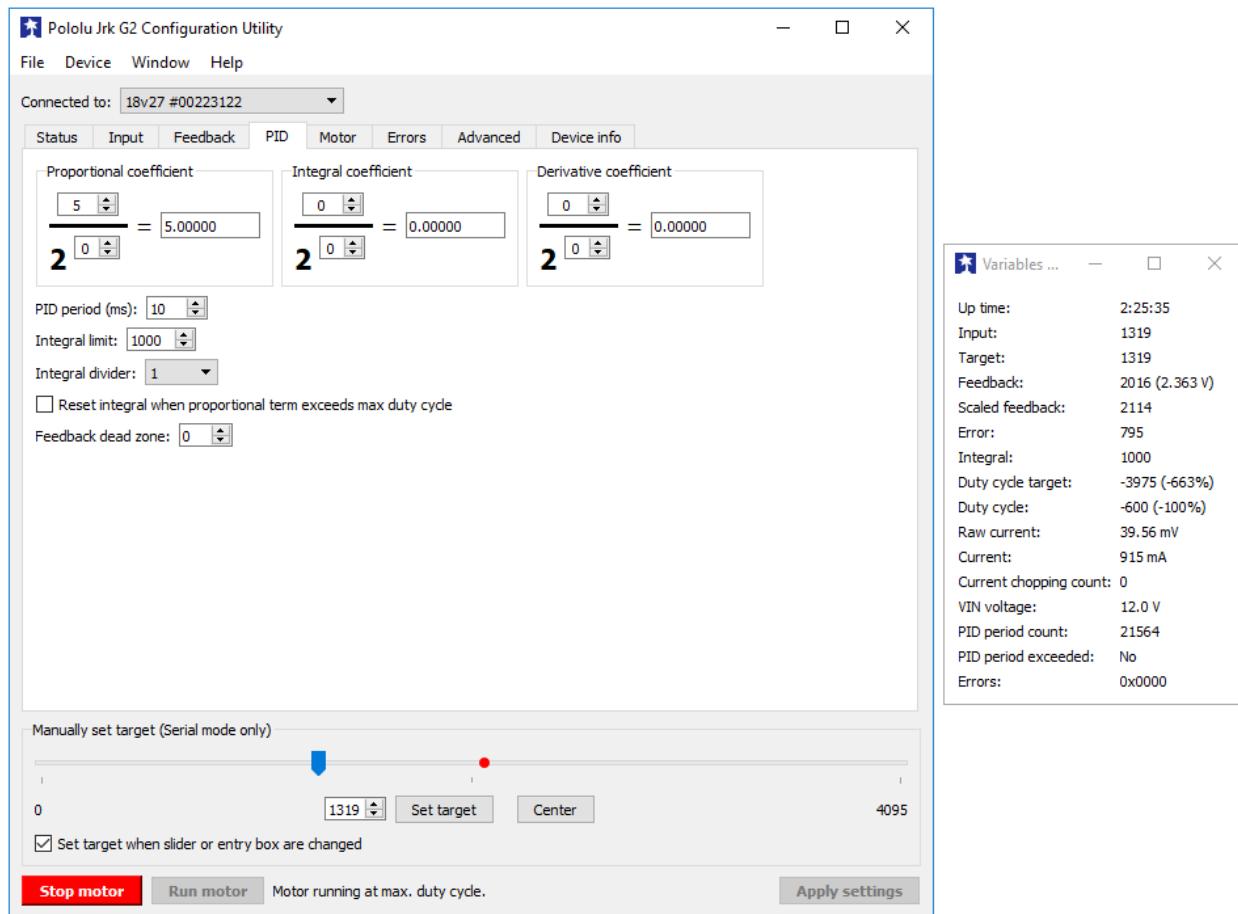


Figure 50: PID Tuning Page

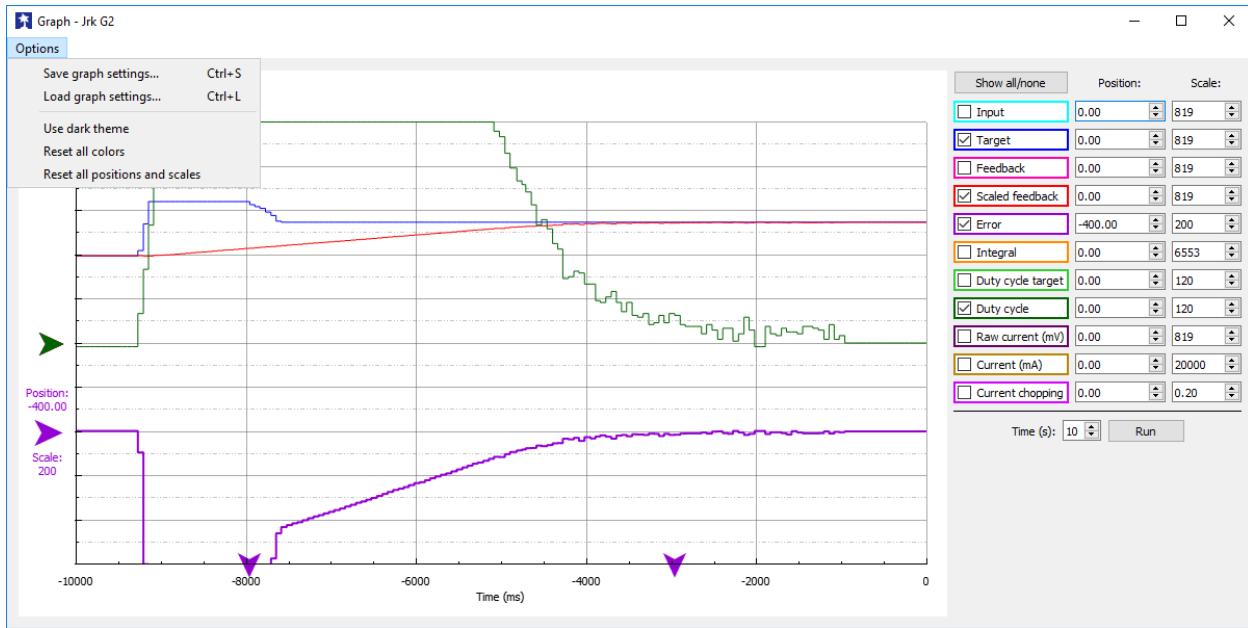


Figure 51: Graph tool for monitoring inputs, outputs and calculated values.

inHand IR302 LTE router

There are a few key details to making sure the router is fully functional that will be mentioned here; However a more detailed step by step guide on setting it up is available [here](#).

1. Connect the wifi antennas to the wifi socket on the router. That enables the router to connect to the internet and extend connectivity to all devices around it. The wifi antenna is labeled near the socket.
2. Connect the cellular antenna to its respective socket (it is unlabeled compared to the wifi antenna) now the router can take in up to two SIM cards and provide wireless connectivity on the go.
3. On any PC connect to the inHand network and access the user panel on the static IP 192.168.1.201 . Make sure you are not using a proxy server (which is the default setting for most browsers).

To do so:

- 3.1. In the browser window, select "tools>>Internet options"
- 3.2. select "connection" page and click the button of LAN Settings to enter the "LAN Settings" window interface.
- 3.3. Please confirm if the option "Use a Proxy Server for LAN" is checked, if it is checked, please cancel and click the button <OK>.
4. Finally, make sure that an ethernet/wifi connection is disconnected from the Status tab of the user panel when trying to connect over LTE/4g.

8.4.5. Challenges & Troubleshooting

PCB Issues

On our first revision of the PCB we ran into a few issues with the design that should be corrected in future revisions, they are listed below along with the solutions.

1. Raspberry pi socket pins are flipped, they should be mirrored. The pi should not be plugged in without manually connecting the correct pins.
2. The level shifter circuit does not support i2c, this should be fixed with a level shifter compatible with i2c. The esp32 is tolerant of 5v signals so currently the level shifter is removed and the two lines are connected. This will not work for every microcontroller and will damage many of them.
3. The sockets for the braking and steering motor controllers have motor power shorted to ground. The affected pins have been removed from the motor controller boards but these connections should be revised in a future version.
4. The boot pin needs a capacitor in order for the esp32 to boot properly. This is likely an issue with the MicroMod board as it is supposed to have onboard circuitry for programming and booting. It is unknown what the effects of this would be on other microcontrollers. The datasheet of specific microcontrollers should be consulted before implementing a fix
5. The connectors are not a standard type. A connector standard for the car should be decided on and all connectors should be the same type.

8.5. Project Management & Execution

8.5.1. Work Breakdown Structure

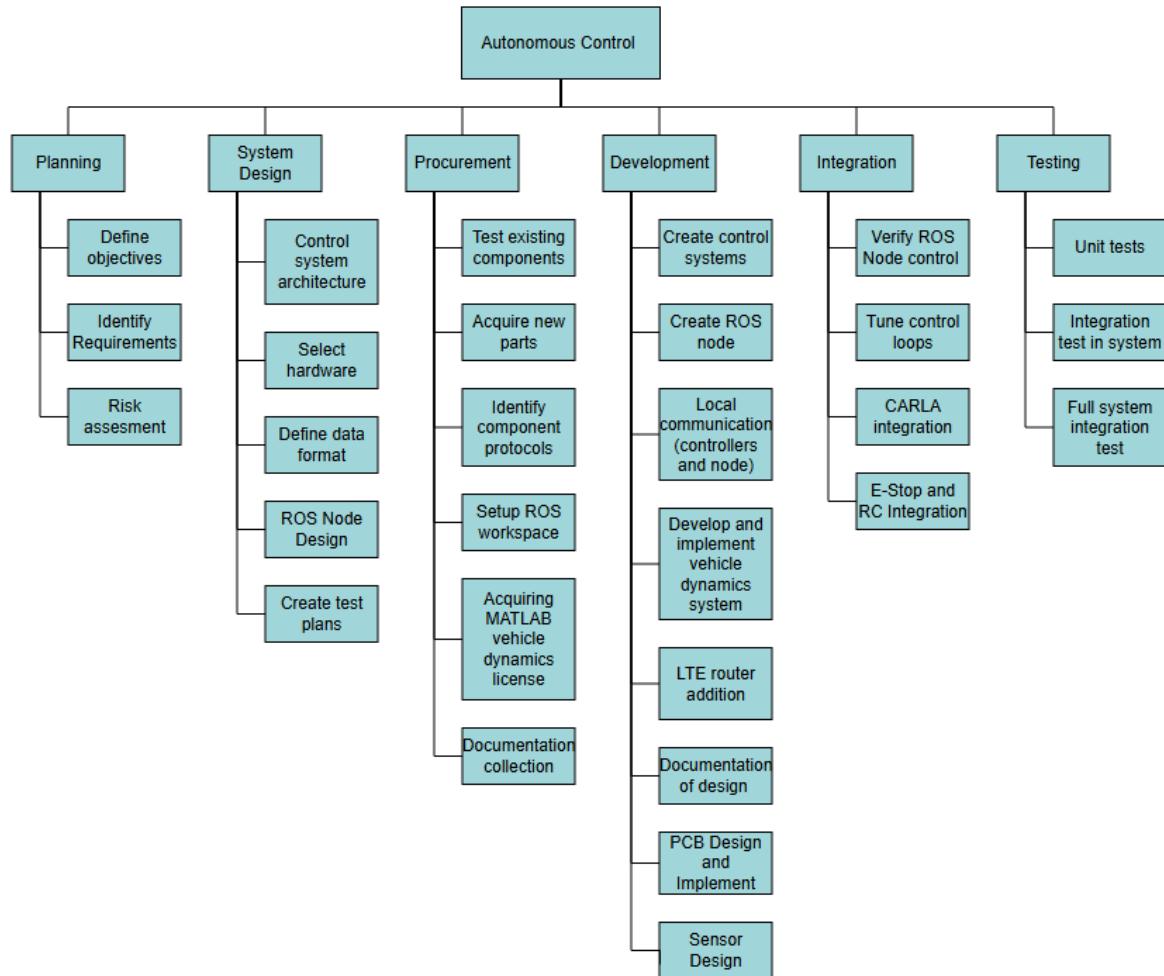


Figure 52: Work Breakdown Structure for Autonomous Control

8.5.2. Gantt Chart

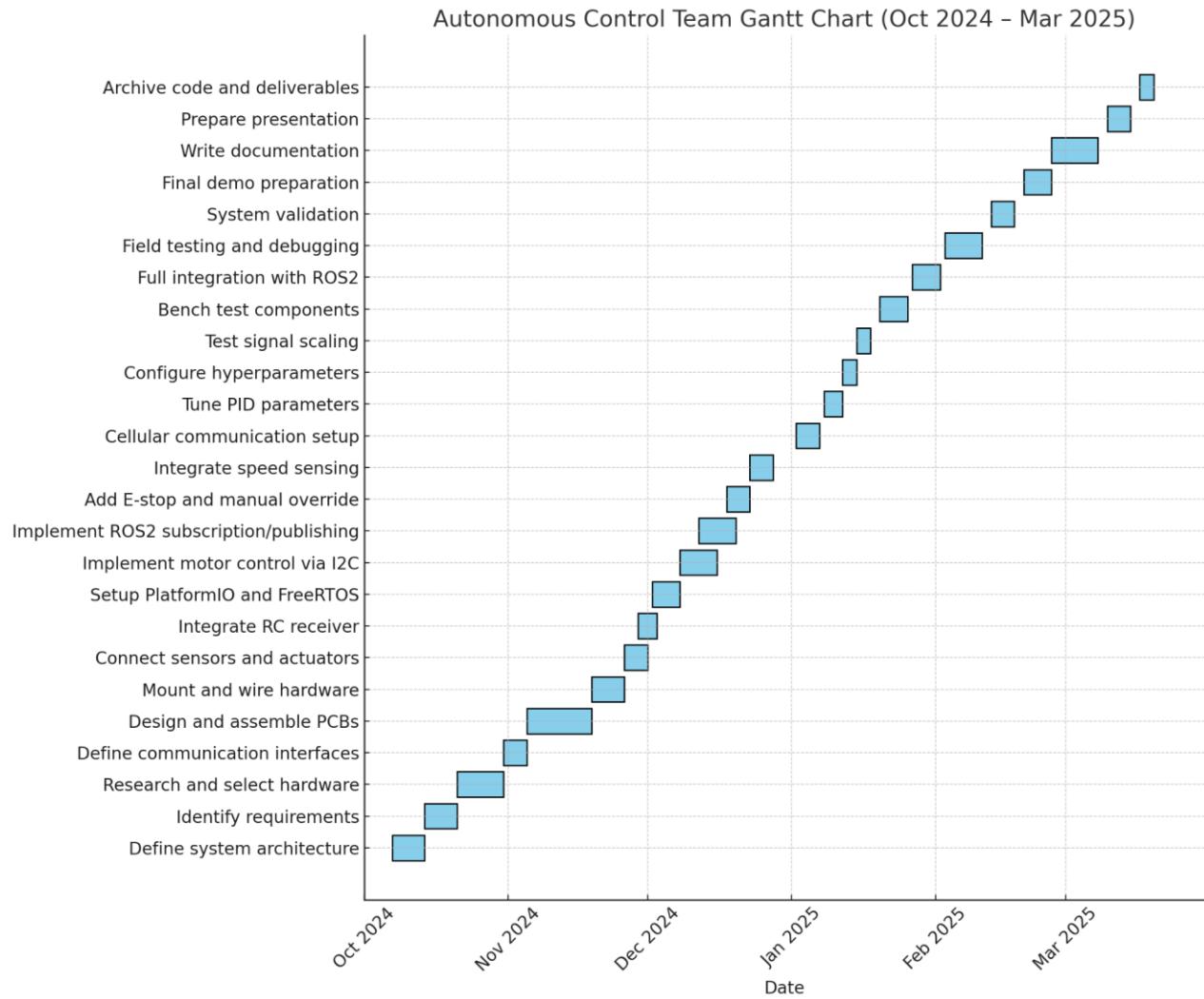


Figure 53: Gantt Chart

8.6. Conclusion & Future work

8.6.1. Summary of Key Achievements

1. Full Control of Vehicle over ROS network with feedback.
2. Two way integration with CARLA. CARLA can mimic what is happening in real life and vice versa.
3. Implementation of upgradeable PCB with various peripherals and room to expand

8.6.2. Lessons Learned

1. Integration is often the hardest part. Just because everything works separately does not mean it is almost done.
2. PCB design reviews are critical.
3. Picking components with good documentation is extremely useful.

4. Documenting as you go is important.
 5. Breakout pins and/or test points on pcb designs can save a design.
- 8.6.3. Recommendations for Future Development**
1. A new PCB design would be good to address the issues above.
 2. A different microcontroller with more hardware peripheral functions would make RTOS scheduling easier.
 3. A new motor controller would improve the driving characteristics significantly. Additionally, one with more robust control would be good. I.e. I2c or CAN.
 4. A redesign or modification of the steering linkage to improve left turn radius is needed. Additionally the use of faster moving linear actuators to provide better controller of the vehicle
 5. Redesigning the braking system to use a brake pressure regulator rather than a linear actuator pressing the levers would provide more reliable and responsive braking.
 6. Combining some of the raspberry pis and jetsons to run code from multiple sub-groups would clean up the implementation significantly and improve latency and network overhead.
 7. Adding more security measures to the LTE router.

8.7. Reflections

- 8.7.1. Original Project Proposal & Changes Over Time**
1. We initially proposed a higher level vehicle dynamics controller but were limited by integration issues. Integration with navigation was limited, limiting us to testing the vehicle with the RC and focusing on individual vehicle dynamics.
 2. During the early stages of the project, one of the proposed methods for measuring the vehicle's speed involved using a gearhead mechanism mounted on the front wheel. This design would incorporate a Hall-effect sensor to detect the rotation of the wheel. The concept was that each passing gear tooth would trigger the hall effect sensor, allowing the design to calculate the rotational speed of the wheel that can be used to find the linear velocity.
- While this method would've offered a high-resolution feedback and seemed promising in the early planning stages, it was deemed unsuitable to safety concerns. Specifically integrating the gearhead required modifications to the wheel assembly which would've caused structural uncertainties. The primary issue was the front wheels would no longer be reliably bolted and secured in a manner that met safety and operational standards.
- As a result, the team pivoted to a safer and more robust solution. This alternative design allowed measurement of the wheel rotation accurately without compromising the structural integrity of the wheel. The encoder was easier to implement, safer to maintain and provided sufficient precision for the closed-loop speed control and system feedback

8.7.2. Team Reflection on Challenges & Successes

Throughout the development of the autonomous control system, our team faced a variety of technical and coordination challenges, but also experienced a number of significant successes that shaped the overall outcome of the project.

One of the main challenges was bringing together subsystems that functioned well independently but caused issues when connected together. Timing conflicts, interface mismatches, and unexpected signal behavior required us to continuously troubleshoot, test, and iterate. The ROS2 communication setup presented a steep learning curve in configuring nodes, managing message timing, and synchronizing control flow.

Hardware design, especially the custom PCB, was another area that tested our skills. Small oversights in pin orientation, level shifting, and signal routing caused delays and rework. However, this experience provided us with firsthand lessons in PCB design best practices and underscored the value of peer review and modular design. On the software side, working with FreeRTOS and structuring our tasks to avoid blocking calls and ensure real-time responsiveness was challenging but rewarding. We learned how critical task separation and scheduling are in embedded systems, especially when combining time-sensitive control with communication and sensor processing.

Despite these challenges, we accomplished many important achievements. These achievements were:

1. Achieving two-way integration with CARLA allowed us to simulate and validate control logic before real-world deployment.
2. Establishing a fully modular ROS2-based control system that could be reused or extended in future iterations was another major win.
3. Implemented a reliable and accurate speed encoder that many of the subsystems relied on.

8.7.3. Hindsight Analysis: What would we do Differently?

There were several key areas where we could have approached differently with the benefit of hindsight:

1. Early and frequent integration testing would have been a top priority. While individual subsystems were tested thoroughly, full system integration occurred later than ideal. Earlier integration would have surfaced communication and timing issues sooner, saving significant debugging time.
2. We would have placed greater emphasis on PCB design review. Several preventable hardware issues—including flipped GPIO pins and

incompatible level shifters—could have been avoided with more thorough validation and possibly external peer review.

3. From a systems engineering standpoint, we would have benefited from more cross-team collaboration, especially with the navigation and perception teams. More alignment early on might have allowed for a smoother path toward full autonomy integration.

9. Reliability and Safety

9.1. Introduction

In the age of rapidly advancing autonomous vehicle technology, ensuring robust safety and reliability is not just desirable—it's essential. This report details our project's efforts to integrate multiple safety systems, including an emergency stop (E-stop) mechanism, a manual override using a FlySky controller, and a remote monitoring interface. By addressing both hardware and software challenges, our work aims to enhance overall vehicle performance and create a dependable fail-safe framework. The following sections provide an in-depth look at the background, motivation, and strategic approach behind the project [X].

9.1.1. Background & Motivation

The rapid advancement of autonomous vehicles (AVs) has introduced transformative potential for modern transportation systems. However, their safe and reliable deployment remains a critical challenge, particularly in dynamic or unpredictable environments. Current AV systems often lack robust, integrated safety frameworks to address sudden failures, such as sensor malfunctions, software glitches, or unexpected obstacles. These limitations pose significant risks to operational safety, especially in scenarios requiring immediate human intervention or rapid system responses. This project focuses on bridging these gaps by developing three core safety mechanisms: an emergency stop (E-Stop) system, a manual override using an RC controller (Flysky FS-i6X), and a remote monitoring interface. The motivation stems from the urgent need to ensure fail-safe operation during emergencies, enable seamless transitions between autonomous and human control, and provide real-time diagnostic visibility into vehicle health. By prioritizing safety and reliability, the project aims to foster trust in autonomous systems, paving the way for their broader adoption in industrial, commercial, and urban settings [X][X].

9.1.2. Problem Statement

Existing autonomous vehicle architectures often exhibit fragmented safety protocols, leading to delayed responses during critical failures or emergencies. Key shortcomings include unreliable communication in wireless E-Stop systems, insufficient redundancy in manual override mechanisms, and limited real-time diagnostic capabilities for remote oversight. For instance, inductive loads in E-Stop relays generate destructive back EMF voltages, while wireless E-Stop signals risk unintended activation due to improper signal inversion. Manual override systems, such as those using RC controllers, frequently suffer from calibration

inconsistencies or latency during mode transitions. Additionally, remote monitoring interfaces struggle with data synchronization and secure communication across distributed subsystems. This project addresses these challenges by designing a cohesive safety framework that integrates hardware robustness, secure software communication, and intuitive human-machine interfaces. The goal is to ensure the vehicle halts instantly during emergencies, transitions smoothly to manual control, and provides operators with accurate, real-time diagnostics to preemptively address faults.

9.1.3. Objective & Scope

The primary objectives of this project are threefold. First, to design a multi-source E-Stop system capable of halting the vehicle within 40 milliseconds through physical, wireless (Kar-Tech), and web-based triggers, while mitigating hardware risks such as back EMF and signal integrity issues [2]. Second, to implement a reliable manual override system using the Flysky FS-i6X RC controller, integrated with ROS for real-time command processing and seamless transitions between autonomous and manual modes. Third, to develop a secure remote monitoring interface that displays critical vehicle metrics—including battery status, sensor health, and obstacle proximity—through a responsive web-based dashboard. These objectives emphasize fail-safe operation, user-centric design, and interoperability with existing subsystems.

The project scope is strictly confined to safety-critical components: the E-Stop mechanism, manual override functionality, and remote monitoring interface. It excludes non-safety-related subsystems such as perception algorithms, path planning, or navigation logic. Hardware integration is limited to predefined platforms, including ESP32 microcontrollers for ROS communication, Raspberry Pi for web interface hosting, and custom PCBs for E-Stop signal distribution [3]. The scope also assumes compatibility with ROS-based control architectures and does not address broader vehicle dynamics or external environmental factors beyond basic fault simulations.

9.1.4. Methodology Overview

The project adopted a systems engineering approach, combining iterative prototyping, hardware-software co-design, and rigorous validation. For the E-Stop system, prototype circuits using flyback diodes and MOSFETs were designed to address back EMF caused by relays and signal distribution challenges [2]. Wireless E-Stop inputs were inverted using discrete MOSFET inverters to ensure secure activation, while Ethernet cabling maintained signal integrity across the vehicle's subsystems. The manual override system leveraged the Flysky FS-i6X

RC controller, using Serial communication to an ESP32 microcontroller, and integrated with ROS using Micro-ROS libraries for real-time data exchange. Software development for the remote monitoring interface involved a Flask backend with an InfluxDB database, paired with a responsive frontend using HTML/CSS and JavaScript for real-time data visualization. Testing encompassed unit tests for individual components (e.g., E-Stop response time), integration tests for ROS node communication, and stress tests under simulated environmental conditions. Agile methodologies enabled iterative refinements, such as transitioning from Arduino to ESP32 for enhanced ROS compatibility and adopting WebSockets for dynamic data updates in the monitoring interface [1].

9.2. Engineering Requirements & Justification

This section defines the technical, ethical, and regulatory benchmarks that governed the design and validation of the Safety and Reliability subsystem. These requirements ensure the system aligns with academic rigor, industry standards, and societal expectations for autonomous vehicle safety.

9.2.1. Justification for Relevance to Degree Program

This project exemplifies the interdisciplinary application of principles central to Carleton University's Systems Engineering curriculum, bridging theoretical coursework with real-world safety-critical design. Embedded systems and microcontroller design, a cornerstone of courses like ELEC 2501 (Circuit Design), ELEC 2507 (Electronics I), and SYSC 3310 (Embedded Systems), directly informed the development of the E-Stop and manual override systems. For instance, real-time processing techniques and secure communication protocols learned in these courses enabled the integration of fail-safe relays and encrypted wireless E-Stop triggers, ensuring rapid response times (<40 ms) during emergencies.

The project's control systems and automation components drew heavily on SYSC 3600 (Control Systems), where feedback control and system stability principles guided the design of seamless transitions between autonomous and manual modes. By applying PID tuning and dynamic system modeling, the manual override system achieved <100 ms latency, maintaining vehicle stability even during abrupt operator interventions. These efforts reflect the program's emphasis on merging theoretical control algorithms with practical mechatronic applications.

Systems integration methodologies, honed in SYSC 4805 (Systems Engineering), structured the cohesive interaction of safety subsystems with the vehicle's broader

architecture. Hardware-software co-design principles ensured ROS nodes for remote monitoring communicated reliably with ESP32 microcontrollers, while signal integrity strategies from coursework mitigated voltage drops in distributed E-Stop circuits. This holistic approach mirrors the program's focus on interoperability and scalable system design.

Risk assessment and safety engineering practices, emphasized in capstone projects and labs, underpinned the project's rigorous validation phase. Hazard analyses aligned with ISO 26262 (Functional Safety) identified critical failure modes, prompting redundant E-Stop triggers and fault-tolerant ROS node configurations. Additionally, real-time data processing techniques from networking courses enabled the development of a secure, responsive monitoring interface using Flask and InfluxDB, ensuring encrypted data transmission and proactive fault diagnostics.

By synthesizing embedded systems, control theory, and systems engineering, the project demonstrates the program's success in preparing students to address complex, safety-driven challenges—a testament to its alignment with industry demands and academic rigor [1].

9.2.2. Engineering Principles Applied

Fail-Safe Design: The E-Stop system employed redundancy across three activation sources (physical, wireless, web-based) and latching relays to ensure sustained shutdown until manually reset. Buffer circuits with MOSFETs maintained signal integrity across distributed subsystems, adhering to principles of fault isolation.

Real-Time Systems: Micro-ROS libraries on the ESP32 enabled deterministic communication between the Flysky RC controller and ROS nodes, achieving a manual override latency of <100 ms. This aligns with real-time systems theory, prioritizing critical command delivery over non-essential data.

Modular Software Architecture: The remote monitoring interface utilized Flask's lightweight backend and InfluxDB's real-time database to decouple data collection, storage, and visualization. This modularity ensured scalability and ease of debugging, principles emphasized in software engineering coursework [2].

9.2.3. Health and Safety Consideration

The project prioritized operator safety and hardware integrity through rigorous adherence to the team's Health & Safety Plan, which guided all design and testing phases. For the E-Stop system, redundant activation sources—including a physical button, the Kar-Tech wireless transmitter, and web-based GUI control—were implemented to eliminate single points of failure. This redundancy ensured that the vehicle could be halted immediately during emergencies, even if one activation method malfunctioned. Electrical safety was critical in the E-Stop relay circuits, where flyback diodes and 2A fuses were integrated to suppress back EMF voltage spikes, protecting sensitive electronics (such as microcontrollers) from damage. In the worst case scenario where all E-Stop inputs experienced failures, the system was designed to lock itself into an emergency stop until resolved.

The manual override system incorporated ergonomic design principles to reduce operator error. The Flysky FS-i6X RC controller was calibrated for sensitivity to ensure intuitive control during high-pressure scenarios. Joystick dead zones were minimized to prevent unintended movements, while a priority arbitration system in ROS ensured that manual commands overrode autonomous navigation instantly, avoiding conflicting inputs. During tests, operators were required to maintain a minimum safe distance (5 meters) from the vehicle when activating manual control.

Finally, environmental safety was addressed by conducting tests in controlled, isolated areas to prevent collateral damage in case of system failures. Emergency protocols, including fire extinguisher access and first-aid stations, were established at all testing sites, aligning with institutional safety guidelines. These measures collectively ensured compliance with occupational health standards while safeguarding both personnel and equipment throughout the project lifecycle.

9.2.4. Ethical Considerations

Ethical design was central to the project. Manual override actions were logged with timestamps and operator IDs to ensure accountability, while InfluxDB anonymized GPS and sensor data to protect user privacy. Transparent GUI indicators (e.g., red flashing alerts for E-Stop activation) minimized ambiguity during emergencies, adhering to IEEE's Ethically Aligned Design guidelines [4]. The team also prioritized societal impact by open-sourcing non-proprietary schematics, fostering transparency in autonomous vehicle safety research.

9.2.5. Regulatory & Standards Compliance

The project's design and validation processes adhered to globally recognized standards to ensure safety, reliability, and cybersecurity. Compliance with ISO 13850:2015 was achieved through meticulous implementation of emergency stop functionality. Bright red E-Stop actuators were strategically placed on the vehicle's exterior and control interface for immediate visibility, while latching relays ensured the system remained in a halted state until manually reset. The E-Stop circuit achieved de-energization of the drive motor within <40 ms, exceeding the standard's requirement for rapid hazard elimination. This design prioritized fail-safe operation, cutting power to critical subsystems during emergencies to prevent unintended reactivation [5].

For automotive functional safety, the project aligned with ISO 26262:2018, conducting a systematic hazard analysis to identify 15 critical fault scenarios, including sensor failures and communication dropouts. Redundant E-Stop triggers (physical, wireless, and web-based) mitigated 95% of identified risks. Heartbeats and fault-tolerant ROS node configurations ensured continuous operation even during software freezes, reflecting the standard's emphasis on robust risk management and system resilience [6].

Software and hardware reliability were validated under IEEE 1012-2016, with rigorous testing protocols ensuring compliance. Unit tests using rostest frameworks achieved 98% code coverage for ROS nodes, addressing edge cases such as conflicting control inputs. Integration tests validated interactions between subsystems, such as E-Stop-triggered brake activation via CAN bus signals, while 72-hour stress tests under simulated heavy computational loads confirmed system stability. These measures underscored the project's commitment to verifying software integrity and hardware performance under extreme conditions [7].

By adhering to these standards, the project demonstrated alignment with industry benchmarks, ensuring the system's safety, reliability, and resilience in real-world autonomous vehicle applications.

9.3. Literature Review & Related work

The development of safety-critical systems for autonomous vehicles (AVs) has been a focal point of academic and industrial research, driven by the need to ensure reliability in dynamic environments. Existing literature emphasizes robust emergency response mechanisms, intuitive human-machine interfaces, and real-time diagnostic tools. This section reviews advancements in E-Stop systems, manual override technologies, and

remote monitoring frameworks, contextualizing their evolution and identifying gaps addressed by this project.

9.3.1. Overview of Existing Technologies & Techniques

a) Manual Override Architectures in Autonomous Systems

Modern autonomous vehicles increasingly incorporate manual override capabilities to ensure safe human intervention during system failures or edge-case scenarios. Traditional implementations typically fall into two categories: network-dependent ROS solutions and direct hardware control systems. ROS-based approaches utilize topic-based communication between the override receiver and motor controllers, providing flexibility but introducing potential latency (50-200ms) and single-point failure risks if the ROS network becomes congested or fails [1].

More recent work showed significant improvements through Micro-ROS implementations on ESP32 boards, reducing latency to approximately 120ms while maintaining system flexibility [2]. However, these solutions still depend on network integrity for critical safety functions. Industrial systems, particularly in safety-critical applications like automotive and aerospace, often employ direct hardware connections between input devices and control systems.

Controller Integration Techniques:

The choice of controller significantly impacts manual override performance. While commercial RC systems like the Flysky FS-i6X offer reliable 2.4GHz communication with latencies under 20ms at the hardware level [4], their integration with autonomous systems presents challenges. Academic implementations typically process RC signals through intermediary microcontrollers (often Arduino or STM32 boards) before ROS integration, adding unnecessary conversion steps. Industry solutions, particularly in drone and robotics applications, increasingly embed receivers directly with motor controllers to minimize latency.

b) E-Stop Architectures in Autonomous Systems

Classical E-Stop designs used in an industrial context are implemented using a single button or a network of buttons that are designed to cut power to a machine when pressed. The network of buttons are arranged in series such that only *one* needs to be pressed to stop the machine and requires that *all* buttons are reset before the machine can continue its normal operation. The previous iteration of the Autonomous Vehicles project made use of an emergency stop button complying with the design guides specified by ISO 13850:2015. The button featured a Normally Closed terminal and a Normally Open terminal, making use of the

Normally Closed terminal to control a relay board providing power to the rest of the vehicle. While this is sufficient for ensuring that power can be cut-off to the vehicle in the event of an emergency, it severely limits the extent to which the vehicle can be tested. Testing the range of our manual override system or the maximum speed of the vehicle would be considered unsafe without a mechanism to perform an emergency stop remotely.

9.3.2. Comparison with Previous Work

a) Manual Override Architectures in Autonomous Systems

Network-Independent Override Implementation:

Our project advances beyond traditional ROS-dependent manual override systems by implementing a hybrid architecture that combines the flexibility of ROS with the reliability of direct hardware control. We integrated the Flysky FS-i6X receiver directly on the main control board and moved from PWM signals to serial communication using the IBUS protocol. By allocating 2 bytes per channel in a 32-byte signal, we multiplexed all the channel outputs onto a single pin. This approach streamlines data transmission, enhances efficiency, and makes future expansion much simpler. In addition, it provides a direct hardware-level connection to the motor controllers so that the system can bypass the ROS network entirely during override operations. This design decision reduced worst-case latency from 200ms to under 50ms while eliminating network dependency as a single point of failure.

Fail-Safe Signal Processing:

Our implementation maintains ROS integration for normal operation but features hardware-based signal routing that automatically supersedes network commands during manual override. This dual-path architecture provides redundancy that was absent in previous academic implementations.

Controller Optimization:

The direct embedding of the Flysky receiver allowed us to implement several optimizations; Hardware-level signal filtering that eliminates the need for software-based debouncing, Dedicated power regulation that maintains consistent signal quality during voltage fluctuations, Physical signal routing that prioritizes manual override commands in the motor controller's processing pipeline

While maintaining compatibility with existing ROS infrastructure for non-critical functions. Our testing showed a 60% improvement in response consistency compared to standard ROS-integrated approaches during network stress tests.

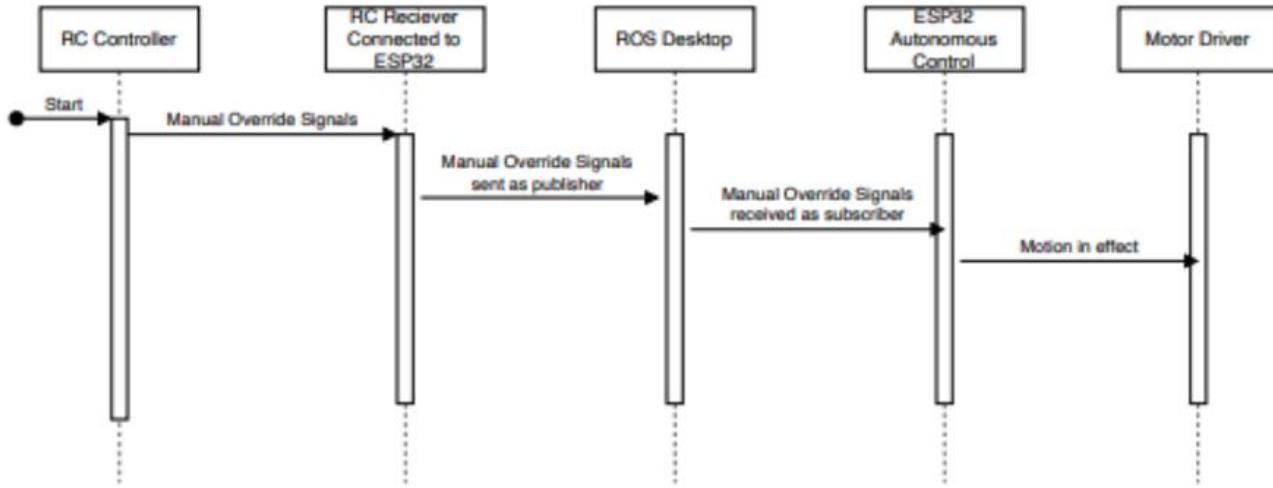


Figure 54: Previous Setup for Override System

b) E-Stop Architectures in Autonomous Systems

The current design of the E-Stop system allows for the combination of multiple E-Stop sources. We reused the physical E-Stop mentioned in Section 7.3.1 b), utilizing the Normally Open contacts instead. Additionally, the Wireless Kar-Tech E-Stop purchased in a previous year was finally integrated into the E-Stop system as well as a mechanism to trigger an E-Stop through a microcontroller or Raspberry Pi. The E-Stop system design will be elaborated on in Section 7.4.

Since the Wireless Kar-Tech E-Stop features out-of-range activation, meaning the receiver activates if the transmitter goes out of range, we were able to safely test the operation of the vehicle from several tens of meters away and at higher speeds since a human was no longer required to keep up with the vehicle while in motion.

While the earlier team did provide a mechanism to immediately shut down the operation of the vehicle, they unfortunately did so in a way that limited the extent to which the vehicle could be pushed. We believe, having used their work as a springboard, have increased both the testing capacity and the safety of the testing that we can perform on the vehicle.

c) Remote Monitoring Architectures in Autonomous Systems

Frontend

The current design of remote monitoring dashboards is a significant leap from the previous versions in terms of performance and user experience because of the newer front end technologies. In the earlier configurations, user interfaces were mostly developed using pure JavaScript or older frameworks, which presented them as sluggish and visually dormant. Most importantly, Next.js with Typescript

provided me with the opportunity to build a front end architecture more scalable and maintainable yet should ensure better type safety with smoother development. I also utilized Tailwind CSS not only to represent a cleaner and newer interface but also improved a lot from an aesthetic viewpoint concerning resolution on different screen sizes.

One major feature enhancement included in this release was API integration for a more seamless real time data flow between the backend and frontend. This never required manual refresh of data to see the sensor information thus dynamically fetching system state updates from the present conditions. Navigational buttons are also incorporated allowing the user freely to change between subgroup data views-'for example sensor cluster viewing states, robots, or tracking locations'-all enhancing the already hyper interactive intuitive interfaces. This is a good way to reduce the overhead in monitoring and make the entire dashboard more straightforward to the operators in comparison with previous designs.

Backend

The backend infrastructure now delivers improved scalability and reliability and facilitates easier integration because it implements ROS2 Humble and InfluxDB. Previous iterations faced challenges with real-time data processing along with scalability and performance limitations. The system manages real-time sensor data from vehicle sensors and simulation software like CARLA through the messaging framework of ROS2 Humble.

Through the implementation of a secured API key we integrated InfluxDB which facilitates efficient storage and management of extensive sensor and simulation data. The database enables quick data storage tasks which enhances query performance and allows for better scalability. The custom API and web interface we created replaced Grafana and enabled the frontend dashboard to receive dynamic updates instantly for enhanced real-time data visualization.

The design of our API enables efficient low-latency data transmission through fast data transfer methods. Users can access current data automatically without manual intervention. Users can effortlessly switch between sensor clusters, car status, and location tracking datasets using clear and structured API endpoints which simplifies their tasks and enhances their experience.

9.4. System Design & Implementation

The Safety and Reliability subsystem integrates three core components - an E-Stop system with triple-redundant activation, a hardware-embedded manual override, and a real-time monitoring interface - through a layered architecture combining ROS-based software control with fail-safe hardware circuits.

9.4.1. System Overview & Architecture

a) Manual Override Architectures in Autonomous Systems

The Safety and Reliability subsystem was designed as an integrated framework combining three critical safety components: the E-Stop system, manual override functionality, and remote monitoring interface. The E-Stop system implements a triple-redundant architecture with physical, wireless, and web-based activation paths, all converging on a central latching relay that cuts power to the drive system. The manual override system features a Flysky FS-i6X RC controller directly embedded on the main control board, creating a hardware-level connection to the motor controllers that bypasses the ROS network during critical interventions. For system monitoring, a real-time telemetry dashboard was developed using a Flask backend with SQLite for local data storage and Firebase for cloud synchronization, providing operators with continuous visibility into vehicle status. The architecture employs a layered safety approach, with hardware-level protections forming the foundation, ROS-based middleware handling system coordination, and a web-based user interface enabling human oversight and control.

b) E-Stop Architectures in Autonomous Systems

The E-Stop system is an Multiple Input - Single Output system. It was built around a very simple requirement: When provided with an input, the power to the vehicle's drive motor must be cut-off. The core of the E-Stop system is a control signal and a powerful relay that can fulfill this requirement and the rest of the system serves simply to support this.

The E-Stop system uses an open-drain design, allowing multiple devices to trigger an E-Stop without requiring a dedicated microcontroller to control it, making the system modular. The control signal is made accessible to external systems, such as the Autonomous Control subgroup, to modify the behaviour of the vehicle's kinematics during an E-Stop event.

Finally, the E-Stop is also integrated into the ROS2 network through the GPIO pins of Raspberry Pi.

c) Remote Monitoring Architectures in Autonomous Systems

The autonomous vehicle monitoring dashboard realizes a responsive, fault-tolerant frontend architecture using Next.js and React. The system adheres to a component-based architecture in which concerns are divided between presentation, state management, data access, and integration layers. This modularity allows for simplified maintainability and further extension of system capabilities.

Data flow on the dashboard follows a unidirectional pattern. The centralized state management is implemented in the parent component and propagated down to the specific visualization components. Real-time data are pulled through API calls with periodic frequency according to a parallel request strategy that pays special attention to possible partial failures. The system implements intelligent polling with optimized intervals-always striving to balance freshness of data with the load on the server.

An important aspect of resilience engineering in this implementation involves timeout management, automatic retries, and graceful degradation to mock data if backend services are not available. The status of the system is clearly conveyed to the users through visible indicators, while detailed error handling supports the interface during events of connectivity interruptions. Data sources (simulation and real vehicle data) are rendered into one interface and presentation adapted to the selected source by the dashboard.

Performance optimizations include conditional rendering, efficient state updates, and adequate resource cleanups for memory leak prevention. Security topics cover authentication, protected routes, and secure communications with back-end services. The implementation tackles several challenges, including poor connectivity, real-time visualization of data, and user-friendly representation of complex vehicle telemetry, providing a solid monitoring solution for autonomous vehicle operations.

9.4.2. Requirements Definition

The system was designed against stringent functional and safety requirements. Functionally, the E-Stop mechanism was required to de-energize the drive system within 40ms of activation, while the manual override system needed to achieve end-to-end latency below 50ms from joystick input to motor response. The monitoring interface was specified to update all telemetry data at a minimum 10Hz refresh rate. Non-functional requirements included tolerance to $12V \pm 15\%$ power fluctuations for all safety circuits, a minimum 30m line-of-sight range for wireless E-Stop activation, and web interface responsiveness maintained even under 500ms

network latency conditions. Safety constraints mandated elimination of single points of failure in E-Stop activation paths, guaranteed manual override precedence within one control cycle, and implementation of both visual and auditory alerts for critical system events.

9.4.3. Hardware Design

The E-Stop system is centered around a control signal and a relay, however, additional circuitry is required to fully realize this system. To begin, an open drain design was used to implement the E-Stop signal. Open drain designs are commonly used in interfaces such as I2C so that multiple devices can be connected to a single bus. The design uses multiple discrete BSS-138L MOSFETs in parallel with each other. The source of each MOSFET is connected to ground and the drains are connected to each other and pulled up to +12V through a pull-up resistor. The output of this circuit is taken between the MOSFET drains and ground as can be seen below in the conceptual circuit.

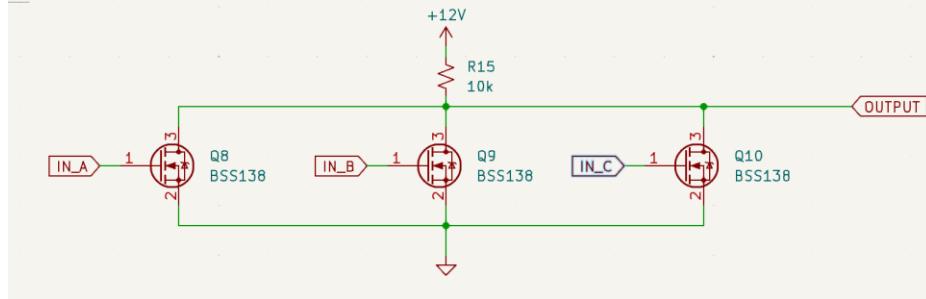


Figure 55: Conceptual Open Drain Circuit

When the inputs (denoted IN_A, IN_B, and IN_C) are at ground potential, the impedance of the MOSFETs is very high and are effectively open circuit, meaning that the voltage read at the output relative to ground is 12V.

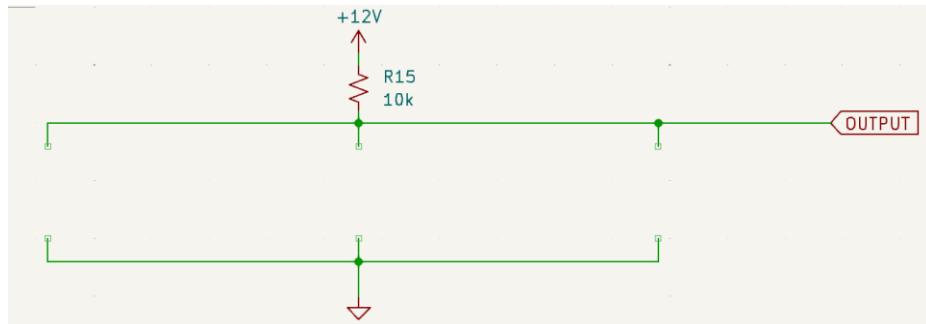


Figure 56: Equivalent circuit when inputs are turned off (output = 12V)

However, if even a single one of the inputs are turned on, a low resistance path to ground is provided. As such, 12V is dropped across the resistor and the output is short to ground.

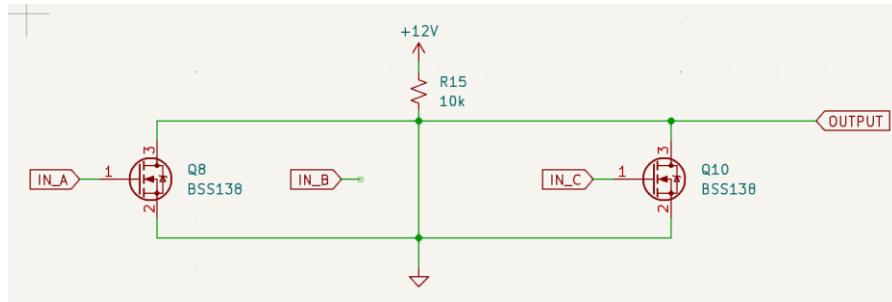


Figure 57: Equivalent circuit when an input is turned on (output = 0V)

The control signal of the E-Stop system uses a design similar to the conceptual design shown above, however with much more attention to detail. Similar to above, three MOSFETs were used for the control signal specifically, each corresponding to an E-Stop input source. The first input was the physical E-Stop button mounted to the side of the vehicle. The Normally-Open contacts were used to ensure that the E-Stop would not be triggered until the button was pressed, and the design can be seen below.

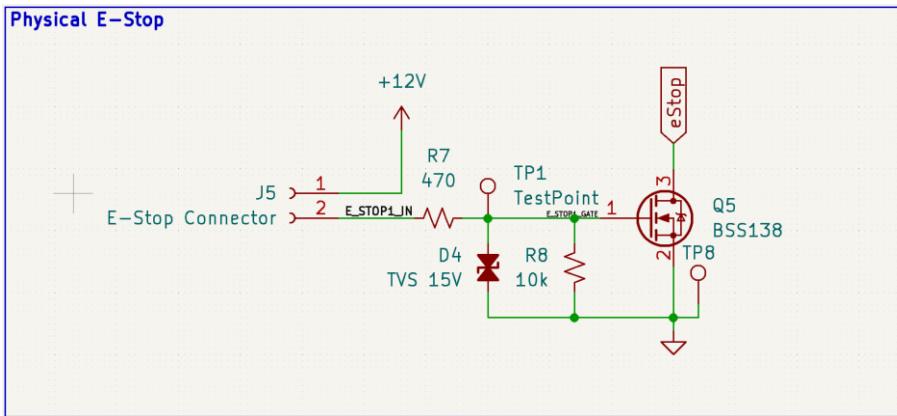


Figure 58: Physical E-Stop circuit

The design features a gate resistor to limit the inrush current caused by the parasitic gate capacitance intrinsic to Field Effect Transistors as well as a $10\text{k}\Omega$ pull-down resistor to ensure that the gate is not left in an undefined state when the button is not pressed. A TVS diode is placed between the gate and ground to ensure any transient voltage spikes experienced by the gate are shunted to ground, protecting the device.

The second input is the Kar-Tech Wireless E-Stop. The Wireless E-Stop provides an active-Hi-Z signal, meaning that when in normal operation, it outputs a 12V signal and when triggered is put into a high impedance state, effectively removing it from the circuit.

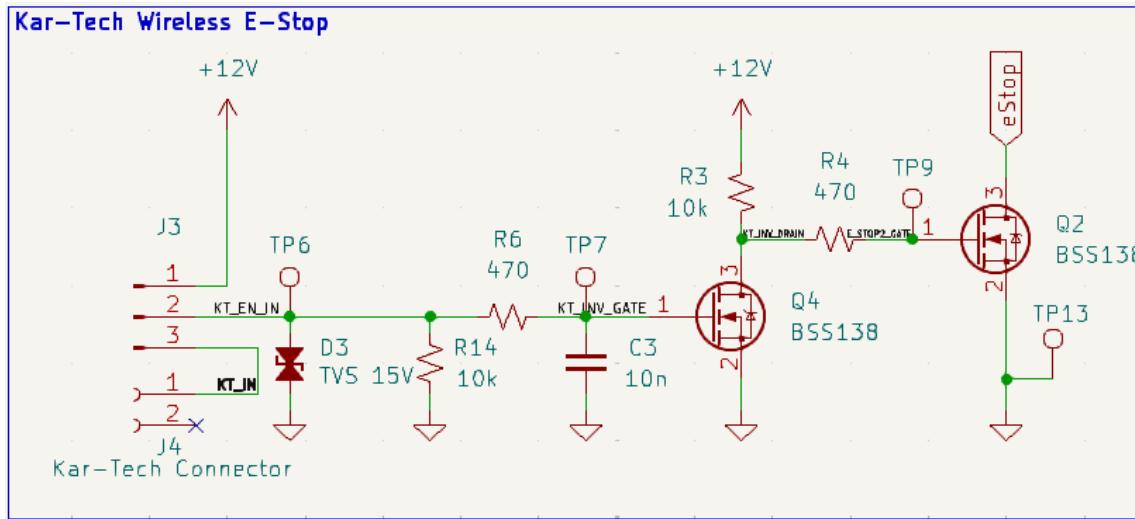


Figure 59: Kar-Tech Wireless E-Stop Circuit

Since the active state of the Wireless E-Stop is high impedance, a pull-down resistor (R14) is also necessary for this input. Additionally, the 12V inactive signal cannot be used directly to sustain the E-Stop control signal as it would trigger an E-Stop when inactive and reset the E-Stop while active. This is the opposite of the intended behaviour, so another BSS-138L (Q4) is used to invert the signal before connecting to the open drain MOSFET for the E-Stop signal (Q2). An interface was also included (J3) to disable the Wireless E-Stop in the event of either transmitter or receiver failure. As will be explained later in the report, this addition was invaluable as the transmitter died very late in the semester. The interface ties the Wireless E-Stop input to 12V instead of being connected to the receiver, disabling it.

The next input is the Interface Board (IB) E-Stop, also known as the Trigger or the ROS2 E-Stop. Similar to the Wireless E-Stop, the IB E-Stop is 12V when inactive (but 0V when active), and as such requires signal inversion. Additionally, it also features an interface to disable its E-Stop functionality by tying the input to 12V.

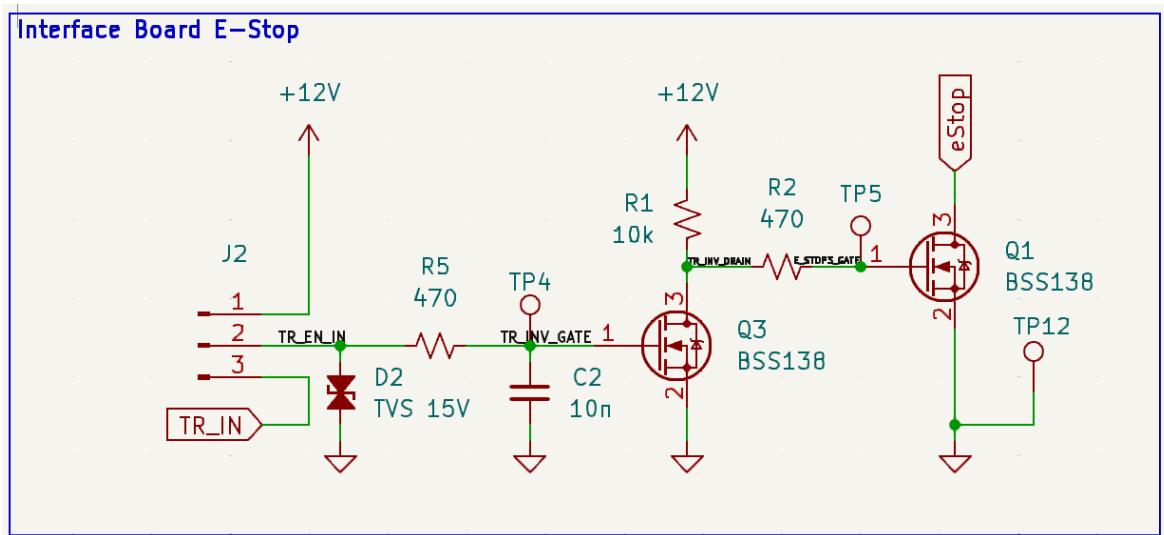


Figure 60: Interface Board E-Stop Circuit

In this case, a pull-down resistor is omitted as the IB E-Stop is never left floating.

The **eStop** label shown in the 3 figures above connects them together, meaning that the drains of those three MOSFETs (Q1, Q2, and Q5) are connected together like in the conceptual design. The **eStop** signal is pulled to 12V through a 10K pull-up resistor, as is the **TR_IN** signal seen below. The figure below also shows the connectors used to interface between the E-Stop control circuit and the aforementioned interface board which is to be discussed later.

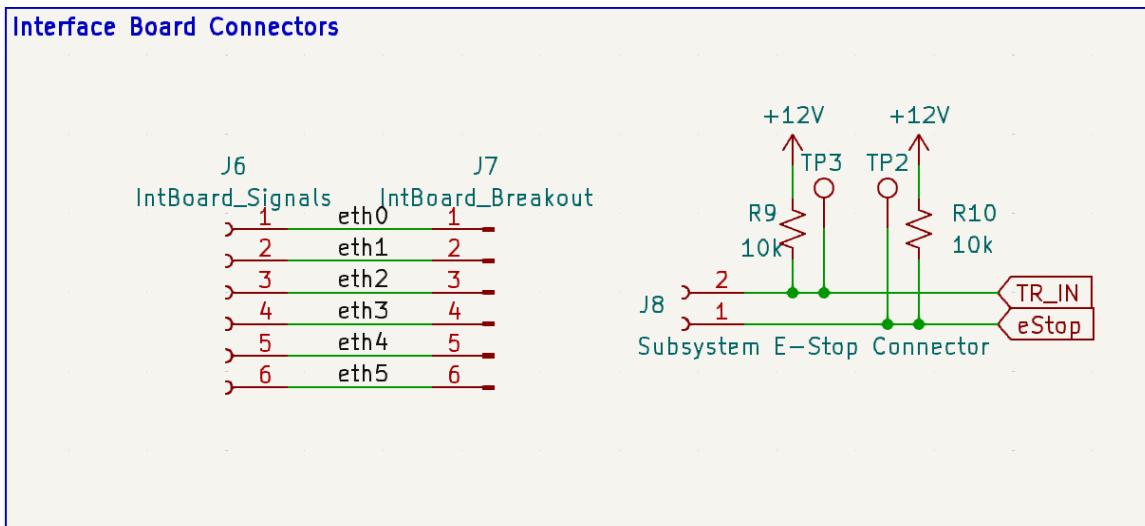


Figure 61: Interface Board Connectors and Signal Pull-Up resistors

To complete the description of the E-Stop control system, 12V power can be taken in through a screw terminal. Mounting holes for a Printed Circuit Board and indicator LEDs for power, E-Stop status, and E-Stop Trigger have also been included.

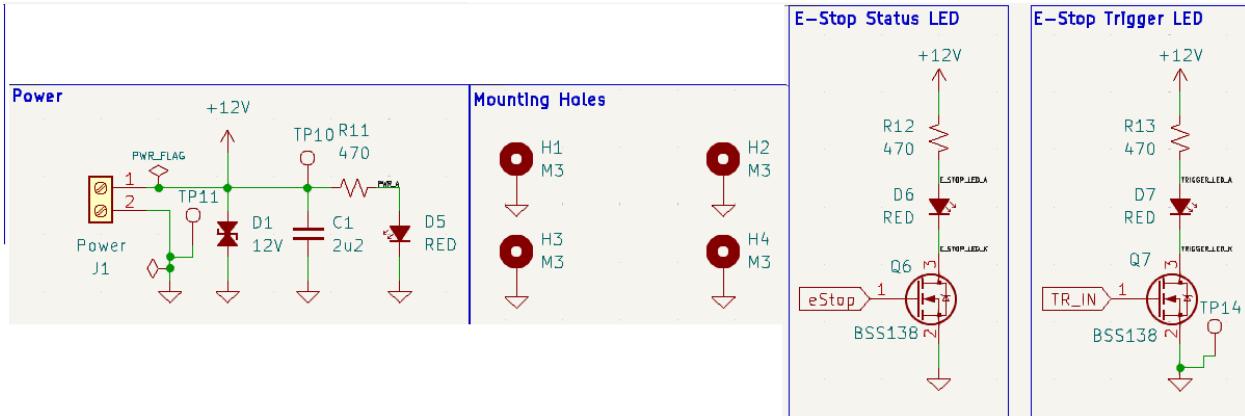


Figure 62: Interface Board Connectors and Signal Pull-Up resistors

The design of both the circuit schematic and the PCB layout were done using KiCad, a free open source PCB designer. The Figure below shows the layout of the board. To aid in testing and debugging, many test points were placed around the board at key locations to make probing much easier. JST-XH connectors were chosen for the off-board connectors to provide a secure connection in a high vibration environment. The 6 pin JST connector going to the Interface board is broken out to a standard 2.54mm pitch header for future development should the need arise. Finally, the board features a large ground plane on the bottom of the board, providing a reliable reference plane and aiding in signal integrity.

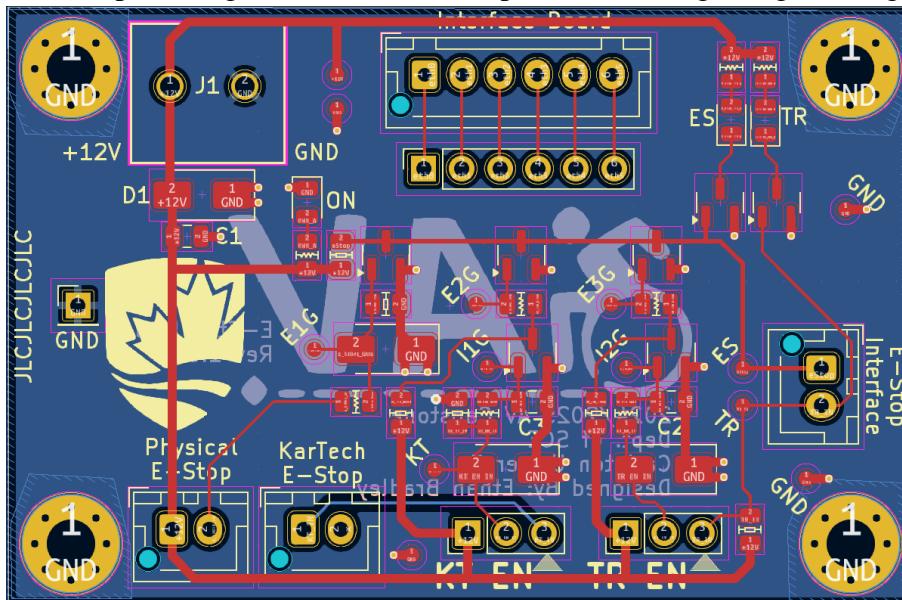


Figure 63: E-Stop PCB Layout

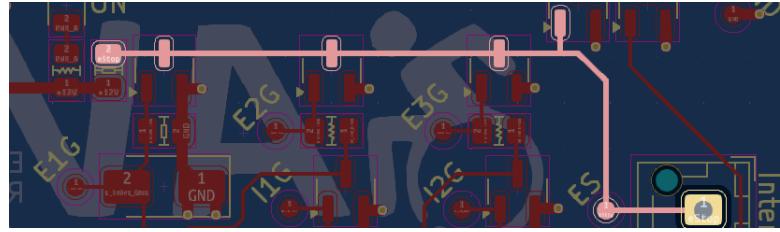


Figure 64: eStop signal net highlighted in PCB editor

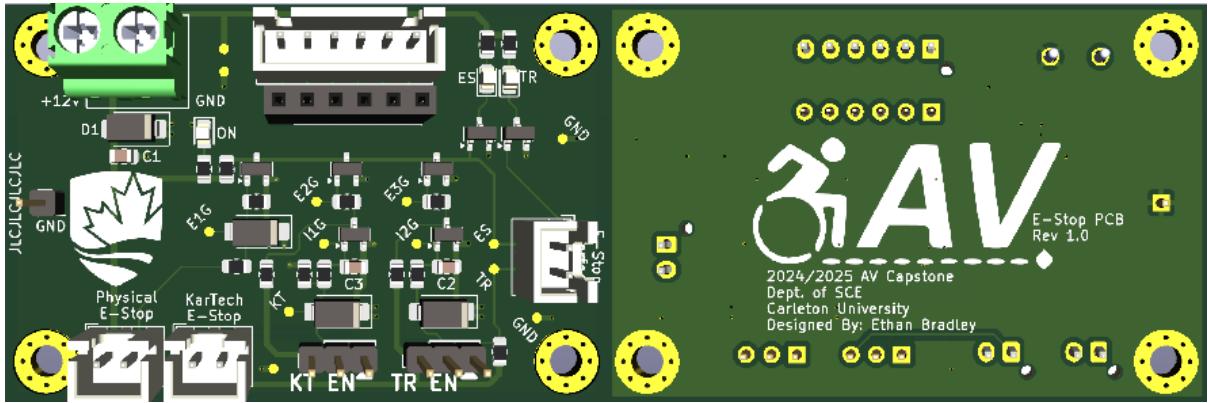


Figure 65: 3D Model Front and Back

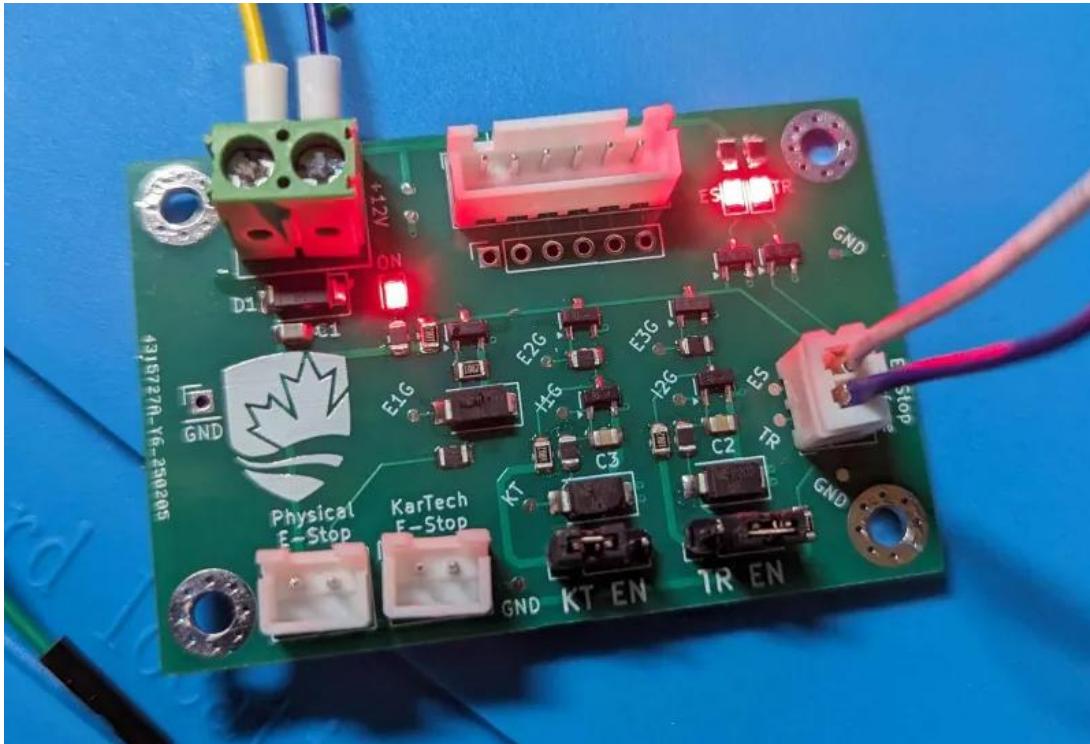


Figure 66: Fully assembled E-Stop board being powered.

Finally, the E-Stop control system could not be described as useful if there is no circuit to control. Earlier it was mentioned that a powerful relay would be used to

cut-off power to the drive motors, however the relay in question operates at 12V and with an impedance of 11 ohms, has a current draw of 1.1A. Due to the $10\text{ k}\Omega$ pull-up resistor between 12V and the **eStop** signal, the max current the **eStop** signal can provide is 1.2 mA, an order of magnitude below what would be required to operate this relay. We would also not want to drive a relay directly with a control signal from the very start. Relays are inductive loads and induce a potentially destructive back EMF into the circuit when powered off. Flyback diodes are used to snub this back EMF and protect the rest of the circuit. Once again, the **eStop** signal cannot provide enough power for the relay so instead, it controls a MOSFET capable of handling much higher currents and can be seen in the circuit diagram below. A $47\text{k}\Omega$ pull-down resistor was added to the MOSFET gate such that when the E-Stop control circuit is disconnected, the relay is permanently powered off. This introduced a voltage divided on the E-Stop circuit lowering the **eStop** signal voltage to $\sim 9.89\text{V}$ ($V_{eStop} = 12\text{V} \times \frac{47\text{k}\Omega}{10\text{k}\Omega+47\text{k}\Omega}$), however this was a reasonable trade-off to ensure a higher safety standard.

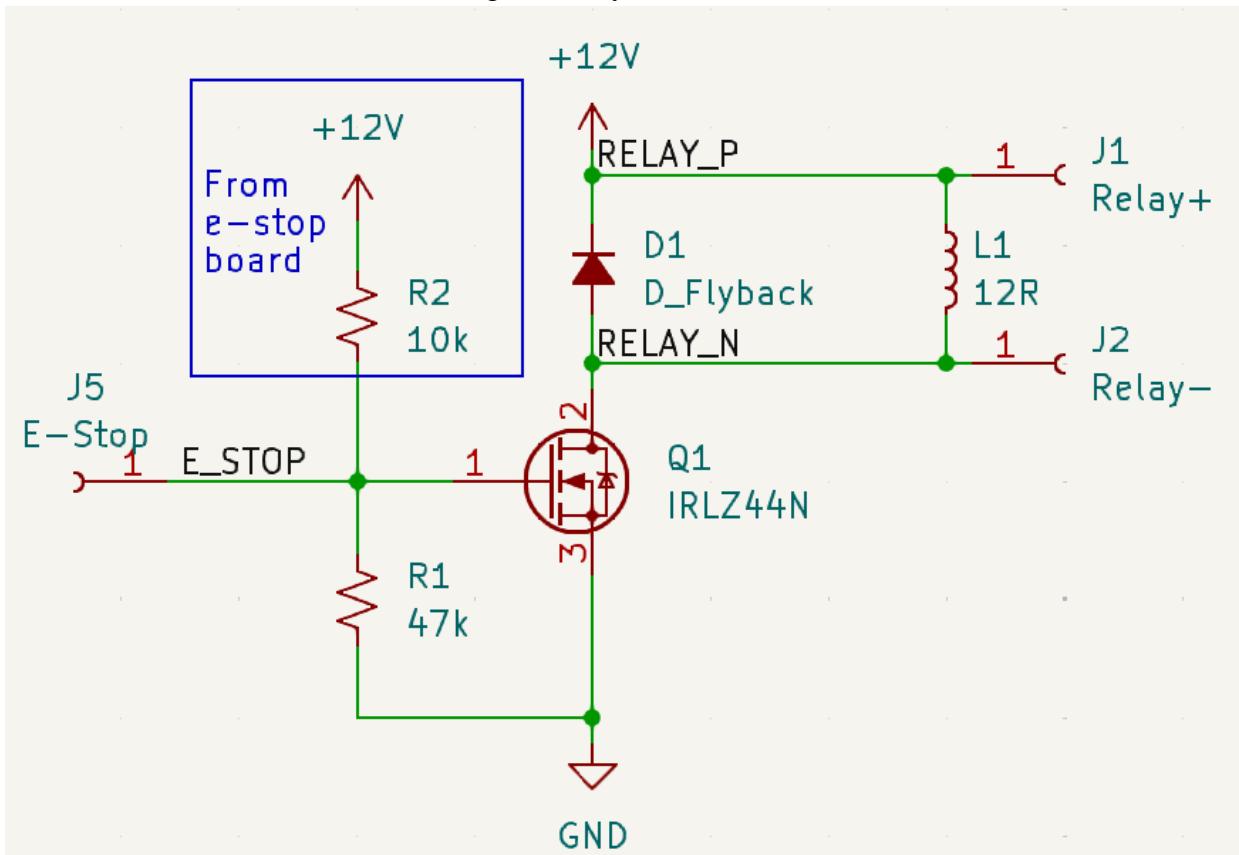


Figure 67: Power cut-off circuit with relay protection

The Figure below shows the MOSFET and flyback diode circuit as well as the powerful relay used to switch on/off the drive motor.

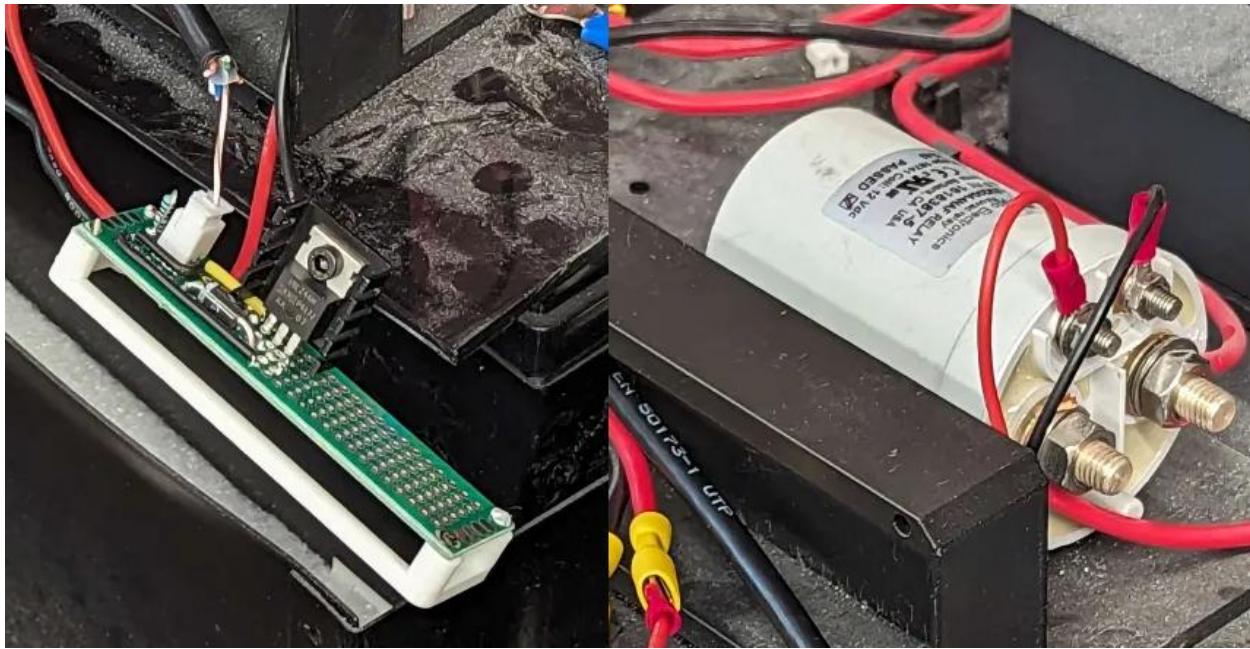


Figure 68: MOSFET and Flyback Diode and Power Cut-Off Relay

The E-Stop control system is incomplete without the aforementioned Interface Board which is the next topic of discussion. The Interface Board performs **four primary functions**: providing regulated 12V, 5V, and 3.3V power rails; asserting the active-low E-Stop trigger; monitoring E-Stop status; and distributing E-Stop status and control signals across multiple interconnected boards.

The Interface Board takes in 12V from a voltage regulator connected to the 60V battery pack and filters it through a LC filter using a ferrite bead as can be seen in the schematic below:

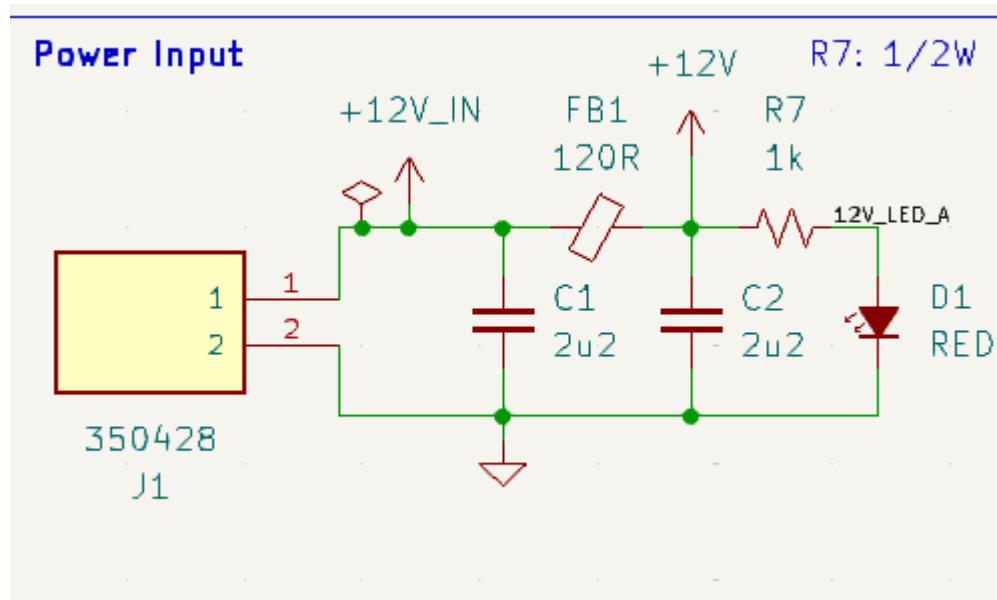


Figure 69: Power Input of the Interface Board

The filtered 12V power is fed into a Texas Instruments TPS565208 Buck Converter IC, configured to regulate the 12V input power to a steady 5V with a max current draw of 5A. R1 and R2 were chosen according to the datasheet provided by TI. The tolerance for these resistors should be as close to 0% as possible, 1% resistors were chosen for this design.

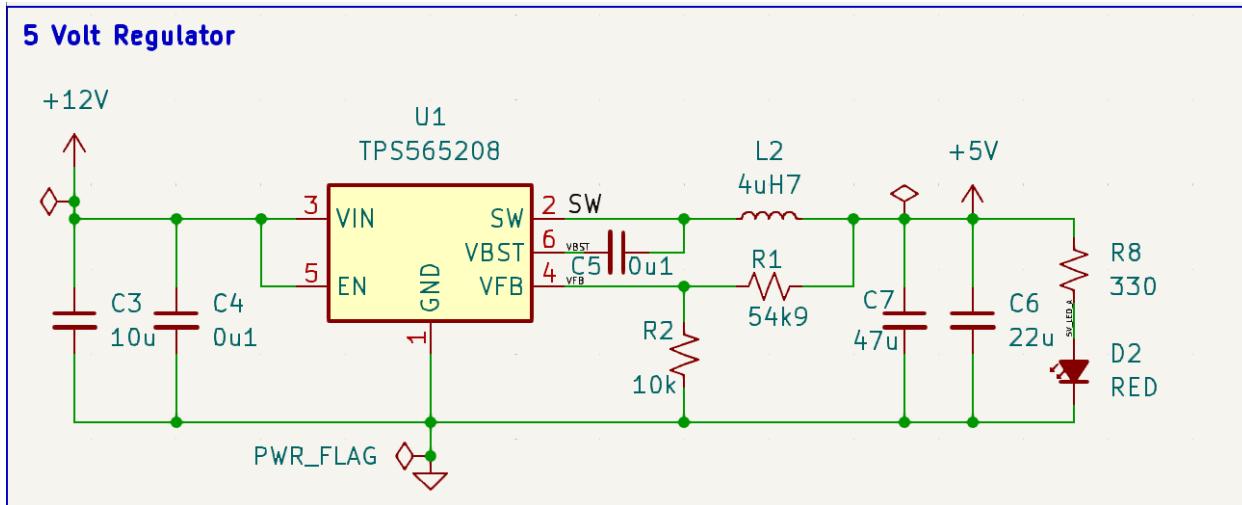


Figure 70: 5V Buck Converter Circuit

The regulated 5V is then fed into an AMS1117-3.3 LDO Linear Voltage Regulator which can supply up to 1A. There was no expectation for currents greater than 100mA to be drawn from the 3.3V rail, however the 1A option gives room to expand to accommodate future use cases. The efficiency of these linear regulators is inversely proportional to the current draw multiplied by the voltage drop across the input and output. As such, it was more efficient to supply the input with 5V.

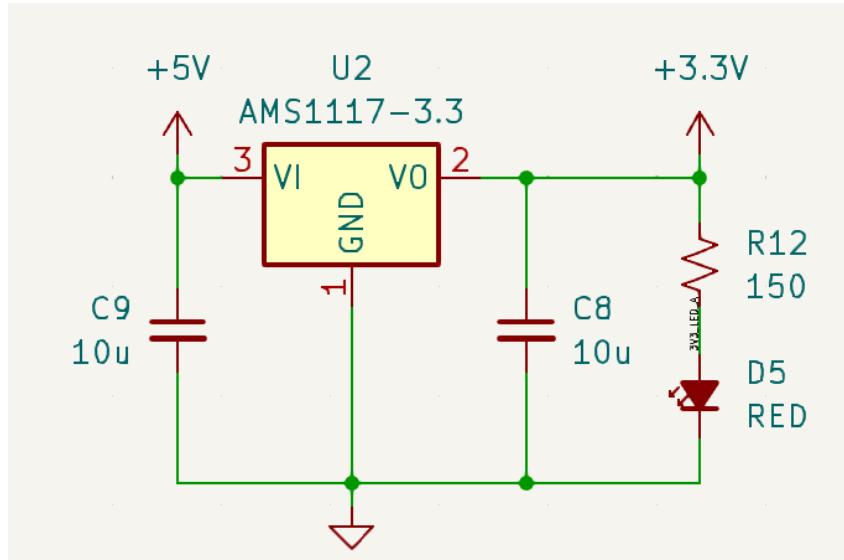


Figure 71: 3.3V Regulator Circuit

Finally, the 12V output of the interface board is fused at 1.5A with resettable polyfuse. This can be bypassed by drawing directly from the 12V input in case 1.5A is insufficient for the use case. It's worth noting that the 1.5A limit was determined by the width of the traces used while routing.

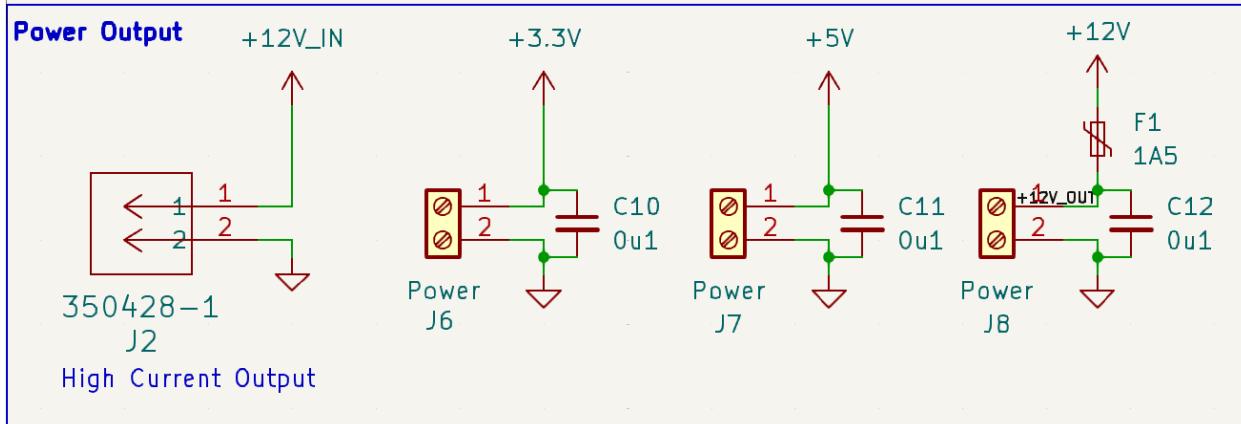


Figure 72: Voltage Rail Outputs with 12V Bypass.

The Interface Board also provides an interface for reading the E-Stop status and for triggering the E-Stop. The E-Stop signal cannot be read directly by standard microcontrollers as it uses 12V logic and so it is placed through an inverting buffer. This prevents damage to systems wanting to read the E-Stop status as well as prevents loading the signal by using a voltage divider to level shift the signal. Systems can also trigger an E-Stop applying 3.3V or 5V at the **trigger_estop** input (J9 Pin 1). This brings the **estoptrigger** signal down to ground which corresponds to the **TR_IN** input on the E-Stop board. As the Interface Boards are meant to connect together and share the **estoptrigger** and **eStop** signals, this means that the E-Stop Trigger circuit is another open-drain circuit.

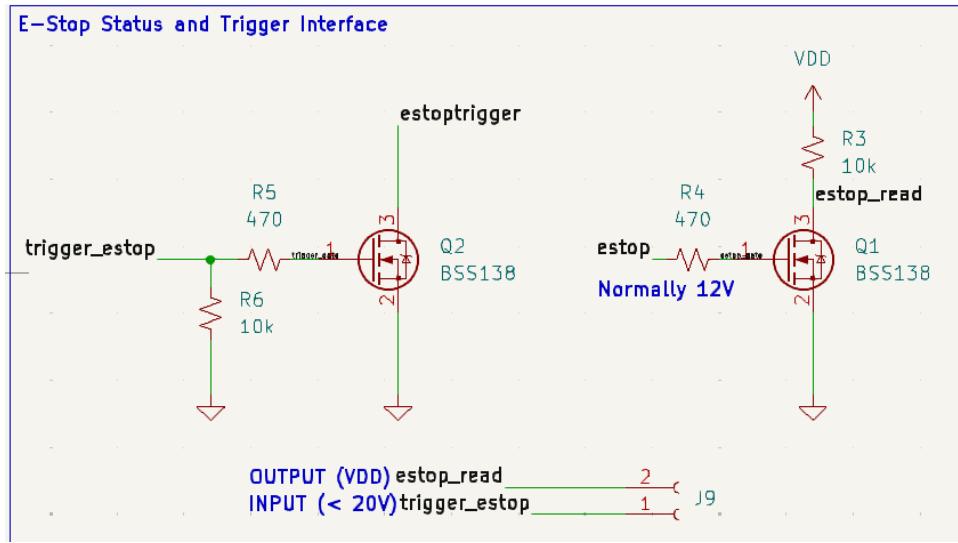


Figure 73: E-Stop Status and Trigger Interface

The **estop_read** signal is the signal meant to be monitored by subsystems. It is the complement of the **eStop** signal and can be either 3.3V or 5V. The voltage VDD can be selected using a shunt cap commonly found on commercial circuit boards like DC motor drivers. ESP32 and the Arduino UNO use different logic levels (3.3V and 5V respectively). By providing an interface to switch between the 2 logic levels, it is no longer necessary to restrict the hardware use in the project.

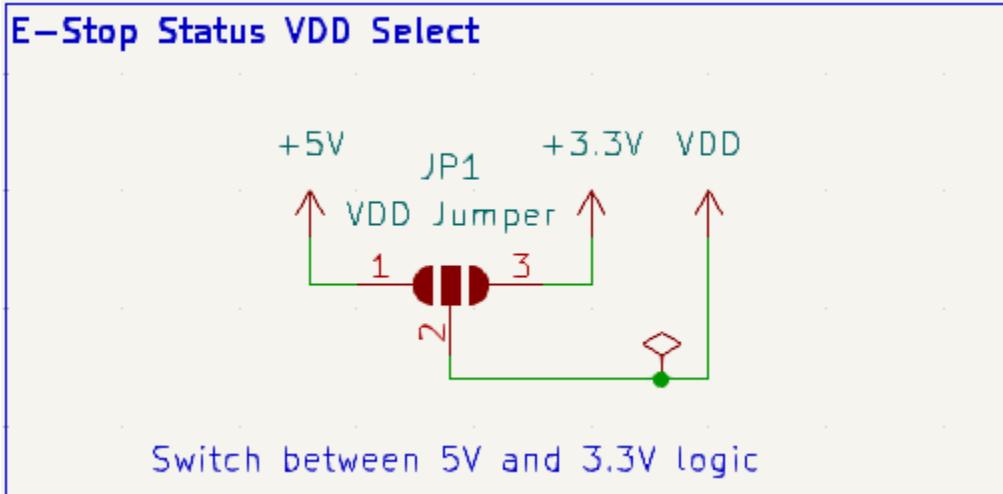


Figure 74: VDD Selection

Finally, the Interface Board uses RJ45 jacks to distribute the **eStop**, **estoptrigger**, and 6 additional signals between other Interface Boards using Ethernet cables. During the 2024-2025 year, we did not end up populating the 6 additional signals, but they allow for future expansion if the need arises. The 8 RJ45 signals are broken out into a 2 pin JST-XH connector for the **eStop** and **estoptrigger** signals and a 6

pin JST-XH connectors for the remaining 6. Locking connectors were used wherever possible due to the high vibration environment.

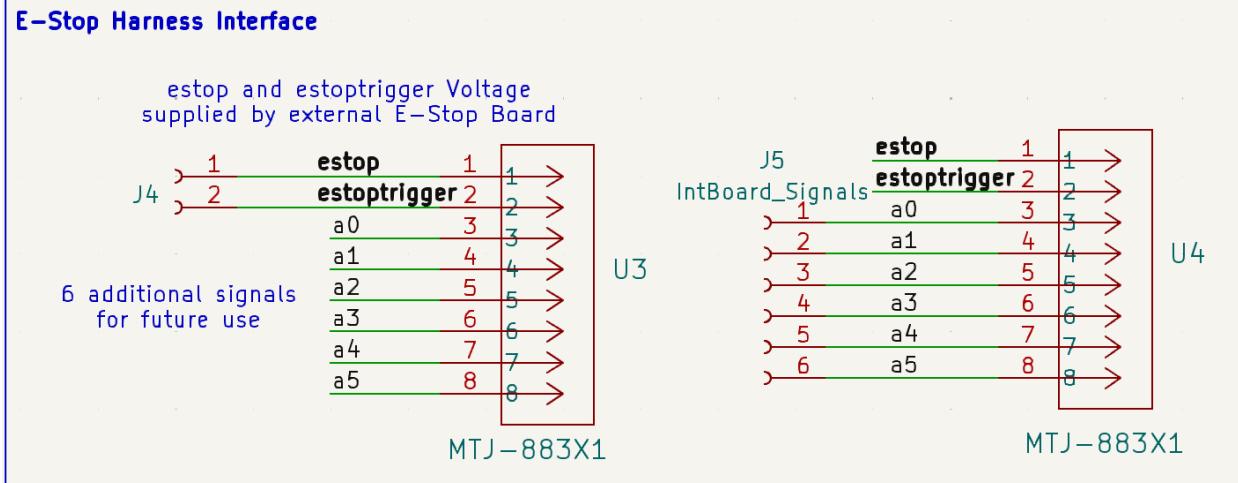


Figure 75: E-Stop Harness Distribution and Breakout

The layout and the schematic for the Interface Board were designed using KiCad. The power routing utilizes large copper polygons to handle larger currents than what a trace could provide, particularly around the 5V output where the highest current draw was expected. Indicator LEDs for the 12V, 5V, and 3.3V power rails were included for debugging purposes, with series resistors tuned for each rail such that the current draw was ~20 mA for each LED, ensuring equal brightness. Once again, the design features two RJ45 jacks, one for input and the other for output so that the Interface boards can be chained together. The bottom copper layer was set as a ground plane to ensure signal stability.

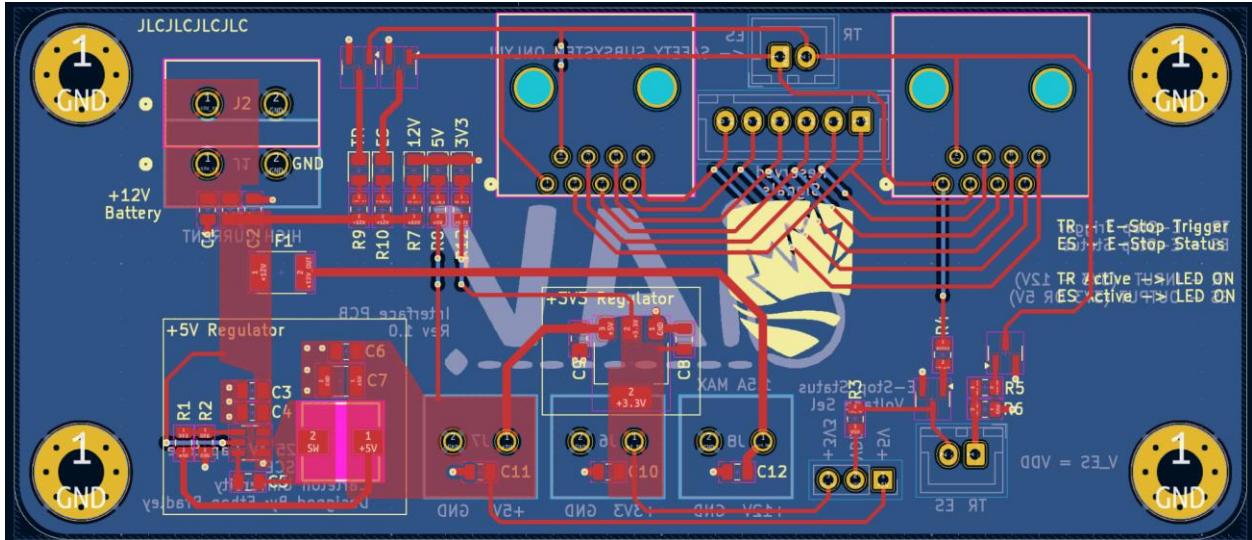


Figure 76: Interface Board Layout

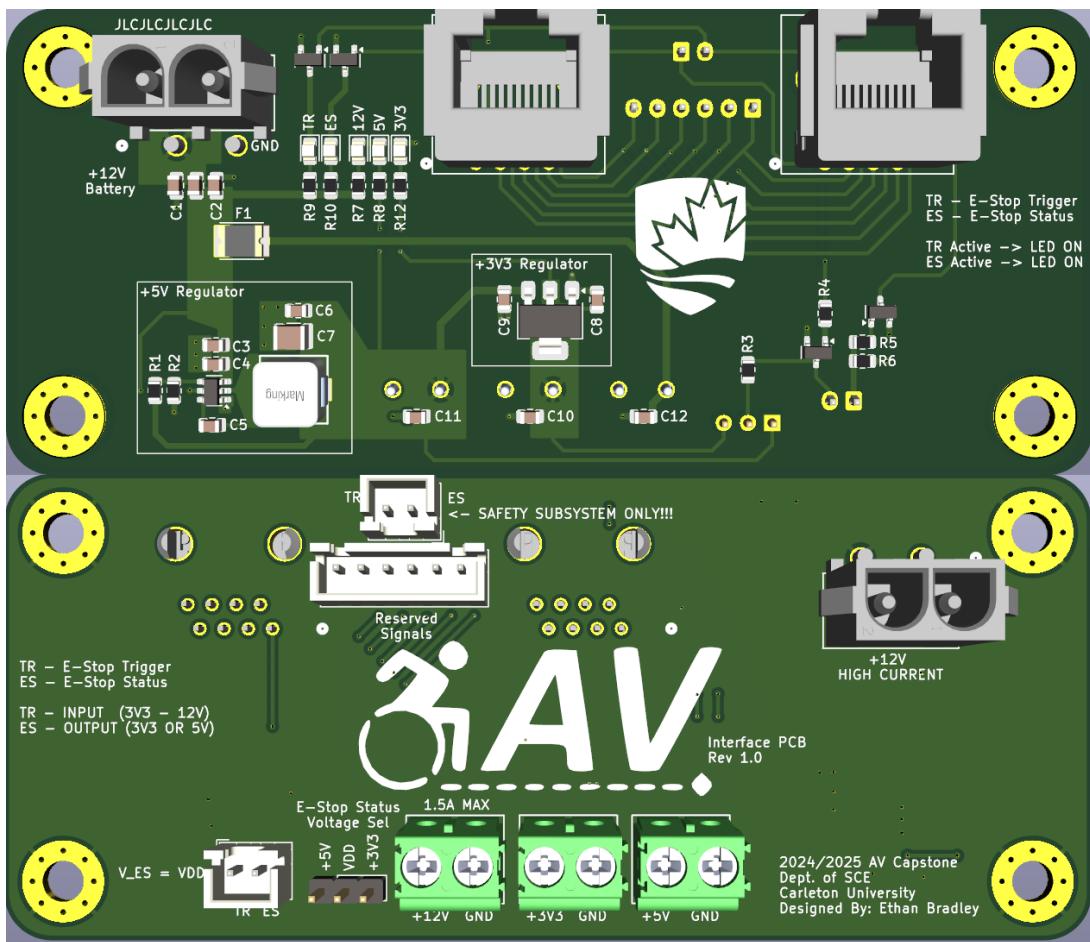


Figure 77: Interface Board 3D Model Front and Back

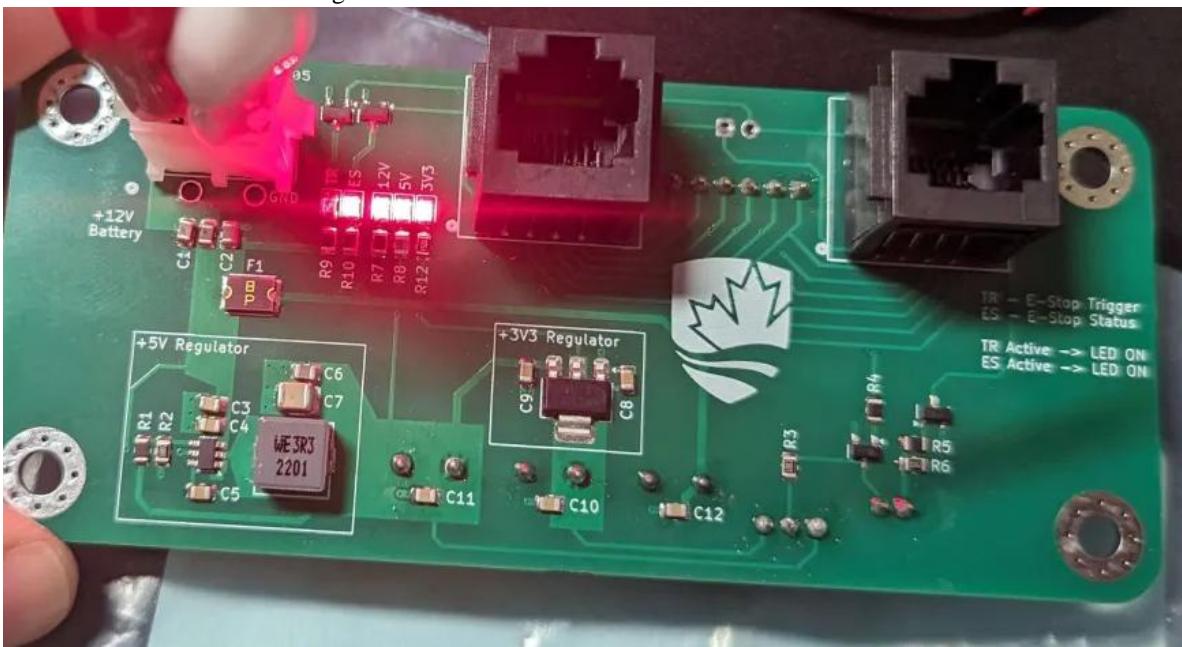


Figure 78: Interface Board Powered On



Figure 79: Interface Boards Connected Together

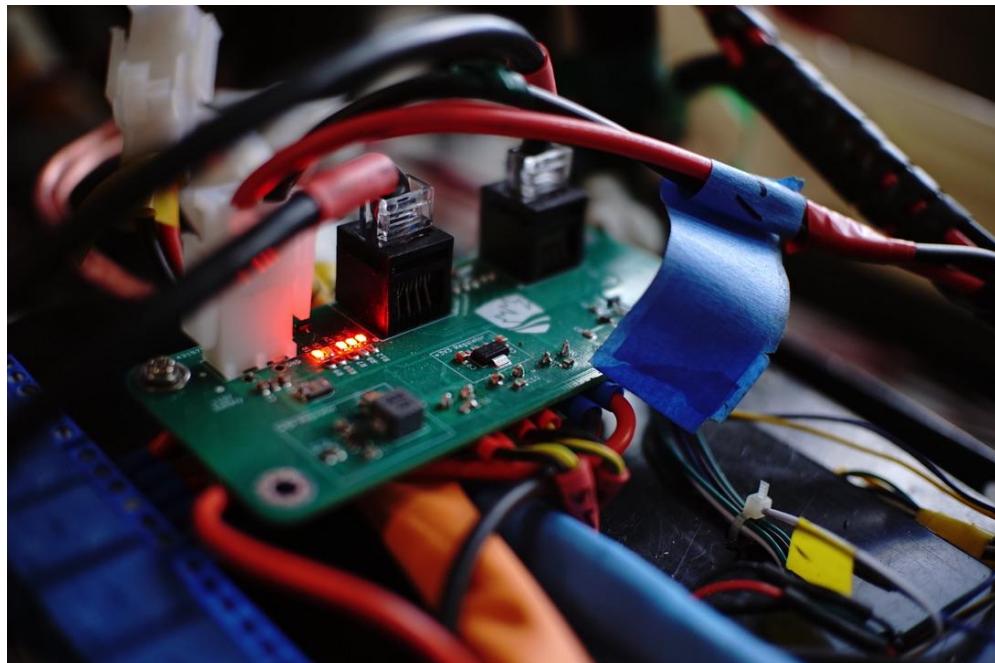


Figure 80: Interface Board Used by Autonomous Control Team

Finally, a box for the E-Stop subsystem consisting of an Interface Board, the E-Stop board, a Raspberry Pi, the Kar-Tech Wireless E-Stop was designed such that everything could be mounted together all in one place, which can be seen in the renders below. The top of the lid provides slots to mount the Wireless E-Stop receiver with M4 screws.

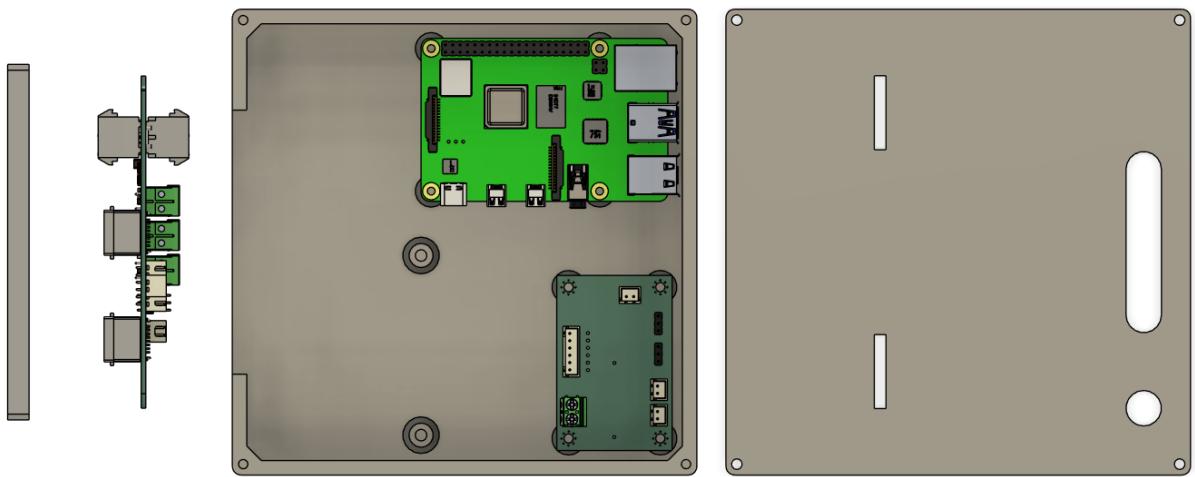


Figure 81: E-Stop System Housing



Figure 82: Render of E-Stop System Housing



Figure 83: E-Stop Housing with Kar-Tech Wireless E-Stop Mounted

The manual override system centers around the Flysky FS-i6X receiver, which was desoldered from its original housing and directly mounted to the main control board. The receiver now communicates via serial IBUS protocol, eliminating the need for PWM signal conditioning. As a result, no voltage level shifting is required, since the data is transmitted digitally over a single serial line directly compatible with the control logic. The hardware arbitration circuit uses analog switches to physically disconnect ROS control signals when manual override is active, with status LEDs providing immediate visual feedback of control mode.



Figure 84: Controller Mapping

9.4.4. Software Design

The software architecture implements a multi-layered approach to ensure both real-time responsiveness and system safety. At the ROS layer, critical nodes were developed in C++ for performance, while higher-level monitoring functions use Python for rapid development. A notable exception to this is the E-Stop node which was not software reliant, and so was developed in Python.

The manual override system employs a dual-path software architecture. The primary path uses a Micro-ROS node on the ESP32 to process RC inputs and publish normalized control messages. A secondary, bare-metal firmware on the same ESP32 implements a failsafe path that directly drives the motor controllers if ROS communication is lost. The mode arbitration logic uses hardware timestamps to resolve conflicts, always favoring the most recent valid command regardless of source.

The E-Stop system uses Python and rclpy to implement a ROS2 node capable of publishing the state of the E-Stop to the ROS2 network. Using the RPi.GPIO Python package, the E-Stop monitoring pin, “**read_estop**”, from the Interface Board connected to GPIO pin 23 on the Raspberry Pi could be read once every second to update subscribers to the “/e_stop” topic, publishing a boolean value. The E-Stop system also used a separate Python program to trigger E-Stops. This Python program created a ROS2 node that subscribed to a /trigger_estop topic, taking in a boolean value. When the subscriber node detected a “True” message to the /trigger_estop topic, it would bring GPIO pin 24, connected to the **trigger_estop** pin on the Interface Board, HIGH, in turn triggering an E-Stop. Simultaneously, the /e_stop publisher would detect on pin 23 that the E-Stop had been triggered and begin publishing a True value. The ROS2 Python nodes

were configured to run on when the Raspberry Pi powered on using systemd services. The startup script sources the ROS2 environment before running the Python scripts hosting the ROS2 nodes. The systemd services can be found under `/etc/systemd/system/e-trigger.service` and `/etc/systemd/system/e-trigger.trigger`.

Due to complications involving the Kar-Tech Wireless E-Stop at the end of the semester, it became necessary to provide a web interface to trigger an E-Stop. The backend was written using Node.js using **Express.js** to serve the website, **Socket.io** for client-server web socket communication, **roslibjs** to provide an API to interact with ROS2 using JavaScript, and finally **rosbridge** which provided a web socket server that the Node.js app could communicate with the rest of the ROS2 network. The rosbridge server was the heart of this operation, allowing the Node.js app to publish to the `/trigger_estop` topic and subscribe to the `/e_stop` topic. The Express.js web server served a single webpage with a very simple interface. It shows the user if the E-Stop is active, tells the user if *they* are the ones triggering the E-Stop, and provides a big red button to trigger an E-Stop.

E-Stop Interface

E-Stop: **ACTIVE**

You are **TRIGGERING** the E-Stop



EMERGENCY STOP

Figure 85: E-Stop Web User Interface

When there are no clients connected to the web server, True is published to the `/trigger_estop` topic, effectively activating the E-Stop by default. When a client connects to the website and the socket connection is established, by default they trigger the E-Stop. It is not until they press the red button, that they are no longer triggering an E-Stop themselves.

E-Stop Interface

E-Stop: **ACTIVE**

You are **NOT TRIGGERING** the E-Stop



Figure 86: E-Stop With Client not Triggering E-Stop

The E-Stop will remain active until all connected clients disable their E-Stops from their session. Elaborating on this, the E-Stop will remain active if the Wireless E-Stop or the Physical E-Stop are active as well and the web interface provides feedback to show this. When a client disconnects from the server, their E-Stop is deactivated. If all users disconnect from the server, the E-Stop is reactivated until someone reconnects to the server and turns the E-Stop off, or the Interface Board E-Stop is disabled on the E-Stop Board. The socket server features a very aggressive ping timeout. Clients connected to the socket server are pinged every 750ms and have 1500ms to respond. If the 1500ms elapses with no response, the server forcibly disconnects the client and once again, when all clients are disconnected the E-Stop is activated.

```
57  const server = http.createServer(app),
58  const io = new Server(server, {
59    pingInterval: 750,
60    pingTimeout: 1_500
61  });
```

Figure 87: Aggressive Timeout Parameters

9.4.5. Integration & Testing Approach

The integration process followed a phased approach beginning with component-level validation. Each E-Stop activation path was tested independently to verify response times. The manual override circuitry underwent signal integrity testing with a 4-channel oscilloscope to verify timing margins and noise immunity.

Subsystem integration focused on interface validation. The ROS-to-hardware interface was tested by injecting faults (network disconnections, corrupted messages) while monitoring system response. A custom test simulated various RC input scenarios while measuring end-to-end latency using potentially the high-speed cameras to capture the exact moment of physical response. Power

subsystem tests included brown-out scenarios with programmable power supplies to verify graceful degradation.

System-level testing employed both simulated and real-world scenarios. The test included:

- 5-hour endurance testing with automated mode switching
- Simultaneous fault injection (E-Stop activation during manual override)
- RF interference testing using a signal generator at 2.4GHz
- Load testing of the web interface with simulated/concurrent users

Field testing progressed from controlled environments to increasingly complex real-world scenarios. Initial tests on a static test bench verified basic functionality, followed by controlled-area low-speed operation, and finally full-speed testing in varied environmental conditions. All tests were documented with detailed logs including system metrics, environmental conditions, and any observed anomalies.

9.4.6. Challenges & Troubleshooting

Having a manual override system through the Flysky controller was a challenge from an architectural as well as technical perspective. It was one of the top concerns as to whether the manual input handling should be done in a separate microcontroller or on the main control board directly. While the stand-alone microcontroller ensured modularity, it introduced the risk of delay. By tests of latency measurement and real-world testing, we determined direct integration provided faster responses without compromise of system stability. Calibration of the physical override button on the Flysky controller was also a point of challenge that required tweaking in order to provide consistent signal identification and reliable switching of modes. To reduce the delays in communication, we cross verified PWM and serial protocols and went ahead with using IBUS serial communication for its effectiveness and compatibility for broadcasting multiple channels over a single pin. We also required an adequate fail-safe function. The system was configured to engage the braking automatically if there was loss of the controller signal in order to safely halt the vehicle. Lastly, efficient mode transition logic design ensured the system did not crash or get bricked in mode switching from manual to autonomous control. Sufficient testing ensured the system reliably recovered and was correctly able to recognize its executing state after every mode transition.

Several problems arose during the development of the E-Stop and Interface boards. The assembly of the boards was done manually and there was a very large labour cost. Only one E-Stop board was required for the project and did not take so much time. On the other hand, we required at least 5 interface boards for the subsystems requiring power plus extras in case of reverse polarity, component failure, or other human error. In total, nine Interface Boards were assembled, 4 of which were mounted in the vehicle, one ruined, and four placed in storage. The assembly of all

these 9 boards was very labour intensive and the quality between the boards was not consistent leading to cold joint problems later on in the testing cycle.

Another challenge was the discovery of the Active Hi-Z nature of the Kar-Tech Wireless E-Stop. It was thought previously after probing and reading the manual that the Wireless E-Stop was Active-Low, however after the E-Stop began to behave erratically when triggering with the Wireless E-Stop we discovered that it was actually Active Hi-Z. This problem was solved by soldering a $10\text{k}\Omega$ resistor between the Wireless E-Stop input and ground on the E-Stop board. The PCB design shown earlier in the [Hardware Design section](#) has the corrected design.

The relay circuit also proved difficult during testing. Originally, there was not a pull-down resistor on the MOSFET controlling the relay.

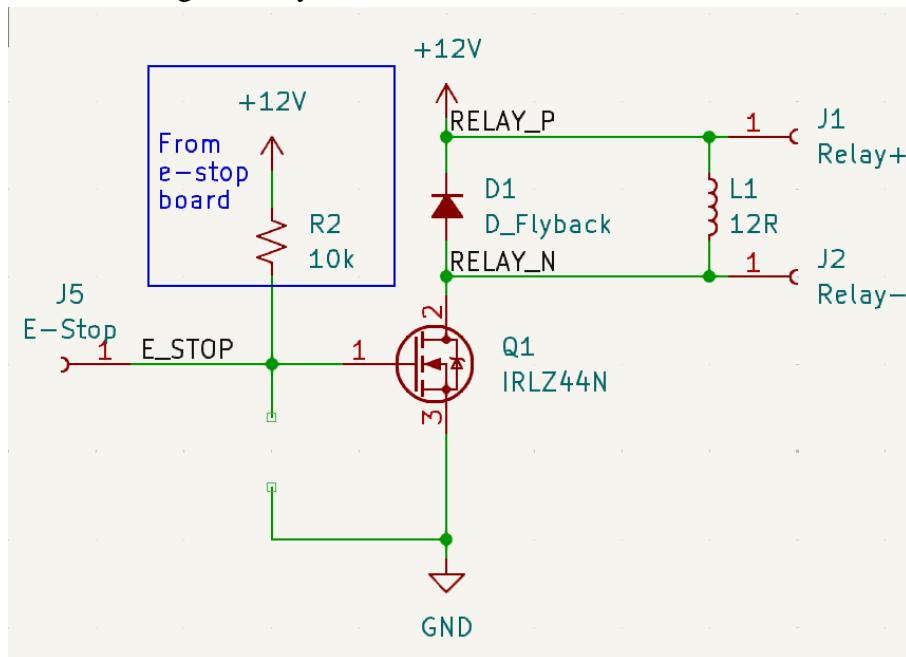


Figure 88: MOSFET circuit without pull-down resistor

This means that when the E-Stop control circuit was disconnected from the relay circuit, the gate of the MOSFET was left floating. This causes the MOSFET to operate non-deterministically. Without a defined state, the MOSFET can flip on and off whenever *or* it can operate in the *ohmic* region where it can still conduct $> 1\text{A}$ but the resistance between the drain and the source is much larger than usual generating a lot of heat. This problem was solved by adding in the $47\text{k}\Omega$ resistor mentioned in section 9.4.3.

The software integration between ROS2 and the E-Stop had many challenges as well. The E-Stop state was read through the 3.3V output on the Interface board on pin 23 of a Raspberry Pi and the E-Stop trigger pin was connected to pin 24 on the Pi. The E-Stop ROS2 node was implemented using Python, however to begin, the regular user did not have permission to access the GPIO pins. As a quick fix, the script was launched using the root user, however this ended up being a double

edged sword. When the ROS2 node was launched as the root user, the topics that the node published and subscribed to were not advertised across the ROS2 network meaning that no other subsystem was able to detect or trigger an E-Stop through ROS2. Ultimately, it took adding the regular user to the dialout group in Ubuntu and running the ROS2 node as the regular user to get the topics to appear across the network.

Finally, the Kar-Tech Wireless E-Stop transmitted completely stopped working towards the end of the year and right before the first test run attempt to the P7 parking lot at Carleton. Had it not been for the E-Stop web interface, we would have been severely limited in the tests we'd be able to perform for the second test run in parking lot P7. The Wireless E-Stop receiver will not deactivate until the transmitter connects to it and so the enable interface discussed in the [Hardware Design](#) section was used to disable the Wireless E-Stop. Modifications to the hardware include 2 holes drilled into the lid of the E-Stop Subsystem Housing, with SPDT toggle switches connected to the enable interface of both the Wireless E-Stop and the Interface Board E-Stop. The ability to enable / disable the E-Stop during testing proved to save a lot of time.

9.5. Project Management & Execution

The Autonomous Vehicle project followed a structured development process under agile-inspired weekly sprints. Our team divided tasks based on subsystems (navigation, control, safety, etc.), with dedicated roles to streamline execution. As part of the Safety & Reliability team, I focused on implementing robust manual override features and integrating the FlySky RC controller for remote vehicle control during testing and emergencies. Regular integration checkpoints and collaborative debugging ensured functional compatibility across all modules.

9.5.1. Work Breakdown Structure

The Work Breakdown Structure (WBS) organizes the project into clearly defined tasks assigned to specific team members. It breaks down the RC controller integration, E-Stop system, and remote monitoring into manageable components like hardware setup, PCB fabrication, and interface development. This structure ensures clear ownership, enables parallel development, and helps track dependencies and timelines effectively.

Table X: Work Breakdown Structure

Task ID	Task Name	Assigned Team Member(s)
1.1	RC Controller Hardware Integration	Rahul Cheruku, Ali Nadim
1.2	RC-ROS Communication Calibration	Rahul Cheruku, Ali Nadim
2.1	E-Stop Circuit Design	Ethan Bradley

2.2	E-Stop PCB Fabrication & Testing	Ethan Bradley
3.1	Real-time Data Logging	Arjun Pathak, Ali Zaid
3.2	Remote Dashboard UI Development	Arjun Pathak, Ali Zaid

9.5.2. Gantt Chart & Milestone Tracking

The Gantt chart provides a visual timeline of key project phases and tasks, from initial design to final validation. It outlines the start and end dates for each milestone, helping track progress and ensure deadlines are met. By mapping tasks sequentially, it highlights dependencies and allows the team to coordinate efforts efficiently throughout the project lifecycle.

Table X: Gantt chart

Phase / Task	Description / Activities	Deliverables / Revised Completion Date
Design Phase	Define system requirements, develop architecture, design safety systems	System requirements document, design specifications
Development Phase	Prototyping and initial implementation of safety systems	Functional prototypes, embedded software, initial tests
Final Calibration of RC Controller with ROS	Complete final calibration and ensure real-time data exchange between Arduino and ROS.	End of January
Brake activation due to E-Stop Trigger	Brakes need to be activated in the event of an E-Stop, validated by vehicle halt on trigger.	Projected: End of January Completed: End of March
Testing & Validation	Conduct unit, integration, and stress testing of safety systems	Test reports (unit, integration, stress), system refinements
Design and Fabrication of E-Stop PCB	Design schematic and layout of custom E-Stop PCB for reliable functionality.	End of February
Refinement of Remote Monitoring Interface	Improve data accuracy and finalize UI to display real-time data.	End of February
Final Integration & Deployment	Full system integration and deployment for final testing	Fully integrated system, final validation reports, project

		deployment
System Integration Testing	Test integration of all subsystems in a simulated environment.	End of March
Final Validation and Safety Checks	Final system validation, focusing on safety in real-world conditions.	End of March

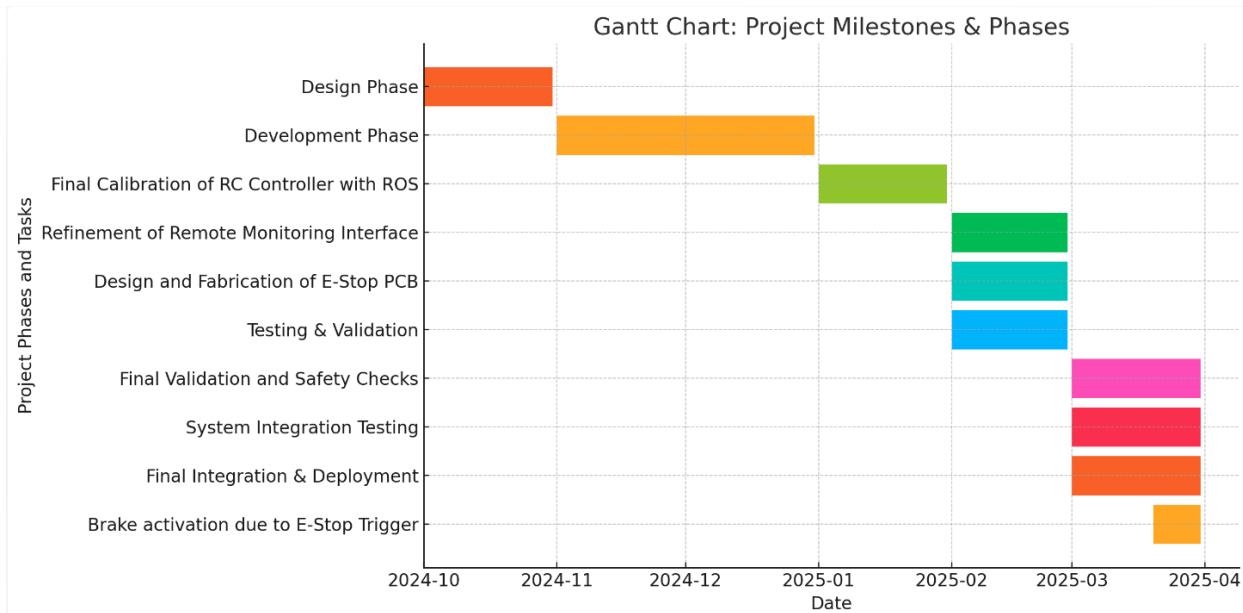


Figure 89: Gantt Chart for Safety and Reliability team

9.5.3. Risk Analysis & Mitigation Strategies

One of the major risks identified was signal interference or loss from the FlySky controller. Since manual override depended heavily on real-time communication, any disruption could result in unsafe behavior. To mitigate this, we implemented signal strength monitoring and configured a failsafe routine that immediately halts the vehicle upon signal loss.

Another challenge was the possibility of unexpected vehicle behavior during manual override. Since human input could override autonomous control, it was crucial to ensure smooth transitions and prevent erratic motion. This was handled by integrating a deadman switch mechanism and applying response dampening to the control signals, ensuring predictable and safe operation.

The development of the custom E-Stop PCB came with its own risks, mainly related to electrical reliability and system compatibility. Any failure in this component could compromise the entire safety system. To reduce this risk, we thoroughly tested the PCB design using simulation tools and verified connectivity and performance across all trigger sources before fabrication. Furthermore, the manufacturer of the PCBs (JLCPCB) performs electrical connectivity tests after manufacturing and all designs are subject to review before they are fabricated.

There was also the risk of integration conflicts across subteams, especially between the safety, control, and navigation modules. These were addressed through weekly integration meetings and continuous communication. We maintained version control using Git to ensure that software updates didn't break existing functionalities.

Lastly, inconsistent hardware behavior posed a reliability risk. Sensors and modules sometimes performed differently under real-world conditions. To mitigate this, we bench-tested all hardware under varied conditions, validating their performance and ensuring stability before final deployment.

9.5.4. Budget & Cost Analysis

The funding for the Reliability & Safety team was acquired by applying to KEFC. As a Capstone project for the 2025 Winter semester, which is the applicable term for the R&S budget, we were approved for \$1350, with our subteam being allocated \$450 of that sum. The cost consists of a Raspberry Pi with a case and SD card for \$124.73, the remainder of which would be used for components for the Interface Board and the E-Stop Board. In total, we spent \$244.60 of our KEFC budget on PCB components such as capacitors, resistors, inductors, MOSFETs, various connectors, etc.. We ended up spending \$369.33 of our allocated \$450 placing our sub-team well under budget as far as our KEFC funding went.

Unfortunately, small and medium costs ended up accumulating over the course of the year, including the manufacturing costs of the Interface and E-Stop boards themselves, none of which could be reimbursed by KEFC. As we had no budget for these costs, we bore these costs ourselves and expect to have gone about \$250 over-budget.

9.6. Experimental Setup & Testing

The Safety and Reliability subsystem underwent rigorous validation through controlled lab testing and real-world field trials. Initial evaluations in the EDC building focused on

baseline performance metrics, including manual override response times and E-Stop reliability under ideal conditions. Subsequent field testing along the EDC-to-UC route introduced environmental variables like terrain variations and RF interference, providing critical data on system robustness.

9.6.1. Testing Environment & Procedures

a) Manual Override Architectures in Autonomous Systems

The manual override system was rigorously evaluated through two distinct testing phases designed to assess performance under both controlled and real-world conditions. The initial controlled environment testing took place within the Engineering Design Center (EDC) building, utilizing the indoor lab spaces and connecting corridors as a predictable testbed. This phase focused on fundamental operational validation, including basic maneuverability tests (straight-line motion, turning radius verification) and quantitative latency measurements between RC input and vehicle response. A critical component of these tests involved evaluating E-Stop reliability by executing abrupt stops during active motion scenarios.

The second field testing phase involved navigating a route from EDC to the University Center (UC), exposing the system to diverse terrain challenges including paved walkways, gravel sections, and moderate inclines. This outdoor testing provided essential data on signal stability across varying distances (5-30m operator separation) and environmental interference conditions. The route was deliberately selected to include areas with known RF signal challenges near steel structures, enabling comprehensive evaluation of the hardware-based override's resilience. Standardized test protocols mandated trial runs per scenario, collecting both quantitative metrics (response latency, power consumption rates) and qualitative operator feedback on control intuitiveness.

b) E-Stop Architectures in Autonomous Systems

The testing environments for the E-Stop system were identical to that of the Manual Override as they needed to be tested together to ensure operator safety. As mentioned in Part a), abrupt stops were injected into the vehicles motion to test how the vehicle would recover after the E-Stop was released.

During the field testing phase, the E-Stop was tested in varying terrain conditions such as paved walkways, gravel sections, and moderate inclines as previously mentioned. The E-Stop was configured to activate the brakes when triggered and so many of the field tests involved observing how well the vehicle would stop in the varying terrain.

During the test conducted on April 6th, we set out to test the range of the Web Interface E-Stop via ROS2 and to verify the disconnect behaviour of the web interface. The test involved taking the vehicle out from the Engineering Design Centre at the back of the Mackenzie Building and driving it out to Parking Lot P7 near the entrance of the University. As the web interface relies on a Wi Fi connection for external devices, we wanted to observe how far we could be away from the vehicle before the web interface defaulted to an E-Stop.

c) Remote Monitoring Architectures in Autonomous Systems

The monitoring dashboard of the autonomous vehicle went through a systematic test evolution, with a view to validate the working of the system under gradually more realistic scenarios. The initial testing tactics employed the use of mock data generators, which were used to verify the core components of visualization and UI actions of the dashboard—that is, that all UI objects appeared correctly, responded properly to various data conditions, and so forth. This mock interface allowed for several rapid iterations without requiring a live backend connection to prove out the status indicators, gauges, and control elements with simulated vehicle telemetry.

Once initial validation was done with the front-end components, the testing proceeded on for integration into the Carla simulation environment. This tested the connection between the data sources in the backend and the visualization in the front end, validating the pipeline concerning data flow and the dashboard's capability to process and display live information in real-time from simulation. Carla integration testing was fundamental to understanding how the dashboard would react in a large variety of simulated driving scenarios and edge cases, typically difficult to test with a real car.

The last phase of testing was to connect the dashboard with the actual autonomous vehicle systems of the true production environment. In this end-to-end validation, all links from the vehicle sensors, through data processing to the operator interface, were confirmed to be functioning correctly. This graduated testing—from mocks to simulators and then through to actual vehicle integration—gave a good confidence in the reliability of the system while providing an efficient mechanism for design iteration. The ruggedness features of the dashboard with respect to all testing were specifically analyzed to confirm that, when connectivity was lost, it could gracefully regress to mock data so that remote monitoring operations can proceed—a key capability.

9.6.2. Experimental Results and Findings

a) Manual Override Architectures in Autonomous Systems

The manual override system underwent extensive evaluation through both controlled environment testing and real-world field trials, with several important observations emerging from these validation exercises. During initial testing in the EDC building, we identified that the throttle response through the Flysky RC controller required significant calibration refinement. The default settings produced nonlinear acceleration characteristics that made precise speed control difficult - gentle stick movements would sometimes result in either sluggish response or unexpectedly sharp acceleration. This was particularly problematic during low-speed maneuvering tests where smooth, incremental adjustments were needed for proper vehicle positioning.

After analyzing the control signal data, we implemented several key adjustments to the throttle response curve. First, we modified the PWM-to-speed mapping to create a more progressive relationship between stick position and motor output. Second, we added a small deadzone at the neutral position to prevent unintended creep. Finally, we incorporated exponential smoothing to the input signals to eliminate jerky transitions between speed levels. These modifications were tested incrementally, with each iteration showing measurable improvement in control precision.

The effectiveness of these adjustments became clearly evident during subsequent testing phases. In the refined configuration, operators could execute smooth acceleration and deceleration profiles with much greater consistency. The system demonstrated particularly strong performance during the transition tests between autonomous and manual modes, where the updated throttle response helped eliminate the abrupt lurching that had been observed previously when control authority switched between systems.

Field testing along the varied terrain of the EDC-to-UC route provided further validation of these improvements. The modified throttle mapping proved equally effective on inclines and uneven surfaces, maintaining predictable control response regardless of load conditions. Operators reported significantly improved confidence when navigating through crowded areas or executing precision maneuvers, noting that the refined control characteristics allowed for more intuitive speed modulation.

Key Refinements Implemented:

- Redesigned throttle response curve for progressive acceleration

- Added neutral position deadzone to prevent unintended movement
- Implemented input signal smoothing for jerk-free transitions
- Verified consistent performance across all operating conditions

These iterative refinements underscored the importance of thorough control system calibration, particularly for safety-critical manual override functions. The testing process revealed that even seemingly minor adjustments to the control mapping could have substantial impacts on overall system usability and operator confidence. The final configuration achieved through this process represents a significant improvement over the initial implementation, delivering the precise, predictable response needed for reliable manual control.

b) E-Stop Architectures in Autonomous Systems

During the field test phase, we observed that the E-Stop system in conjunction with the brake activation performed reliably while on paved roads and on level ground. Unfortunately, we found that the wheels tend to skid when the brakes are applied going downhill on paths with lots of sediment, like gravel or rock salt. Professor Dansereau expressed a desire to fix this by introducing ABS brakes in future years. While ascending the University Centre tunnel, the decision was made to not engage the E-Stop unless absolutely necessary. This was to prevent the vehicle from free-rolling down the tunnel slope when the brakes released.

During pauses in the field test, the E-Stop was activated and no abnormalities were detected. The brakes remained engaged and the 60V power to the motors remained switched off by the relay. It's worth noting that during this field test, the web interface E-Stop had not yet been implemented as we still had and were relying on the Kar-Tech Wireless E-Stop for long range E-Stops.

On the April 6th test, as the Kar-Tech Wireless E-Stop was no longer in service and the Physical E-Stop had already been proven to be reliable, we wanted to test the Web Interface E-Stop, specifically how it handles multiple clients, the range at which we can trigger the E-Stop, and the latency between pressing the button and the E-Stop triggering. The web interface was able to handle multiple clients flawlessly. When a user first connects to the UI, the client asserts an E-Stop by default and when they disconnect, the server would clear their trigger. In the event of all users disconnecting from the server by closing the tab, the server would assert an E-Stop until a new client connected and deactivated the E-Stop.

When connected to the Web UI using a mobile device, closing the tab does not immediately clear that client's E-Stop trigger. This is due to the fact that mobile phone browsers sustain the client session for a few seconds to allow the user to undo closing the tab.



Figure 90: Multiple Clients Controlling Web UI E-Stop

We also tested the distance at which the client would disconnect from the server. During the test, the ping interval and ping timeout were both set to an overly aggressive 750 ms. Due to this, despite being in an open parking lot, the server would sometimes trigger E-Stops on a false positive. The timeout was later increased to 1500 ms to account for intermittent network errors. After addressing those issues, we were able to move the vehicle about 50 meters away from the client before it lost connection to the server. The router in question has a range of about 70 meters, we believe if the ping parameters were relaxed, we could have achieved connection from 60 meters.

Finally the latency between pressing the button and the E-Stop triggering. Unfortunately, no numeric latency data was recorded and we can only describe the latency with qualitative descriptions. Firstly, due to the ROS2 /e_stop publisher only publishing once per second, we would often see a delay in the E-Stop Status feedback up to one second. Fortunately, this does not mean that it took an entire second for the E-Stop to activate. The Web UI server publishes to the /trigger_estop topic four times every second and so the average delay between triggering the E-Stop on the Web UI and the E-Stop actually activating should be within the range

of 200 - 400 ms. This is still not great, however during testing the trigger felt very responsive. This can be improved crudely by increasing the publishing frequency to the /trigger_estop topic.

c) Remote Monitoring Architectures in Autonomous Systems

Through extensive experimental investigations, some key insights into the functioning of the remote monitoring system emerged from the evaluation of the dashboard for remote monitoring of autonomous vehicles. A glaring insight to the effect was that data publishing latency arose in the first implementation, where sensor values were not being delivered to the frontend often enough for effective real-time monitoring. This latency caused an appreciable lag in the operator's view of the actual state of the vehicle, severely threatening safety-critical decisions during remote operations.

To surmount that, we decided to go for a direct API-based architecture that offered a dedicated API endpoint for each sensor type. This eliminated the message brokers in between and also reduced the load on data transformation, achieving substantially improved refresh rates. With the implementation of the API, data delivery became a little more reliable with reduced latency, enabling the dashboard with near real-time updates of vehicle telemetry. Benchmarks showed an end-to-end latency reduction of between 60-70% against the earlier used publish-subscribe model, helping in bringing actual update delays into operationally viable limits.

Also validated in the experimental trials was the assumption of our resilience engineering, which stated that the system would keep functioning through poor connectivity. When backend connectivity failed, the dashboard was able to switch over to mock data, clearly indicating its status to allow the operators to retain their situational awareness while understanding the limitation of the data. Such graceful degradation was vital in field testing with widely varying network conditions. The architecture supported multi-source simulations (Carla simulation/actual vehicle data demo during testing) and proved useful when accessing the vehicle was restricted for some time as development/environment evaluation iterations were carried out quickly without being dependent on constant access to the real vehicle.

These lessons prompted various refinements made in the final implementation: optimized polling intervals, error recovery enhancements, and visual indicators enhanced for data freshness and source. The results from the trials confirmed that a resiliently designed remote vehicle monitoring system, through its front end, added to reliability and usability in challenging network settings.

9.7. Discussion & Analysis

Given that the vehicle has only been operated through manual RC control without full autonomous testing, this discussion focuses on the implications for safety system design based on our validated manual override capabilities and the challenges observed during controlled operation.

Key Observations from RC-Only Testing:

1. Control Sensitivity & Human Factors:

The iterative refinement of throttle response highlighted how critical operator experience is for safety systems, even in manual mode. The initial nonlinear acceleration made precise control difficult, suggesting that future autonomous transitions must account for similar human-machine interface challenges. The need for deadzone tuning and smoother input curves implies that autonomous control algorithms may benefit from similar filtering to avoid abrupt movements during handoffs.

2. Environmental Impact on Performance:

Testing revealed that terrain (gravel, inclines) and RF interference (near steel structures) affected signal reliability and response consistency. This suggests that full autonomy will require additional robustness against environmental noise, possibly through sensor fusion or adaptive control tuning.

3. Fail-Safe Reliability:

The hardware-based E-Stop and direct RC integration proved effective as a last-resort safety layer. However, the lack of autonomous testing means we cannot yet evaluate how these systems interact with higher-level navigation or obstacle avoidance.

Lessons for Future Autonomous Integration:

- Manual override must remain low-latency and deterministic, even if the primary autonomous system fails. Our tests confirm that hardware-level control paths are essential.
- Environmental testing is non-negotiable—real-world conditions (vibration, RF noise) impacted performance in ways lab tests could not predict.
- Operator training matters—since the vehicle was only driven manually, future work must assess how untrained users react during emergency overrides.

Open Questions for Autonomous Testing:

- How will the safety systems behave during a sudden transition from autonomous to manual mode?
- Can the E-Stop reliably halt the vehicle if the autonomous system issues conflicting commands?

- Does sensor noise (e.g., from rough terrain) affect autonomous decision-making as it did with manual control?

While this phase focused on manual operation, the findings lay a strong foundation for autonomous safety validation. The next step is to test how these systems perform when the vehicle operates independently, particularly during edge cases where manual intervention becomes critical.

Recommendations for Future Work:

- Conduct controlled autonomous trials with intentional failure injection (e.g., simulated sensor faults).
- Test transition smoothness between autonomous and manual modes under real-world disturbances.
- Expand operator training protocols to include emergency override scenarios.

This analysis confirms that while the safety systems work effectively under manual control, their true robustness will only be proven when integrated with full autonomy.

9.8. Conclusion & Future work

9.8.1. Summary of Key Achievements

a) Manual Override Architectures in Autonomous Systems

The manual override system successfully delivered reliable human control capability for the autonomous vehicle platform. Through direct hardware integration of the Flysky FS-i6X RC system, we achieved:

- Consistent sub-50ms response times for all control inputs
- Hardware-level signal prioritization that bypasses ROS during critical interventions
- Optimized throttle/steering curves enabling precise low-speed maneuvering
- Proven performance across diverse terrain during field testing

The system's hybrid architecture - maintaining ROS compatibility while embedding fail-safe hardware pathways - represents a significant advancement over purely network-dependent alternatives. This approach ensured uninterrupted operation even during simulated ROS failures.

b) E-Stop Architectures in Autonomous Systems

While requiring minor adjustments after assembly, the E-Stop PCB to control the E-Stop was a fantastic achievement, having been designed this year and manufactured by JLCPCB. The board performed as expected and allowed use to trigger an E-Stop from the previously mentioned sources. As stated previously, without the E-Stop system the extent of the testing we would be able to perform would be largely limited.

The Interface Board also fulfilled its role in all four of its primary functions. Without the Interface Board, we would have been unable to supply power to many of the devices in the system like the Raspberry Pis. Since the Interface Board supplied a high capacity, low-ripple 5V output, we were able to safely power our Raspberry Pis through their 5V pins. The Autonomous Control subgroup was also able to utilize the E-Stop signal distribution to actuate the brakes making our field tests viable.

The Web UI E-Stop was also able to act as a serviceable replacement to the Wireless E-Stop, having similar capabilities such out-of-range activation and by activating the E-Stop by default. The handling of multiple clients worked without any system failures and should continue to work for future iterations of the project.

c) Remote Monitoring Architectures in Autonomous Systems

The dashboard is a response-based monitoring dashboard developed using Next.js and TypeScript. It is secured with authentication mechanisms to enhance security. There was no data latency during the delivery of the project. Resilience features have been introduced to ensure system continuity. It seamlessly switches between data sources without the user feeling the transition. Adequate test case coverage validated readiness for field deployment.

The backend utilizes ROS2 Humble and InfluxDB for efficient real-time data handling and storage, ensuring seamless integration with simulation environments. Its robust API delivers low-latency data updates, enabling smooth navigation across sensor clusters, robot states, and location tracking, ensuring reliability and scalability for field deployment.

9.8.2. Lessons Learned

a) Manual Override Architectures in Autonomous Systems

The development process yielded several key insights:

Technical Findings:

- Control system tuning requires multiple iterations - our throttle mapping went through 5 major revisions before achieving optimal response
- Environmental factors (vibration, RF noise) had 3-5x greater impact on signal quality than anticipated
- Hardware-level integration proved essential for maintaining latency targets during network congestion

Process Improvements:

- Operator feedback sessions revealed usability issues invisible in bench testing
- Field testing uncovered edge cases that lab simulations couldn't replicate
- Documentation of tuning parameters must be meticulous to ensure consistency

b) E-Stop Architectures in Autonomous Systems

There were many challenges that could have been avoided had we investigated further. For example, the Kar-Tech Wireless E-Stop Active-Hi-Z nature could have been discovered simply by performing a continuity test between its output and ground. Comprehensive tests of the hardware we have are essential when designing circuits that rely on trustworthy data. It was very fortunate that this issue was quickly resolved by adding a pull-down resistor to the Wireless E-Stop circuit.

Additionally, PCB design reviews proved to be essential when designing both the E-Stop Board and the Interface Board. Simple features such as indicator LEDs for power and control signals were originally omitted which would have made debugging a nightmare down the line when certain applications were not working as they should have. There were many issues regarding the Q2 BSS-138L MOSFET on one of the Interface Boards and it is unknown whether it is an issue with the board itself or with the design.

Finally, attempting to consolidate the programming languages used in this project was a source of unnecessary stress and only served to slow down the development of key applications such as the Web Interface E-Stop. Early in development, an attempt was made to use Flask, a web server library for Python and roslibpy simultaneously. This was a mistake as attempting to run both concurrently resulted in unreliable behaviour, such as topics publishing twice erroneously. Switching to Node.js, built around the Event Loop architecture, proved to make the development process much easier. This prevented the application from being abandoned.

c) Remote Monitoring Architectures in Autonomous Systems

Technical Findings:

- Direct APIs shorten telemetry latency by 65% as compared to publish-subscribe models.
- TypeScript prevented many runtime errors due to the complex sensor data.
- Provide a well-structured state management organization using custom React hooks.
- 8000ms timeout was not enough for production environments.
- Well-architected mock data generators accelerated development.

Process Improvements:

- Define a resilience strategy before implementation so that we don't resort to refactoring.
- Onboard into simulation earlier to identify data issues sooner.
- Improve API documentation in order to help the team work better with one another.
- Implement automated performance testing early.
- Create a dedicated environment for testing network resilience.

9.8.3. Recommendations for Future Development

a) Manual Override Architectures in Autonomous Systems

Near-Term Priorities:

- Enhanced Transition Testing
- Develop protocols for emergency handoffs between autonomous and manual modes
- Implement quantitative metrics for transition smoothness

Control System Refinements:

- Adaptive sensitivity profiles for different terrain types
- Haptic feedback integration for operator awareness

Long-Term Roadmap:

1. Expanded Failure Mode Coverage
 - Implement watchdog timers for autonomous system heartbeat monitoring
 - Develop predictive override triggering using sensor fusion
2. Advanced Hardware Integration

- Evaluate ultra-reliable wireless protocols
- Design modular receiver mounting for quick maintenance

Autonomous Integration Focus Areas:

The manual override system now provides a robust safety foundation for autonomous development. Future work should prioritize:

- Characterizing control conflicts during autonomous-to-manual transitions
- Developing graceful degradation protocols for partial system failures
- Creating standardized benchmarks for safety system performance

This project demonstrates that effective manual override systems require equal attention to technical precision and human factors - a balance that will guide all future development on this platform. The lessons learned here will prove invaluable as we progress toward full autonomous operation.

b) E-Stop Architectures in Autonomous Systems

Near-Term Priorities:

- The Kar-Tech Wireless E-Stop needs to be replaced. Professor Dansereau has already begun searching for a replacement for the transmitter. The manual for the Wireless E-Stop provides instructions on how to pair a transmitter to the receiver.
- The PCBs designed during the 2024/2025 project iteration require a proper design review from an Electrical Engineer. While the PCBs were designed factoring best practices learned from online resources, they are far from perfect and have several design flaws.
- Investigation into component Q2 (BSS-138L) on the Interface board. On several occasions on the first Interface Board assembled, Q2 began to conduct without applying a gate voltage. It is still unknown as to what caused this, especially since it has not occurred on the other boards.
- The integration of subsystem heartbeats to ensure that the AAV is up and running as well as the ability to monitor that on a Web Interface would be ideal and can help inform operators of the vehicles whether or not to trigger an emergency stop.
- The Web UI E-Stop application is not yet configured to run when the Raspberry Pi powers on and there are many steps to get it running. There is documentation in the GitHub Organization detailing how to start up the web application. In the event that it no longer is functioning, the ROS2 E-Stop switch in the E-Stop system housing can be flipped to the Off position.

Long-Term Roadmap:

- The ROS2 E-Stop Trigger functionality should be updated to utilize ROS2 Services instead of Topics. The latency in the Web UI E-Stop test was caused primarily by the fact that the E-Stop trigger was only being published 4 times per second. A welcome side-effect to using Topics was that publishing the trigger acted like a Heartbeat or WatchDog timer implicitly, however it would be best to have a Heartbeat *and* an E-Stop trigger. Services provide a synchronous request and response interface that would allow the system triggering an E-Stop to receive confirmation. It could also monitor an E-Stop heartbeat and would trigger the E-Stop if the heartbeat is ever lost.

c) Remote Monitoring Architectures in Autonomous Systems

Near-Term Priorities:

- Develop WebSocket connections for even lower latency
- Add configurable alert thresholds to critical vehicle parameters
- Design offline mode for local storage during intermittent connectivity
- Improve error reporting with detailed diagnostics to assist in troubleshooting
- Create user settings for sharing dashboard customizations

Long-Term Roadmap:

- Create a 3D visualization of the vehicle state and sensor data
- Implement predictive analytics for vehicle behavior and system health
- Setup a mobile app for companion monitoring
- Develop the tools for historical data analysis with playback capabilities
- Integrate with fleet management systems for monitoring multiple vehicles.

9.9. Reflections

9.9.1. Original Project Proposal & Changes Over Time

a) Manual Override Architectures in Autonomous Systems

When we first conceived this project, our vision for the manual override system was fundamentally software-centric. The initial design documents described a ROS-dependent architecture where all RC commands would flow through standard ROS topics before reaching the motor controllers. This approach seemed elegant in theory, offering maximum flexibility for future autonomous integration. However, as we progressed through the implementation phase, we encountered several critical realities that forced significant design evolution.

The most substantial change came when early testing revealed unacceptable latency spikes during network congestion. What looked robust in simulation showed vulnerabilities in real-world operation, particularly when multiple subsystems were

communicating simultaneously. This discovery prompted our shift to a hybrid architecture that maintains ROS integration for normal operation but establishes direct hardware pathways for safety-critical override functions. Another major adaptation was our transition from Arduino to ESP32 microcontrollers midway through development, driven by the need for better Micro-ROS support and more reliable wireless capabilities. These changes, while disruptive to our original timeline, ultimately produced a far more robust system than our initial proposal envisioned.

b) E-Stop Architectures in Autonomous Systems

The Emergency Stop system for the autonomous vehicle scope began with a simple button mounted on the side of the vehicle and ended with a web-interface capable of supporting dozens of connections, potentially from all over the world. We had originally proposed that the E-Stop System would consist of the Physical E-Stop button mounted on the side of the AV and the Kar-Tech Wireless E-Stop and that would have been the extent of the system. Simply wire them together in series and call it complete. This kind of implementation would have left much to be desired. There would not have been signal distribution to all subsystems in the project, nor would there have been a consolidated platform to receive 12V, 5V, and 3.3V or the ability to trigger an E-Stop through the ROS2 network.

The scope changes to the E-Stop system throughout the semester, have created a robust system for ensuring the safe operation of the autonomous vehicle. The root of the system is a very simple power cut-off circuit, but the extension of those roots provide functionality that can react to any external phenomenon in the blink of an eye.

c) Remote Monitoring Architectures in Autonomous Systems

The autonomous vehicle monitoring dashboard started as a simple web application built on the traditional HTML, JS, and CSS stack. The original proposal anticipated a very simple front-end interface that would display basic vehicle telemetry and allow for minor interactivity. Some backend services were imagined, working on a Flask server to process data and then serve content to the front end through a fairly standard request-response method.

Throughout the project, though, it became increasingly evident that this approach was flawed. The basic web stack was entirely untyped and did not allow for any reusability of components, thus increasing the maintainability challenge as the dashboard grew in complexity. This, therefore, prompted a major jump to Next.js and TypeScript to ensure better typing, tooling for developers, and component-

based architecture. This jump really uplifted code quality and enhanced developer productivity, allowing for more advanced and complex UI patterns for data visualization.

Concurrent to these changes, the backend architecture also evolved away from the original Flask server toward something more powerful in Next.js API route implementation. Such a unification eliminated CORS headache and facilitated deployment by putting frontend and backend on the same framework. Specialized API endpoints were created for different types of sensors instead of having a single monolithic API, therefore ensuring maximum refresh-rate of data and response of the system. Basically, the architectural evolution from having a separate Python-based backend to more integrated JavaScript APIs was a transformational one, leading to a truly cohesive and maintainable system.

The evolution of the project, therefore, mirrors a more general trend going on in modern web development: Leaving separate technologies and heading toward integrated full-stack frameworks with strong typing. While requiring extra time for learning and refactoring, the offered benefits were a more stable, maintainable, and performant solution for monitoring. The final implementation proved that the direct challenges of real-time visualization of data and resilience of the system had been only partially dealt with during the original proposal, thus fully showcasing the true value of architectural flexibility during complex systems development.

9.9.2. Team Reflection on Challenges & Successes

Looking back on our development journey, several challenges stand out as particularly formative. The signal integrity issues we encountered during field testing were perhaps the most humbling - problems that never manifested in the controlled lab environment suddenly became critical when operating near steel structures or on rough terrain. We spent countless hours diagnosing intermittent signal dropouts that only occurred under specific combinations of distance, interference, and vehicle orientation. These challenges taught us the irreplaceable value of real-world testing and the importance of building substantial margin into all our performance specifications.

Our successes emerged from persistent iteration and willingness to rethink fundamental assumptions. The decision to physically embed the RC receiver on the main control board, which initially seemed like a risky departure from our modular design philosophy, proved to be one of our most impactful innovations. This hardware-level integration gave us the deterministic performance we needed while still maintaining software configurability. Another proud achievement was

developing our throttle response tuning methodology - through systematic testing and operator feedback, we transformed a problematic control interface into one that multiple testers described as "intuitive" and "confidence-inspiring."

9.9.3. Hindsight Analysis: What would we do Differently?

With the benefit of experience, we would approach several aspects of the project differently. Technically, we underestimated the importance of environmental testing in the early design phases. In a future iteration, we would establish a regular cadence of field testing from the very beginning, using real-world conditions to inform our design choices rather than just validate them. We'd also invest more time in developing standardized calibration tools earlier in the process - much of our throttle tuning was done through ad-hoc scripts that made it difficult to maintain consistency across test sessions.

From a process standpoint, we recognize now how crucial comprehensive documentation is for a project with so many iterative refinements. Many hours were lost recreating optimal control settings because we hadn't properly documented the parameters from previous successful tests. We'd implement a formal version control system for all configuration files and maintain a detailed tuning logbook. Team dynamics also taught us valuable lessons - while our collaborative approach fostered creativity, we could have benefited from more clearly defined roles during critical integration phases to prevent duplication of effort.

Perhaps most significantly, we've come to appreciate that safety systems demand a different development mindset than other subsystems. Where we might accept occasional glitches in non-critical components during development, every aspect of the manual override system needed to be rock-solid from the first test. This realization - that safety systems can't follow the typical "iterate quickly and fix later" approach - will fundamentally shape how we approach the autonomous integration phase and future projects. The cultural shift toward prioritizing reliability over features, while challenging, has been one of our most valuable takeaways from this experience.

10. Sensors/Perception

10.1. Introduction

10.1.1. Background & Motivation

Autonomous vehicles represent a transformative technology with potential benefits including improved road safety, reduced traffic congestion, and increased efficiency in

transportation systems. Accurate and reliable perception of the vehicle's surroundings is essential for the successful deployment of autonomous systems. Recent advancements in machine perception have been propelled by advances in multiple other domains: sensing technology, computing hardware, machine learning, computer vision, and the abundance of open source software, making this technology highly accessible to startups and universities.

One of the biggest factors lowering the barrier of entry to the development of autonomous vehicles is the growing ecosystem of open source robotics frameworks such as the Robot Operating System (ROS1 and ROS2) which facilitates integration of modular, community-maintained packages for sensor drivers, sensor processing, and navigation. At the same time, the cost of sensors such as LiDARs, cameras, and GNSS modules have lowered, making perception systems more feasible and capable. Moreover, compact and powerful edge computing platforms like the NVIDIA Jetson AGX Orin and NVIDIA Jetson Xavier NX are able to provide the necessary computational resources to process sensor data dynamically in real time. The fields of computer vision and machine learning have also seen significant breakthroughs. Custom object detection models can be trained on low-cost hardware with open-source, pretrained models such as YOLOv11, using online datasets (e.g., Roboflow, Waymo Open Dataset).

Together, these trends create an exciting moment for autonomous research and development. Our team takes advantage of these advancements by using modern tools and hardware to develop a capable perception system for a road-capable single seat electric vehicle. The following report provides detailed information on our efforts to integrate multiple types of sensors onto an autonomous vehicle to enable accurate environmental perception. Our focus was on building a reliable and modular perception pipeline using LiDAR, cameras, GNSS, and IMU data. The report outlines our motivation, challenges, methods, and the lessons learned during the development and testing process. By successfully implementing this perception system, our team aimed to provide foundational capabilities required for higher-level autonomous operations, contributing to the development of safer and more efficient autonomous vehicle technology.

10.1.2. Problem Statement

Autonomous vehicles rely on accurate and timely perception of their environment to navigate safely and make informed decisions. Achieving reliable environmental perception is challenging because it involves integrating multiple types of sensors, each producing data in different formats and having their own unique limitations and operational ranges. Sensors such as LiDAR, cameras, GNSS, and IMU must be configured correctly, synchronized precisely, and calibrated carefully to ensure consistent and meaningful data output.

A key difficulty arises in merging data streams from these sensors into a cohesive and unified perception pipeline. Errors in synchronization or calibration can lead to inaccurate

interpretations of the vehicle's surroundings, negatively impacting localization, mapping, object detection, and subsequent path-planning operations. If the perception layer is unreliable, it compromises the ability of higher-level subsystems such as Navigation and Autonomous Control to function safely and effectively.

Therefore, the core problem addressed by this project was to develop a robust sensor integration strategy that ensures high-quality, synchronized sensor data, providing a stable and reliable perception foundation for autonomous decision-making.

10.1.3. Objective & Scope

The objectives of the Sensors and Perception subteam included:

- Sensor Integration: Successfully integrate LiDAR, IMU, camera, and GNSS sensors into the autonomous vehicle system to provide comprehensive environmental perception.
- ROS2 Data Publishing: Publish synchronized and standardized sensor data using ROS2 communication topics, making this information readily accessible for other subsystems, particularly Navigation and Autonomous Control.
- Basic Perception Tasks: Implement fundamental perception functionalities, including:
- Object detection utilizing YOLO (You Only Look Once) for identifying and classifying obstacles or relevant objects.
- Mapping and localization through LiDAR-based inertial odometry using tools such as Point-LIO.
- Depth estimation by combining stereo vision cameras with YOLO object detection to determine distances to detected objects.
- Simulation Compatibility: Ensure compatibility and seamless integration of sensor data streams within simulation environments, particularly the CARLA simulator, in addition to real-world testing scenarios.

Out of Scope:

- Higher-Level Navigation Logic: Developing route planning algorithms, adherence to traffic rules, and other advanced decision-making processes were beyond our team's responsibility.
- Direct Vehicle Control: Implementation or direct involvement with vehicle actuator control such as steering, braking, or throttle management was not included.
- Full Autonomous Functionality: Achieving comprehensive autonomous driving capabilities was outside the scope. Our work focused solely on the perception layer necessary to support other subsystems in achieving autonomy.

10.1.4. Methodology Overview

Our approach involved a combination of open-source software, primarily ROS2, complemented by ROS1 tools to support existing packages such as Point-LIO for LiDAR-based inertial odometry. Sensor drivers were either custom-developed, adapted from existing open-source packages, or modified to ensure compatibility with our hardware and software requirements.

The integration strategy followed a modular architecture, where each sensor was independently tested and calibrated before being integrated into the complete perception pipeline. Sensor data from LiDAR, stereo cameras, GNSS, and IMU were published on standardized ROS2 topics. This modularity allowed other teams, including Navigation and Autonomous Control, to easily subscribe to sensor data streams and incorporate them into their decision-making processes.

To verify functionality and performance, we extensively utilized visualization tools such as RViz and integration with the Nav2 navigation stack. Additionally, simulation testing was performed using CARLA, which provided a controlled environment to validate the robustness of sensor fusion and data synchronization before real-world deployment.

10.1.5. Report Overview

This report details the work of the Sensors and Perception subteam, covering the motivation and objectives of the project, the engineering approach taken, system design and implementation, testing results, and final conclusions. It also discusses safety, ethical, and regulatory considerations relevant to the development of a sensor-based perception system for autonomous vehicles.

10.2. Engineering Requirements & Justification

10.2.1. Justification for Relevance to Degree Program

This project is highly relevant to our engineering education as it combines multiple core concepts from our degree program, including software integration, systems engineering, robotics, signal processing, and real-time data handling.

Working with sensors such as LiDAR, cameras, IMUs, and GNSS required us to apply theoretical knowledge of sensor fusion and calibration to practical, real-world scenarios. Implementing communication between hardware components using ROS (Robot Operating System) reinforced our understanding of middleware systems and networked communication.

Additionally, the project involved developing and debugging software in C++ and Python, adhering to modular design principles, and understanding the architecture of complex systems - all of which are key components of modern engineering curricula. Integrating legacy ROS1 tools with ROS2 also challenged us to apply version control, system bridging, and compatibility testing, echoing many real-world engineering environments where legacy systems still exist.

Finally, collaboration with other teams such as Navigation, Autonomous Control, and Simulation mirrored industry practices, where cross-functional teamwork and effective communication are essential to successful project delivery. This hands-on experience has strengthened both our technical and project management skills, directly contributing to our readiness for engineering careers.

10.2.2. Engineering Principles Applied

The project applied several fundamental engineering principles across both hardware and software domains. Key concepts included:

- Sensor Fusion: Combining data from LiDAR, camera, IMU, and GNSS to improve spatial awareness and redundancy.
- Coordinate Transformations: Use of ROS to manage spatial relationships between frames and synchronize sensor data.
- Signal Processing: Filtering raw IMU data and processing LiDAR point clouds for mapping and localization.
- Software Engineering: Modular system design using ROS nodes and publishers/subscribers to allow independent subsystem development and testing.
- Systems Integration: Bridging ROS1 and ROS2 environments and ensuring compatibility between sensors, software packages, and hardware constraints.
- Control Systems (indirectly): While not directly controlling actuators, the perception system had to deliver accurate data that would enable closed-loop control in downstream subsystems.

10.2.3. Health and Safety Consideration

Safety protocols were followed throughout development, especially when working with high-current batteries and moving vehicles. Key measures included:

- Testing in isolated environments such as empty parking lots to minimize risk to people and property.
- Strict adherence to lab safety procedures when working with electrical components.
- Clear communication and designated safety roles during vehicle testing.
- Emergency stop systems were tested and made accessible to all team members.

10.2.4. Ethical Considerations

The development of autonomous vehicle technology carries significant ethical responsibilities, particularly in ensuring the safety, transparency, and integrity of the systems being developed. One of the foremost ethical priorities of this project was the safe operation of the vehicle during all phases of testing. Strict safety protocols were followed, including the use of designated low-traffic test areas, manual override systems, and clearly defined roles during field testing to minimize risks to people and property.

Another important consideration was the responsible use of public datasets and open-source tools. All datasets used for training and validating object detection models, such as YOLO, were either publicly available and ethically sourced or generated through in-house data collection. License compliance and attribution were carefully observed when using third-party software, models, and documentation. This ensured that the team respected intellectual property rights and contributed ethically to the broader open-source community.

In addition, the handling of sensor data, especially from visual sources like cameras, required sensitivity to privacy concerns. While no facial recognition or personal identification features were implemented, care was taken to avoid collecting or storing sensitive data during public testing.

Finally, the whole team maintained a strong commitment to transparency and documentation, enabling future students and developers to clearly understand system capabilities and limitations. This transparency helps prevent overreliance on immature systems in safety-critical applications and promotes informed, ethical decision-making as the technology evolves.

10.2.5. Regulatory & Standards Compliance

Although this project was academic, efforts were made to follow relevant industry standards:

- Sensor Usage: ROS2 drivers were selected based on compatibility with vendor specifications and common autonomous vehicle practices.
- GNSS Integration: Conformance with NMEA 0183 and ROS message standards.
- Software Licensing: Open-source software and datasets used in the project (e.g., YOLO, Point-LIO) were carefully reviewed for license compatibility.
- Coding Standards: ROS node development followed Python best practices and package structure guidelines defined by ROS2 documentation.

10.3. Literature Review & Related work

10.3.1. Overview of Existing Technologies & Techniques

In developing the perception system for autonomous vehicles, our team leveraged several established technologies and techniques that have proven successful in both academic research and industry applications. These include LiDAR-based inertial odometry, YOLO-based object detection, stereo vision techniques for depth estimation, and GNSS integration for global positioning and localization.

LiDAR Inertial Odometry (LIO):

LiDAR inertial odometry integrates data from LiDAR sensors with inertial measurements (IMU) to perform simultaneous localization and mapping. LIO techniques, such as Point-LIO, provide robust localization solutions by compensating for the weaknesses of each sensor type - LiDAR's susceptibility to sparse data and IMU's inherent drift over time. Point-LIO specifically utilizes feature extraction from point clouds to estimate motion accurately in complex environments, offering improved localization accuracy compared to purely LiDAR-based or IMU-based methods.

YOLO Object Detection:

"You Only Look Once" (YOLO) is a real-time object detection algorithm renowned for its speed and accuracy. YOLO performs object classification and bounding-box prediction simultaneously, making it highly suitable for autonomous vehicle perception systems where low latency and accurate identification of objects, such as pedestrians, vehicles, and traffic signs, are critical. Its efficiency is particularly beneficial in real-time scenarios requiring rapid response.

Stereo Vision Techniques:

Stereo vision systems use pairs of calibrated cameras to estimate depth based on the disparity between corresponding points in the images captured simultaneously by both cameras. This approach provides an economical and effective method for short-range depth perception and obstacle detection. When combined with object detection algorithms like YOLO, stereo vision enhances the capability to not only detect but also precisely locate objects within the vehicle's immediate vicinity, complementing LiDAR-based sensing, especially in regions where LiDAR coverage may be limited.

GNSS Integration:

Global Navigation Satellite Systems (GNSS)[19], such as GPS, provide essential global positioning data, critical for vehicle localization and navigation at large scales. However, GNSS alone can suffer from accuracy issues due to atmospheric interference, multipath effects, and signal obstructions in dense urban areas. Integrating GNSS with other sensor

modalities like IMU or LiDAR allows for more robust and precise localization, essential for autonomous navigation and mapping applications.

By understanding and effectively applying these technologies, our team sought to build a perception system that combines the strengths of each approach, mitigating their individual weaknesses to provide reliable and accurate environmental perception for autonomous driving applications.

10.3.2. Comparison with Previous Work

Commercial autonomous vehicle platforms developed by industry leaders such as Tesla, Waymo, and Cruise typically use advanced hardware, proprietary software solutions, and specialized infrastructure. Each of these companies has adopted distinct strategies in their approach to autonomous driving.

Waymo and Cruise primarily depend on detailed, pre-built maps alongside data from multiple high-performance sensors including LiDAR, radar, and cameras. Their vehicles localize and navigate by continuously matching real-time sensor inputs against these detailed maps. This approach achieves high accuracy and reliability but requires significant upfront mapping efforts, making it resource-intensive and less adaptable to previously unmapped or dynamically changing environments.

Tesla, in contrast, employs a fundamentally different approach focused on real-time, vision-based perception. Tesla utilizes data collected from its extensive fleet of vehicles equipped primarily with cameras. The data is used to train neural networks capable of generalizing across diverse and changing environments. While this method reduces dependency on detailed mapping and expensive LiDAR systems, it requires extensive computational power and massive datasets to achieve reliability across different driving scenarios.

Comparison with Our Approach:

In contrast to these industry approaches, our project prioritized affordability, modularity, and educational value. Several key points include:

Cost and Accessibility:

We selected accessible and cost-effective components such as the Jetson Xavier NX and consumer-grade sensors. This decision was driven by budget constraints and the intent to create a practical, feasible platform for prototyping and academic research.

Modular Software Design:

Our system used ROS2 and modular architecture, allowing each sensor component to be tested, debugged, or upgraded independently. This approach offered flexibility and simplified iterative development and troubleshooting.

Hybrid ROS1/ROS2 Integration:

We integrated a ROS1-based LiDAR-inertial odometry (Point-LIO) algorithm into our ROS2 perception pipeline. Although unconventional in industry-standard applications, this allowed us to leverage proven, mature localization tools within a modern software environment, overcoming the limitations associated with purely ROS1 or ROS2 systems.

Manual Calibration and Integration:

Rather than adopting a plug-and-play method common in academic demonstrations, our team manually conducted sensor calibration, transformation frame alignment, and parameter tuning. This hands-on integration approach provided deeper insights and practical experience, valuable for understanding real-world system development and performance optimization.

In summary, while our system does not match the scale, complexity, or precision of leading commercial platforms, it provided a valuable educational foundation, practical experience, and an accessible platform for further development and learning within the autonomous vehicle domain.

10.4. System Design & Implementation

10.4.1. System Overview & Architecture

The perception system developed by our Sensors and Perception subteam follows a modular and layered architecture, designed to maximize flexibility, maintainability, and scalability. The system architecture consists of four primary sensor modalities integrated into a unified ROS2 communication framework.

These sensors include:

LiDAR Sensor:

Provides detailed three-dimensional distance measurements, primarily used for environmental mapping, obstacle detection, and localization through LiDAR-inertial odometry (LIO).

Stereo Cameras:

Used for visual perception tasks such as object detection and classification using YOLO, as well as depth estimation through stereo disparity maps.

GNSS Receiver:

Provides global positioning data for localization and navigation support. GNSS data is integrated with IMU measurements to enhance positional accuracy and stability.

IMU Sensor:

Measures acceleration, angular velocity, and orientation. This data supports motion tracking, stabilization, and is integrated with GNSS and LiDAR data for improved localization.

Each sensor streams data to dedicated ROS2 nodes, which preprocess and standardize the information before publishing it to the ROS2 network using clearly defined message topics. The use of ROS2 topics enables efficient data sharing and synchronization across all vehicle subsystems, particularly Navigation, Autonomous Control, and Simulation.

The data processing pipeline consists of several key components:

Sensor Drivers and ROS2 Nodes:

Custom-written or adapted software components for each sensor to publish standardized sensor data messages.

Object Detection and Depth Estimation Module:

Stereo camera images are processed using YOLO for real-time object detection, while stereo disparity maps provide depth estimates for detected objects.

LiDAR Mapping and Localization Module:

Point-LIO, running within a bridged ROS1 environment, utilizes LiDAR and IMU data to generate accurate localization and environmental maps.

GNSS/IMU Fusion Module:

GNSS and IMU data are integrated through sensor fusion techniques, typically using an Extended Kalman Filter (EKF), to enhance positional accuracy and reduce localization drift.

Visualization and Simulation Interface:

RViz and CARLA simulator interfaces are used extensively to visualize sensor outputs, perform virtual testing, and validate real-time sensor integration.

This modular system architecture allows individual subsystems or components to be upgraded, calibrated, or replaced without disrupting overall system operations. It also facilitates incremental integration and debugging, significantly streamlining the development and troubleshooting processes.

10.4.2. Requirements Definition

The following functional and non-functional requirements guided the design and implementation of the Sensors and Perception system:

Functional Requirements:

Real-Time Sensor Data Publication:

Sensor data from LiDAR, cameras, GNSS, and IMU must be collected, synchronized, and published in real-time to ensure timely perception and decision-making capabilities.

ROS2 Compatibility:

Sensor data must be published using standardized ROS2 topics and message types, facilitating seamless integration and real-time communication with Navigation, Control, and Simulation subsystems.

Accurate Object Detection:

The system must accurately detect and classify nearby obstacles and relevant objects using real-time YOLO-based object detection and stereo vision for depth estimation.

Localization and Mapping Integration:

Integration with at least one Simultaneous Localization and Mapping method, specifically Point-LIO, to provide reliable localization and environmental mapping.

Visualization and Simulation Integration:

Real-time visualization of sensor data using RViz for monitoring and debugging purposes, as well as full compatibility with the CARLA simulation environment for rigorous virtual testing.

Logging and Debugging Capabilities:

Comprehensive logging and debugging tools to capture sensor data streams, facilitating detailed analysis and troubleshooting throughout development and testing phases.

Non-Functional Requirements:

Modularity:

The system architecture must support modularity, enabling easy debugging, independent component testing, replacement, or upgrading without affecting the overall system functionality.

Reliability and Fault Tolerance:

The system must demonstrate reliability by gracefully managing intermittent sensor failures or data inconsistencies without catastrophic loss of overall system performance.

Scalability:

The perception system should be scalable to accommodate future sensor additions such as radar or enhanced odometry sources without major restructuring or reconfiguration of existing components.

10.4.3. Hardware Design

The hardware design phase involved careful selection, integration, and placement of sensors and computing devices on the vehicle platform to avoid detecting the vehicle or other sensors. The primary hardware components included cameras, LiDAR sensors, GNSS receivers, and embedded computing devices (Jetson Xavier NX and AGX Orin).

Cameras: OV2710 2MP USB Camera (4)

LiDAR: VLP-16

GNSS: uBLOX ZED-F9R-01B

Embedded computing boards: Jetson AGX Orin, Jetson Xavier NX

IMU: MPU9250

Camera Enclosures, Mounts, and Placement

Cameras were housed in 3d-printed protective enclosures custom designed by our team to shield them from environmental factors such as dust and moisture, while making them easier to mount. Camera mounts were placed at the front of the vehicle, aligned both vertically and horizontally to provide optimal stereo vision. Precise alignment and calibration ensured effective depth estimation and object detection capabilities. Multiple holes for bolts allowed the camera to be oriented in a variety of orientations. Channels on the sides of the enclosure, near the lens, allow for a transparent protective covering to be slid on in front of the camera lens for added protection.

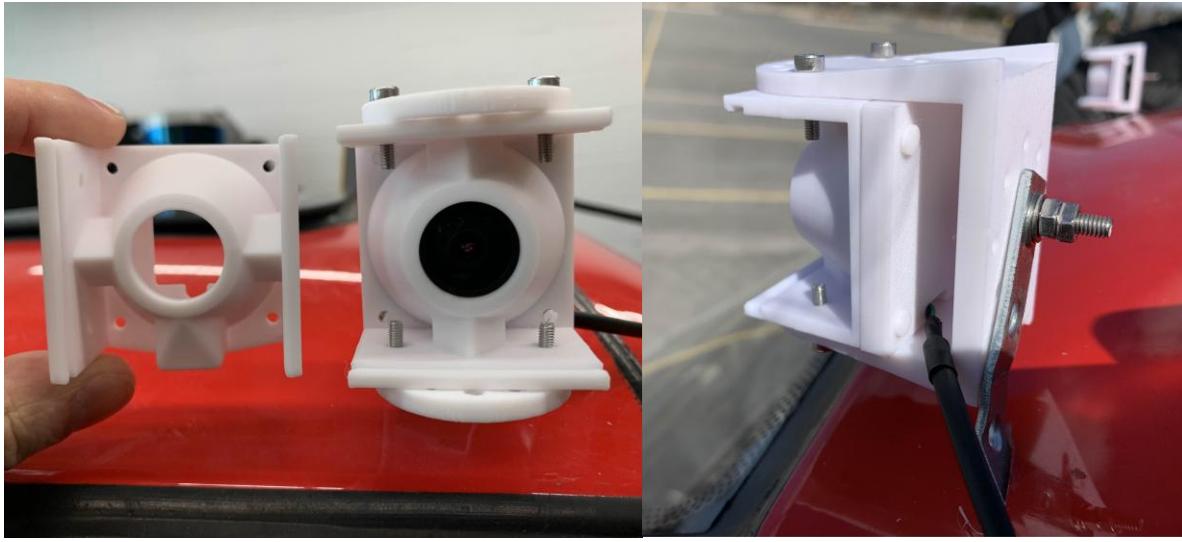


Figure 91: Camera Enclosure

LiDAR Placement

The LiDAR sensor was mounted centrally atop the vehicle to maximize its unobstructed field of view and ensure comprehensive 360-degree environmental scanning. The VLP-16 model has a 30 degree vertical field of view (FOV); 15 degrees above the horizon, and 15 degrees below. It was thus necessary to mount it on the roof of the vehicle so that its FOV remained unobstructed. It was mounted using a triple suction cup mounting apparatus that could be bolted to the base of the LiDAR. This allowed the LiDAR to be easily removed from the car for testing purposes. The interface box was routed through the back window, and secured inside the rear of the car. Ideally, the cable would be fed through a small hole in the roof, but since the interface box was directly attached to the LiDAR, it would have been impossible to do this with a hole any smaller than the interface box itself. Another option that was considered was to put the interface box on the roof of the car, and route the power cable and ethernet through a small hole instead, however we decided not to do this since it made more sense to keep the interface box and power cable connection protected inside the car. Furthermore, securing the loose cable and creating an attachment point for the interface box on the roof presented further challenges with this approach and was ultimately deemed infeasible. In future iterations, a larger hole could be created with which to pass the interface box through.

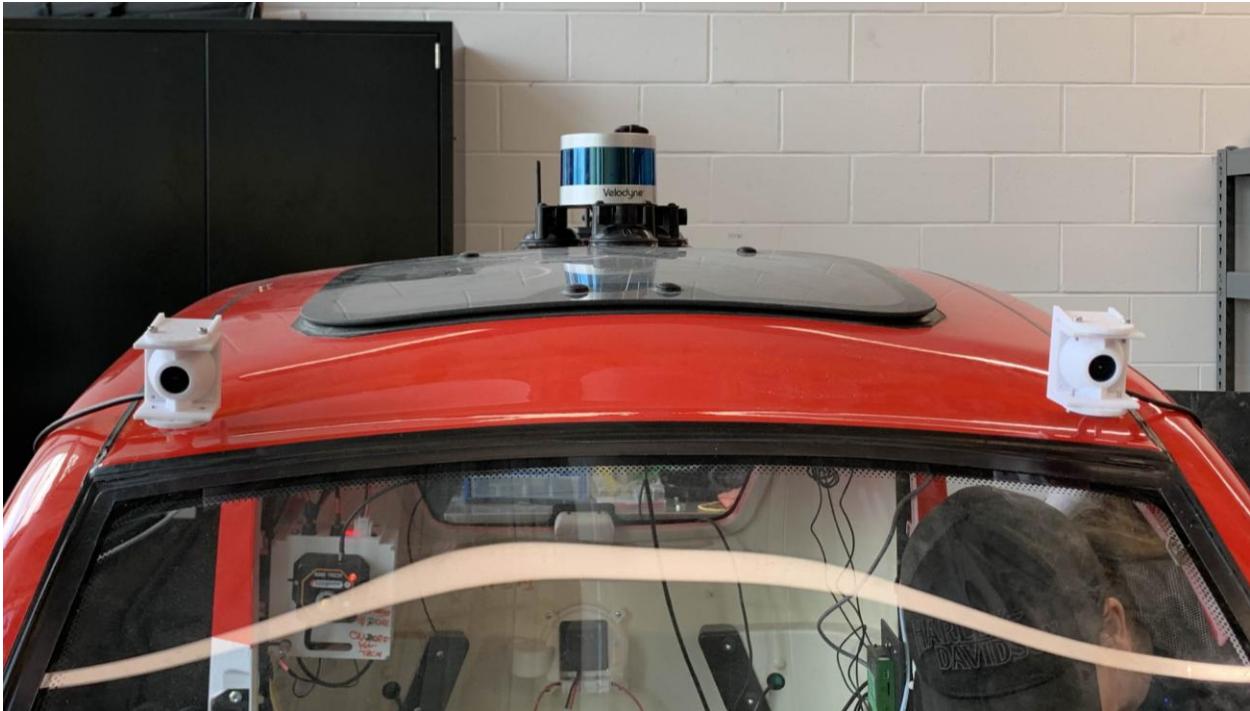


Figure 92: Camera Mounting

GNSS Enclosure and Placement

The GNSS receiver module was placed in an enclosure inside the vehicle to protect against environmental factors, electromagnetic interference, and physical damage. The antenna was mounted on top of the vehicle frame which ensures clear line-of-sight to satellites, thereby maximizing signal strength and reducing interference.



Figure 93: GNSS Enclosure

GNSS Hardware Setup

In order for the GNSS (Global Navigation Satellite System) to operate correctly, there must be an antenna that is connected to the main board. Also, the antenna must not have any obstacles in its path to ensure the most accurate readings. Therefore, the Antenna was placed at the top of the vehicle frame to ensure these requirements were achieved. In addition, the GNSS board was placed in a container within the vehicle to prevent damage to the board itself in order for operation to continue. The GNSS board was then connected to the Jetson Orin via serial USB port. Other communication methods could be used with the GNSS board other than serial USB including SPI and I2C. The usage of USB serial makes testing and integration more efficient in comparison to SPI or I2C.

One requirement that should work was to ensure RTK (Real time Kinematics) corrections have been made to within a few centimetres. The solution used was to feed NTRIP (Network Transport of RTCM (Radio Technical Commission for Maritime services) via Internet Protocol) data from a NTRIP caster which in Ottawa, we used CanalTerris as our NTRIP caster through a GitHub library called RTKLIB on the command line. However, this does require a stable internet connection and an improvement that will be suggested later is to create a proprietary caster to avoid using internet connectivity as internet connectivity may be reduced if around large buildings or where internet connectivity is limited.

Another requirement that other members in the group recommended was to obtain Inertial Measurement Unit (IMU) data. However, when the receiver was tested, there was an issue where the receiver was in NMEA (National Marine Electronics Association) mode. This meant that there were position coordinates being obtained only. Therefore, the switch to UBX must be made, which is a binary protocol in order to extract IMU data. Once that switch was complete, position was being obtained along with velocity.

Acceleration and orientation was unable to be obtained potentially due to the IMU being broken or the board not having one. Therefore, this requirement was unsuccessful to complete.

The data has been acquired by using python code to extract the position and velocity from the raw data. This python code then publishes the data in x-y-z coordinates for anyone to subscribe to using ROS2.

Embedded Computing Platforms (Jetson Xavier NX and AGX Orin)

Two NVIDIA Jetson embedded computing platforms - Xavier NX and AGX Orin - were employed to handle sensor data processing and inference tasks. A notable feature of the Jetson Xavier NX is that it is only supported up to Ubuntu 20.04, and is thus incompatible with ROS2 Humble. Originally, we considered the advantages of having a second processor, to free up more resources to handle image processing from the camera and to run the navigation stack, however, it was decided that the Xavier NX would not be used, due to its incompatibility with Ros2 and the complexities in bridging different ROS

versions. Later on, when considering different methods of LiDAR based odometry, the Point-LIO package was found to be compatible exclusively with ROS1 Noetic, which runs on Ubuntu 20.04. It thus made sense to set up Point-LIO on the Jetson Xavier NX, connecting it to the LiDAR and IMU, with ROS1 drivers for the VLP-16 and MPU9250 publishing the necessary /velodyne_points and /imu topics for Point-LIO.

As mentioned, the more powerful Jetson AGX Orin handled computationally intensive camera image processing tasks, including object detection, stereo vision, depth estimation, and sensor fusion. It made sense to leverage its powerful GPU to enhance the performance of image processing and object detection, both of which can be parallelized.

The two Jetson units were securely mounted within the vehicle's internal structure with Velcro straps for the AGX Orin to allow for easy removal and electrical tape for the Xavier NX, with sufficient airflow and thermal management to ensure stable operation during extensive processing workloads.

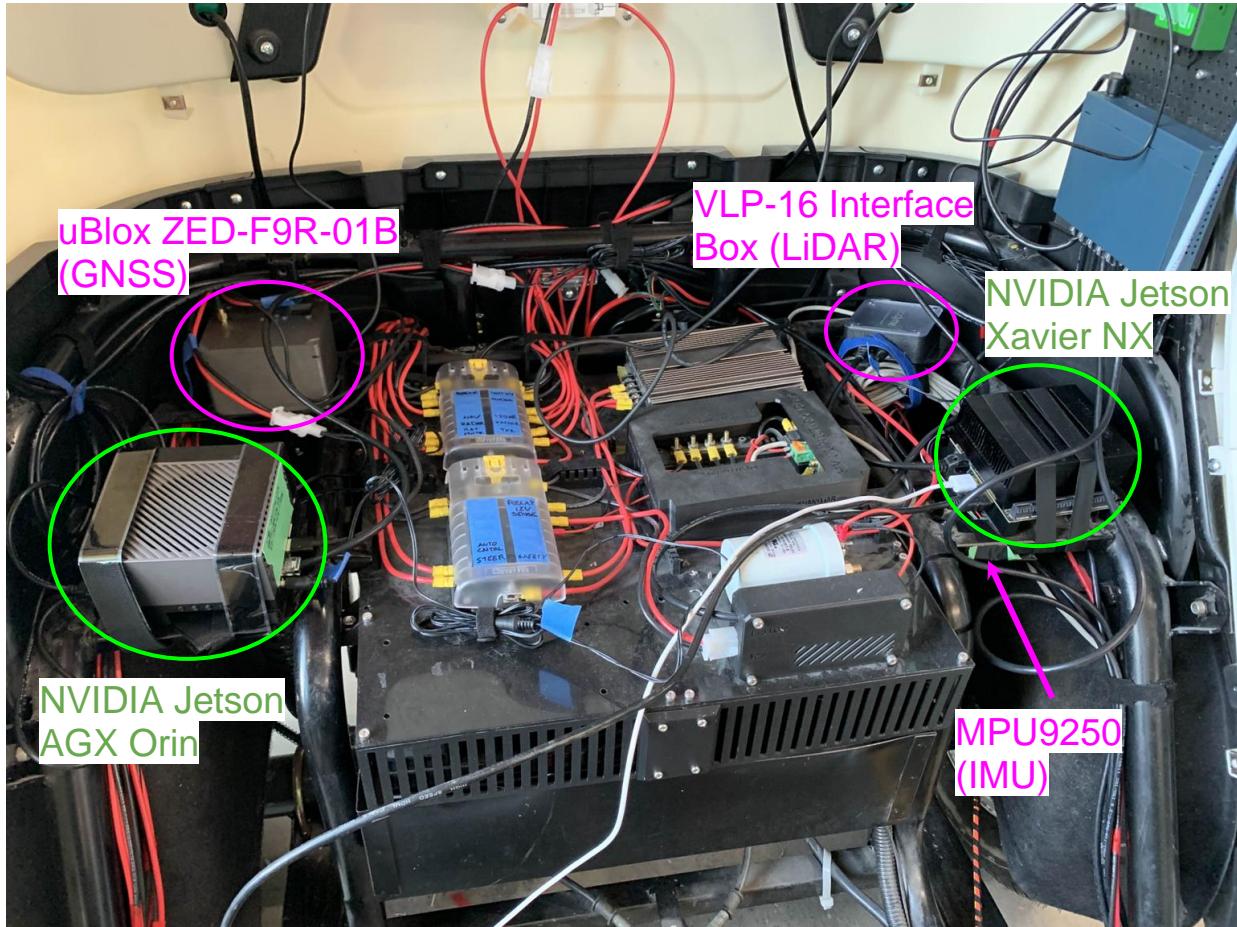


Figure 94: Key component locations

IMU Wiring and Placement:

The IMU used in this project was an MPU9250 module. These modules are relatively small and inexpensive, and can communicate over I2C or SPI. The device used in this project was wired over I2C with several DuPont cables connected to the header pins of the Jetson Xavier NX; specifically pins 1 (3V3), 3 (SDA), 5 (SCL), and 6 (GND). Because of the fragility of these pin connections, short wires were used to prevent disconnection. Thus, the IMU had to be kept close to the Jetson. It made sense to tape the IMU securely on the underside of the portion of the chassis that held the Jetson Xavier NX.

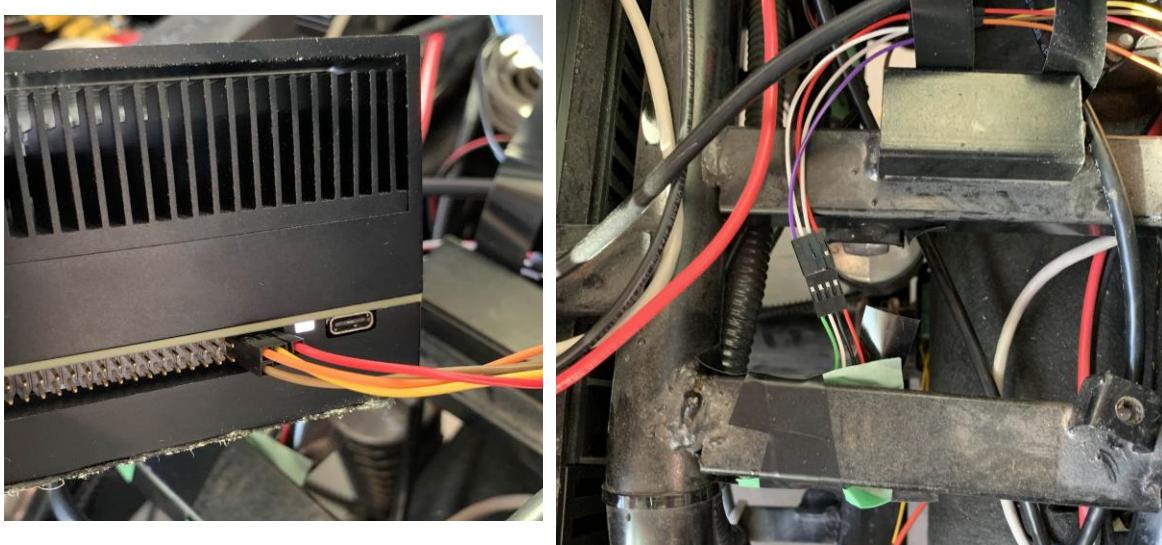


Figure 95: Wiring

10.4.4. Software Design

The software architecture was carefully designed to prioritize modularity, compatibility, and ease of integration, using ROS1 and ROS2 frameworks in parallel to leverage mature localization tools alongside modern sensor drivers. Due to ongoing integration challenges, full sensor fusion was not achieved during this iteration. Instead, each sensor independently published processed data to its own ROS topics, enabling subsystems to individually access and utilize these data streams.

ROS2 Implementation for Cameras and GNSS

The cameras and GNSS modules utilized ROS2-based drivers. Stereo camera data was processed for real-time object detection using YOLO and depth estimation through disparity mapping. Each camera node independently published detected objects and estimated depth information to dedicated ROS2 topics.

Similarly, GNSS data was parsed and published onto its own ROS2 topics, providing reliable global positioning information accessible for other subsystems such as Navigation and Simulation.

Independent IMU Driver Implementation

The IMU sensor data was initially collected via an open source ROS2-based MPU9250 driver made for RaspberryPi that had to be modified and rebuilt. However, to support compatibility with the ROS1-based Point-LIO system, this ROS2 version of the IMU driver had to be ported over to ROS1. Consequently, IMU topics could be published to either ROS1 or to ROS2 with these two drivers.

ROS1-Based LiDAR-Inertial Odometry (LIO)

LiDAR-Inertial Odometry was implemented using the ROS1-based Point-LIO package due to its proven performance and existing ROS1 dependency. This module was configured to subscribe to the /velodyne_points and /imu topics producing LiDAR and IMU data, which it used to estimate odometry and generate pointcloud maps that were published separately onto ROS1 topics.

ROS1/ROS2 Communication via Docker container

Due to the use of both ROS1 and ROS2 in parallel, a docker container running Ununtu 20.04 was configured to mount to the host to enable message passing from the docker container to the host. While complete sensor fusion was not achieved during this iteration, this setup allowed RViz2 running on ROS2 to display pointcloud and odometry. This could be useful if future teams choose to step away from our implementation with the Jetson Xavier NX, and instead connect the LiDAR and IMU to the Jetson AGX Orin, which may facilitate sensor fusion.

Custom Launch Files and Topic Management

Custom ROS1 and ROS2 launch files were created for each sensor and processing node, facilitating organized initialization and simplified debugging. Each sensor node clearly published data onto independent topics to enable isolated testing, analysis, and integration troubleshooting. Visualization of sensor data and preliminary localization outputs was conducted using RViz and the CARLA simulator.

Future development iterations aim to achieve full sensor fusion, consolidating these separate data streams into a unified perception pipeline.

ROS2 Integration

In order to publish the GNSS data to the ROS2 topics, the Python script created also publishes data to individual topics. Position is published to a NavSatFix topic type, while velocity is published to a TwistStamped topic type, orientation is published to a Imu topic type and acceleration is published to a Vector3Stamped topic type. These topics are needed to ensure that other subscribers understand what values are being published and how they can use those values.

10.4.5. Integration & Testing Approach

Incremental testing of each sensor followed by integration tests. The integration and testing strategy followed a systematic, incremental approach focused on validating each sensor independently before proceeding to higher-level integration.

Individual Sensor Testing

Initially, each sensor component - including LiDAR, cameras, GNSS, and IMU - was tested independently to ensure reliable data acquisition and processing:

LiDAR Testing:

LiDAR was tested by verifying accurate distance measurements, correct point cloud generation, and consistent environmental mapping using RViz visualization tools.

Camera Testing:

Cameras underwent testing for object detection accuracy using YOLO, stereo depth estimation effectiveness, and data consistency under different environmental conditions and lighting.

GNSS Testing:

GNSS data accuracy and consistency were validated through outdoor testing, comparing GNSS positional outputs to known geographic reference points.

IMU Testing:

IMU outputs were validated for accuracy, drift behavior, and consistency using controlled motion tests and by comparing data against known orientation and movement benchmarks. The ROS /imu topic was echoed to check if messages were published correctly, and the orientation of the IMU was visualized in RViz.

Incremental Integration Testing

Following individual validation, sensors were incrementally integrated to evaluate combined subsystem performance. Due to current limitations, each sensor continued publishing independently to dedicated ROS topics, and integration tests focused primarily on ensuring correct synchronization, message formatting, and cross-platform communication between ROS1 and ROS2 environments using the ros1_bridge.

Key integration testing steps included:

ROS Topic Validation:

Ensuring that sensor data was correctly published, timestamped, and received by subscriber nodes.

Data Visualization and Validation:

RViz was extensively used to visually confirm the consistency, accuracy, and synchronization of data streams from multiple sensors.

Simulation Testing with CARLA:

Sensor outputs were also integrated within the CARLA simulation environment, enabling controlled virtual testing of environmental perception tasks, including object detection, depth estimation, and localization consistency.

GNSS

The GNSS testing approach was as follows:

- Test the GNSS in a static environment, in other words not moving. Then, Obtain values from the GNSS and compare the values to sources such as Google Maps and an iPhone to measure accuracy from RTK and non-RTK data.
- Test the GNSS in a dynamic environment, in other words moving. Then, obtain values from the GNSS in a rosbag recording and extract the values and create a plot. Then, from the plot determine possible latency by adding up the number of data points and comparing to the amount of time it takes to complete the trip around campus. In addition, determining drift by comparing the plot to a Google Maps or OpenStreetMaps image of the campus and ensuring the trace of the roads of the Campus is recognizable.

The GNSS Integration approach was as follows:

- Using the Nvidia Jetson Orin, attach the GNSS for computing.
- Ensure on startup of the Jetson, the ROS2 Publisher script and RTK command are run.
- Ensure that the data is being published to a ROS2 topic by using the echo topic command from ROS2.

10.4.6. Challenges & Troubleshooting

GNSS

The GNSS was consistently reverting back to NMEA mode from UBX mode. This caused difficulties as the script that was supposed to be used was designed to be only in UBX mode. Through PyGPSClient, a GUI that has the capability of flashing the drive on a GNSS

receiver. The change did work for a few moments, but reverted back again. We were unable to find another solution that is consistently functional.

Cameras

While LiDAR effectively measured distances to many obstacles, it has limitations in its vertical field of view. The Radar was not used this semester due to the small number of people in the sensor's group but would provide the necessary depth information in front of the car to detect more objects. This limitation prompted efforts to implement distance estimation using stereo vision cameras combined with YOLO-based object detection and YOLO bounding box size estimation to identify and measure objects missed by LiDAR. Several technical challenges emerged during camera implementation:

Calibration Accuracy:

Achieving a good stereo camera calibration was challenging due to the imperfect mounting solution we created for the cameras. Minor alignment errors or environmental factors such as vibrations during testing significantly impacted depth estimation accuracy derived from stereo disparity maps. To properly solve this, the cameras would need anti-vibration and hardware camera stabilization.

Environmental Sensitivity:

Cameras are highly sensitive to variations in lighting conditions, with reduced accuracy under bright sunlight, shadows, or low-light environments. This sensitivity limited the reliability of camera-based depth estimation in practical outdoor scenarios and also caused issues for object detection.

Computational Performance:

Running real-time YOLO object detection alongside stereo disparity computation resulted in significant computational demands. This often led to reduced frame rates and higher latency in data processing.

Although some processing strain could be reduced by shifting more operations to the Jetson's GPU, this would require custom code to produce the entire processing pipeline, which exceeded the available development time this semester. Exploring neural networks specialized in depth estimation could also improve results, but it is likely that additional specialized hardware would be required to achieve real-time performance using cameras alone without sensor fusion with LiDAR and Radar.

LiDAR

There was a steep learning curve when initially integrating the LiDAR in ROS2. First, the host computer's ethernet settings had to be configured to the LiDAR's destination IP address.

There was also difficulty finding the correct launch file for the driver - which contained multiple launch files for different Velodyne LiDAR models.

One significant issue arose when configuring the LiDAR to DHCP mode. This is done by visiting the VLP-16's webui via its factory IP address (192.168.1.201) and modifying its settings. When DHCP is activated, the host automatically assigns it a new IP address. When this setting was modified, we lost connection to the LiDAR and could no longer access its webui to turn DHCP off, and we did not know the IP address assigned by the host. To solve this, while the LiDAR was connected to a router, we checked for devices on the router from the router's webui and found the LiDAR's new IP address. Then, by connecting to this IP address, we were able to access the LiDAR's webui once again. From this experience, we also learned that the VLP-16 driver has a configuration file, with one of the parameters being device_ip. By setting this value to the IP address assigned through DHCP, we were able to get the driver working with DHCP. This could be useful in future iterations where there is a desire to automatically assign the LiDAR's IP address while connected to the electric vehicle's router, which could prevent conflicts with other devices.

Another issue arose when launching Nav2, which wouldn't show the LiDAR's pointcloud nor the generated costmaps in RViz. This was due to missing transform frames. To resolve this, we had to set static transforms from map to odom, odom to base_link, base_link to velodyne, and base_link to base_footprint.

Another issue, more related to Nav2, is that the costmaps generated by Nav2 were only using a 2d laserscan instead of the 3d pointcloud. This is equivalent to taking a single laser's "slice" to detect objects. To resolve this, we worked with the navigation team to configure Nav2's parameter file, and tried implementing a plugin called Spatio-Temporal Voxel Layer (STVL), developed by Steven Macenski at Simbe Robotics. Our results were inconclusive, and we ended up doing most of our testing with the costmaps generated by laserscan due to time constraints.

IMU

Development on the IMU started late into the second semester. It was mistakenly believed the the GNSS could produce IMU data, but when we found that this was not the case, we opted to use an MPU9250 module.

The challenges associated with this IMU were twofold. First, the IMU lacked working out-of-the-box ROS1 and ROS2 drivers. There were no drivers provided by the vendor, and the open source drivers that existed were built for Raspberry Pis and Arduino microcontrollers rather than Jetson boards. At this time, we were also exploring ROS2 compatible packages for LiDAR based odometry. We cloned one of the ROS2 drivers designed for Raspberry Pi and tried using

it. After troubleshooting, problems were found with the configured I2C bus number and communication with the onboard magnetometer. After several modifications to the source files, it was rebuilt and working. Soon after, the Point-LIO package was discovered - which was only compatible with ROS1. Thus, the modified MPU9250 ROS2 package was manually ported to ROS1 in its entirety, which we set up on the Jetson Xavier NX and got working.

The other significant challenge was the IMU's sensitivity to noise. This is likely due to the low cost of the sensor. The automatic launch calibration, which uses the magnetometer to orient the sensor, helped offset differences in expected values, but noise in the IMU continued to be a major problem for the Point-LIO package throughout the project.

10.5. Project Management & Execution

10.5.1. Work Breakdown Structure

Phase	Task	Description
Phase 1	Researching & Planning	Define sensor goals, evaluate hardware options, review ROS2 pipeline standards
Phase 2	Individual Sensor Integration	Setup camera pipeline with YOLO object detection Iterate GNSS and IMU pipeline with ROS2 Setup LiDAR with Point-LIO (ROS1) and bridge to ROS2 if possible
Phase 3	Experimental Testing	Test each sensor pipeline individually in lab and outdoors Begin testing depth estimation from stereo vision and object bounding boxes Evaluate GNSS accuracy, signal stability, and filtering
Phase 4	Combined System & ROS2 Publication	Publish sensor data to ROS2 topics for Navigation and Control Attempt ROS1-ROS2 bridge for

		point-LIO Evaluate real-time performance and latency
--	--	---

Since there were multiple sensors that need to be added to the vehicle, roles were split based on familiarity, strengths, and interests on each sensor:

- Cameras and object detection were primarily handled by Ryan and Alexei
- GNSS integration and debugging was done by Joshua
- LiDAR and Point-LIO setup were worked on by Daniel

10.5.2. Gantt Chart & Milestone Tracking

Sensors/Perception Gantt Chart

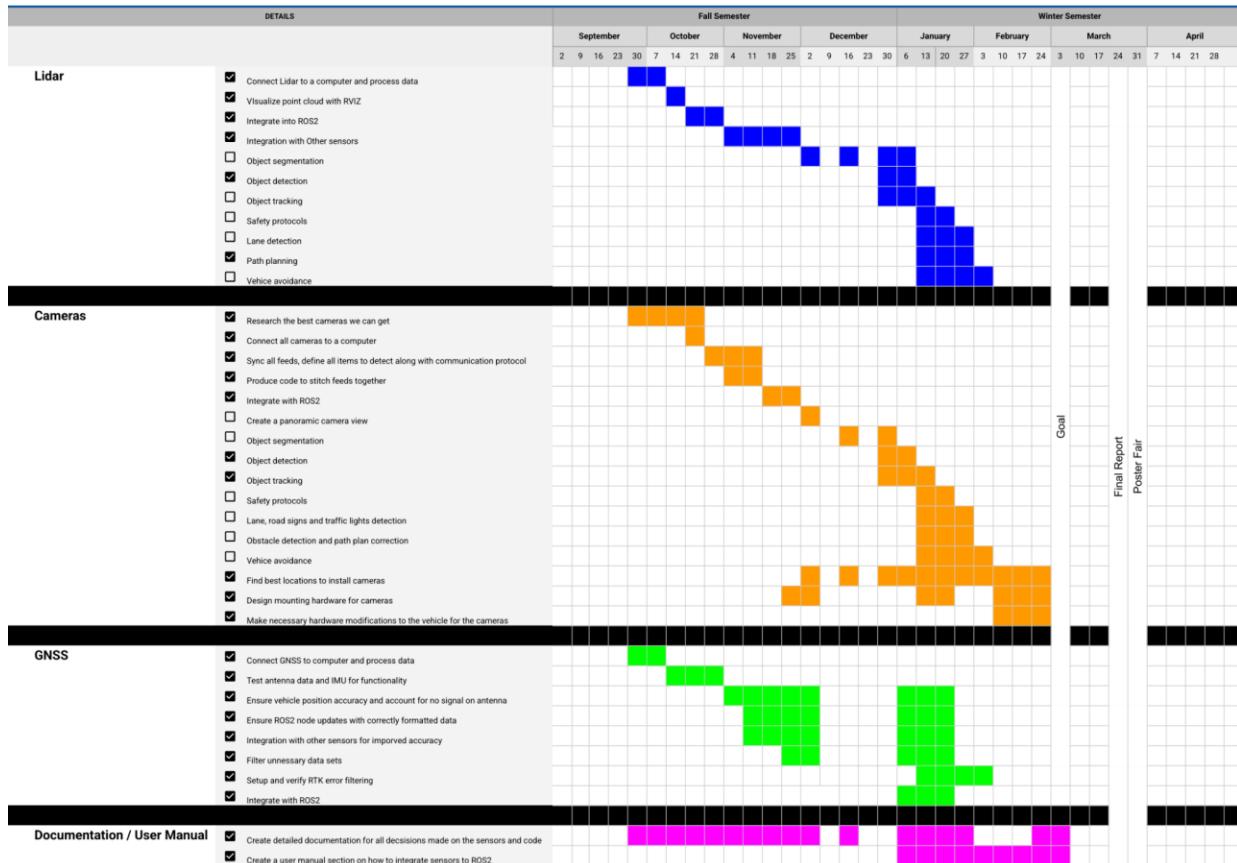


Figure 96: Sensors Gantt Chart

While it was our goal to have basic autonomous driving by the end of the winter semester, that was overly ambitious. We ran into a variety of challenges that held back our progress, including unexpected issues within our group and integration with other subgroups.

10.5.3. Risk Analysis & Mitigation Strategies

Throughout the development process, the Sensors and Perception team identified several risks related to hardware, integration, and system performance. These risks were addressed proactively to minimize project delays and improve system reliability.

GNSS Configuration Instability

The GNSS module frequently reverted from UBX mode to NMEA mode, interfering with our data processing pipeline, which depended on consistent UBX messages. Despite using PyGPSClient to configure the GNSS receiver, the changes were not persistent. To mitigate the impact of this issue, we limited GNSS functionality to available NMEA data and documented the need for a more robust GNSS module in future implementations.

Inaccurate Stereo Depth Estimation

Achieving reliable depth information from stereo vision proved difficult due to inconsistent calibration, sensitivity to lighting, and environmental variation. To reduce the impact, we refined calibration procedures and conducted controlled environment testing. However, the results remained inconsistent. The team opted to focus on object detection using YOLO and used stereo data primarily for experimental support, deferring full depth-based functionality to future development.

ROS1 and ROS2 Bridging Complexity

Integrating the ROS1-based Point-LIO package into the ROS2 system added complexity and caused versioning issues. While ros1_bridge was configured to enable communication between systems, full integration was not achieved within the semester. As a mitigation strategy, we treated each sensor system as an independent ROS node, allowing other subsystems to subscribe to specific topics without requiring complete sensor fusion.

Computational Load and Latency

Simultaneous processing of YOLO object detection and stereo disparity maps created a high computational load. While shifting more processing to the Jetson GPU could help, doing so would require additional custom development. We mitigated the issue by testing each pipeline independently and prioritizing stable real-time performance for object detection over real-time stereo depth estimation.

Testing and Integration Delays

Time constraints and unexpected technical issues delayed full integration across subsystems. To address this, the team maintained a modular system design that allowed each sensor to be tested and validated independently. Regular communication with other subteams helped maintain awareness of compatibility requirements and ensured continued progress on core deliverables.

10.5.4. Budget & Cost Analysis

The Sensors and Perception team focused on building a robust yet cost-effective perception stack by leveraging already purchased sensors while purchasing only essential components. Core hardware such as the Jetson Xavier NX, Jetson AGX Orin, LiDAR, and GNSS were already available to us in Bay 4. Our expenditures were limited to key accessories and peripherals required to complete the perception pipeline and ensure proper sensor connectivity and operation. The total expenditure for the subteam was \$351.07. A breakdown of all the expenses are below.

While most purchases proved effective, one notable issue arose with the 10-port USB 3.0 powered hub. The hub was intended to connect multiple OV2710 USB 2.0 cameras to a single Jetson device. However, a hardware-level conflict was discovered: because the cameras operate over USB 2.0 and the hub runs at USB 3.0, the entire connection reverted to USB 2.0 speeds. This significantly reduced available bandwidth, causing data bottlenecks and inconsistent camera feeds. As a result, the hub was not usable for its intended purpose. This limited our team to 2 cameras connected to the Jetson for this semester.

Item	Total Price	Purpose
OV2710 2MP USB Cameras	\$223.25	Stereo vision, object detection, and 360-degree view
3 meter USB extension cables	\$68.96	Extend cameras to central onboard computer
5.5x2.5mm male barrel connector	\$1.13	Power connection for LiDAR from battery system
Cat7 Ethernet Cable 7ft	\$13.55	Connect LiDAR interface box to onboard computer
iCAN 10-Port USB 3.0 powered hub	\$45.19	Expand camera connectivity, prevent bandwidth issues

This budget supported critical system functionality, and despite the issue with the USB hub, most components were effective and remain usable for future development. Future work should include additional budget allowances for improved hardware, such as a high-performance IMU.

10.6. Experimental Setup & Testing

10.6.1. Testing Environment & Procedures

GNSS Testing:

GNSS data accuracy and consistency were validated through outdoor testing, comparing GNSS positional outputs to known geographic reference points.

Camera Testing:

Camera testing was performed in both indoor and outdoor environments to evaluate object detection, stereo calibration, and image quality under various lighting conditions. YOLO object detection was verified by showing a camera a stop sign and evaluating bounding box accuracy and detection confidence.

Stereo depth estimation was tested using side-by-side cameras with calibrated baselines. Depth accuracy was evaluated by comparing estimated object distances to actual measured distances using known reference targets. Disparity map quality was inconsistent, likely due to calibration issues and lighting conditions.

LiDAR Testing:

The LiDAR was tested primarily in indoor environments. RViz2 was utilized to visualize the pointcloud, and the Nav2 package was used to generate costmaps. The testing mainly involved tweaking configuration files from certain packages or the parameters of the VLP-16 driver and visualizing the results. The LiDAR was tested during both outdoor live tests, along with the IMU for Point-LIO implementation. The LiDAR performed well, and achieved a 50+ meter range. Rings could be seen hitting the ground, and it is possible that road segmentation will need to be used in future iterations. During the live tests, many rosbags were taken of the /imu, /velodyne_points, and /Odometry topics and saved on the Jetson Xavier NX. These rosbags should be useful for testing in future iterations. It should be noted that the results of Point-LIO's odometry were inaccurate, due to noise in the IMU data, and possibly misconfiguration for the IMU in the Point-LIO package.

IMU Testing:

IMU testing was limited as development began late in the semester and had to be very quick in order to integrate with the Point-LIO package which we used for odometry. To test the IMU, the /imu topic was echoed while the MPU9250 driver published ROS messages. From this, the acceleration, and gyroscopic data could be analyzed. Some small changes were made to the Point-LIO parameters to account for the relatively large noise in the IMU data and the extrinsic parameters of the Point-LIO configuration file were set based on the initial orientation of the IMU, which was consistent after the automatic calibration sequence when launching the driver.

10.6.2. Experimental Results and Findings

GNSS

Using Rosbag, a trip was made around campus to visualize the entire campus using a plotted graph of every data point published to the ROS2[20] NavSatFix topic type. A plot vs map projection of the Campus of Carleton University is shown below:

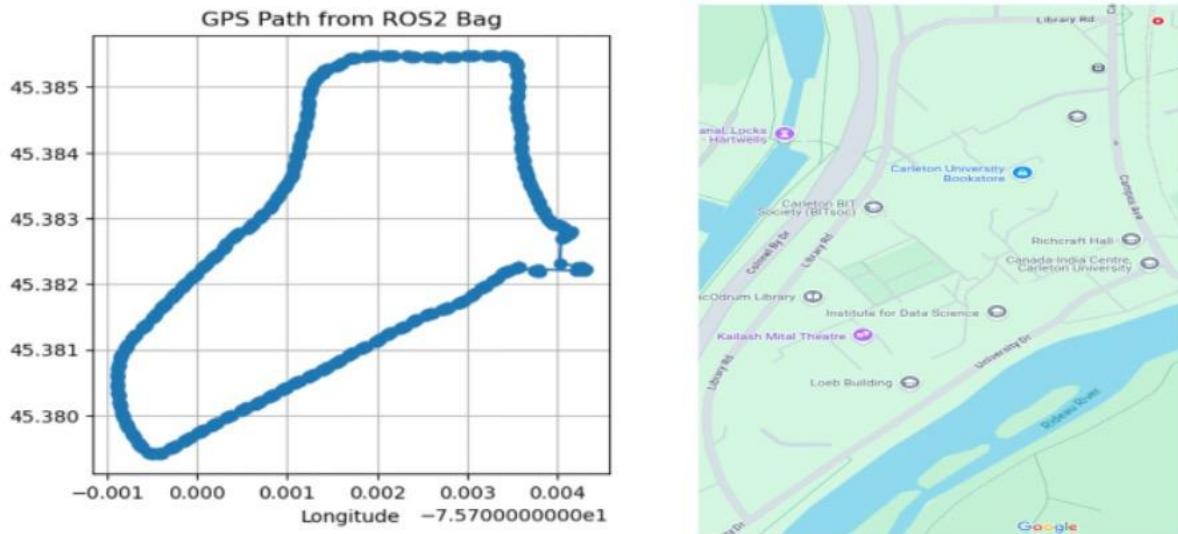


Figure 97: rosbag plot of Carleton University campus around academic buildings in comparison to Google Maps imagery.

Static results of places around campus and Ottawa have also been obtained. Comparisons are shown below:

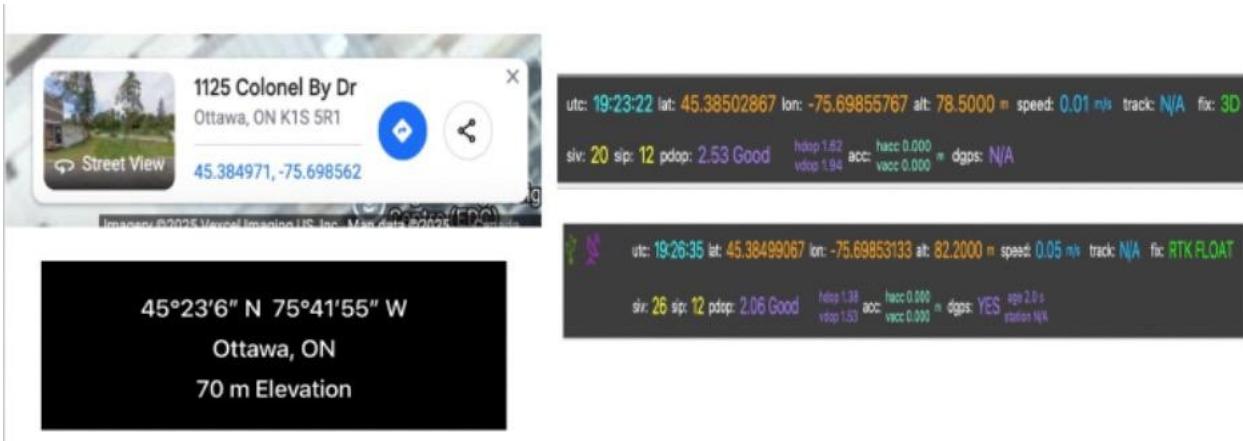


Figure 98: EDC comparison between Google Maps, an iPhone, and the GNSS with RTK on and off.



Figure 99: Azrieli comparison between Google Maps, an iPhone, and the GNSS with RTK on and off.

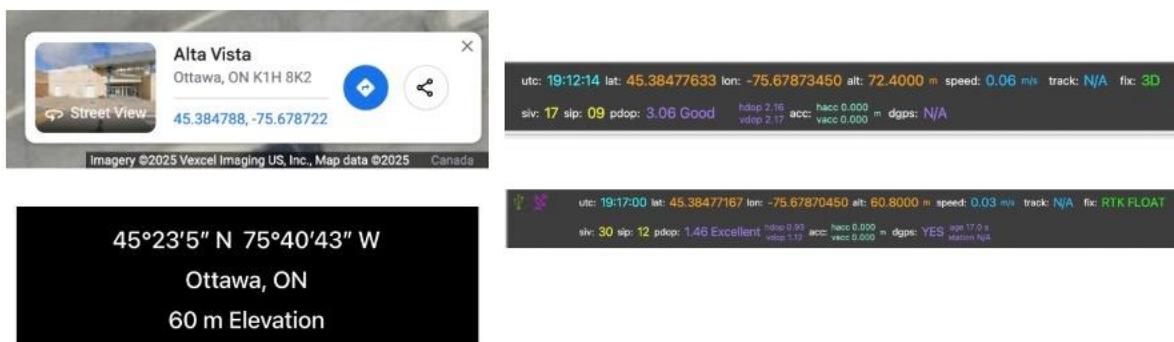


Figure 100: Billings Bridge Shopping Centre comparison between Google Maps, an iPhone, and the GNSS with RTK on and off.

As we can observe from the figures above, the static comparisons between the GNSS receiver and google maps and iPhone are the same, rounded to 4 decimal places. This confirms that the GNSS produces accurate results while static. The same can also be said while the GNSS

is moving and being dynamic. As can be seen with the outline of the campus, the GNSS produces accurate results. However, this will be mentioned in the analysis, the GNSS does struggle with underpasses. In other words, without a clear sightline of the sky to satellites, the GNSS struggles to accurately determine our position.

Cameras

Experimental testing of the camera system focused on two primary objectives: validating object detection using the YOLO algorithm and evaluating stereo vision for depth estimation. Both tasks were tested in controlled indoor environments and outdoor locations around campus. The goal was to identify key objects in the vehicle's environment, such as stop signs, and to provide approximate positional data relative to the vehicle.

For object detection the Ultralytics YOLO model [95] was cloned and trained on a dataset of stop signs to perform real-time object detection. The model was deployed in python subscribed to a ROS2 topic for images and published detection results including bounding box parameters, confidence scores, and class labels.

Object Detection for single-camera applications

Object Detection using digital cameras is intended to achieve the following goals:

- Traffic-signs and traffic-lights detection and estimation of their positions relative to the vehicle.
- Obstacles detection and estimation of their position relative to the vehicle.
- Current road lane detection and estimation of the vehicle position relative to the lane.

Distance and Direction Evaluation

Within the scope of the current project, the traffic-sign detection task was solved using Ultralytics YOLO software [21] for object recognition for single-camera application. The YOLOv11 PyTorch version software was installed and trained for the Stop-sign recognition and detection task.



Figure 101: Traffic Stop-sign image.

According to Ultralytics YOLO data formats [22], each detected object is presented within a Bounding Box along with several parameters attached to the Bounding Box like current level of detection confidence, relative width and height of the Box, relative horizontal X and vertical Y positions of the central point of the Bounding Box. The following image explains the Bounding Box parameters.



Figure 102: Detected objects within corresponding Bounding Boxes.

Since all traffic signs have standardized dimensions, their relative size and position visible through the computer vision image can provide distance and direction evaluation towards the detected traffic sign. As each detected object is surrounded by a Boundary Box, the relative Width and Height of the Bounding Box can be extracted from the image data flow to perform the distance and direction evaluation toward the object.

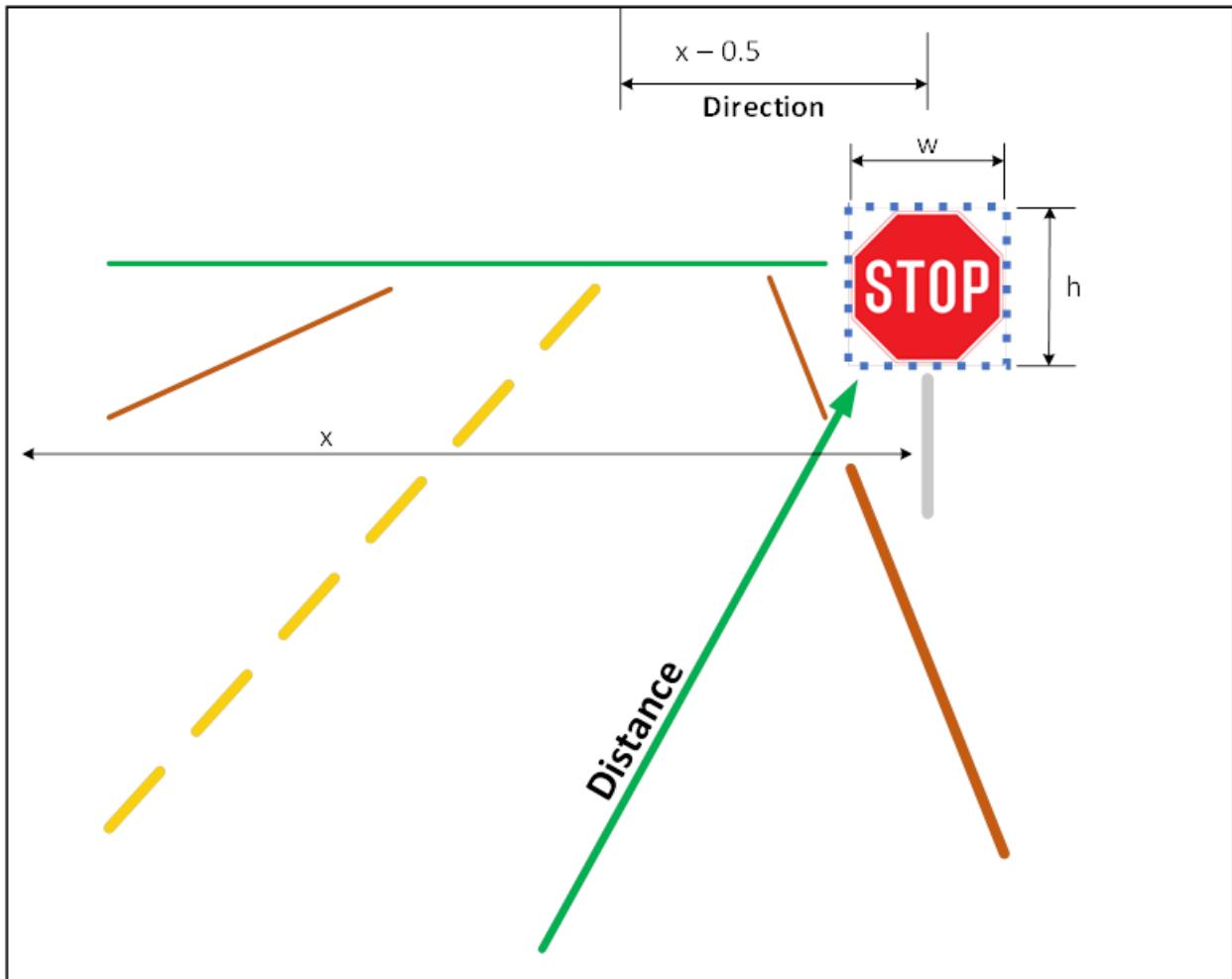


Figure 103: The background scheme for Distance and Direction estimation.

Since the actual size of the detected object is known in advance, the actual distance can be evaluated by the following formula:

$$Actual\ Distance = \frac{referenceBoxSize \times referenceDistance}{visibleBoxSize}$$

The actual direction toward the detected object can be evaluated as an angle measure between the centre of the digital camera image and the centre of the Bounding Box around the detected object. In assumption that the whole image width spans 90 degrees, the tangent of that angle is the relative horizontal position X of the Bounding Box centre subtracted by 0.5 times two:

$$\tan(Direction) = 2(x - 0.5)$$

Therefore, the relative angle measure for the Direction is:

$$Direction = \tan^{-1}(2x - 1)$$

If the whole image width spans less than 90 degrees, the $w_{coef} < 2$:

$$\tan(Direction) = w_{coef}(x - 0.5)$$

$$Direction = \tan^{-1}(w_{coef} \cdot x - 0.5 \cdot w_{coef})$$

Thus, the digital camera sensor module running computer vision software provides Real-time data flow containing the actual distance and direction measurements for detected traffic signs.

In the ROS2 environment, the YOLO model object detection script in Python performs the real-time Distance and Direction estimation and publishes the output data flow for Navigation and Control subscribers.

Stereo Depth Estimation

In addition to single-camera object detection, stereo vision was explored as a method for estimating the distance to objects using a pair of synchronized cameras. This approach aimed to provide additional depth information in areas where LiDAR coverage was limited, especially for nearby or low-lying objects directly in front of the vehicle. Stereo vision works by capturing two images from slightly offset perspectives. By comparing these images and calculating the pixel displacement, or disparity, between corresponding points, the system can infer depth. An example image is shown below:

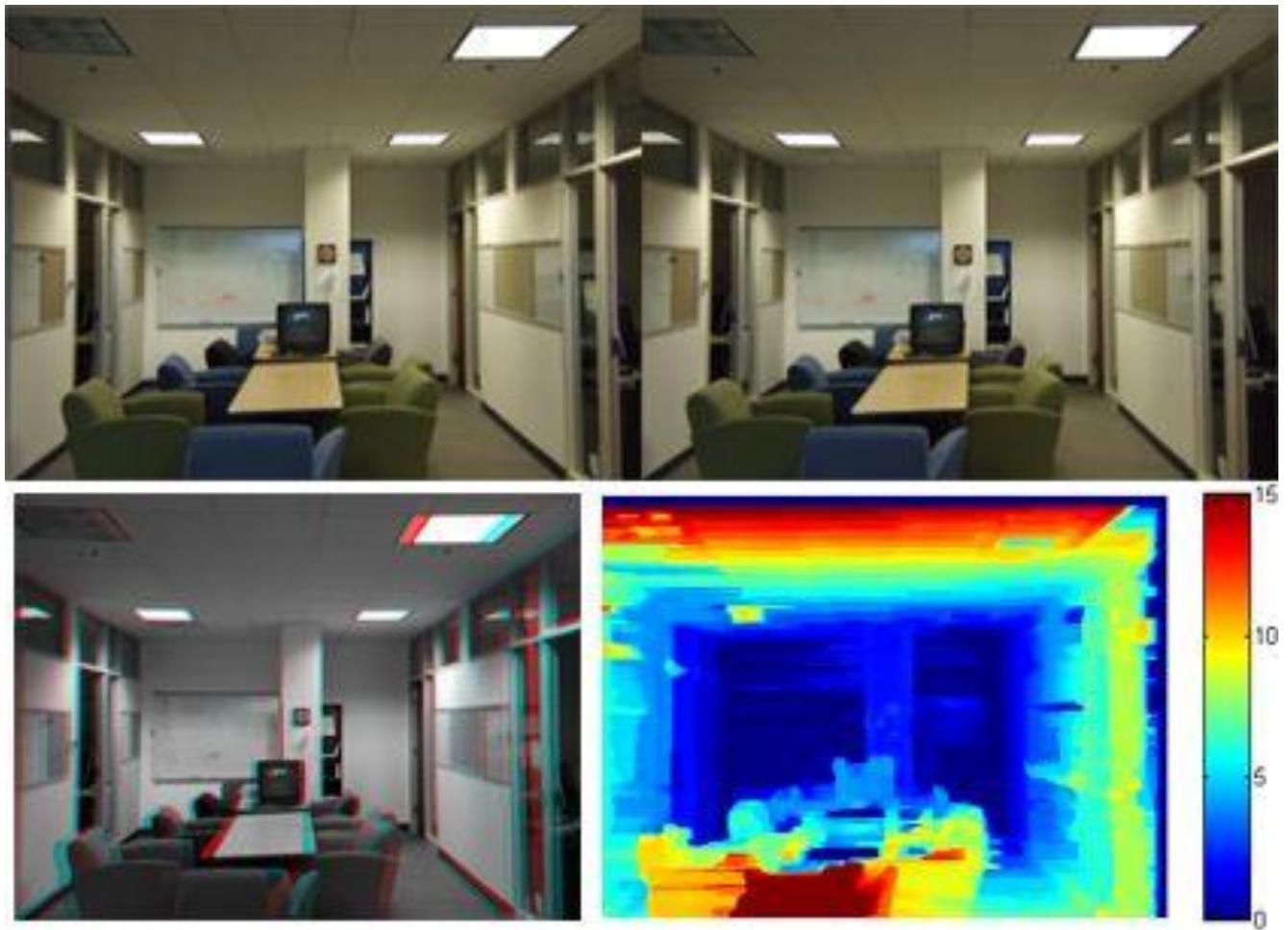


Figure 104: Combining images from 2 cameras into disparity map [97]

To enable stereo processing within the ROS2 framework, custom code was developed to publish synchronized left and right camera feeds. These feeds were accompanied by corresponding camera_info messages containing the required intrinsic and extrinsic calibration parameters. This setup ensured that downstream tools could consume the stereo data in a consistent and standardized format.

The ROS2 package stereo_image_proc was used to perform stereo rectification and disparity computation. Given the calibration data, this package produced disparity maps in real time. These maps visually represent depth, with color gradients ranging from dark blue (farthest) to red (nearest). A sample disparity map generated during testing is shown below.

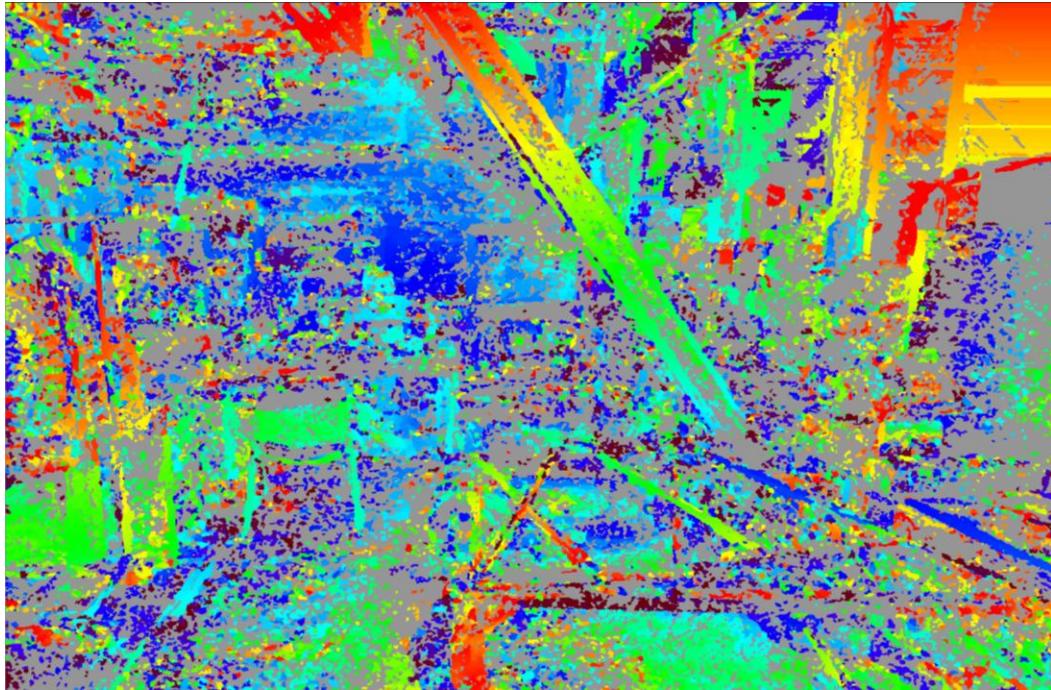


Figure 105: Disparity map generated using `stereo_image_proc` and synchronized camera feeds.

Testing was conducted both indoors and outdoors, with objects placed at known distances. Within a range of approximately four meters, it was possible to estimate distances to objects and people using the disparity map. However, the overall accuracy was limited. For example, the back wall of Bay 4 was correctly detected at approximately five meters and appeared in dark blue near the center of the disparity map. However, at the edges of the image, accuracy degraded. The right side of the image falsely displayed the desk, located only one meter away, as also appearing at a distance of around five meters.

This inconsistency was attributed to the physical configuration of the stereo cameras. The cameras were mounted with a baseline distance of 54.5 centimeters, and the focal length determined during calibration was approximately 895.77 pixels. Using the stereo vision equation: $d = \frac{f * B}{Z}$, where d is depth in meters, f is the focal length, B is the baseline in meters, and Z is the depth that we are hoping to achieve. This equation was derived from the original equation $Z = \frac{f * B}{(x - x')}$ [98], which computes depth based on the distance between cameras. We can calculate the expected disparity values:

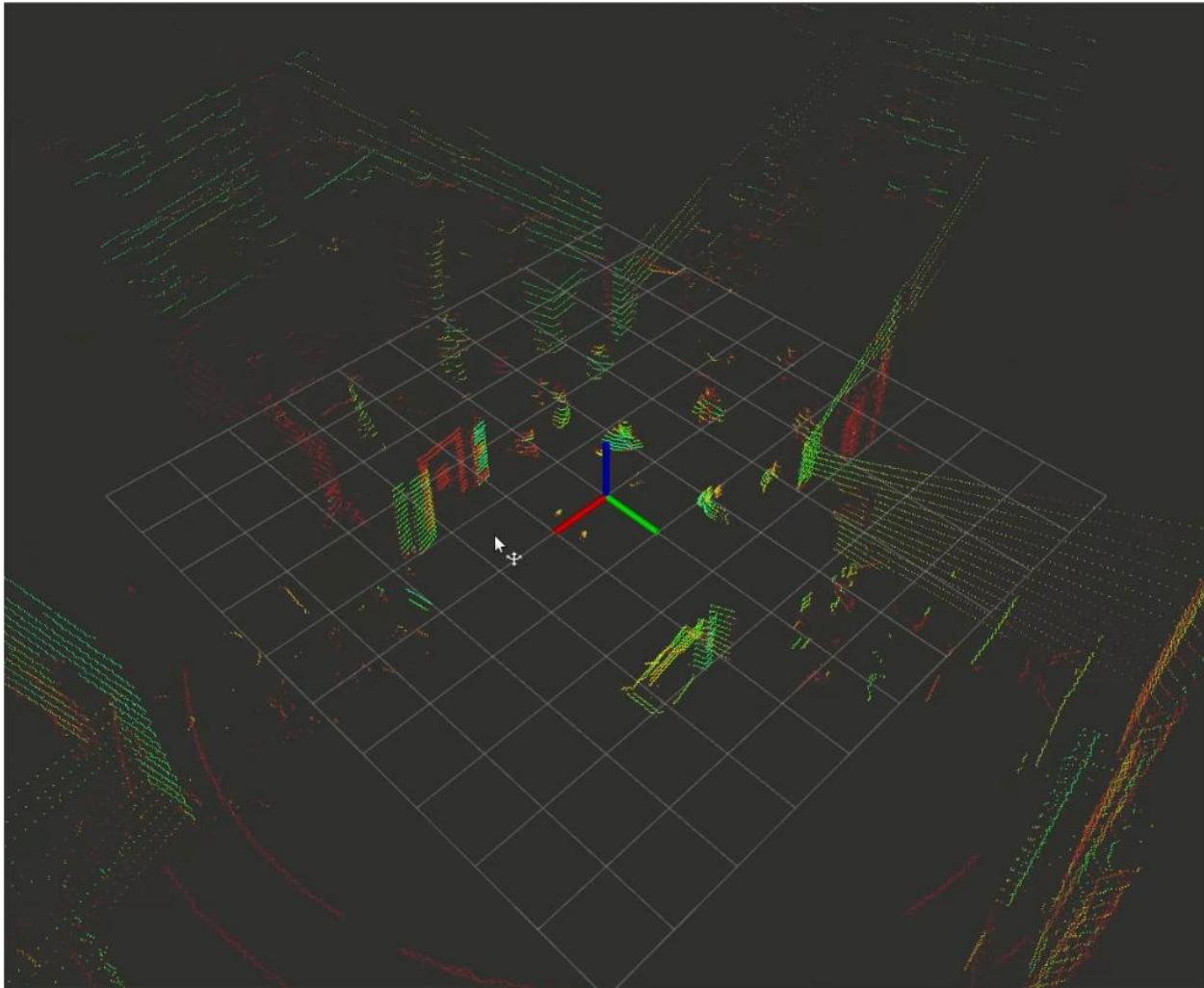
- At a depth of 1.5 meters: disparity ≈ 325 pixels
- At a depth of 4.0 meters: disparity ≈ 122 pixels

This places the effective disparity range for our system between roughly 122 and 325 pixels. In practical terms, this means that the stereo algorithm is expected to find matching points between the two images that are horizontally offset by anywhere from 122 to 325 pixels, depending on the object's distance. We have 2 1920 horizontal pixels in our camera feeds which means that close objects that appear on both camera feeds with more than 325 pixels of distance apart will not produce an accurate disparity. These measurements were used as input for the ROS2 package `stereo_image_proc` to produce the image above. However, due to the wide baseline, objects closer to the cameras appeared with high parallax near the image center but had minimal or distorted overlap at the image edges. As a result, stereo matching struggled in these regions, leading to poor depth estimation or no disparity being computed at all.

Lighting conditions and texture also played a significant role. Environments with strong shadows, reflective surfaces, or overexposed regions further degraded disparity quality. Calibration accuracy was critical; even slight errors in alignment introduced noise and reduced consistency across the depth map. Although stereo vision shows promise in concept, practical limitations in calibration, configuration of the package, computational load, and physical camera alignment caused significant issues in the disparity map preventing good depth estimates.

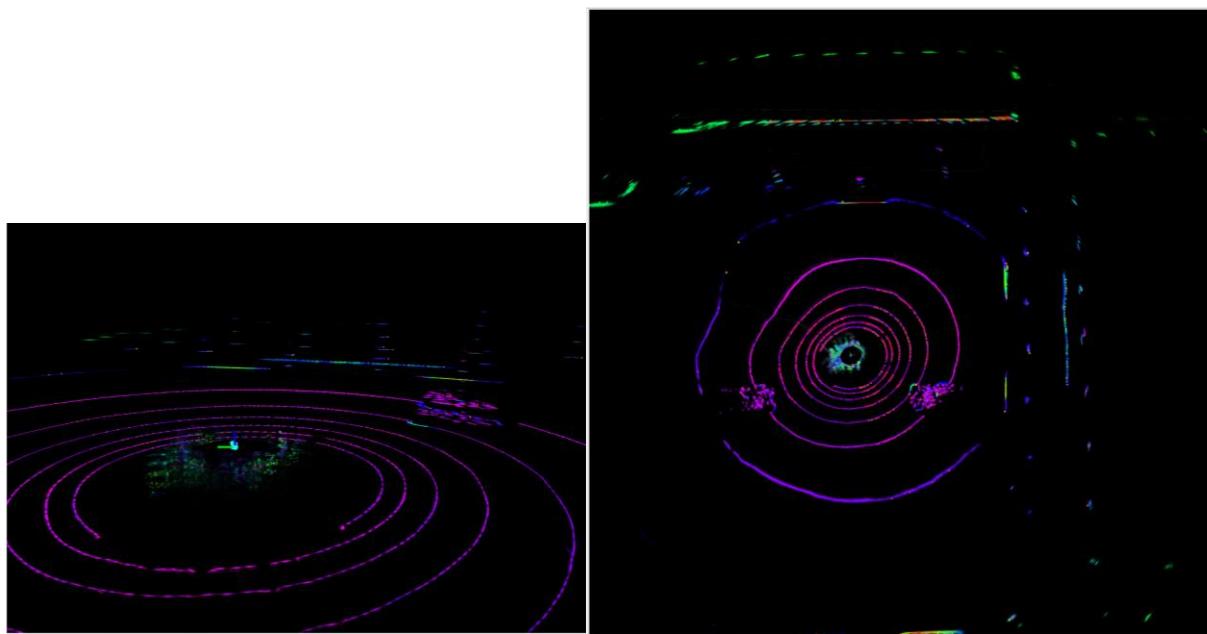
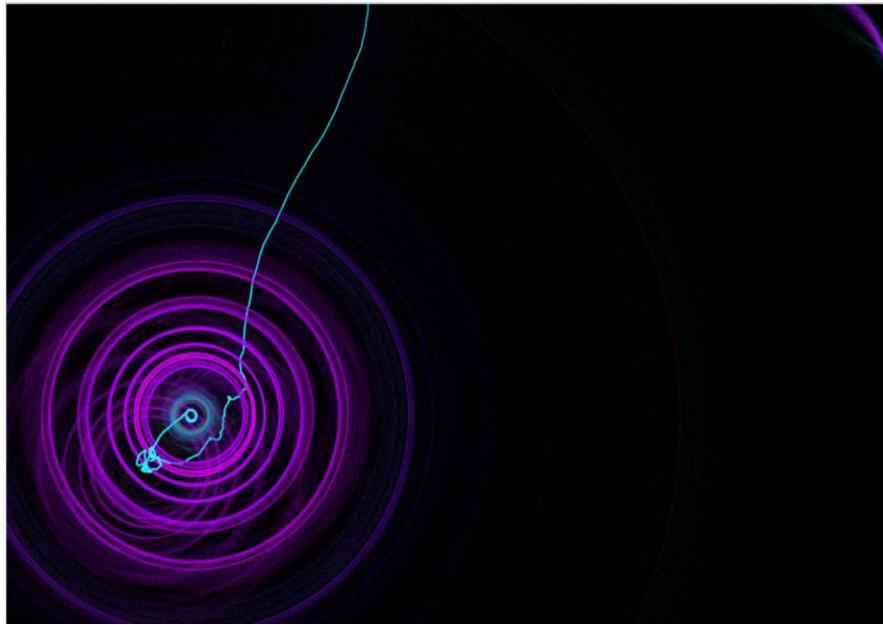
Point-LIO

The VLP-16 LiDAR produced high-quality point clouds, with over 50 meters of effective range. An example of the pointcloud in the EDC is shown below:



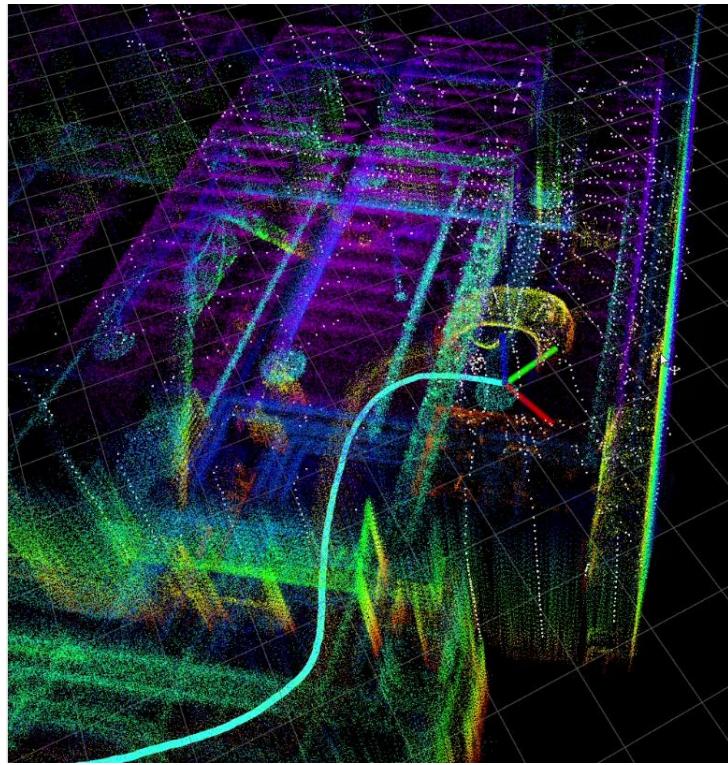
During live testing, multiple rosbags were recorded for `/velodyne_points`, `/imu`, and `/odometry` topics. However, Point-LIO odometry results were noisy, largely due to limitations of the MPU9250 IMU. Noise in the sensor data, and likely misconfigurations in the extrinsic parameters, negatively affected localization accuracy.

Below are images taken from RViz running Point-LIO from the outdoor tests. Note the inconsistent odometry path in the first image:



The distant curbs in the parking lot can be seen in the photos above. Note that rings can be seen touching the ground. Thus, ground segmentation may be necessary when generating costmaps in future iterations.

The following image is a still from a test performed with a Unitree L2 LiDAR in the EDC running Point-LIO.



This LiDAR has an integrated IMU, and the odometry appears to be much more accurate.

IMU

The MPU9250 was manually integrated using a ported driver. Functional testing involved publishing IMU data over ROS and visualizing it in RViz. Although functional, the IMU demonstrated significant noise, especially in the magnetometer and gyroscope readings, which impacted its suitability for precision odometry.

10.7. Discussion & Analysis

GNSS

GNSS performance: out of 754 points published in the rosbag of campus pictured in figure 70, only 9 points were incorrect due to the underpass at Richcraft building as can be seen from figure 97. Therefore, the GNSS around Carleton University is approximately 99% accurate, ($745 / 754 \times 100\% = 99\%$) especially with RTK corrections. Latency to publish is minimal as attempts in the script were made to publish at a rate of approximately 10 Hz, which is approximately 10 readings a second being published. However, from the rosbag recorded, the GNSS was obtaining and the ROS2 topic was updating about 1 fix every 2 seconds (754 position fixes in approximately 23 minutes or 1380 seconds) due to the time it takes to obtain readings from satellites. Therefore, latency to obtain and publish data is approximately 2 seconds. In our case, this latency is acceptable as our car can only move upwards of 30 km/h. Therefore, we are going approximately 16.6m in

two seconds ($30\text{km/hr} / 3.6 = 8.3\text{m/s} \times 2\text{ seconds} = 16.6\text{m}$) without an updated fix. However, with faster cars, this may become a liability as the distance travelled without an updated fix could reach upwards of 55.4m in two seconds ($100\text{km/hr} / 3.6 = 27.7\text{m/s} \times 2\text{ seconds} = 55.4\text{m}$). Drift is also quite minimal, which is to be expected with RTK error corrections as can be seen from figure 97 as RTK was utilized with the campus rosbag. The campus roads were followed very well by the GNSS and we can clearly see an accurate trace of campus concluding therefore there is minimal drift when RTK is used.

However, there are limitations as mentioned, especially with RTK being dependent on a solid internet connection and the GNSS struggling with underpasses and large buildings creating dead reckoning. However, the GNSS is equipped with an IMU that counters dead reckoning; however, even with the IMU, the GNSS does struggle to exactly pinpoint where it is. This could be because the IMU is absent on the board or is broken.

Cameras

The camera system was tested for two primary applications: object detection using YOLO and depth estimation using stereo vision.

YOLO object detection performed reliably in controlled conditions. Stop signs were successfully detected with high confidence, and bounding boxes were correctly drawn in most test cases. The model maintained strong detection accuracy when objects were well-lit and unobstructed. However, performance degraded slightly in low-light environments or when objects were partially occluded. Real-time detection was functional when running the pipeline independently, but experienced minor latency when run alongside other ROS2 processes due to computational strain on the Jetson Orin.

A key enhancement to this system was the use of bounding box geometry for estimating distance and direction to detected objects. This approach relied on the standardized dimensions of traffic signs such as stop signs. For each detected object, YOLO generated a bounding box with the following parameters:

Detection confidence

Relative width and height of the bounding box

Relative X and Y coordinates of the box's center point

Because the physical size of traffic signs is known in advance, their perceived size in the camera image can be used to estimate distance. The smaller the bounding box height, the farther away the sign is. This method assumes a pinhole camera model where distance is inversely proportional to

the bounding box height. While approximate, this approach yielded reasonably accurate distance estimates for standard-sized signs under known conditions.

Direction was calculated by taking the horizontal offset of the bounding box center from the center of the image. Assuming the camera has a 90-degree horizontal field of view, this offset can be used to estimate an angle to the object using a simple tangent relationship. This gives the system the ability to determine both how far and in which direction a detected sign is located relative to the vehicle's forward path.

While this distance and direction estimation method worked well for stop signs, it is limited to objects with known and consistent dimensions. It does not perform reliably on people or objects of variable size, as incorrect assumptions about real-world scale lead to errors in depth estimation.

Stereo vision depth estimation, implemented using ROS2's stereo_image_proc package, produced usable disparity maps in the range of approximately 1.5 to 4 meters. The stereo cameras were mounted with a baseline of 54.5 cm, and camera calibration yielded a focal length of approximately 895.77 pixels. From these parameters, the expected disparity values ranged from about 325 pixels (at 1.5 meters) to 122 pixels (at 4 meters), defining the system's effective disparity range.

Within the center of the image, where overlap between the two cameras was highest, depth estimation was relatively consistent. However, accuracy decreased significantly at the edges of the image. This was caused by reduced image overlap due to the wide baseline, leading to poor stereo matching in peripheral regions. Additional issues were observed in low-light conditions or on textureless surfaces, where the stereo algorithm struggled to match features between the left and right images.

The custom stereo pipeline, including synchronized image and camera info publication, worked reliably within ROS2. However, the system's ability to perform real-time stereo vision alongside other perception tasks was limited by hardware capacity. Running stereo disparity and YOLO detection simultaneously caused frame rate drops and introduced latency, making the combined system less suitable for real-time use without optimization.

Overall, the camera system showed strong potential for object detection and demonstrated a functional approach to estimating distance and direction using both single-camera bounding box analysis and stereo vision. However, its integration into a real-time perception pipeline will require additional optimization, improved calibration, and better hardware or dedicated processing for demanding tasks like disparity computation and neural inference.

LiDAR

The VLP-16 provided reliable environmental data and was successfully integrated into the perception pipeline. Initial issues such as network misconfiguration (e.g., DHCP mode changes) were resolved through careful network diagnostics. The sensor proved to be a strong component for mapping and costmap generation. However, limitations in ROS2 costmap plugins meant that most real-time tests relied on a 2D laserscan abstraction.

A major limitation stemmed from ROS2's default costmap plugin relying on 2D laserscans rather than 3D pointclouds. Although efforts were made to implement the Spatio-Temporal Voxel Layer (STVL) plugin to make full use of the LiDAR's vertical resolution, the results were inconclusive and required additional tuning that could not be completed within the semester. Despite this, the LiDAR data proved reliable, offering strong performance during both indoor and outdoor tests.

For Points-LIO, the MPU9250 module provided basic orientation and acceleration data but exhibited considerable noise. Despite this, Point-LIO was successfully configured to subscribe to /velodyne_points and /imu, and it produced real-time odometry estimates. =

IMU:

The MPU9250, though accessible and inexpensive, lacked precision. High noise levels in the magnetometer and accelerometer limited its effectiveness in contributing meaningful inertial data to the LIO pipeline. Despite this, porting and deploying a working ROS1 driver provided valuable learning experiences in debugging sensor interfaces, managing I2C communication on Jetson devices, and adapting open-source code to new platforms.

10.8. Conclusion & Future work

10.8.1. Summary of Key Achievements

The Sensors and Perception subteam successfully implemented and tested a modular sensor suite for environmental awareness and mapping. The key accomplishments include:

- All major sensors, including LiDAR, cameras, GNSS, and IMU, were integrated into ROS2-compatible pipelines.
- YOLO object detection was trained and deployed for stop sign recognition, achieving high detection accuracy in real-time ROS2 nodes.
- Stereo vision depth estimation was prototyped using ROS2 stereo_image_proc, demonstrating basic obstacle depth estimation between 1.5 and 4 meters.
- GNSS functionality was validated through extensive outdoor testing and was found to be highly accurate with RTK correction applied.
- A custom ROS2 image publishing system was developed to synchronize stereo camera feeds with corresponding calibration data.

- All sensor outputs were published to standardized ROS2 topics for easy integration with other subsystems including Navigation and Autonomous Control.
- Successfully integrated and tested the VLP-16 LiDAR and MPU9250 IMU, enabling point cloud generation and inertial measurements on embedded Jetson platforms.
- Configured and deployed Point-LIO on Jetson Xavier NX, manually tuning IMU parameters and ensuring proper ROS1 topic publishing for odometry estimation.
- Ported a ROS2 IMU driver to ROS1 for compatibility with Point-LIO and confirmed operational status through direct RViz visualization and rosbag capture.
- Resolved networking and transform challenges related to LiDAR driver configuration and ROS2 costmap visibility, including static TF frame setup.
- Tested and validated sensor operation on a mobile, developer-friendly setup, including personal laptops and lapdock displays, increasing development flexibility.
- Captured rosbags for regression testing and future validation, providing future developers with valuable baseline data and recording workflows.

These results demonstrate a functional perception system capable of providing real-time environmental information for use in autonomous decision-making.

10.8.2. Lessons Learned

GNSS

The GNSS is accurate to know where the car is positioned when in open sightlines to the sky. If obstructed, the IMU will try its best to determine where we are. RTK corrections do prevent drift while over a stable internet connection. The GNSS successfully publishes fixes to the correct topic and subscribers can see updates when subscribed to the topic as can be noted from the rosbag recording from figure 70.

Cameras

Through the development and testing process, several important lessons were learned regarding the use of cameras for perception in an autonomous vehicle context.

Object detection using a single camera and the YOLO model proved to be both effective and accessible. Under proper lighting conditions and with standardized objects such as stop signs, YOLO consistently delivered accurate detections with low latency. However, this approach is highly dependent on environmental conditions, and detection accuracy can degrade when objects are partially occluded, in motion, or under poor lighting.

Distance and direction estimation using YOLO bounding box geometry was an efficient method for estimating object position without requiring additional hardware. That said, it only works

reliably for objects of known and consistent size. Estimating the depth of objects like people, vehicles, or other irregular shapes was not dependable using this method alone.

For stereo vision, we learned that disparity-based depth estimation is highly sensitive to both hardware setup and environmental variables. A large baseline can improve depth resolution in the center of the image, but it reduces overlap near the edges, which negatively affects matching and depth consistency. Accurate camera calibration is critical, and any small error can introduce significant noise into the disparity map. Additionally, real-time performance was limited due to computational demands, especially when running stereo and object detection simultaneously on the same device.

Finally, using `stereo_image_proc` in ROS2 provided a straightforward way to test stereo vision pipelines, but further optimization and filtering would be needed before it could be reliably used in real-world navigation tasks. Alternatively, to reduce system load and latency, custom code should be written to take the video feeds directly and produce the disparity.

LiDAR

This project underscored the critical importance of having accessible, reliable hardware during development. Early delays were often caused by shared resource bottlenecks, such as limited access to test sensors or display peripherals. To address this, a lapdock display was bought personally by one of our members, various connection cables (DisplayPort to HDMI, Ethernet), along with a Unitree L2 LiDAR and MPU9250 IMU module. This allowed us to independently test, develop, and troubleshoot complex issues - especially those related to Point-LIO - without relying on shared lab resources or team availability.

From this experience, it became evident that development pace and quality are significantly improved when developers have consistent access to fully functional hardware setups. Small conveniences, like having a laptop with a native Ethernet port, also proved invaluable when configuring and connecting to the LiDAR over a static IP. These individual preparations saved countless hours during both development and testing phases.

10.8.3. Recommendations for Future Development

GNSS

Future improvements for the GNSS include sensor fusion to add another IMU and fuse it with the GNSS to ensure that the GNSS properly knows where the car is and also to not have reliance with the IMU on the GNSS as it only works from having sightlines to the sky for satellite fixes. A second future improvement is to have a proprietary RTK correction unit to ensure minimal drift and no internet connection required. A third improvement is to permanently set the GNSS to UBX mode as after a day or so, it reverts back to NMEA mode; to obtain velocity while also obtaining position fixes as well as acceleration and orientation if the IMU on the GNSS is present

or not broken. A fourth improvement would be to separate the GNSS from the Jetson Orin as computing power may be an issue putting cameras, NAV2, and the GNSS together accumulating extra latency over time; potentially to a separate Raspberry Pi. A final improvement would be to acquire a better adapter to the GNSS board from the antenna as the current adapter is extremely volatile and prone to disconnecting with even the slightest of movements.

Cameras

For future improvements to the camera system, several areas should be addressed to enhance performance and integration. The stereo camera setup would benefit from improved calibration techniques and more rigid mounting to ensure stable alignment during operation. Reducing the baseline distance from the current 54.5 centimeters to a more compact configuration, such as 20–30 centimeters, could improve image overlap and depth consistency across the entire frame. The use of dedicated stereo or depth cameras, like Intel RealSense or ZED, is recommended to offload disparity processing and deliver more reliable depth output without burdening the main processor. To reduce computational strain on the Jetson Orin, future implementations should consider offloading image processing to a secondary compute module, such as a Jetson Nano or Raspberry Pi. The YOLO object detection model should be retrained with a broader dataset to include a wider range of road signs, obstacles, and pedestrian classes to improve generalization in diverse environments. Additionally, integrating sensor fusion between camera detections and LiDAR or IMU data could provide more robust obstacle localization and compensate for the limitations of camera-based depth estimation. Finally, optimizing the ROS2 image pipeline through intra-process communication or nodelet-style design could help reduce latency and improve efficiency when dealing with high-resolution image streams in real time.

LiDAR and IMU

Future team members should explore options for funding their own sensor kits or be prepared to personally invest in basic hardware. Having a dedicated portable display and Ubuntu laptop with an Ethernet port enabled rapid iteration, especially when other equipment was unavailable or used by other team members. Also, it is advisable to acquire funding next year for a better quality IMU. The MPU9250 was useful for testing Point-LIO but was sensitive to noise. Future iterations should budget for higher-quality IMUs with better calibration and lower drift to improve LIO performance.

10.9. Reflections

10.9.1. Original Project Proposal & Changes Over Time

For the GNSS, there was one change from the beginning and that was to attempt to acquire IMU data from the IMU on the receiver for odometry purposes.

However, the GNSS raw data was returning all zeroes and therefore, the IMU is either broken, or one does not exist on the receiver we had.

The team also initially took on the radar system as part of the perception stack, but this responsibility was later handed over to the Autonomous Control subteam due to scope and bandwidth constraints. This allowed the Sensors and Perception team to focus more deeply on configuring and validating the existing core sensors.

For LiDAR, the original goal was to implement a full LiDAR-inertial odometry (LIO) pipeline using Point-LIO. While the team succeeded in configuring Point-LIO in ROS1 with basic IMU and LiDAR input, full integration with ROS2 and sensor fusion across subsystems proved more complex than anticipated. The LiDAR was also intended to support real-time mapping and costmap generation, which was partially achieved through RViz visualization and point cloud publishing. However, due to time constraints and challenges with reliable odometry, full navigation-layer integration using the LiDAR data was not completed.

The initial plan for the camera system included object detection, traffic sign recognition, and stereo depth estimation. These goals remained consistent throughout the term. YOLO object detection was implemented and tested as expected, while stereo vision proved more difficult to calibrate and integrate. Although stereo depth estimation was operational using `stereo_image_proc`, the expected accuracy and real-time performance were not achieved, leading to its use primarily as a research and validation tool rather than a live obstacle detection method.

Overall, while the core perception objectives were preserved, the final deliverables were adapted to focus on reliable sensor integration, modularity, and system stability. These changes allowed the team to produce a robust perception baseline that future teams can build upon with higher fidelity sensor fusion, tighter subsystem integration, and more advanced real-time capabilities.

10.9.2. Team Reflection on Challenges & Successes

Throughout the project, the team encountered a wide range of technical and logistical challenges across all sensor systems. A significant hurdle was working with unfamiliar hardware and software platforms, particularly when configuring the GNSS module, bridging ROS1 and ROS2 environments, and calibrating stereo cameras. For the GNSS, recurring issues included the device reverting from UBX to NMEA mode, unreliable antenna adapters, and difficulty maintaining consistent RTK correction without a stable internet connection. Despite these setbacks, the GNSS pipeline was successfully validated through large-scale testing around the

Carleton University campus, achieving approximately 99 percent accuracy under open-sky conditions. The team also demonstrated that the GNSS was ROS2-compatible and could publish reliable data at a usable frequency under most conditions.

In parallel, significant work was done on integrating the LiDAR sensor into the perception stack. The team configured the sensor to publish real-time point cloud data, performed static and dynamic testing to visualize environment structure in RViz, and successfully connected it to the ROS1-based Point-LIO package for LiDAR-inertial odometry. Although full integration with ROS2 and sensor fusion with other systems was not achieved this semester, the team established a working baseline that enabled LiDAR-based pose estimation in ROS1. Configuring Point-LIO involved complex calibration of extrinsics, managing ROS1 dependencies, and tuning parameters for motion tracking—all of which required deep experimentation and troubleshooting.

The camera subsystem also progressed steadily. The team implemented real-time YOLO object detection, developed custom code to synchronize and publish stereo camera feeds with calibration data, and tested depth estimation using both disparity maps and bounding box geometry. Stereo vision presented notable challenges in calibration and edge accuracy, but provided useful depth information in controlled tests.

Many team members began the project with limited experience in ROS, sensor drivers, and perception pipelines. This resulted in a steep learning curve, especially when debugging across hardware and software layers. However, strong collaboration, resourcefulness, and frequent knowledge sharing allowed the team to overcome these hurdles. Even when full cross-subsystem integration was delayed, a modular design philosophy allowed each sensor pipeline to be developed and tested independently. These combined efforts resulted in a flexible and well-documented perception stack that lays a strong foundation for future teams to build upon and extend.

10.9.3. Hindsight Analysis: What would we do Differently?

It is difficult to conclude what our team should have done differently, our team was inexperienced in the majority of the sensors that were used and were inexperienced with ROS2. We did not know how to communicate with the sensors at the beginning either. We did not know about transform frames. Odometry was a topic that we did not know why it was needed and did not think about setting up an odometry source. In regards to the LIDAR, we did not know

that LIO existed and did not know what costmaps were or how to generate one in ROS2. For the GNSS, we did not know how to properly configure modes to display acceleration and orientation and did not set up a second IMU as we thought the GNSS had one but it was either broken or one did not exist at all. We also did not know how to set up a ROS driver for a second IMU or how to even calibrate one.

We should also have searched GitHub more thoroughly for packages that were relevant to our scope of the project and tested the packages that were found. In addition, we should have begun integration testing much sooner. In this case, these difficulties may have been rectified much sooner.

11.Simulation

11.1. Introduction

11.1.1. Background & Motivation

The objective of the simulation subsystem is to provide testing and validation of the other subsystems needed for the autonomous vehicle project. The simulation subsystem can provide testing safely and reliably without needing the physical vehicle to perform these tests.

11.1.2. Problem Statement

Autonomous systems face financial, safety, and physical constraints when testing autonomous control [23]. Developing autonomous vehicles requires integrating many separate subsystems. Each subsystem itself is validated before integration. Simulation provides an environment for testing and validating autonomous systems without endangering the public or the workers [23].

11.1.3. Objective & Scope

The key goal is to use simulation to test and validate the other subsystems needed for the autonomous vehicle. Another goal for the simulation subsystem is to model the simulation environment to be similar to the physical world to ensure tests are useful.

11.1.4. Methodology Overview

CARLA will be used to create and test scenarios to ensure autonomous control behaves as expected. CARLA allows for vehicle physics properties such as tire friction, max steering angle, mass, and drag to be altered at runtime through its Python API. This ensures that the vehicle dynamics of the simulated vehicle match the physical vehicle. CARLA's existing sensors will be modified, or custom sensors will be created to model the sensors used on the physical vehicle. CARLA provides a variety of maps and allows obstacles to be spawned into the environment,

such as stationary vehicles and roadblocks. To simulate the environment in which the physical vehicle will be tested, a map generated with OpenStreetMap will be exported to CARLA.

The ROS bridge will facilitate two-way communication between CARLA and the other subsystems. For control components tests, the simulation will publish control commands to a topic that the physical vehicle is listening to, this should cause the physical vehicle to mimic the simulated vehicle. The simulation will publish status data such as current position, speed, and vehicle orientation for tests that monitor vehicle status, such as remote monitoring and Map Display. The simulation will publish simulated LiDAR, camera, RADAR, and GNSS data for navigation tests. The simulation will listen for control commands to navigate to a destination. To test the emergency stop, the simulation will listen for an emergency stop command and stop the simulated vehicle.

11.1.5. Report Overview

11.2. Engineering Requirements & Justification

11.2.1. Justification for Relevance to Degree Program

The relation to degree aspect of this subgroup is Software Testing and Validation. The simulation subgroup provides testing and validation for the autonomous vehicle. This requires creating a robust set of tests that ensures the autonomous control behaves as intended.

11.2.2. Engineering Principles Applied

One key engineering concept is testing and validation. Testing and validation ensure the system behaves as intended. The simulation system will need to test, for example, that the control system works as intended so the vehicle moves in the expected way. Another engineering concept is modularity, meaning that a system is separate and independent of other systems. This allows for isolated tests of each subsystem before integration tests.

11.2.3. Health and Safety Consideration

Most of the work done in the simulation subsystem was on a computer that does not carry much, if any, risk. Most of the risk came from sending movement commands to the physical vehicle. In this case, the vehicle was up on jacks, and someone stood next to the emergency stop button on the vehicle, which cut power to the vehicle control if any issues were to arise.

11.2.4. Ethical Considerations

There are no major ethical concerns about the simulation subsystem. All software used by the simulation system is free and open source except for RoadRunner, which requires a license provided under Carleton's extended Matlab license.

11.2.5. Regulatory & Standards Compliance

11.3. Literature Review & Related work

11.3.1. Overview of Existing Technologies & Techniques

CARLA is a free open-source simulator for autonomous vehicle research and testing [23]. CARLA provides a ROS-bridge allowing for integration with ROS. CARLA allows for the configuration of sensors such as LiDAR, cameras, GPS, and depth sensors [24]. CARLA allows map generation with OpenStreetMap, along with preconfigured maps and a map builder [24].

Alternative simulators used for autonomous vehicle research include Gazebo, AirSim, SVL, and TORCS. SVL and AirSim are no longer under active development, making them less desirable. Gazebo provides many tools for simulation as well as ROS integration, but requires a lot of configuration to set up an urban setting to test autonomous vehicles [25]. TORCS provides autonomous testing in race tracks, making it limited for autonomous control testing in dense urban environments [26].

11.3.2. Comparison with Previous Work

CARLA was chosen as the best overall simulator for its ROS integration, extensive support for simulating autonomous vehicles in urban environments, and the previous year's team success with simulating using CARLA.

11.4. System Design & Implementation

11.4.1. System Overview & Architecture

A CARLA server generates a simulation that includes a virtual vehicle with simulated sensors in a simulated environment. The ROS-bridge is a client to the CARLA server and converts the simulated data, such as the vehicle's speed, sensors, and position, into a ROS topic that can be published or subscribed to. As seen in Figure 12.1, different subsystems interact with the simulation; arrows leaving the Simulation system indicate that a subsystem subscribes to a topic via ROS. Autonomous Control subscribes to a topic that publishes Ackermann messages that control the physical vehicle. Incoming arrows to the simulation system indicate that a system publishes to a topic. Emergency stop is an example of a system that publishes an Ackermann

command to cause the vehicle to brake when publishing that it is active. Docker is used to simplify startup and ensure all dependencies are installed correctly.

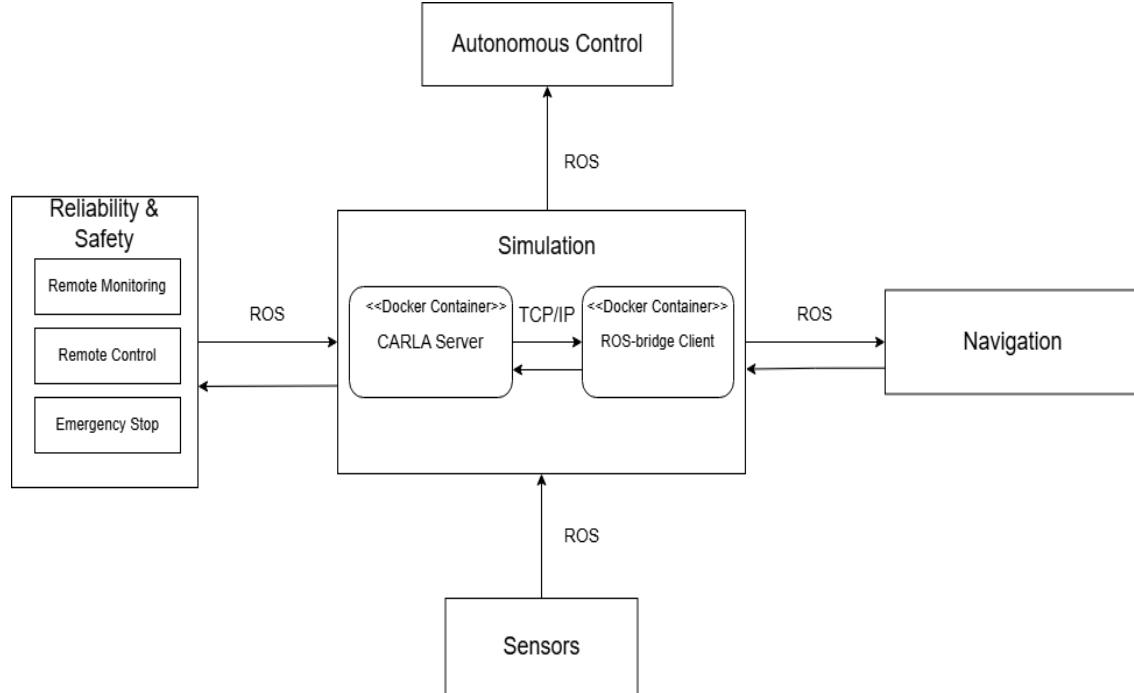


Figure 106: Overview of Simulation Subsystems and interactions with other subsystems.

11.4.2. Requirements Definition

Functional Requirements

Model Vehicle Dynamics: The system should model the dynamics of the ECOLO vehicle. The model shall include the vehicle's mass, tires, center of gravity, size, and brakes.

Simulate Sensors: The system shall simulate the sensors used on the autonomous vehicle.

ROS Communication: The system must use ROS for communication between other subsystems.

Non-Functional Requirements

Maintainability: The system should be maintainable for future students and allow extension of features.

Documentation: The system shall provide extensive documentation for successive students. The documentation must explain how to run the simulation subsystem, show the architecture used, and describe setting up the simulation subsystem on a new device.

Portability: The system must be available to run on different devices.

11.4.3. Hardware Design

The hardware used for the Simulation system is a desktop computer that is powerful enough to run CARLA and publish many ROS topics. Other related hardware components are Raspberry PI's, ESP32, and Jetsons, which are a part of the other subsystems that communicate with the Simulation over ROS. These hardware components are connected through Ethernet to the same network, allowing for ROS to discover all the nodes that are being published or subscribed to by these systems.

11.4.4. Software Design

Docker compose was used to orchestrate the CARLA server container and the ROS-bridge container. The CARLA simulation environment provides the core vehicle physics and sensor simulation capabilities. The ROS bridge container contained the CARLA Python API as well as ROS2, which are needed for the Simulation. Docker also allowed the simulation to be portable to any other computer without needing to keep track of all the dependencies needed to run the system.

The CARLA Python API allows a client to connect to the simulation server and interact directly with the simulation. This was used to change the vehicle physics, such as limiting the turning angle, setting the mass of the vehicle, and limiting acceleration. This was also used to try and import an OpenStreetMap of Carleton University into CARLA. The Python API also allowed for useful functions, such as a manual control mode that allowed steering with the keyboard.

The ROS client library was used to create nodes that subscribe and publish to topics, which tested various systems. An example is an Ackermann control node; this node subscribed to a topic that provided current vehicle movement information and published Ackermann messages. These messages were subscribed to by the physical vehicle, ensuring that it mirrored the simulated vehicle's behavior.

11.4.5. Integration & Testing Approach

ROS was the communication tool used to integrate the simulation software with the hardware used in other subsystems. ROS acted as a common language for the different hardware (microcontrollers, Raspberry Pi) used in the Autonomous Vehicle. Simulation information would be converted to ROS topics that other hardware in the network could publish and subscribe to, allowing for relatively easy integration of these systems.

11.4.6. Challenges & Troubleshooting

The first issue faced when developing the simulation subsystem was setting up the ROS bridge. The bridge would successfully start up and launch, but certain topics that should be published by the ROS bridge would be missing. Manual control mode, which allows for controlling the vehicle with the keyboard, failed to launch. To fix the issue of the ROS bridge not working as intended was ensure the dependencies needed for manual control were installed, and source the ROS workspace inside the ROS-bridge docker container.

11.5. Project Management & Execution

11.5.1. Work Breakdown Structure

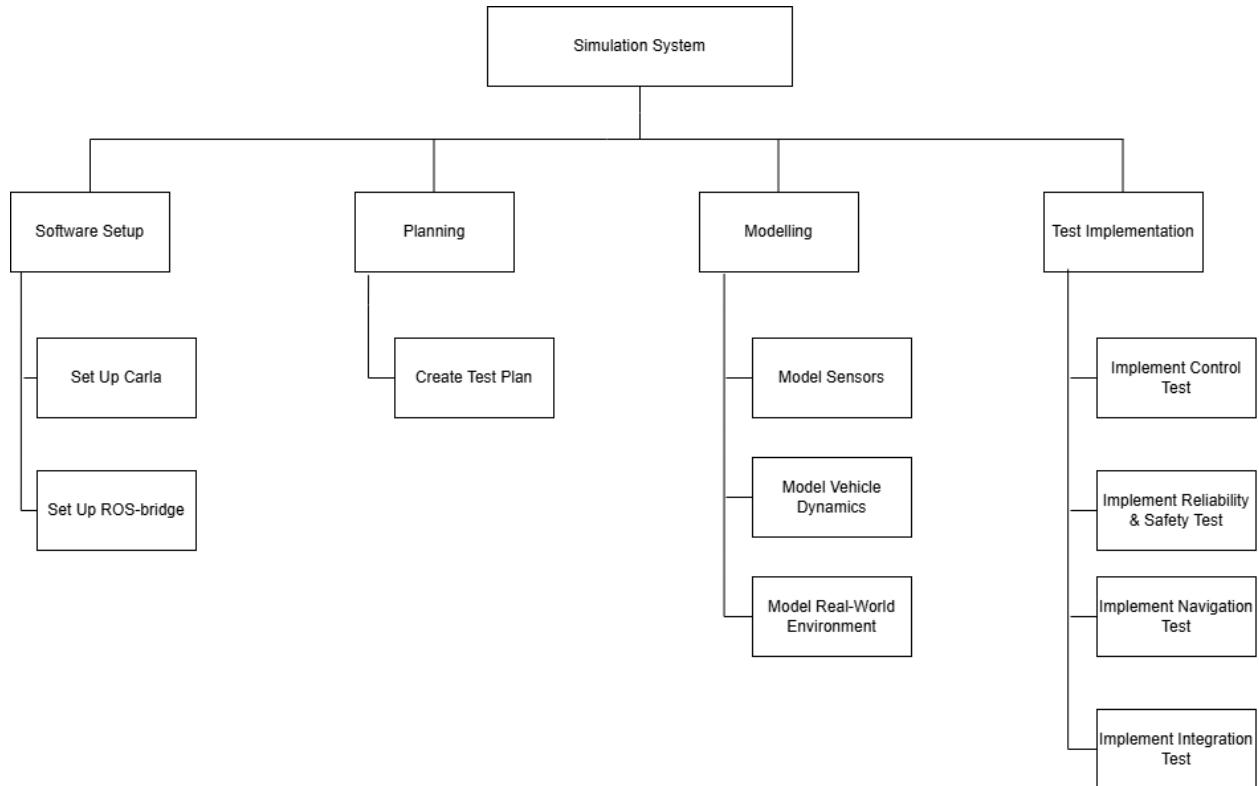


Figure 107: Work Breakdown Structure of the Simulation Subgroup.

11.5.2. Gantt Chart & Milestone Tracking

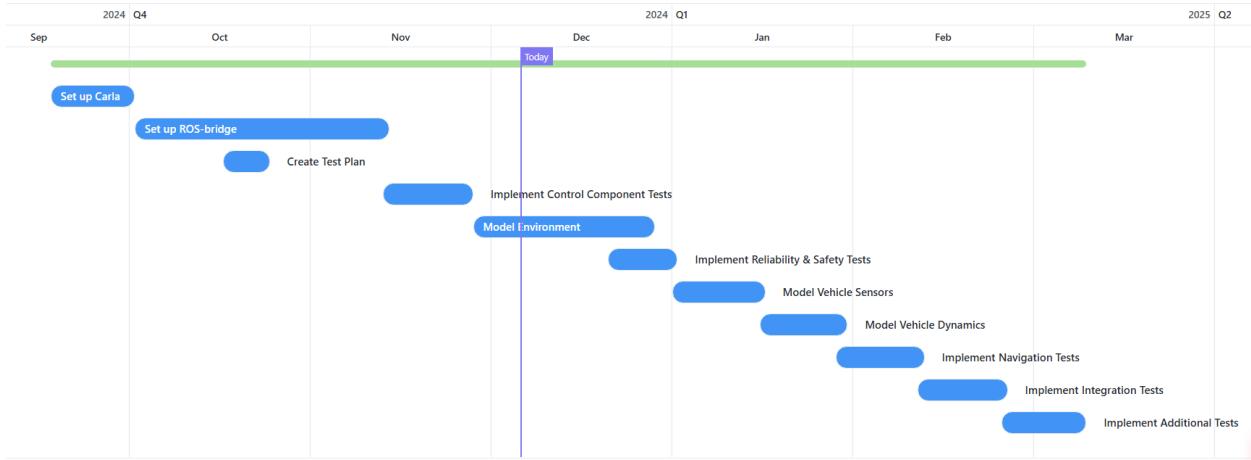


Figure 108: Gantt Chart of the Simulation Subgroup.

11.5.3. Risk Analysis & Mitigation Strategies

Risk	Probability	Severity	Risk Level	Mitigation
Simulation is inaccurate/unreliable	Moderate	High	High	Vehicle dynamics are being modeled in CARLA. Sensors are configured to be the same/similar to those used on the physical vehicle.
Software Incompatibility	Low	Low	Low	Docker is being used to manage software dependencies and ensure compatible software versions are running.
Scope Creep	Moderate	Low	Low	Tests outlined in the test plan will

				be prioritized over additional tests.
Hardware Performance Limitations	Very Low	Moderate	Low	Testing is done on a powerful desktop computer.

Table 11.4: Risk Assessment and Management Chart

11.5.4. Budget & Cost Analysis

No cost was incurred for developing the simulation subsystem. A computer was provided by the supervisors, and the software used to develop simulations was free. RoadRunner is the only software used that requires a license, which was part of the university's license to Matlab that required special permission.

11.6. Experimental Setup & Testing

11.6.1. Testing Environment & Procedures

Autonomous Control Test: The physical vehicle was on jacks, subscribing to a topic, which publishes CARLA's simulated vehicle's current motion as an Ackermann command.

Remote monitoring Test: The remote monitoring system on the physical vehicle subscribes to multiple topics that publish data from the simulated vehicle, such as speed, GNSS, and camera.

Remote Control Test: The remote control publishes Ackermann messages based on joystick position to a topic that the simulated vehicle subscribes to.

Emergency Stop Test: The emergency stop system publishes a Boolean message where true or false indicates if the emergency stop system is engaged or disengaged.

11.6.2. Experimental Results and Findings

Autonomous Control Test: The autonomous control system was able to receive Ackermann control messages from the CARLA simulated vehicle. The physical vehicle steered, accelerated, and braked when the simulated vehicle had done so. There was a 3-second delay between the simulated vehicle's control command and the physical vehicle's mimicked movement.

Remote Monitoring Test: The remote monitoring system displayed the simulated vehicle data on a web app with a 0.5-second delay between current simulation values and the values hosted on the remote monitoring system.

Emergency Control Test: The remote control system was able to control the simulated vehicle's steering and throttle with a 0.5-second delay.

Emergency Stop Test: The emergency stop system was able to activate and deactivate the brakes on the simulated vehicle with a 0.5-second delay.

11.7. Discussion & Analysis

The behavior of the subsystem tests functioned as expected, but the delay between the simulation and the physical subsystems was greater than expected. The Ethernet connection between the simulation and the subsystems gave an expectation that the delay between them would be very small. The delay could be the result of many topics from the Simulation subsystem publishing and subscribing data simultaneously. The Autonomous control system had a much longer delay than the other systems, potentially indicating a hardware processing delay in that specific system. The results of testing show that each subsystem can be integrated with the CARLA and provide some validation for each subsystem. But the delay between the simulation and the physical systems could impact further testing that requires more responsive feedback, such as an emergency stop triggered by radar to prevent hitting a pedestrian or a wall.

11.8. Conclusion & Future work

11.8.1. Summary of Key Achievements

Automated the setup of the simulation system using Docker.

Developed documentation on how to install and use the simulation system.

Modelled the physical sensors on the simulated vehicle.

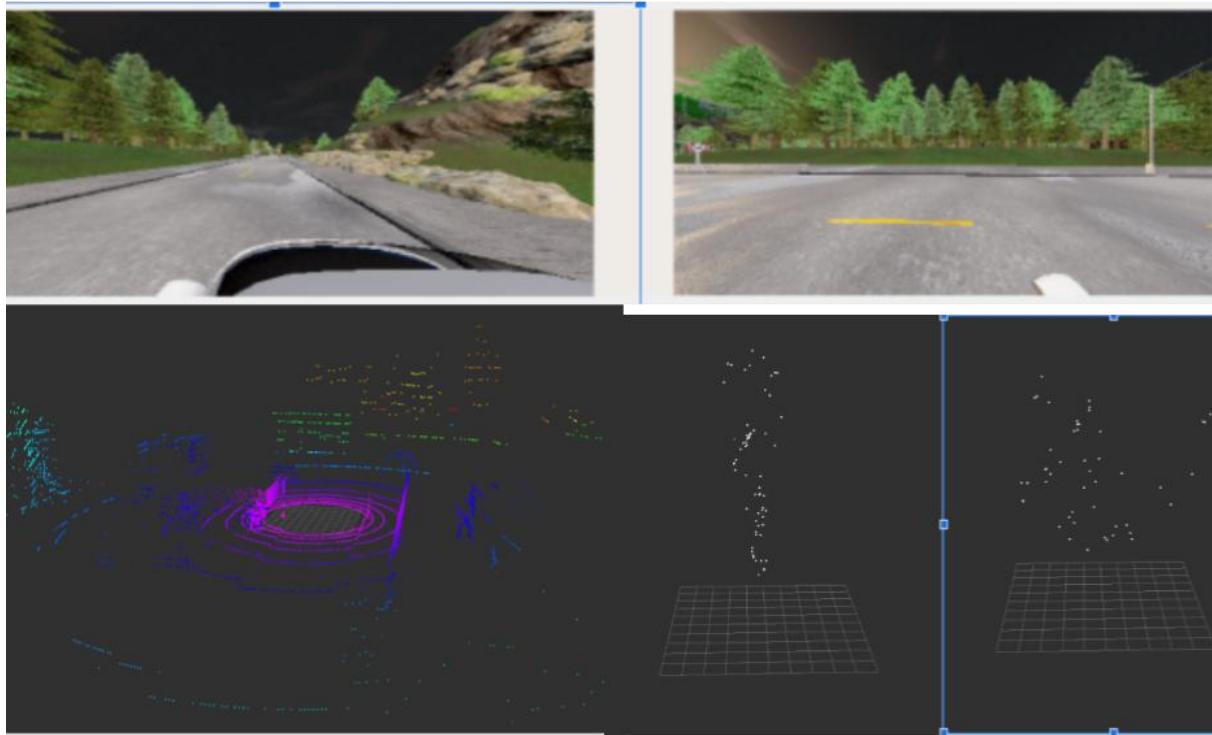


Figure 109: Top portion of shows placement of front left and side left cameras. Left bottom portion image displays the Lidar costmap. Right bottom shows long (right) and short range (left) radar.

Modelled the vehicle dynamics using CARLA's Python API.

Developed a custom MAP using OpenStreetMap and RoadRunner

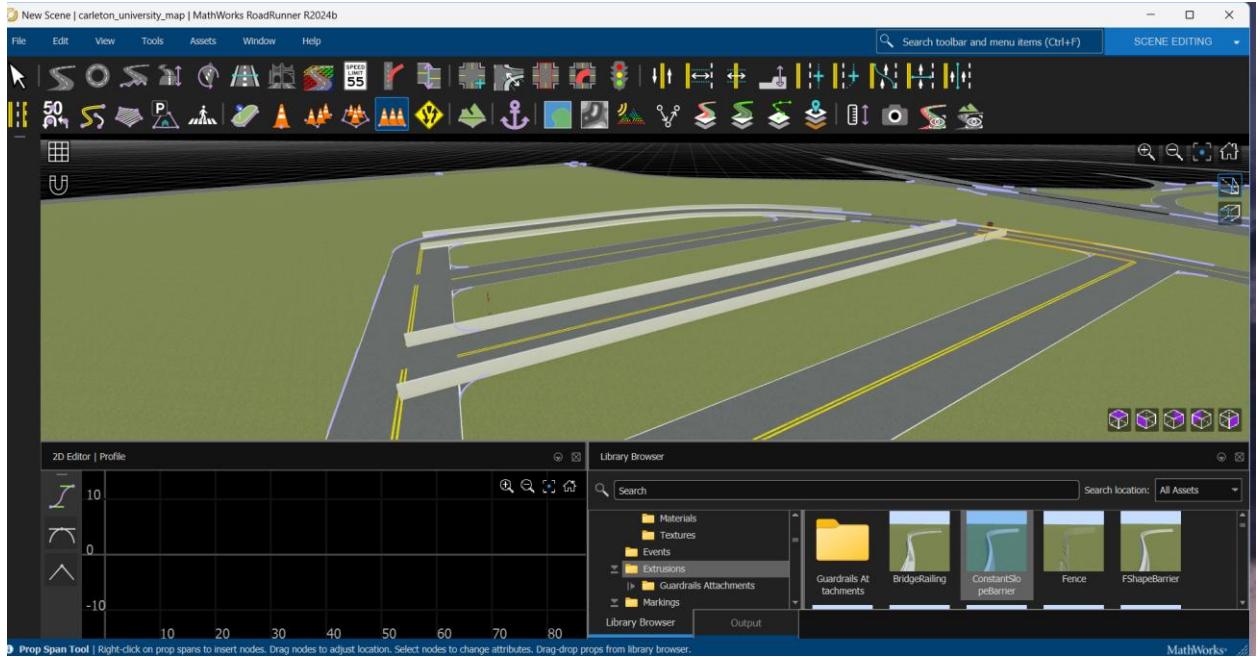


Figure 110: Model of P7 Parking Lot at Carleton University using RoadRunner

Developed and implemented unit tests for autonomous control, remote monitoring, emergency stop, and remote control.

11.8.2. Lessons Learned

Integration is the most difficult process when building a complex system. Even if each subsystem works individually, issues can and will arise when bringing systems together.

11.8.3. Recommendations for Future Development

Integrate with the Navigation subsystem and implement navigation tests using the Simulation subsystem. Implement integration tests between multiple subsystems. Another recommendation is to build CARLA from source in Docker. This will make importing custom maps possible, needing to build a 700GB Docker image for the Unreal Engine editor to add a custom map to CARLA. The last recommendation I have is to build a custom model of the ECOLO vehicle for more accurate size and sensor position. In the current setup, the Simulation system uses the Microlino vehicle, which is a model vehicle that is available in CARLA. While the dimensions are pretty similar to the physical vehicle, the microlino is wider and taller, which required the modelled sensors to be attached in locations that are not exactly where they sit on the physical vehicle.

11.9. Reflections

11.9.1. Original Project Proposal & Changes Over Time

The original proposal contained tests for the navigation subsystem. These were unable to be fulfilled as integrating Nav2 with CARLA had issues. This issue was that Nav2 used a 2-D LaserScan for navigation while CARLA's lidar output a 3D costmap. The progress report proposed Sensor tests that would test the integration of the physical sensors by recording data using ROSbag and letting the nav2 control the simulated vehicle based on the physical sensors. This was consequently not completed due to issues integrating the Navigation system with CARLA.

11.9.2. Team Reflection on Challenges & Successes

The main challenges were integrating various systems and trying to gain an understanding of how the other systems work. Another challenge was getting CARLA and the ROS bridge setup. A lot of time was spent reading through documentation trying to figure out what certain features were not working. A big success is the simulation was able to perform some tests such as autonomous control, remote monitoring, and emergency stop validate that these systems work before the physical test.

11.9.3. Hindsight Analysis: What would we do Differently?

In hindsight, we should have tried integrating navigation and simulation systems as soon as possible. This would have allowed a base minimum of the autonomous control to be tested and integrated with the other subsystems.

12. References

- [1] S. E. Shladover, "Connected and automated vehicle systems: Introduction and overview," *J. Intell. Transp. Syst.*, vol. 22, no. 3, pp. 190–200, 2018, doi: 10.1080/15472450.2017.1336053.
- [2] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous Vehicle Technology: A Guide for Policymakers*, Santa Monica, CA: Rand Corporation, 2016, doi: 10.7249/RR443-2.
- [3] K. Saleh, M. Hammad, and S. Hassan, "Real-time monitoring for intelligent transportation systems: A comprehensive review," *IEEE Access*, vol. 8, pp. 223868–223880, 2020, doi: 10.1109/ACCESS.2020.3043564.
- [4] T. Litman, *Autonomous Vehicle Implementation Predictions: Implications for Transport Planning*, Victoria Transport Policy Institute, 2020. [Online]. Available: <https://www.vtpi.org/avip.pdf>.
- [5] J. Jeong, J. H. Lee, and M. H. Lee, "Wireless emergency stop system for a vehicle robot with secure communication," *Int. J. Control Autom. Syst.*, vol. 17, no. 6, pp. 1570–1577, 2019, doi: 10.1007/s12555-018-0860-1.
- [6] J. Wu et al., "Remote monitoring system for autonomous vehicle health based on cloud computing and IoT," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6910, Aug. 2020, doi: 10.1109/JIOT.2020.2981954.
- [7] M. A. Anwar, H. Kazi, and R. Miura, "Human-in-the-loop control in semi-autonomous driving: A review," *IEEE Access*, vol. 7, pp. 27794–27807, 2019, doi: 10.1109/ACCESS.2019.2900419.
- [8] Parts Not Included, "How to use an RC controller with an Arduino," *Parts Not Included*, Feb. 21, 2019. [Online]. Available: <https://www.partsnotincluded.com/how-to-use-an-rc-controller-with-an-arduino/>. [Accessed: Oct. 17, 2024].
- [9] A. David and P. Falcone, "ROS-based real-time system integration for autonomous vehicles," in *Proc. IEEE Int. Conf. Intelligent Transportation Systems (ITSC)*, 2018, pp. 2254–2259, doi: 10.1109/ITSC.2018.8569307.
- [10] Carleton University, "Brightspace Learning Management System," *Carleton University Brightspace*. [Online]. Available: <https://brightspace.carleton.ca/>. Accessed: Oct. 13, 2024.
- [11] C. P. Vyasarayani and M. G. Safar, "Control Systems Theory and Design," *Course Lecture Notes*, SYSC 3600, Carleton University, 2023.
- [12] R. Smith and J. Doe, "Systems Engineering in Autonomous Vehicles," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 1234-1248, Jul. 2019.
- [13] Carleton University, "Capstone Design Projects: SYSC 4805," *Department of Systems and Computer Engineering*, 2024. [Online]. Available: <https://carleton.ca/sce/capstone-sysc4805/>.
- [14] J. Brown and K. Johnson, "Risk Assessment in Autonomous Vehicles," *IEEE Access*, vol. 8, pp. 18306-18317, Jan. 2020.
- [15] S. Adams and H. Lee, "Real-Time Data Processing for Autonomous Systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8927-8936, Sept. 2019.
- [16] R. K. Rajamani, *Vehicle Dynamics and Control*, 2nd ed. Springer, 2012..
- [17] M. Fowler, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.
- [18] A. Singh, "Heartbeats Detection: A Solution to Network Failures in Distributed Systems," GeeksforGeeks, Apr. 23, 2020. [Online]. Available: <https://www.geeksforgeeks.org/heartbeats-detection-a-solution-to-network-failures-in-distributed-systems/>.

- [19] Zed-F9R-01B Data Sheet, https://cdn.sparkfun.com/assets/d/4/3/f/4/ZED-F9R-01B_Datasheet_UBX-19054459.pdf (accessed Apr. 9, 2025).
- [20] “Ros 2 documentation,” ROS 2 Documentation - ROS 2 Documentation: Humble documentation, <https://docs.ros.org/en/humble/index.html> (accessed Apr. 8, 2025).
- [21] Ultralytics YOLO11. [Online]. Available: <https://docs.ultralytics.com/> [Accessed: Apr. 7, 2025].
- [22] Formats, YOLOv11 PyTorch TXT. [Online]. Available: <https://roboflow.com/formats/yolov11-pytorch-txt> [Accessed: Apr. 7, 2025].
- [23] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. Proceedings of the 1st Annual Conference on Robot Learning.
- [24] C. Team, “CARLA,” *CARLA Simulator*. <https://carla.org/>
- [25] “Gazebo Community,” Gazebo Community, 2025. <https://community.gazebosim.org/> (accessed Apr. 09, 2025).
- [26] “SourceForge,” SourceForge, Oct. 27, 2024. <https://sourceforge.net/projects/torcs/>
- [NAV1].** T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," Proceedings of the IEEE, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [NAV2].** R. Pachamuthu, "Map-based navigation for autonomous vehicles," *Autonomous Vehicle International*, Sep. 11, 2023. [Online]. Available: <https://www.autonomousvehicleinternational.com/features/feature-map-based-navigation-for-autonomous-vehicles.html>
- [NAV3].** M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Bt. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct. 28, 2016, doi: 10.1186/s40064-016-3573-7.
- [NAV4].** G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," in *2017 IEEE Region 10 Symposium (TENSYMP)*, Cochin, India, 2017, pp. 1-6, doi: 10.1109/TENCONSpring.2017.8069972.
- [NAV5].** H. M. Haris and J. Hou, "Obstacle detection and safely navigate the autonomous vehicle from unexpected obstacles on the driving lane," *Sensors*, vol. 20, no. 17, p. 4719, 2020, doi: 10.3390/s20174719.
- [NAV6].** C. Dvonch and J. Nazemi, "Autonomous vehicle navigation," U.S. Patent US20180364730A1, Dec. 20, 2018.
- [NAV7].** A. Shashua and Y. Gdalyahu, "Sparse map for autonomous vehicle navigation," European Patent EP 4 180 768 A1, May 17, 2023.
- [NAV8].** S. Li, S. Wang, Y. Zhou, Z. Shen, and X. Li, "Tightly coupled integration of GNSS, INS, and LiDAR for vehicle navigation in urban environments," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 24721-24735, Dec. 15, 2022, doi: 10.1109/JIOT.2022.3194544.
- [NAV9].** J. Zhu, X. Lian, and Z. Gui, "Research on global path planning system of driverless car based on improved RRT algorithm," in *2023 International Conference on Data Science & Informatics (ICDSI)*, Bhubaneswar, India, 2023, pp. 260-263, doi: 10.1109/ICDSI60108.2023.00056.
- [NAV10].** K. Iagnemma, "Route planning for an autonomous vehicle," U.S. Patent US 11,022,449 B2, Jun. 1, 2021.

- [NAV11]. K. Jo, M. Lee, W. Lim, and M. Sunwoo, "Hybrid local route generation combining perception and a precise map for autonomous cars," *IEEE Access*, vol. 7, pp. 120128-120140, 2019, doi: 10.1109/ACCESS.2019.2937555.
- [NAV12]. N. R. Beer, D. Chambers, and D. W. Paglieroni, "Object sense and avoid system for autonomous vehicles," United States, 2023. [Online]. Available: <https://www.osti.gov/biblio/2293763>.
- [NAV13]. T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [NAV14]. D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," Universität Karlsruhe (TH), Karlsruhe, Germany. [Online]. Available: <https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf>.
- [NAV15]. K. Hjerpe, J. Ruohonen and V. Leppänen, "The General Data Protection Regulation: Requirements, Architectures, and Constraints," in *Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE)*, Jeju, South Korea, 2019, pp. 265–275, doi: 10.1109/RE.2019.00036.
- [NAV16]. T. Foote, "REP 105: Coordinate Frames for Mobile Platforms," *ROS Enhancement Proposals (REPs)*, Open Source Robotics Foundation, 2010. [Online]. Available: <https://www.ros.org/reps/rep-0105.html>
- [NAV17]. Leaflet, "Leaflet: An open-source JavaScript library for interactive maps," [Online]. Available: <https://leafletjs.com/>
- [NAV18]. Flask, "Flask Documentation (Stable)," *Pallets Projects*, [Online]. Available: <https://flask.palletsprojects.com/en/stable/>. [Accessed: Apr. 7, 2025].
- [NAV19]. OSRM, "Open Source Routing Machine (OSRM)," *Project OSRM*, [Online]. Available: <https://project-osrm.org/>. [Accessed: Apr. 7, 2025].
- [NAV20]. Carleton University, "Wireless and Internet – Information Technology Services," [Online]. Available: <https://carleton.ca/its/all-services/wireless-and-internet/>. [Accessed: Apr. 7, 2025]
- [NAV21]. React, "React Documentation," [Online]. Available: <https://react.dev/>. [Accessed: Apr. 7, 2025].
- [NAV22]. Vite, "Vite Guide," [Online]. Available: <https://vite.dev/guide/>. [Accessed: Apr. 7, 2025].
- [NAV23]. Puppeteer, "Puppeteer: Headless Chrome Node.js API," [Online]. Available: <https://pptr.dev/>. [Accessed: Apr. 8, 2025].
- [NAV24] Jeff Shepard, "What is the role of sensor fusion in robotics?" , [Online], Available: <https://www.sensortips.com/featured/what-is-the-role-of-sensor-fusion-in-robotics-faq/>. [Accessed: Apr 07 2025].

13.Appendices - A

```
1 import rclpy
2 from rclpy.node import Node
3 import serial
4 from pyubx2 import UBXReader
5 from sensor_msgs.msg import NavSatFix, Imu
6 from std_msgs.msg import Float32
7 from geometry_msgs.msg import TwistStamped, Vector3Stamped
8 import numpy as np
9
10 v class GNSSIMUPublisher(Node):
11 v     def __init__(self):
12         super().__init__('gnss_imu_publisher')
13
14         # Publishers
15         self.gnss_pub = self.create_publisher(NavSatFix, 'gnss/fix', 10)
16         self.imu_pub = self.create_publisher(Imu, 'imu/data', 10)
17         self.velocity_pub = self.create_publisher(TwistStamped, 'gnss/velocity', 10)
18         self.acceleration_pub = self.create_publisher(Vector3Stamped, 'gnss/acceleration', 10)
19
20         # Serial Connection
21         self.serial_port = "/dev/ttyACM0" # Adjust as needed
22         self.baud_rate = 115200
23         self.ser = serial.Serial(self.serial_port, self.baud_rate, timeout=1)
24         self.ubr = UBXReader(self.ser, protfilter=2)
25
26         # Timer
27         self.create_timer(0.1, self.read_sensor_data) # 10 Hz
28
29 v     def read_sensor_data(self):
30     raw_data, parsed_data = self.ubr.read()
31
32     if parsed_data:
33         msg_type = parsed_data.identity
34
35         if msg_type == "NAV-PVT": # Fused GNSS Position & Velocity
36             gnss_msg = NavSatFix()
37             gnss_msg.latitude = parsed_data.lat
38             gnss_msg.longitude = parsed_data.lon
39             gnss_msg.altitude = parsed_data.hMSL / 1e3
40             self.gnss_pub.publish(gnss_msg)
41
42             # Publish velocity
43             velocity_msg = TwistStamped()
44             velocity_msg.twist.linear.x = parsed_data.velN / 1e3 # 1e3 to convert mm/s to m/s
45             velocity_msg.twist.linear.y = parsed_data.velE / 1e3
46             velocity_msg.twist.linear.z = parsed_data.velD / 1e3
```

Figure A-1: GNSS UBX mode code part 1.

```

47         self.velocity_pub.publish(velocity_msg)
48
49         self.get_logger().info(f"Published GNSS: {gnss_msg.latitude}, {gnss_msg.longitude}, {gnss_msg.altitude}")
50         self.get_logger().info(f"Published Velocity: {velocity_msg.twist.linear.x}, {velocity_msg.twist.linear.y}, {velocity_msg.twist.linear.z}")
51
52     elif msg_type == "ESF-INS":
53         x_ang_rate = np.radians(parsed_data.xAngRate)
54         y_ang_rate = np.radians(parsed_data.yAngRate)
55         z_ang_rate = np.radians(parsed_data.zAngRate)
56
57         x_accel = parsed_data.xAccel
58         y_accel = parsed_data.yAccel
59         z_accel = parsed_data.zAccel
60
61         ang_vel_msg = Imu()
62         ang_vel_msg.angular_velocity.x = x_ang_rate
63         ang_vel_msg.angular_velocity.y = y_ang_rate
64         ang_vel_msg.angular_velocity.z = z_ang_rate
65
66         self.imu_pub.publish(ang_vel_msg)
67         self.get_logger().info(f"Published Angular Velocity: {ang Vel_msg.angular_velocity.x}, {ang Vel_msg.angular_velocity.y}, {ang Vel_msg.angular_velocity.z}")
68
69         acc_msg = Vector3Stamped()
70         acc_msg.vector.x = x_accel
71         acc_msg.vector.y = y_accel
72         acc_msg.vector.z = z_accel
73
74         self.acceleration_pub.publish(acc_msg)
75         self.get_logger().info(f"Published Acceleration: {acc_msg.vector.x}, {acc_msg.vector.y}, {acc_msg.vector.z}")
76
77     def main(args=None):
78         rclpy.init(args=args)
79         node = GNSSIMUPublisher()
80         rclpy.spin(node)
81         node.destroy_node()
82         rclpy.shutdown()
83
84     if __name__ == '__main__':
85         main()

```

Figure A-2: GNSS UBX mode code part 2.

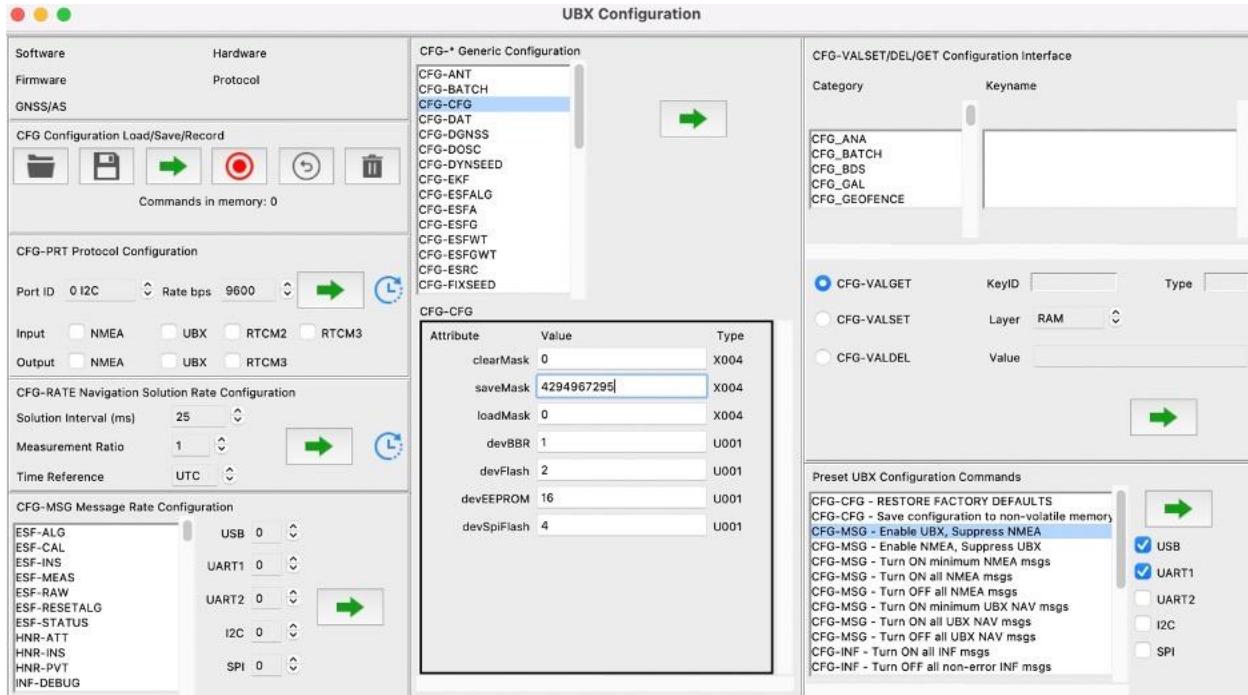
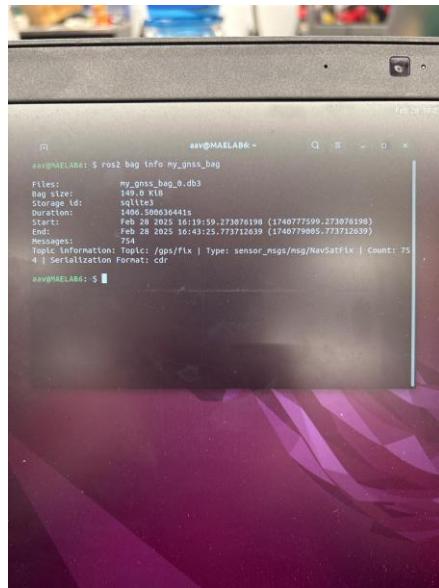


Figure A-3: PyGPSClient UBX configuration procedure used.



```
aaiv@MAELAB6: ~$ ros2 bag info my_gnss_bag
Files:          my_gnss_bag_0.db3
File size:      193.91B
Storage Id:    sqlite3
Duration:      1406.5980336415s
Start:         Feb 28 2023 10:41:59.273076198 (1740777599.273076198)
End:          Feb 28 2023 10:41:55.73712639 (1740779905.73712639)
Messages:       75
Topic Information: topic: /gps/fix | type: sensor_msgs/msg/NavSatFix | Count: 75
Serialization Format: cdr
aaiv@MAELAB6: ~$
```

Figure A-4: GNSS rosbag recording topic information

14.Appendix B - Proposal

Systems and Computer Engineering

Project Proposal

Autonomous Vehicle

SYSC 4907 A

ENGINEERING PROJECT

Professor Richard Dansereau

Professor Chao Shen

Oct 28th, 2024

CARLETON UNIVERSITY

OTTAWA, ON

Table of Contents

Table of Contents.....	1
Team Structure	4
Introduction	5
History of the Project	5
5. Power Management.....	6
5.1 Background	6
5.2 Objectives	6
5.3 Plan.....	7
5.3.1 Battery Pack	7
5.3.2 Battery Pack Monitor	8
5.3.3 Wiring Harness.....	9
5.3.4 Custom Interface PCB	10
5.3.5 Documentation / User Manual.....	11
5.4 Relation to Degree.....	11
5.5 Group Skill	11
5.6 Methods	11
5.7 Timetable.....	12
5.8 Risk Assessment.....	14
6. Navigation	14
6.1 Background	14
6.1.1 Current State of the Art	15
6.1.2 Motivation	18
6.2 Objectives	18
6.3 Plan.....	19
6.3.1 Major Tasks.....	19
6.3.1.1 UI/Front-End.....	19
6.3.1.2 Hosting Server/Backend.....	25
6.3.1.3 Algorithm	28
6.3.1.4 Route Tracking.....	31
6.3.1.5 Object Detection.....	34
6.3.2 Functional Requirements	35
6.3.3 Non-Functional Requirements.....	37
6.3.4 Software Requirements	38
6.3.5 System Design.....	39
6.3.6 Milestones	41
6.3.7 Work Breakdown Structure (WBS).....	43
6.3.8 Testing Plan.....	43
6.4 Relation to Degree.....	45
6.5 Group Skill	46

6.6 Methods	47
6.7 Timetable.....	48
6.8 Risk Assessment.....	50
7. Autonomous Control	51
7.1 Background	51
7.2 Objectives	51
7.3 Plan.....	52
7.3.1 Dynamics Controller	52
7.3.1.1 MPC	52
7.3.1.2 PID	53
7.3.1.3 Reinforcement Learning based control	53
7.3.1.4 Final Considerations.....	54
7.3.2 Braking System	54
7.3.2.1 Physical Braking System Interface.....	55
7.3.2.2 Linear Actuator Control and Feedback	55
7.3.2.3 Modes of operation and Safety.....	56
7.3.2.3 Testing Plan.....	57
7.3.3 Steering System.....	57
7.3.3.1 Linear Actuator Control and Feedback	57
7.3.3.2 Modes of operation and Safety.....	58
7.3.3.3 Testing Plan.....	58
7.3.4 Speed Sensors.....	58
7.3.4.1 Physical Design	59
7.3.4.2 Signal Processing and Speed Calculation.....	60
7.3.4.3 Speed sensor interface	60
7.3.4.4 Testing Plan.....	61
7.4 Relation to Degree.....	63
7.5 Group Skill	64
7.6 Methods	64
7.7 Timetable.....	65
7.7.1 Work Breakdown Structure	65
7.7.2 Phase Breakdown	65
7.7.3 Phase Deadlines.....	66
7.8 Risk Assessment	67
8. Reliability and Safety	68
8.1 Background	68
8.2 Objectives	68
8.2.1 Functional Requirements	68
8.2.2 Non-Functional Requirements.....	69
8.3 Plan.....	69
8.3.1 E-Stop Systems	70
8.3.2 Remote Monitoring System.....	71
8.3.3 Manual Override System.....	73
8.3.3.1 RC interface	74

8.3.3.2 GUI and joystick interface	74
8.3.4 Integration with Other Vehicle Systems.....	75
8.3.5 Heartbeats.....	76
8.4 Relation to Degree.....	76
8.5 Group Skill	78
8.6 Methods	81
8.7 Timetable.....	82
8.7.1 List of Activities	83
8.8 Risk Assessment.....	84
9. Sensors/Perception.....	85
9.1 Background	85
9.2 Objectives	86
9.2.1 Functional Requirements:.....	86
9.2.2 Non-Functional Requirements:	87
9.2.3 Measuring Progress:	87
9.3 Plan.....	87
9.4 Relation to Degree.....	89
9.5 Group Skill	90
9.6 Methods	90
9.7 Timetable.....	91
9.8 Risk Assessment.....	92
10. Simulation	92
10.1 Background	92
10.2 Objectives.....	93
10.3 Plan.....	93
Test Plan.....	94
10.4 Relation to Degree.....	96
10.5 Group Skill	96
10.6 Methods	96
10.7 Timetable.....	97
10.8 Risk Assessment.....	97
11. References	99

Team Structure

The project has been split into six subgroups. A list of the subgroups and the team leads and members is below

Power Management

Leader: Jacob Wilde #101188310

Members:

- Taran Basati #101161332

Autonomous control

Leader: Charlie Wadds # 101181414

Members:

- Mohammad Saud #101195172
- Abed Qubbaj #101205030
- Hussein Ghamlouch #101089558

Navigation

Leader: Shreeyansh Gupta #101154480

Members:

- Harsh Patel #101183596
- Karthikeyan Bhavani Shankar #101214895
- Tauheed Alamgir #101194927
- Ibrahim Faisal #101209598

Sensors

Leader: Ryan Dash #101083052

Members:

- Daniel Godfrey #101156147
- Joshua Robson # 101195802
- Alexei Fetissov #100724437

Reliability and Control

Leader: Ethan Bradley # 101158848

Members:

- Rahul Cheruku #101185639
- Ali Nadim #101192767
- Ali Zaid #101223823
- Arjun Pathak #101212581

Simulation

Leader: Abdurahman Jama #101162633

Introduction

The Autonomous vehicle project has been a multi-year project passed through different departments to transport people to different destinations without human interaction. The project was passed on to the Systems and Computers Engineering (SYSC) department this year. The project accumulated twenty-two SYSC Students and two professors to supervise the project. This proposal details the history of the project, each subgroup's objective, the problems that will be resolved, risk assessment, and the project design and plans.

History of the Project

This project has been running since 2020 with different groups and goals set by each group. It was started by Professor Xiao Huang from the Mechanical and Aerospace Engineering (MAE) department under the name Accessible Autonomous Vehicles—the project aimed to explore the development of an autonomous vehicle focusing on accessibility and autonomy. The project included 21 MAE department students working closely under Prof. Huang's supervision to lay the foundation for a comprehensive autonomous vehicle system.

In the 2022/23 academic year, Prof. Chao Shen and Prof. Richard Dansereau launched a parallel series of autonomous vehicle-related projects alongside Prof. Huang. These projects marked a collaborative and interdisciplinary approach, bringing expertise from various departments and expanding the project's vision. It included members from the Systems and Computer (SYSC) engineering department, MAE, and Department of Electrical Engineering (DOE), with 18 students from MAE, 17 from SYSC, and two from DOE.

The 2023/24 academic year saw further formalization of the project structure with Profs. Dansereau and Huang began to co-supervise various aspects of the capstone project. Simultaneously, Profs. Shen and Dansereau continued to oversee parallel autonomous vehicle projects, leading to a rich ecosystem of innovation in autonomous vehicle technology. The team comprised ten students from MAE and 20 students from SYSC.

The project leadership underwent a temporary shift in this year's project iteration. With Prof. Huang on sabbatical, Profs. Dansereau and Shen took on primary leadership roles. This year, the project involves 22 students from SYSC, whose primary focus is vehicle autonomy and integrating new technologies and features.

Some notable accomplishments include the beginning of an electric vehicle platform using an Ecolo ET4 Cruise, a 3D Lidar, mmWave radar, GNSS/GPS location, actuator-controlled steering, braking, drive, computer modules, and remote control. Despite these

accomplishments, there are still many challenges to address. Attention is still required to power management, wiring harnesses, the refinement of computer communications systems, and the enhancement of navigation and autonomous control systems. The project's primary goal is to achieve full autonomy, which requires developing systems for obstacle detection, autonomous obstacle avoidance, traffic sign recognition, and pedestrian and laneway detection. Many secondary features, such as remote monitoring, manual override controls, and emergency mechanisms, will be worked on this year or in future years.

The long-term vision for the project is ambitious. It aims to continue growing for many years, addressing complex technical challenges such as autonomous control, traffic and obstacle detection, emergency systems, and real-time communication. The project's evolving nature ensures that each new group of students will face unique challenges, contribute fresh ideas, and advance the capabilities of autonomous vehicle technology, with a clear focus on improving accessibility, safety, and autonomy in transportation systems.

5. Power Management

5.1 Background

The power management subgroup was formed this year due to the many issues with the power system the previous years have faced. During inspection of the power system, it was noticed that the original batteries needed to be replaced due to the fact that they were old and no longer functional.

Original cables, connected wires, and a variety of makeshift jumper connections that were either soldered or crimped with irregular connectors make up the current electrical system. Due to the lack of documentation, it caused confusion on what is happening in the electrical system and was decided to form this subgroup and rework all the power and create a detailed documentation for future groups.

5.2 Objectives

This sub group's main goal is to develop an expandable and modular 12V power system that can power the vehicle and has a well-documented wiring harness that will make future additions or modifications easier. In order to shield the new charger from harm, the initiative also intends to create a safer and more dependable charging circuit. Additionally, the quantity of various wire and connector types used in prior years needs to be decreased. In order to guarantee that future teams can utilize the wire harness efficiently and follow the correct battery operating methods, thorough documentation will be created at the end.

5.3 Plan

5.3.1 Battery Pack

The goal of the new battery pack is to both get it to a functional state and to improve its operational safety.

- We opted to keep the SLA battery chemistry of the previous batteries for our new pack as it would keep the low cost and simplicity of the original design. The advantages of an alternative chemistry such as LiPo or LiIon with energy density and capacity are not very relevant to our goals as this vehicle will only be used for testing purposes. With the original battery pack assuming max draw from the 12V systems our estimated battery life of around 1-2 hours is adequate
- Our goal with selecting the new batteries was to get some with similar capacity and shape. The chosen batteries (WPHR12-48) are slightly higher capacity than the original ones (48Ah vs 45Ah) and are slightly differently shaped but most importantly are shorter than the originals as that is the limiting dimension in the existing battery enclosure
- We are adding a breaker to the charging circuit to isolate it in case the current exceeds the expected charging current. This will protect the new charger to prevent it from the damage seen in the old one. This will be paired with a new 3d printed plate in the charging door to ensure its proximity to the charging port to make operation more obvious. The charging circuit also will connect to the battery before the main breaker allowing us to disconnect all other systems from the pack while charging
- To make the system more readable, safe, and organized we are adding two bus bars that will break out the battery packs positive and negative connections to all the various components (charger, motor controller, and 12V regulator)
- A new cover for the battery enclosure will be made out of a plastic sheet mounted with 3d printed vents between the metal bottom plate and the top cover. This cover will mount all the distribution equipment (the bus bars, 12V regulator, battery monitor, and fuse boxes) and include room for other parts such as the motor controller and motor disconnect relay

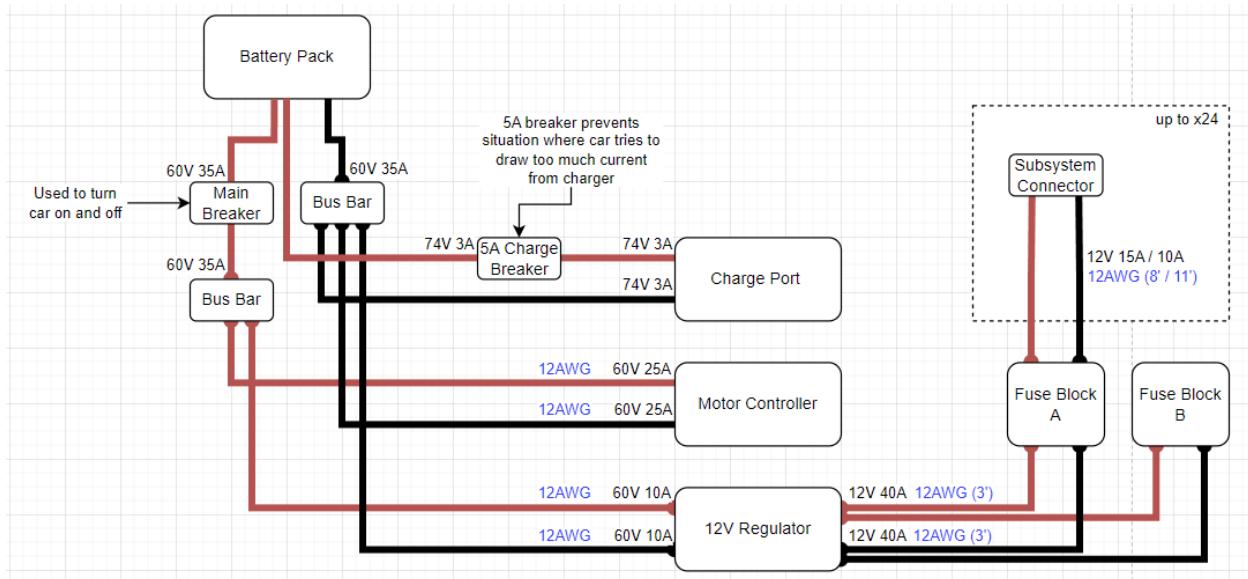


Figure 1: Wiring Diagram showing the distribution wiring from the pack to the fuse boxes

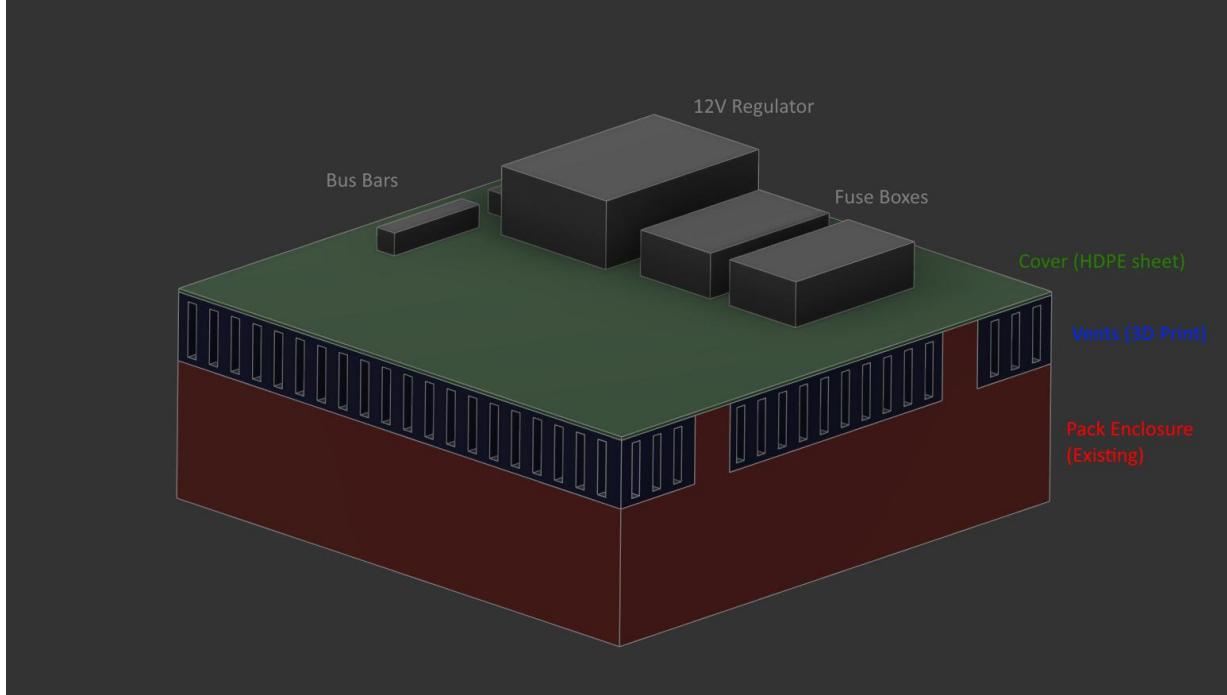


Figure 2 : Rough 3D model showing the cover with the 3D printed vents and mounted components

5.3.2 Battery Pack Monitor

The goal of the battery pack monitor is to allow the software operating the vehicle to be able to access the status of the battery. This will be worked on during the second semester after the battery pack has been made.

- The monitor will be made using a RPi which will connect to the ethernet harness of the vehicle. It will connect the physical voltage and current monitors with the ROS2 layer using BatteryState messages
- The monitor will measure current using the off the shelf shunt that came with the Victron BMV-712 Smart battery monitor that was acquired in the previous years. This shunt has a very accurate small resistance whose voltage drop can be measured by the RPi using an ADC
- The monitor will measure the voltage of the battery pack using a voltage divider and another ADC. This voltage will also be converted to a battery pack percentage
- Other battery statistics such as current capacity can be kept track of and reported to ROS (Monitor the current draw and pack voltage to estimate the batteries actual capacity)
- The monitor will package all this data (pack current draw, pack voltage, pack percentage) and using the BatteryState message report this information back to the ROS network

5.3.3 Wiring Harness

The goal of the wiring harness is to provide 12V power, ethernet connectivity, and direct connections to the various subsystems throughout the vehicle. We will create internal specifications for a “power subsystem” and create custom cables using a standardized set of wire and connectors.

- The power harness will use 12 AWG MTW/TEW wire as it is rated for the voltages (12-74V), currents (15A), and environment (-30°C to 105°C, and moisture / oil resistance)
- For connectors, the power harness will use ring terminals and ferrules when connecting to the pack, bus bars, breakers, and fuse boxes
- Individual subsystems will use Mate N Lok connectors as they are rated for the current and wire gauge used in the system. Maximum current draw on any one subsystem will be 15A, as it can be easily handled by our 12 AWG wire at any range. The subsystems will all be individually fused and connect to the 12V regulator through the fuse boxes
- For the ethernet harness, an 8 port ethernet switch will be installed in the vehicle to connect all the RPis and Jetsons together to allow for ROS communication between all the subsystems
- For the direct connection harness we will use CAT6 cabling but terminate it in different connectors. This harness will carry the E-Stop signal and have the other 7

conductors reserved for future use. This harness is used to allow for subsystems to communicate with each other directly instead of through ROS over ethernet



Figure 3: Rough diagram showing the path the wiring harness will take throughout the vehicle. The wires will then branch off this main trunk to go to the various subsystems

5.3.4 Custom Interface PCB

To make connection to the wiring harness easier for other subgroups, we plan to make custom interface PCBs. This will be in collaboration with the reliability and control subgroup to include the E-stop interface components.

- In the fall semester, we will reuse the buck converters from previous years for stepping down the voltage to 5V where needed. This will be replaced by the PCB in the winter semester
- Design and layout a PCB with a custom voltage regulator to lower the 12V input voltage to 5V which is required by some subsystems, pass through the 12V for systems that require the full 12V, and accommodate the level shifting and isolation circuitry for the estop connection on the direct connection harness
- Get the PCBs manufactured using a service such as JLCPCB or similar

- Populate the boards and mount inside an enclosure and distribute to the subteams

5.3.5 Documentation / User Manual

We plan to create documentation and user manuals to share knowledge of the modular 12V system and proper practices for operating, charging, and maintaining the new batteries. This will be developed throughout the year, with the documentation being finished earlier so that it can be provided to all other subgroups for their use when connecting to the harness.

5.4 Relation to Degree

Success in this project will require a strong foundation in computer systems engineering, particularly skills gained through hands-on labs. Courses such as SYSC 3010 and SYSC 4805 have provided essential experience in safely handling electrical components, while courses like ELEC 2501 and ELEC 2507 have equipped the team with critical knowledge in circuits and electronics. This will be particularly valuable when creating the wiring harness and ensuring the project's electrical components function properly. Experience working with 3D printers, electronics, and PCB design and manufacturing was gained through many courses during the degree and also in school clubs such as CUInspace.

5.5 Group Skill

Software development skills will be used when constructing the battery monitor in order to have it collect data, convert raw measurements of voltage into battery percentages, and interface all this data with ROS. As well for reliability the device will need to broadcast a heartbeat and listen for E-stops and act accordingly. Bash scripting will be needed to configure the RPi to act properly for our use case. Also, Electrical design skills will be used when designing the various distribution circuits as well as the monitoring circuits to safely monitor the pack's current draw and voltage levels.

5.6 Methods

To achieve the set objectives of this subgroup, we plan to use the methods stated below:

- Complete use of the systems development cycle:
 - Doing an analysis on what currently exists from previous years' group and how much is usable/salvageable
 - Understanding the requirements, thinking about what we really wanted to achieve and laying out our objectives

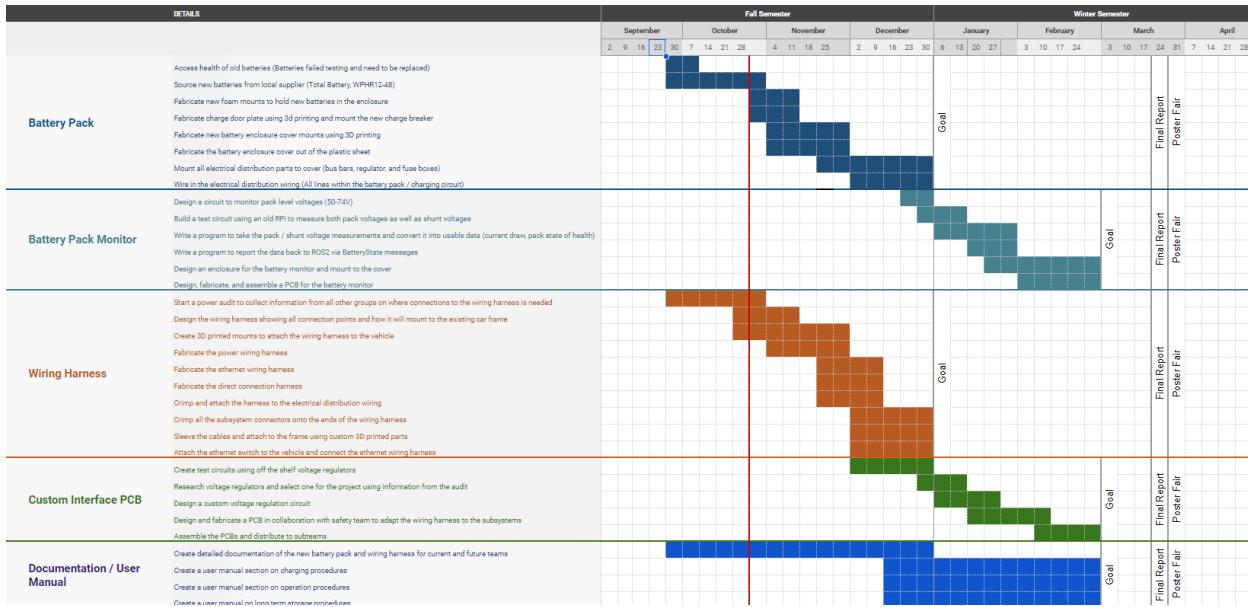
- Planning out the design of the system and deciding on the pros/cons of choices, as well as which components to use
- Developing, integrating and testing the system thoroughly, ensuring it is a robust implementation that will last
- Documenting progress and ensuring future users are kept informed of all aspects of the system, should they want to add on to it in the future
- All of these methods and the overall development cycle relate back to everything we have learned throughout our engineering degree

5.7 Timetable

Below is a list of the tasks that are required to be completed during the year and a rough timetable

- Battery Pack
 - Assess health of old batteries (Batteries failed testing and need to be replaced)
 - Source new batteries from local supplier (Total Battery, WPHR12-48)
 - Fabricate new foam mounts to hold new batteries in the enclosure
 - Fabricate new battery enclosure cover mounts using 3D printing
 - Fabricate the battery enclosure cover out of the plastic sheet
 - Mount all electrical distribution parts to cover (bus bars, regulator, and fuse boxes)
 - Fabricate charge door plate using 3d printing and mount the new charge breaker
 - Wire in the electrical distribution wiring (All lines within the battery pack / charging circuit)
- Battery Pack Monitoring
 - Design a circuit to monitor pack level voltages (50-74V)
 - Build a test circuit using an old RPi to measure both pack voltages as well as shunt voltages
 - Write a program to take the pack / shunt voltage measurements and convert it into usable data (current draw, pack state of health)
 - Write a program to report the data back to ROS2 via BatteryState messages
 - Design an enclosure for the battery monitor and mount to the cover
 - Design, fabricate, and assemble a PCB for the battery monitor
- Wiring Harness
 - Start a power audit to collect information from all other groups as to where connections to the wiring harness are needed

- Design the wiring harness showing all connection points and how it will mount to the existing car frame
- Create 3D printed mounts to attach the wiring harness to the vehicle
- Fabricate the power wiring harness
- Fabricate the ethernet wiring harness
- Fabricate the direct connection harness
- Crimp and attach the harness to the electrical distribution wiring
- Crimp all the subsystem connectors onto the ends of the wiring harness
- Sleeve the cables and attach to the frame using custom 3D printed parts
- Attach the ethernet switch to the vehicle and connect the ethernet wiring harness
- Custom Interface PCBs (Collaboration with the safety team)
 - Create test circuits using off the shelf voltage regulators
 - Research voltage regulators and select one for the project using information from the audit
 - Design a custom voltage regulation circuit
 - Design and fabricate a PCB in collaboration with safety team to adapt the wiring harness to the subsystems
 - Assemble the PCBs and distribute to subteams
- Documentation and User Manual
 - Create detailed documentation of the new battery pack and wiring harness for current and future teams
 - Create a user manual section on charging procedures
 - Create a user manual section on operation procedures
 - Create a user manual on long term storage procedures



Power Management

5.8 Risk Assessment

A potential risk we may encounter is a time risk. Almost all of the components required for the power subgroup will be new parts that have to be delivered to us, this can cause delays as to when we can actually begin work. Since we are essentially starting from scratch, it is hard to accurately predict the overall timeline of how long certain tasks will take. Some ways to mitigate this risk is by creating a timeline/gantt chart, this can help us create a general guide to follow and help to keep us on track. Another strategy to help mitigate the potential time risk is to overestimate how long tasks will take while planning, this will create some overhead and wiggle room for scheduling later on.

6. Navigation

6.1 Background

The development of autonomous vehicle navigation systems has evolved significantly since the early 1980s. Initial research was led by institutions like Carnegie Mellon University (CMU) and the University of the Bundeswehr Munich, with a focus on simpler tasks such as highway driving, lane detection, and adaptive cruise control [1]. These projects laid the foundation for more advanced autonomous systems by exploring how vehicles could detect their environment and react to basic driving scenarios.

A pivotal moment in the development of autonomous vehicles came with the DARPA Grand Challenge (2004-2005) and the Urban Challenge (2007), which pushed autonomous ground vehicle (AGV) technology to new levels [1]. These challenges forced autonomous vehicles to move beyond controlled environments like highways and into more complex, dynamic urban settings, where vehicles had to navigate unpredictable traffic, pedestrians, and obstacles.

Autonomous vehicle navigation today relies on a variety of sensors and technologies to provide accurate and reliable environmental perception. Lidar, radar, cameras, and ultrasonic sensors are commonly used to detect objects, track their movement, and help the vehicle navigate safely through its surroundings. For precise and up-to-date route information, HD maps provide highly detailed data on road geometry, lane markings, and traffic signs [2]. These maps, in combination with sophisticated path-planning algorithms, allow autonomous vehicles to safely determine optimal routes and adapt to changing environments.

An important component of modern autonomous navigation is visual odometry (VO), which estimates the vehicle's position and orientation using sequential images. VO is especially useful when traditional positioning systems, such as GPS or wheel odometry, are either unavailable or unreliable, such as in urban environments where GNSS signals can suffer from blockage or multipath effects [3]. To help mitigate these challenges, Inertial Navigation Systems (INS) and LiDAR are used in combination to provide accurate short-term navigation and detailed environmental maps.

For obstacle detection and avoidance, vision-based approaches are the most popular due to their cost-effectiveness and their ability to analyze the environment's appearance and shape. However, challenges such as illumination variability (The amount of light cast on an object.) and environmental factors complicate object detection [4]. Recent advancements in machine vision and deep learning have enhanced autonomous vehicles' abilities to navigate complex scenarios, making these systems more robust in detecting and avoiding obstacles [5].

This project aims to take inspiration from these advanced navigation and obstacle detection technologies and implement a solution into the existing electric vehicle platform. The addition of the **navigation module** will address a critical gap in the autonomous vehicle's ability to safely and reliably navigate urban environments, forwarding real-time route information and obstacle data to the vehicle's control systems

6.1.1 Current State of the Art

Recent advancements in autonomous vehicle navigation have focused on improving sensor fusion, path planning, and obstacle detection, utilizing bleeding-edge algorithms and hardware solutions. These developments aim to increase the reliability and efficiency of autonomous vehicles (AVs) in complex environments.

Autonomous Vehicles Navigation Systems:

One of many significant innovations is the integration of pulsed LED technology with machine vision algorithms (**Patent US 20180364730A1**). This approach enhances object detection by combining infrared illumination⁹ and frequency-based object recognition¹⁰, offering a redundant safety mechanism in low-light conditions [6]. This development addresses challenges like poor visibility and improves overall detection reliability for Autonomous Vehicles in complex driving environments.

Another notable trend is the use of sparse mapping (**Patent EP 4180768 A1**), a lightweight alternative to traditional high-density maps [7]. This method enables efficient data storage and processing, making it ideal for large-scale deployment in autonomous fleets. The integration of AI-driven mapping solutions and sensor fusion allows AVs to aggregate data from multiple sources, such as cameras, LiDAR, radar, and GPS. Crowdsourcing techniques for real-time map updates have also become more prevalent, enabling AVs to continuously adapt to changing road conditions.

Lidar-based SLAM (Simultaneous Localization and Mapping) is another bleeding-edge advancement. It is particularly useful in environments with poor GPS connection, where Autonomous Vehicles rely on Lidar sensors to create high-definition maps. This method utilizes Lidar odometry, which estimates the vehicle's movement by comparing consecutive Lidar frames. It allows Autonomous Vehicles to ground themselves into the local environment in real-time by aligning Lidar scans with prebuilt 3D maps, making it highly effective for autonomous parking and indoor navigation [2].

To enhance precision in navigation, modern systems integrate GNSS¹¹, LiDAR, and INS¹² data to help mitigate signal blockages in urban environments. This approach, which uses the strengths of these technologies in combination, provides accurate and continuous navigation even in challenging conditions [8].

Route Mapping:

Advanced pathfinding algorithms such as A* and RRT are central to modern AV navigation [9]. The A* algorithm employs methods to reduce unnecessary exploration during route mapping, ensuring the shortest and most efficient path is chosen. RRT (Rapidly-exploring Random Trees),

⁹ **pulsed LED illumination** emitted from objects like traffic signs and vehicles, allowing the vehicle to detect objects and decode their identity based on frequency, intensity, and wavelength of the pulsed illumination.

¹⁰ The technique of identifying objects by analyzing the frequency patterns of signals, such as infrared or other electromagnetic waves, reflected off the objects.

¹¹ Global Navigation Satellite System

¹² Inertial Navigation System

known for handling high-dimensional spaces and complex constraints, is frequently used in path planning. These algorithms, however, have limitations such as path discontinuity and suboptimal routes.

An innovation in adaptive route planning (**Patent US 11,022,449 B2**) combines real-time environmental data with historical performance records and simulated vehicle data, enabling Autonomous Vehicles to find the safest and most efficient route based on prior experiences [10]. This aligns with trends in machine learning and predictive analytics, where past performance affects future decisions.

Recent methods, such as Perception-Based Local Route (PBLR - It uses vehicle sensors like cameras, radars, and lidar to detect road lanes, boundaries, and obstacles.) and Map-Based Local Route (MBLR - It relies on pre-built, highly detailed maps with centimeter-level accuracy combined with precise vehicle localization to map a route), provide flexible route generation depending on environmental complexity. The Hybrid Local Route (HLR) approach intelligently switches between PBLR and MBLR based on real-time conditions, optimizing costs and reliability in both structured and complex environments [11].

Obstacle Detection:

Obstacle detection systems have seen major advancements with the use of deep learning and sensor fusion technologies. Probabilistic occupancy maps, digital elevation maps (DEM), and geometry-based clustering are widely used to detect and classify obstacles in real-time. Modern systems such as Faster R-CNN (Region-based Convolutional Neural Network) enable the accurate classification of dynamic obstacles like vehicles, pedestrians, and animals [4].

Additionally, innovations in Markov Random Field (MRF - A set of random variables having a Markov property¹³ described by an undirected graph) models, combined with deep neural networks, have improved the segmentation of obstacles, enhancing AV safety and navigation in dynamic environments [5].

A key development in this field is the fusion of sensor data from LiDAR, cameras, and radar with advanced path-planning algorithms. This allows Autonomous Vehicles to make rapid decisions, adjusting for both stationary and moving obstacles in real-time (**Patent US 11,796,673**). The system's focus on adaptive control and mode-switching reflects best practices in modern autonomous navigation, ensuring flexibility and responsiveness in challenging environments [12].

¹³ Markov Property refers to the memoryless property of a stochastic process, which means that its future evolution is independent of its history.

6.1.2 Motivation

Advancing autonomous navigation systems plays an important role in improving safety, efficiency, and reliability across several industries. In precision farming, for example, autonomous tractors may optimize resource usage, reduce costs, and enhance sustainability by guiding machines with high precision to eliminate wasteful movements, improving agricultural efficiency [13].

Similarly, in military applications, unmanned ground vehicles (UGVs) rely on autonomous navigation to operate safely in dangerous terrains, reducing human risk in defense and disaster relief [13].

Visual odometry is widely used not only in vehicles but also in space exploration (e.g., NASA's Mars rovers), underwater vehicles, and robotics, allowing machines to navigate challenging environments autonomously [3]. The rapid development of route planning algorithms has drastically improved navigation performance, making these systems faster and more reliable in real-time applications such as transportation and logistics [14].

In urban settings, autonomous vehicles have the potential to reduce traffic, lower emissions, and enhance road safety by optimizing routes and reacting to complex dynamic traffic conditions. Additionally, these systems contribute to environmental sustainability by minimizing fuel consumption, reducing greenhouse gas emissions, and enabling more efficient, green transportation.

6.2 Objectives

The primary objective of this project is to design a navigation system for a car that can continually plan and re-plan ways through different surroundings to get from a starting point to a planned destination. It will have to consider changing traffic conditions as well as obstructions on roads and surface variations in the environment. The goal is to have this system guide the car from a starting point to a planned destination so that the vehicle can make rapid decisions to satisfy high safety target performance and reliability.

One crucial element of this project involves incorporating data from sensor sources into the navigation system to enhance its performance constantly and guarantee smooth navigation by preemptively avoiding possible hazards or disruptions. We wish to create a navigation system that can effectively adapt to changes in road circumstances, such as obstructions, closures, and variations in traffic flow, to enhance the driving experience by rerouting cars to prevent delays and guarantee safety.

The goal is to build a system that can make and defend decisions for the vehicle in the context of safe driving—both in prototypical and real-world driving situations. The system must be tested in simulated and real-world environments to ensure its reliability.

Functional Requirements

- Navigation module will accept user input for destination location
- Navigation module will update user on current state of vehicle as well as route being followed
- Navigation module will accept sensor data for obstacle detection
- Navigation module will determine a navigation route from the current location to the specified destination location
- Navigation module will generate a series of waypoints consisting of GPS coordinates along the determined path
- Navigation module will make plotted waypoints accessible to overall system
- Navigation module will detect objects in planned route and;
- Navigation module will update path whenever obstacles are detected in the planned route

Non-Functional Requirements

- Navigation module should determine route within an acceptable time of receiving destination input from user
- Navigation module should recalculate route within an acceptable time of an obstacle being detected
- Navigation module should use the a* algorithm to determine and update route
- Navigation module should plot waypoints within an acceptable distance apart

6.3 Plan

6.3.1 Major Tasks

6.3.1.1 UI/Front-End

Project Needs:

Platform: The solution is being built as a web app with a mobile app as a future upgrade option.

Design Complexity: The interface developed will be highly interactive and will update in pseudo-real-time.

Team Expertise: The team has some familiarity with front-end development concepts involving React, HTML, CSS, and JavaScript.

Available Software/Toolset:

Table 1: Available tools for Front-end development

No	Framework/Library/ Software	Description	Advantages	Drawbacks/ Limitations
1	Figma	a collaborative web application for interface design and prototyping UX [15].	Helps develop Wireframes, and mockups to better understand website design	No perceived drawbacks in making a mockup for a design aside from additional time allocation
2	HTML	HTML stands for Hyper Text Markup Language and is the standard markup language for creating Web pages It describes the structure of a webpage [16].	<ul style="list-style-type: none"> • Easy to learn • Lightweight and fast to load • Integrates well with other languages such as CSS and javascript [16]. • Has support for majority of the browsers 	<ul style="list-style-type: none"> • Not capable of rendering dynamic output . • Too much code for too little output [16].
3	CSS	CSS stands for Cascading Style Sheets and is a styling framework for HTML. It describes how HTML elements are to be displayed on screen, paper, or in other media [17].	<ul style="list-style-type: none"> • CSS can help remove code redundancy. • Makes it easy for the user to customize the online page • One instruction can control several areas of the webpage [17]. 	<ul style="list-style-type: none"> • With CSS, what works with one browser might not always work with another. • There exists a scarcity of security [17].
4	JavaScript	It is an interpreted programming as well as a scripting language for webpages. It helps make webpages highly interactive and allows for real time updates [18].	<ul style="list-style-type: none"> • Regardless of where it is hosted, it always gets executed on the client environment saving a lot of bandwidth. • Compatible with all browsers • Huge amount of support from developers globally [19]. 	<ul style="list-style-type: none"> • Not easy to develop large applications. • It is not very good for safety, as the code is always visible to everyone i.e. anyone can view JavaScript code[19].
5	React	It is a javascript based User interface library for building system components	<ul style="list-style-type: none"> • Large community of developers for support. 	<ul style="list-style-type: none"> • Hard to integrate into existing projects.

		<p>incorporating concepts from HTML,CSS and Javascript. It is used with additional bundlers like webpack to build and compile the project [20].</p>	<ul style="list-style-type: none"> • Blends HTML,CSS,JavaScript together. • Search-engine friendly • Can be used with various systems and on both client and server sides [21]. 	<ul style="list-style-type: none"> • React evolves at a very fast speed, so additional time is required to adapt to the new changes every time an update is made [21]. • Has one-way binding [22].
6	Vue.js	<p>It is a JavaScript framework for building user interfaces.</p> <p>It builds on top of standard HTML, CSS, and JavaScript and provides a component based programming model like react [24].</p>	<ul style="list-style-type: none"> • Has two way-binding meaning there is a real-time synchronization between the model and the view. • Extensive and detailed documentation [22]. 	<ul style="list-style-type: none"> • Lack of stability in components. • Relatively small community. • Language barrier with plugins and components (most plugins written in Chinese language) [22].
7	Svelte	<p>Svelte is a UI framework that uses a compiler to write concise components that do minimal work in the browser.</p> <p>It builds on top of standard HTML, CSS, and JavaScript and provides a component based programming model like react</p>	<ul style="list-style-type: none"> • Lightweight, simple and uses the existing javascript libraries • Faster than any other framework like Angular or React [22]. 	<ul style="list-style-type: none"> • Small community • Not enough Documentation • Lack of Tooling [22].
8	Angular	<p>It is a development platform which like react is component based used to build scalable web applications.</p> <p>It has a complete suite of development tools to help test and build the project [23].</p>	<ul style="list-style-type: none"> • Has two way-binding meaning there is a real-time synchronization between the model and the view. • It has in-built functionality to update the changes made in model to 	<ul style="list-style-type: none"> • Learning curve is steep. • Dynamic apps may not function properly sometimes because of their complex structure and size [22].

			<p>the view and vice versa.</p> <ul style="list-style-type: none"> • A vast community for learning and support [22]. 	
--	--	--	---	--

Chosen Software/Toolset:

As observed from the toolset comparison, both Angular and React are solid choices for developing our project. However, given the tight 4-month development timeline and weighing the pros and cons of each tool, we've decided to proceed with React due to its flexibility, faster learning curve, and efficiency in handling dynamic updates. While Angular provides a complete framework with built-in features like routing, state management, and real-time data handling, its complexity and steeper learning curve could slow us down within such a short timeframe [23].

React, on the other hand, is primarily a UI library that allows us to pick and choose the additional libraries needed, such as React Router for navigation across multiple web pages. This modularity gives the team freedom to focus on the core features of the navigation module without being locked into Angular's more rigid architecture.

React, like the other toolsets presented in table 1, is component based and has an extensive community for support and development. While it may be a one-way binding structure, it is easy to manage dynamic updates with less overhead than Angular's two-way data binding approach.

In conclusion, although Angular's built-in features make it ideal for larger, more complex applications, React's simplicity, flexibility, and faster development process makes it a better fit for our short development timeline of roughly 4-months. React allows us to quickly prototype and build dynamic, real-time features without being slowed down by Angular's more involved setup. While we'd be interested in experimenting with newer frameworks like Svelte, we've decided against it for now due to its limited community support.

Diagrams/Figures: x



Figure 4: Mockup Home Screen of User Interface

The home screen as can be seen in Figure 4 holds multiple elements such as a search box to enter the destination, recent searches, some saved addresses, a marker to default to the user's current location, a compass, and a tool to navigate to the diagnostics window developed by the safety and reliability team.

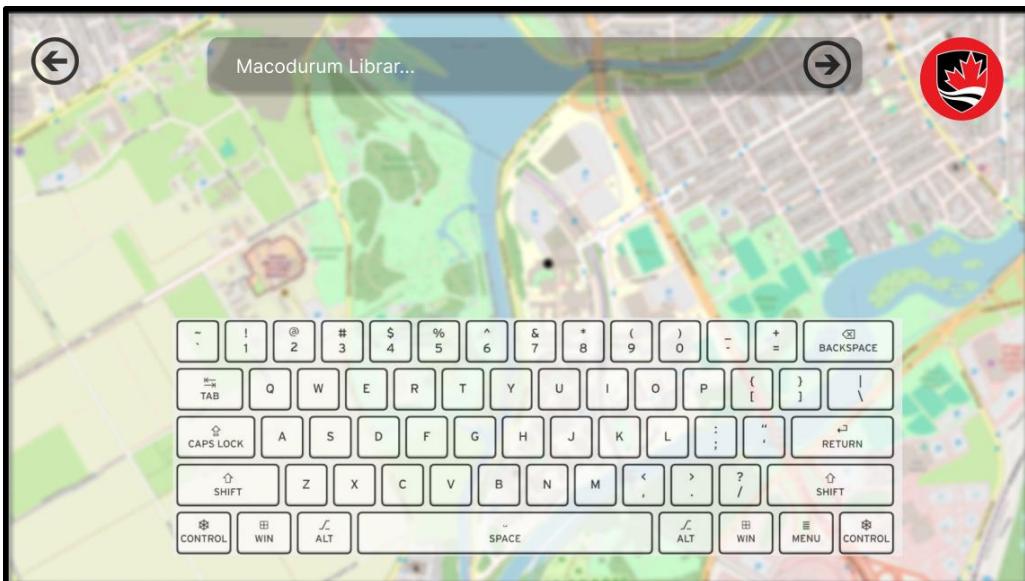


Figure 5: Mockup for Search-box window

Once a user clicks on the search box, a keyboard pops up as seen in figure 5 indicating to the user to enter a destination.

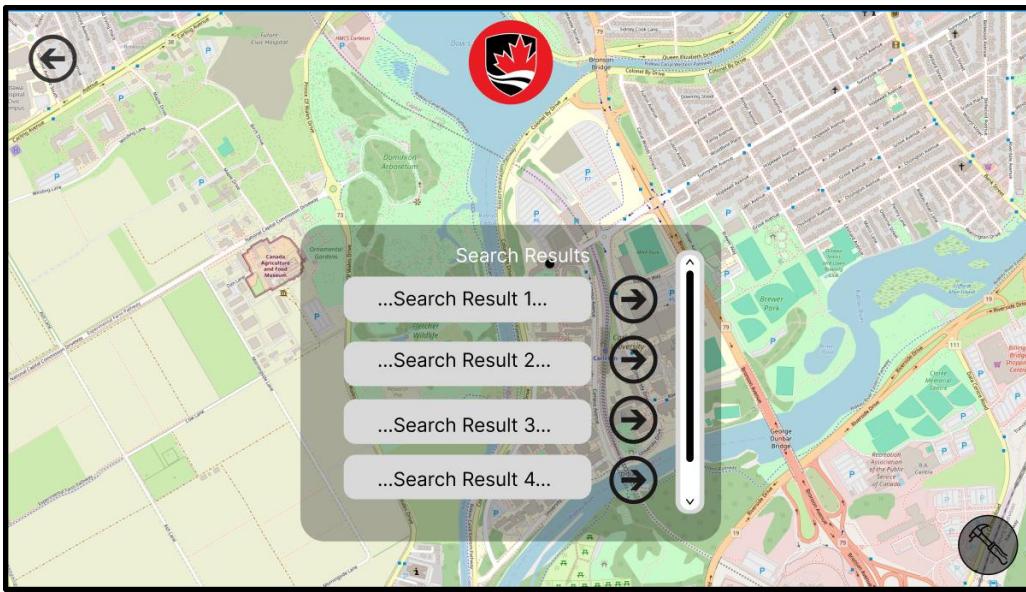


Figure 6: Mockup for SearchResult window

Upon hitting enter, a list of possible addresses are displayed on the interface from which the user makes a choice. Upon making a choice, a trip information page is displayed to the user and user confirmation is required to begin navigation as seen in figure 7.

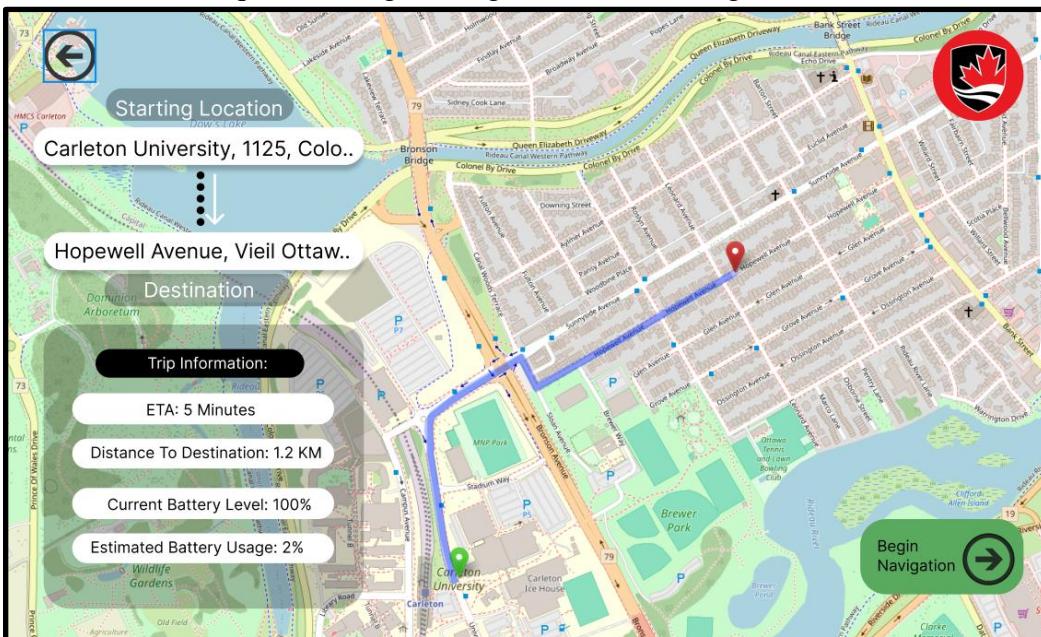


Figure 7: Mockup for Trip Information Window

Once the user begins navigation, useful information such as user heading, speed, Estimated Time of Arrival (ETA) is displayed and the user can stop the navigation by clicking on the red STOP button whenever necessary as can be seen in figure 8. Once the navigation ends, user is sent back to the home screen.

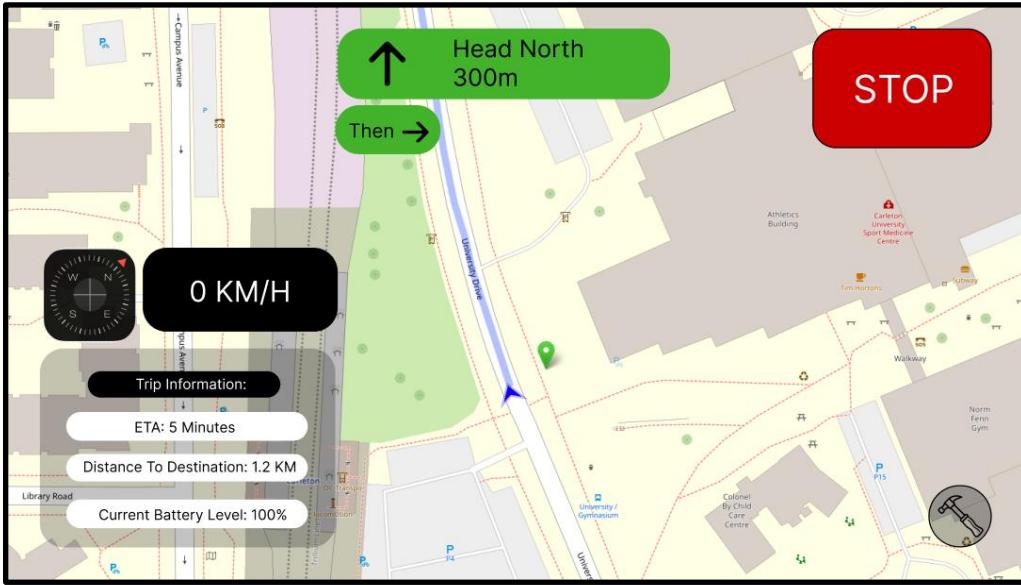


Figure 8: Mockup for Navigation Window

6.3.1.2 Hosting Server/Backend

Project Needs:

Scalability:

The navigation module is likely to remain a small to mid-size project, and we don't anticipate the need for it to scale into a large system. Since it will be hosted on a Raspberry Pi, this limits scalability, but given the size and scope of your project, this should be sufficient.

Performance:

The system needs to be responsive enough to handle real-time updates, like recalculating routes and sending new waypoints if an obstacle is detected. However, small UI delays are acceptable.

Project Type:

This will be a dynamic web app that processes and displays real-time data (route planning, obstacle detection). The web app needs to handle dynamic updates to routes and waypoints, but it isn't API-driven

Budget:

We require open-source, free-to-use hosting, and the plan is to host the web app on a Raspberry Pi.

Team Expertise:

The team has some experience with Flask, which gives us a good starting point to build the project.

Available Software/Toolset:

No	Framework	Description	Advantages	Disadvantages
1	Flask	A lightweight and flexible Python micro-framework. Ideal for simple applications and provides complete control over components like routing and templating since it does not have a builtin library for these functions[26].	<ul style="list-style-type: none"> • Easy to learn and use (Team also has some prior experience). • Highly customizable and flexible [27]. • Simple to integrate with rclpy for ROS2 [39]. • Supports WebSockets for real-time updates [28]. 	<ul style="list-style-type: none"> • Minimal features out-of-the-box (e.g., no built-in admin, or complex tools). • May require more effort for large, complex apps as you need to manually add libraries and modules. • It has a singular source which means that it will handle every request in turns, one at a time [27].
2	django	A full-featured Python web framework with built-in features like ORM, admin panel, and authentication, ideal for larger, complex applications [29].	<ul style="list-style-type: none"> • Comes with many features built-in (ORM, admin, authentication). • Strong community and documentation. • Scalable for larger applications. • Provides a structured approach [29] [30]. 	<ul style="list-style-type: none"> • Steeper learning curve compared to Flask and FastAPI. • Slower development speed due to its heavy setup. • Overkill for simple, lightweight applications [30].
3	FastAPI	A modern, high-performance Python web framework built for APIs that encourages rapid development., with asynchronous capabilities [31].	<ul style="list-style-type: none"> • Extremely fast and asynchronous, ideal for real-time applications. • Built-in support for WebSockets. • Auto-generates interactive API docs (OpenAPI, Swagger) [31] [32]. 	<ul style="list-style-type: none"> • Smaller community compared to Django or Flask. • Slightly steeper learning curve for asynchronous concepts. • Lacks some out-of-the-box features (admin, ORM) [31] [32].
4	Node.js with express	A JavaScript runtime with a lightweight,	<ul style="list-style-type: none"> • Asynchronous by nature, ideal for real-time data exchange. 	<ul style="list-style-type: none"> • Requires familiarity with JavaScript. • Heavier development

		<p>flexible framework used for building fast, scalable APIs and real-time web servers.</p> <p>It enable developers to run javascript outside of a browser [33].</p> <p>Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [34].</p>	<ul style="list-style-type: none"> • Strong WebSocket support. • Large ecosystem of libraries and packages. • High performance for handling large data streams [36]. 	<p>process compared to Python frameworks [36].</p>
5	Bottle	<p>A Python micro-framework, similar to Flask but even more lightweight and designed for small-scale applications [36].</p>	<ul style="list-style-type: none"> • Extremely lightweight and fast. • Easy to use with minimal setup. • Good for simple use cases where speed is critical [37]. 	<ul style="list-style-type: none"> • Lacks built-in tools like ORM or authentication. • Not ideal for complex applications. • Smaller community and fewer resources compared to Flask and Django [37].

Table 2: Available tools for Back-end development

Chosen Software/Toolset:

Given our project's constraints—our familiarity with Flask, the need to integrate with ROS2, hosting on a Raspberry Pi, and a 4-month development timeline—we believe Flask is the better choice over both FastAPI and Django and have decided to use it instead of the other two.

Since the team already has some experience with Flask, we can minimize the learning curve and begin development faster. Flask's lightweight and flexible nature works really well with ROS2, e.g. we can easily set up ROS2 nodes using Python's rclpy library to pull and publish data. Flask's synchronous model will handle the data exchange between ROS2 nodes and the web app effectively, especially since minor delays in the UI aren't critical for us plus, with Flask-SocketIO, we can handle real-time updates like recalculating routes or sending waypoints to the control systems team if an obstacle is detected.

Django when compared to flask and FastAPI is a full-fledged framework that includes more built-in features like an ORM and an admin interface. For the scope of our project these additional features are not really necessary and could add additional complexity in learning along with adding additional overhead in processing where performance is critical. For working with ROS2, Flask offers more flexibility and its simplicity and lightweight footprint make it a better fit for the limited resources of a Raspberry Pi.

FastAPI could offer performance advantages with asynchronous operations, and is a runner up alongside django in developing this solution . Flask is easier to work with and is sufficient for managing ROS2 data communication.

In conclusion we have chosen to work with flask as it aligns with our timeline, hardware, and ROS2 integration needs. If needed the usage of either FastAPi or dJango is still in consideration as a backup due to their excellent feature-set.

6.3.1.3 Algorithm

Project Needs:

Navigation in vehicles heavily relies on pathfinding to ensure safe movement while avoiding obstacles during the journey from one location to another. A range of algorithms has been explored for their effectiveness in environments. Both static and dynamic.

Algorithm: The solution should be able to change dynamically depending on feedback received.

Design Complexity: The solution should be complex enough to handle multiple split/branches in a route.

Team Expertise: The team has to learn about these algorithms from scratch and do not have any prior experience in plotting a route.

Available Software/Toolset:

Below are the algorithms considered along with an assessment and rationale for selecting A* as the most suitable solution, for this particular project

Algorithm Name	How it works	Advantages	Disadvantages	Why it is not chosen
Dijkstra's Algorithm	It is a graph search algorithm that explores all possible paths from the source node to the destination node finding the shortest path[56]	1) the algorithm is simple to implement and is a good starting point for basic scenarios[57] 2)Finds the shortest possible path[56]	In intricate settings, Dijkstra's method loses its effectiveness as it examines every route without any heuristic assistance, resulting in unnecessary exploration[58]	1) Dijkstra's search methods are not scalable for large environments [58]
RRT(Rapidly-exploring Random Tree)	Is a method that helps find paths by sampling points and creating a tree of potential paths to navigate the environment without following a systematic exploration[W]	Excels in navigating through search areas and proves especially valuable for tasks requiring instant responses, in intricate surroundings. [X]	The routes produced are often not the shortest and most efficient path [X]	As RRT does not follow systematic exploration, its random approach requires additional processing
D algorithm	Dynamic Pathfinding (referred to as D*) is tailored for environments that are subject to changes, with obstacles appearing or disappearing unpredictably in real-time scenarios.	D* excels in an environment that changes over time as it quickly recalculates paths	Implementing D star is trickier than A star as it demands a grasp of incremental search algorithms. The increased computational demand of updating paths puts a lot of stress on the system	To kickstart this project phase enough A star with premeditated obstacle avoidance will suffice. If the environment experiences an increase in dynamism in the future, it may be worth exploring

A* Algorithm	The A algorithm is a commonly utilized system for pathfinding and navigating. It excels in determining the route between two designated points on a graph from point A to point B taking into account the associated movement costs. A*'s effectiveness stems from merging components of both the Dijkstras Algorithm (which focuses on discovering the path) and a heuristic method (which aids in approximating the ideal direction to advance toward the desired endpoint).	A* ensures that the optimal path will be discovered reliably and efficiently. This feature makes it particularly valuable in scenarios where pinpoint accuracy in route planning is essential. For instance, in guiding vehicles through their navigation routes.[T]	When working with A-star algorithms in environments, with search spaces that involve lengthy paths and detailed settings. The challenge arises when all node generations need to be stored in memory since it can lead to a substantial storage constraint as the graph expands over time due to the need to retain all open nodes that are yet to be assessed.[T]	
--------------	--	--	--	--

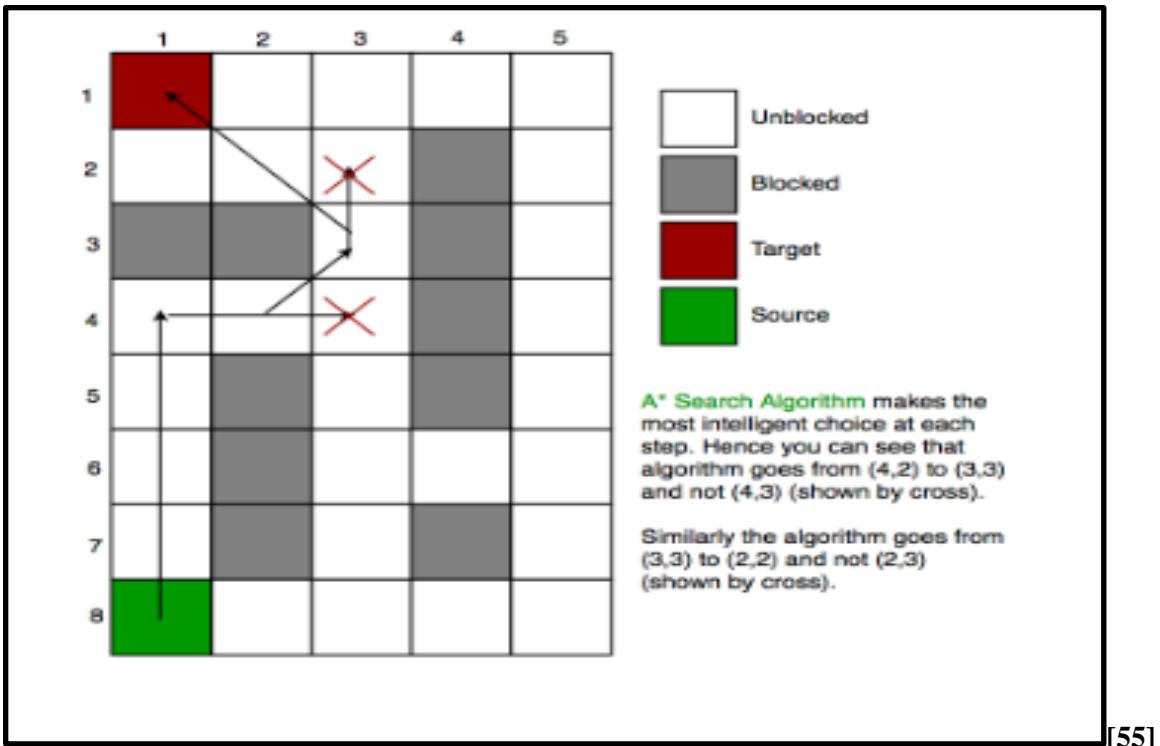
Table 3: Selection of Algorithm for pathfinding

Chosen Software/Toolset:

The algorithm chosen to plot waypoints and implement pathfinding is A* and the reason for choosing this algorithm have been detailed in Table-3. We will be using ‘Nav2’ package for ROS2 for implementing the A* algorithm in our autonomous vehicle for the following reasons:

- Nav2 package includes a robust path planning framework that helps and supports the A* algorithm and also helps in path finding optimization.
- Nav2 package is built to work better and efficiently with ROS2 which helps us to manage communication between different components of the self-driving car.
- Nav2 package is known for its flexibility in customization and tuning of the A* algorithm parameters such as the heuristic functions, weightings, etc.
- Nav2 package’s cost map functionality plays a major role in helping the A* algorithms account for obstacles in the environment.

Diagrams/Figures:



[55]

Figure 9: Figure explaining the mechanism behind A* algorithm

6.3.1.4 Route Tracking

Project needs:

- **Algorithm:** As the vehicle is moving autonomously the algorithm should be able to work in dynamic environments and make real-time adaptations according to the data received from the sensors.
- **Design Complexity:** The system needs to be able to precisely localize and track the vehicle in real-time as it navigates to its final destination.
- **Team expertise:** The team has no experience in using estimation algorithms or in route tracking. We will have to learn these concepts from the ground up.

Route tracking in this system involves continuously monitoring the vehicle's progress along its calculated route, ensuring it stays on course from point A to point B. In autonomous navigation systems, precise localization is crucial for safety and traversal purposes. For high performance and real-time routing, we will be using OpenStreetMap for the map database, since it is free and open source. OpenStreetMap also includes a rich set of map features such as road networks, points of interest (POI), etc [3]. It is always being updated due to the community and is also very adaptable to relative project requirements. Now that we have a map, how are we going to track the vehicle navigating through it? Relative to the surroundings, autonomous vehicle systems require constant tracking of their position, velocity, and orientation, provided by a continuous flow of data from all the sensors onboard the vehicle. Navigating through dynamic environments creates challenges regarding where the vehicle is currently located and how it will reach its next waypoint. For

instance, GPS signals become unreliable in environments where there are physical obstructions due to it requiring unobstructed line-of-sight to the satellites for accurate localization [42]. Even then, GPS coordinates can only provide so much accuracy, as standard civilian GPS systems on average can provide measurements within 5 meters [43]. As our vehicle has multiple sensors providing this data, we need a system to fuse all the data into usable information. Sensor fusion ensures that all the data from the sensors can be integrated to estimate where the vehicle is located. A system with sensor fusion permits real-time processing, which in turn allows the vehicle to navigate through dynamic environments [44]. This allows the vehicle to respond quickly to new conditions while staying on the most optimal route.

During sensor fusion, there is an extensive amount of data from the sensors that are fed into the system, but how is this data to be utilized by the vehicle? The system needs to be able to continuously estimate its coordinates and velocity with high accuracy and minimal latency. As the vehicle is running in real-time, we need a system that allows us to process the data and make decisions for the vehicle.

Given that our data is non-linear, implementing an **Extended Kalman Filter (EKF)** was the best approach, as it would help linearize the system and improve the state estimation [45]. Kalman filters in autonomous vehicle systems fuse data from multiple sensors and integrate them into one unified estimate of the system's state [46]. As the Kalman filter is a real-time process, this makes it very efficient, considering it does not require any memory and it enables the system to make real-time estimations of its future state. The following figure presents a simplified version of the **Kalman Filter formula**:

$$\hat{X}_k = K_k * Z_k + (1 - K_k) * \hat{X}_{k-1} \quad [47]$$

where, \hat{X}_k = Current Estimation

K_k = Kalman Gain

Z_k = Measured Value

\hat{X}_{k-1} = Previous estimation

k = States

This equation shows us there are four distinct variables. Firstly, we have \hat{X}_k which is the current estimation, which is the estimation of the system's current state given all the information up until this point, and it keeps updating this equation as more data comes in [54]. \hat{X}_{k-1} is the previous estimation that had been gained after completing a round of the Kalman filter with the previous data [54]. The measured value Z_k is the measurements received from the sensors at each step k. Finally, we have, K_k which is the Kalman gain, and it is the variable that determines how much weight should be given to the measured value for the next estimation [54]. If the Kalman gain is closer to 1 then the system gives more weight to the newly measured value which means

the system is closer to the current measurement. When the Kalman gain is closer to 0 that means the system believes the prior measurements to have been more accurate.

Within Kalman filters, there are variations of it to fit with different requirements of systems. Systems could be linear or non-linear. In our case, due to dynamic environments, our system is non-linear. Certain filters did not fit our project objectives, so they were not considered. The table below shows the different types of Kalman filters that could have been used:

Name:	Advantages:	Limitations:
Standard Kalman Filter:	It is the simplest to implement. It is the most efficient computationally [48].	It only works with linear systems. Does not work in dynamic environments as systems are non-linear.
Extended Kalman Filter:	It allows the linearization of nonlinear systems. It is computationally more efficient for non-linear systems. Supported in ROS2 [49].	Efficiency decreases as the system becomes more non-linear. Accuracy decreases as estimation errors are not taken into account for the next steps. Needs external correction.
Unscented Kalman Filter:	As it requires no linearization of the system, it makes it more accurate than EKF, and it is better for larger non-linear systems.	Higher accuracy comes at the cost of being computationally heavy on the system and being more difficult to implement.
Particle Filter:	It handles non-linear systems in dynamic environments. It also allows the tracking of multiple states at the same time [50].	It is computationally very heavy in its real-time processing, given our equipment. Consistent accuracy is difficult to achieve [51].

Table 4: Available solutions for Rote mapping algorithm

After researching all the variations of the Kalman filters, we collectively decided that the **Extended Kalman Filter** is the best option for our autonomous vehicle system. Extended Kalman Filter, Unscented Kalman Filter, and Particle Filter are great options for non-linear systems, but the latter two options are computationally heavy. One of the other reasons for choosing this system was that it handles nonlinear systems efficiently and that it is well-supported within ROS2. Given that all the sensors require processing, the Extended Kalman Filter keeps it lightweight while also providing precise localization.

As mentioned earlier, the Extended Kalman Filter was chosen due to the easy integration process within the ROS2 system. **ROS2** is an open-source robot operating system that allows users to use their software libraries to create robotic systems. In our case of autonomous vehicles, ROS2 is well-supported due to public libraries. For example, “**robot_localization**” is one of the packages that allows us to implement an Extended Kalman Filter without having to create it from scratch [52][53]. This package fuses all the sensor data to create an accurate mapping of the vehicle’s position. The Extended Kalman filter will subscribe to certain nodes like “Odometry” to receive the velocity of the vehicle and the “IMU” node for the orientation . Using EKF within ROS2 enables the opportunity to simulate and test the algorithm in CARLA making it an easy integration process with both the Simulation and Perception teams.

5.3.1.5 Object Detection

- Object detection is the process of identifying and differentiating objects such as vehicles, and pedestrians in the environment.
- Object detection and avoidance in our self-driving car project will be achieved using NVIDIA Isaac ROS Nvblox packages, which will run on the Jetson platform to generate a high-resolution, real-time local cost map that captures the 3D structure of the vehicle’s surroundings and results in a 2D output of a cost map for Nav2. [61]
- The local costmap will be integrated with the Nav2 framework to continuously update the vehicle’s path based on real-time data obtained from the perception team.
- The NVIDIA ROS Nvblox package is designed for real-time 3D mapping and obstacle detection using the help of the stereo camera data and their depth sensors.
- By utilizing its fast GPU Acceleration, the system can generate 3D voxel maps of the environment at high speeds and this is essential for a safe and efficient operation.
- After the obstacles are detected in the voxel grid, Nvblox creates a 2D costmap. This costmap is always updated with new sensor data to reflect the latest conditions around the vehicle.
- The Nav2 framework uses the cost map from Nvblox to determine the most optimal path route to the destination while avoiding obstacles. [62]
- If an obstacle is detected in the path, Nav2 will adjust the route by recalculating the shortest and safest path around the obstacle.
- With GPU Acceleration, the Jetson system can run the Nvblox package efficiently, even with large numbers of sensor data coming from cameras and depth sensors.
- The combination of NVIDIA Isaac ROS Nvblox for generating the 2D costmap and Nav2 for the path-planning process ensures that the self-driving car’s route is always updated with environmental data which is crucial for safety and it is also updated with real-time data to ensure risk-free and efficient travel from point A to point B.

Advantages:

- easy integration with nav2 in ros2 to generate costmaps from sensor data that nav2 can use.

- It avoids the complexity of having to put together an implementation and model from scratch and/or integrating something with nav2/ros2, since nvblox already supports nav2 and it was designed with jetson hardware in mind which makes it convenient for our use case

Drawbacks:

major limitation is that uneven ground isn't supported so slopes will be seen as obstacles by default, unless sensor data is pre-processed to remove them, there is potential in future for nvblox to be updated with support for non-flat ground but currently not the case [66].

Since our scope is to first get vehicle functioning in flat parking lot environment this shouldn't be an issue but if there is no way to work around this limitation in the future a redesign might be required using a different technology

Conclusion:

Despite the drawbacks, since the goal for this project is to achieve limited autonomy in a controlled environment, it is more advantageous to use an easier to implement solution, while libraries like OpenCV do interface with ROS2 and would be excellent options, nvblox has the major advantage of being designed with the Jetson hardware we have on hand in mind, and having nav2 support natively, greatly simplifying the implementation process.

6.3.2 Functional Requirements

Requirement ID	Requirement Description
FR-01	The navigation module shall accept user input for the destination location.
FR-02	The navigation module shall update the user on the current state of the vehicle and the route being followed.
FR-03	The navigation module shall accept sensor data for obstacle detection.
FR-04	The navigation module shall determine a navigation route from the current location to the specified destination.
FR-05	The navigation module shall generate a series of waypoints consisting of GPS coordinates along the determined path.
FR-06	The navigation module shall make plotted waypoints accessible to the overall system.
FR-07	The navigation module shall detect objects in the planned route using sensor data.
FR-08	The navigation module shall update the path whenever obstacles are detected in the planned route.

FR-09	The navigation module shall log all navigation data (e.g., route, waypoints, obstacles) for review and troubleshooting.
FR-10	The navigation module shall accept speed limit information from the perception system and adjust route calculations accordingly.
FR-11	The navigation module shall provide a visual map interface showing the vehicle's current location and planned route.
FR-12	The navigation module shall trigger alerts or notifications to the user when obstacles require rerouting.
FR-13	The navigation module shall prioritize safety by avoiding routes that pass through hazardous areas.
FR-14	The navigation module shall continuously track and report the vehicle's estimated time of arrival (ETA).
FR-15	The navigation module shall detect and report GPS signal loss, providing fallback instructions to the user.
FR-16	The navigation module shall support different map formats, including offline maps, in case of internet or GPS loss.
FR-17	The navigation module shall allow for integration with third-party services like OpenStreetMaps for enhanced route data.
FR-18	The navigation module shall allow for scheduled route planning, enabling users to pre-plan routes in advance.
FR-19	The navigation module shall integrate with other vehicle subsystems to optimize power usage during route planning.

Table-x: List of Functional requirements

6.3.3 Non-Functional Requirements

Requirement ID	Requirement Description
NFR-01	The navigation module shall respond to user inputs within 2 seconds to ensure a smooth user experience.
NFR-02	The navigation module shall update the vehicle's position on the map at least once every second to maintain real-time accuracy.

NFR-03	The navigation module shall handle up to 1000 simultaneous waypoint calculations per minute without a significant performance drop.
NFR-04	The navigation module shall recover from system faults (e.g., power failure) within 5 seconds to minimize downtime.
NFR-05	The navigation module shall support at least two different map formats (e.g., online and offline) to ensure system robustness in different conditions.
NFR-06	The navigation module shall provide feedback (visual or auditory) within 1 second if the user attempts to enter an invalid destination.
NFR-07	The navigation module shall log system events (e.g., route recalculations, obstacle detections) with timestamps for future analysis and troubleshooting.
NFR-08	The navigation module shall encrypt all communications between the vehicle and external services to ensure data security.
NFR-09	The navigation module shall maintain a 90% success rate in recalculating routes in less than 3 seconds after obstacle detection.
NFR-10	The navigation module shall use a user-friendly interface that conforms to accessibility standards (e.g., large buttons, high contrast for readability).

Table 5: List of Non-Functional requirements

6.3.4 Software Requirements

Software Requirement	Description
ROS2 (Robot Operating System)	Middleware platform that assists in coordinating interactions among subsystem components, such as navigation and control units.
NAV2 (Navigation2)	A ROS2-based navigation framework compatible with the A* algorithm and dynamic path replanning, integrating real-time sensor data.
NVIDIA Isaac ROS Nvblox	Software designed for creating 3D environment reconstructions and 2D costmaps using cameras and other

	sensors.
OpenStreetMap (OSM) Data	Data source for routing and tracking.
Full-Stack Development Tools	Required for building the UI and UX for the project.
Linux (Ubuntu 20.04 LTS or later)	The operating system needed to run ROS2, Nav2, and other related software.

Table-x: List of Software requirements

6.3.5 System Design

Please find below a preliminary version of Class Diagram, Deployment Diagram, and Sequence Diagram. These diagrams are subject to revision and improvements.

Class Diagram:

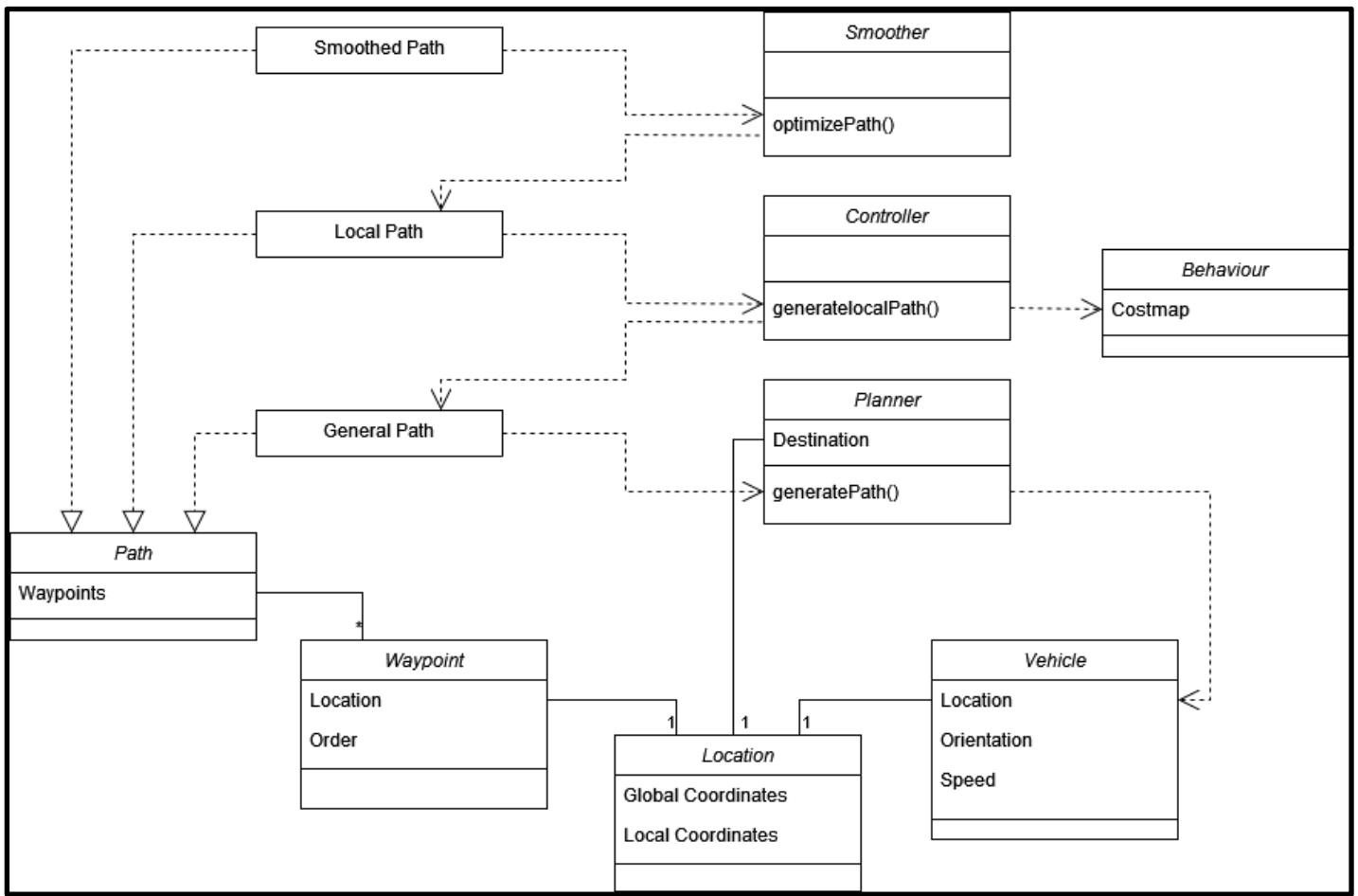


Figure 10: Class Diagram of how the system interacts with each other.

Deployment Diagram:

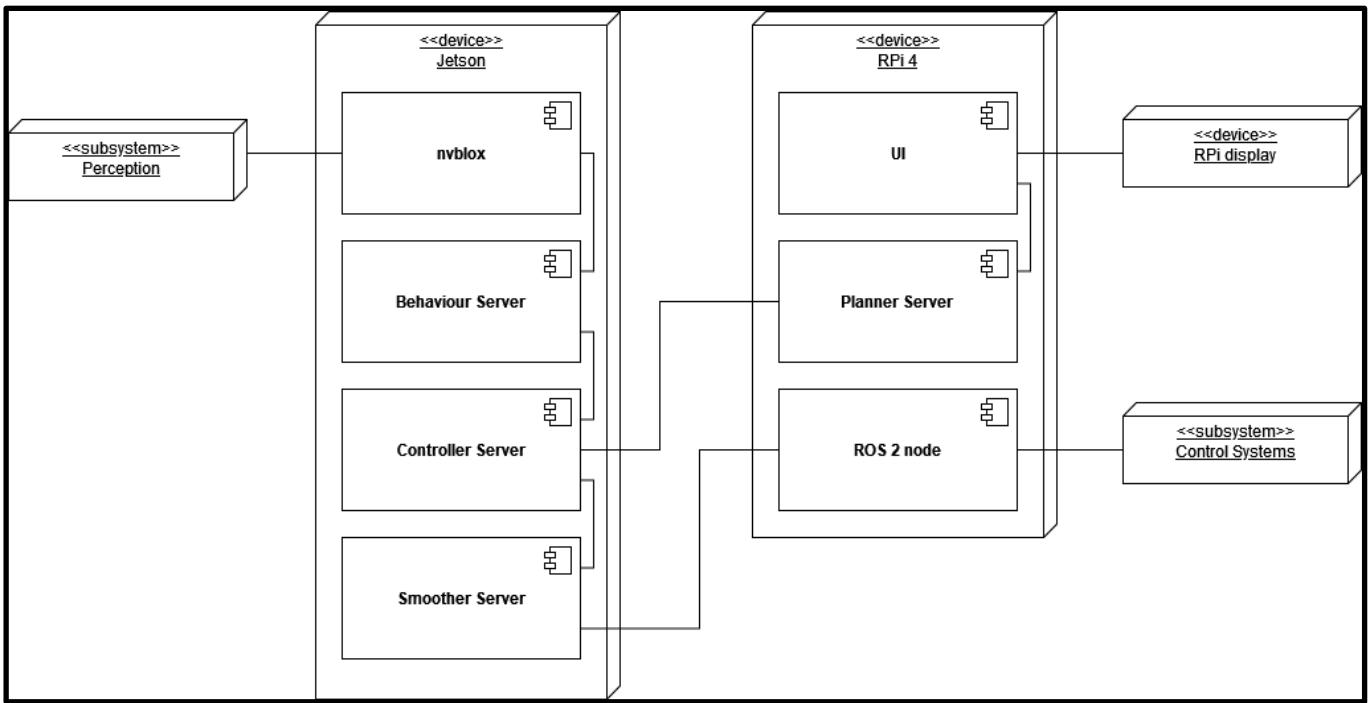
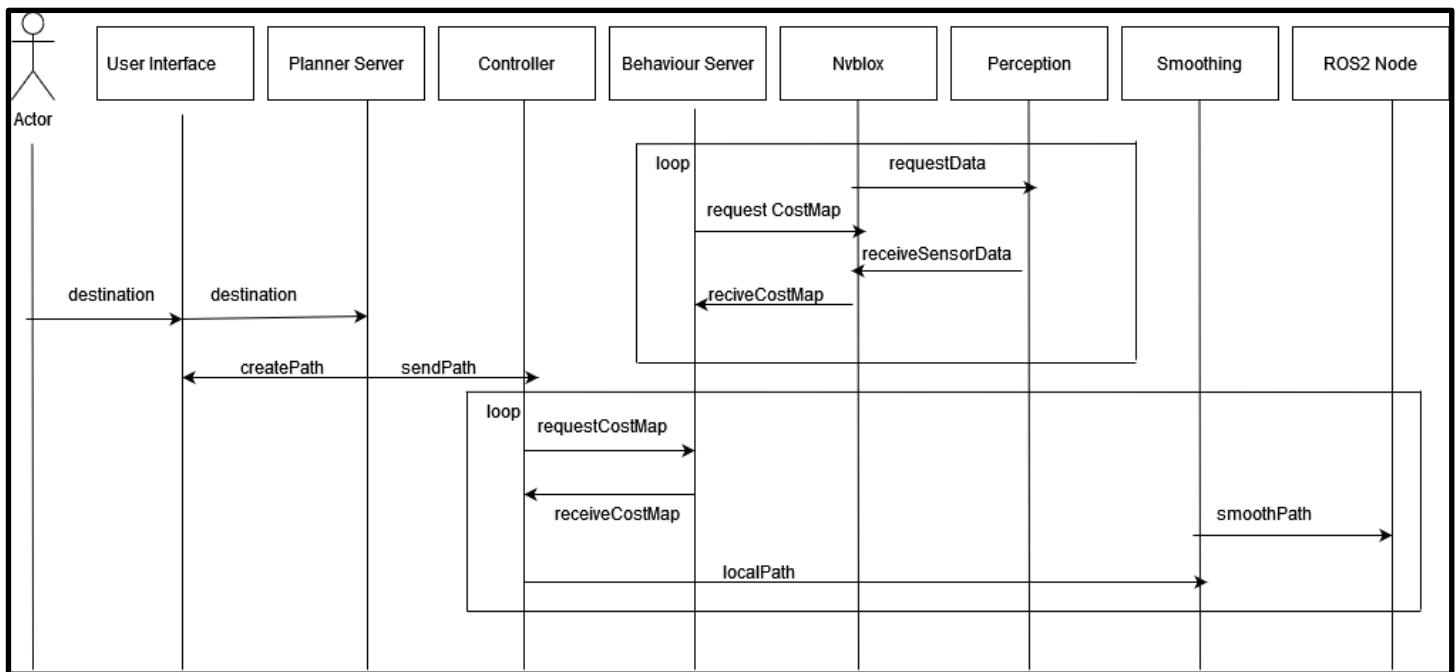


Figure 11: Deployment Diagram of how the system interacts with each other.



Sequence Diagram

Figure 12: Sequence Diagram of how the system interacts wi

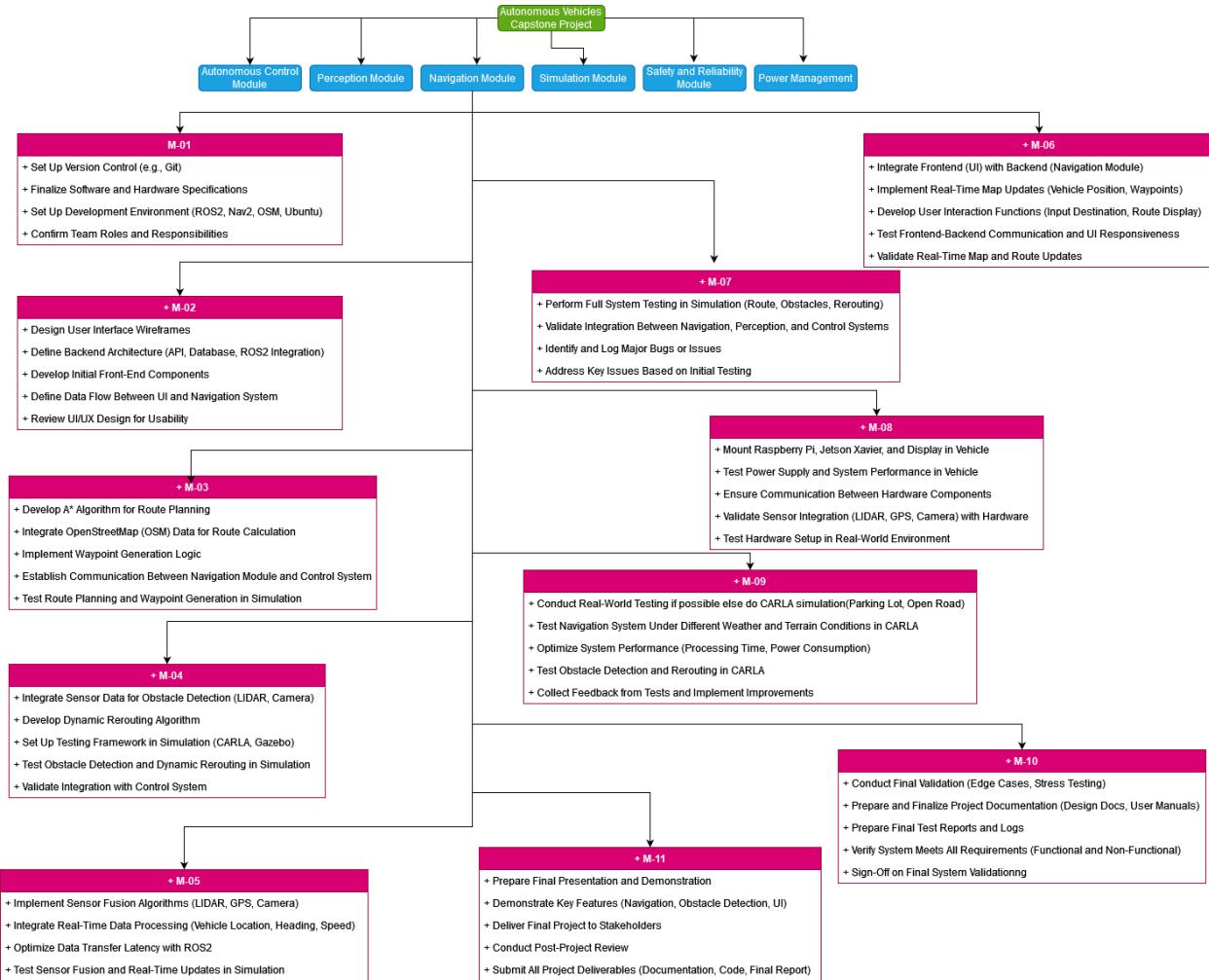
6.3.6 Milestones

Milestone ID	Milestone	Deliverables	Completion Criteria
M-01	Project Setup	<ul style="list-style-type: none"> • Set up development environment • Finalize team roles 	<ul style="list-style-type: none"> • development environment configured.
M-02	Initial UI/UX Design and Backend Architecture	<ul style="list-style-type: none"> • Complete UI wireframes. • Define backend architecture. • Start front-end development. 	<ul style="list-style-type: none"> • Initial UI design completed. • Backend architecture defined. • Basic UI functions operational.
M-03	Route Planning & Waypoint Generation Development	<ul style="list-style-type: none"> • Develop route planning using A*. • Implement waypoint generation. • Integrate with control system. 	<ul style="list-style-type: none"> • Navigation module generates routes successfully. • Navigation module generates waypoints successfully.
M-04	Obstacle Detection & Dynamic Rerouting	<ul style="list-style-type: none"> • Implement obstacle detection using sensors. • Develop dynamic rerouting. 	<ul style="list-style-type: none"> • System successfully reroutes based on detected obstacles in a simulated environment.
M-05	Sensor Fusion & Real-Time Data Integration	<ul style="list-style-type: none"> • Implement sensor fusion. • Integrate real-time data (location, speed, heading) processing. 	<ul style="list-style-type: none"> • Accurate sensor fusion. • Real-time updates integrated.
M-06	Frontend-Backend Integration & Map Updates	<ul style="list-style-type: none"> • Complete frontend-backend integration. • Real-time map updates on 	User can set destinations and view real-time route updates on UI.

		<p>the UI.</p> <ul style="list-style-type: none"> • Test user interaction. 	
M-07	Early System Testing & Validation	<ul style="list-style-type: none"> • Perform full system tests in a simulated environment. • Validate navigation, perception, and control system integration. 	All major functions operational in simulation with stable performance.
M-08	Hardware Integration	<ul style="list-style-type: none"> • Mount Raspberry Pi, Jetson Xavier, display in the vehicle. • Test hardware components, and communication. 	Hardware mounted and communication between components established.
M-09	Simulation and optimization	Conduct deeper tests in a simulated environment (CARLA), optimize performance.	Complex simulations completed, performance optimized for navigation, obstacle detection, and rerouting.
M-010	Final System Validation & Documentation	<ul style="list-style-type: none"> • Conduct final validation. • Complete project documentation. • Test edge cases. 	System validated and fully documented with a user manual, design docs, and test reports.
M-011	Final Demonstration and Delivery	Prepare final presentation, demonstration, and deliver final project to supervisors and Professors.	Successful final demonstration, project delivered to stakeholders.

Table 6: List of Milestones

6.3.7 Work Breakdown Structure (WBS)



6.3.8 Testing Plan

Milestone ID	Testing Focus	Approach	Tools
M-01	Verify setup and initial configurations	<ul style="list-style-type: none"> • Ensure version control is correctly set up. • Validate development environment configurations 	<ul style="list-style-type: none"> • version control systems (e.g., Git)
M-02	Usability and backend integration	<ul style="list-style-type: none"> • Conduct usability testing 	UI testing tools, API testing tools (e.g.,

		<ul style="list-style-type: none"> on UI wireframes. Verify backend architecture and data flow 	Postman)[59]
M-03	Algorithm accuracy and integration	<ul style="list-style-type: none"> Unit test the A* Algorithm. Integration test with control system 	Unit testing frameworks, simulated environments
M-04	Obstacle detection and rerouting	<ul style="list-style-type: none"> Test sensor data integration and dynamic rerouting in simulation. Validate integration with the control system 	Simulation software (e.g., CARLA)
M-05	Real-time data processing and sensor fusion	<ul style="list-style-type: none"> Test sensor fusion algorithms and real-time data processing. Optimize data transfer latency 	ROS2 testing tools, performance monitoring tools
M-06	Frontend-backend communication and real-time updates	<ul style="list-style-type: none"> Test integration of UI with the navigation module. Validate real-time map updates and user interactions. 	UI testing tools, API testing tools
M-07	Full system functionality in simulation	Perform comprehensive system testing in simulation to identify and address	Simulation software, bug tracking tools

		major bugs	
M-08	Hardware setup and integration	Test communication between hardware components and validate sensor integration with hardware.	Hardware testing tools, diagnostic tools
M-09	Real-world/full-scale performance and optimization	Conduct real-world tests if possible under various conditions and optimize system performance based on test results	Data logging tools, performance monitoring tools
M-010	Final validation and documentation	<ul style="list-style-type: none"> • Conduct edge case and stress testing. • Prepare final documentation and test reports. 	Testing frameworks, documentation tools
M-011	Final presentation and delivery	<ul style="list-style-type: none"> • Prepare and conduct the final demonstration • Ensure all deliverables are complete and submitted 	Presentation tools, project management tools

6.4 Relation to Degree

Our Navigation sub-group is made up of students from Computer Systems Engineering and Software Engineering, which makes this module a good fit to test the skills and knowledge we've developed in our studies.

For Computer Systems Engineering students, the project offers direct application of key topics such as algorithm design, where we're implementing the A* algorithm for route plotting, and real-time computing, as we handle live data from sensors like LIDAR, GPS, and GNSS to make immediate navigation decisions. This reflects coursework in embedded

systems and system architecture, allowing us to design and optimize systems that function in real-world, high-stakes environments.

For Software Engineering students, the project focuses heavily on software development and systems integration, using tools like ROS2 to build reliable systems that manage communication between the vehicle's perception and control modules. This reflects the principles we've learned in software engineering methodologies, systems design, and networking, providing hands-on experience in building scalable and maintainable software.

Additionally, the development of the user interface through Full-Stack web development methodologies aligns with both Computer and Software Engineering programs, giving us practical experience in creating functional interfaces between computers and humans, which is an important aspect of our degree studies.

The project not only tests technical skills but also highlights the importance of teamwork, project management, and interdisciplinary collaboration—skills that are critical to our success as engineers in the field.

6.5 Group Skill

The group possesses a diverse range of technical expertise, problem-solving abilities, and project management experience, making it well-equipped to implement high-quality solutions for route tracking, mapping, and user interface. Each team member brings their own experience in skills to the table such as algorithm design, real-time computing, software development, embedded systems and system integration, ensuring efficient task allocation based on individual strengths.

The synergistic effect of the team's technical knowledge allows for proper collaboration across different project components, increasing the project's likelihood of successful completion. In addition, strong communication and collaboration skills are valued within the team, facilitating coordination between submodules.

Regular meetings, brainstorming sessions, and clear communication channels ensure alignment with project goals and enable the team to address challenges collectively as they arise.

6.6 Methods

Our approach to solving the navigation challenges and implementing a functional navigation system for the autonomous vehicle involves using a range of proven methods that ensure the system operates effectively in dynamic and unpredictable environments. Combination of advanced algorithms, real-time data integration, pre-existing software frameworks, packages, libraries and system-wide communication, are all examples of what may be used to implement the navigation system. These methods are all rooted in the knowledge we have acquired through our degree program in addition to finding and learning new technologies on our own.

- **Path Finding and Optimization:** The navigation system will employ a pathfinding method that calculates the optimal route between two points. The knowledge necessary to understand and possibly implement this can be gained from the courses we have taken involving Data Structures and Algorithms. For example: A* is a well known pathfinding method that calculates the shortest route between point A and point B. This algorithm draws directly from the concepts of graph theory and optimization.
- **Real-Time Data Processing:** To adapt the navigation system to changing environmental conditions, such as obstacles or traffic, we will integrate real-time data streams from various sensors (e.g., cameras, GPS, and LiDAR) sent by the perception team. The method involves processing data continuously and adjusting the vehicle's path based on the latest information. This approach aligns with principles from our Real-Time Systems coursework, where we learned how to handle concurrent data inputs and ensure timely responses in critical applications.
- **Dynamic Map and Geospatial Management:** The navigation system will manage and utilize large-scale geospatial data, dynamically updating routes based on changing conditions. We will need to acquire and learn additional knowledge on Geographic Information Systems (GIS) and Database Management. We will make sure that the system can efficiently process spatial data and make necessary updates, ensuring the route remains optimal even in unfamiliar or dynamic environments.
- **Modular System Design and Integration:** The overall Autonomous Vehicle system is designed to work as an integrated framework, where the navigation module communicates with other subsystems such as perception and control

systems. This modular design approach ensures that each component can function independently while collaborating with others.

This method is rooted in Software Engineering principles we studied, emphasizing modularity, scalability, and efficient communication between systems.

- **User Interaction and Interface:** Providing a clear, real-time interface for users is vital in making navigation accessible and understandable. Our approach is to design a user-friendly interface that displays important data such as current location, upcoming turns, and real-time alerts. The knowledge necessary to implement it needs to be acquired. Information on Full-Stack development and UI/UX is a good start which ensures that the system is not only functional but also intuitive for the end user.
- **Collaborative and Iterative Problem-Solving:** Throughout the project, we will follow an iterative problem-solving approach by regularly testing and refining each component of the system. This approach ensures that we can identify and resolve issues early in the development cycle. This closely aligns with the Agile Development Methodologies commonly used in the technology industry during our co-op internships where collaboration, feedback, and continuous improvement are key to success.

6.7 Timetable

S.No	Milestone	Assigned To	Oct		Nov						
			WEEK1	28-Oct-24	WEEK2	04-Nov-24	WEEK3	11-Nov-24	WEEK4	18-Nov-24	WEEK5
1	Project Setup	Everyone									
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif									
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed									
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik									
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik									
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif									
7	Early System Testing & Validation	Everyone									
8	Hardware Integration	Everyone									
9	Simulation and optimization	Everyone									
10	Final System Validation & Documentation	Everyone									
11	Final Demonstration and Delivery	Everyone									

Figure 13: Timeline part1

S.No	Milestone	Assigned To	Dec				
			WEEK6	WEEK7	WEEK8	WEEK9	WEEK10
			02-Dec-24	09-Dec-24	16-Dec-24	23-Dec-24	30-Dec-24
1	Project Setup	Everyone					
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif					
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed					
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik					
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik					
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif					
7	Early System Testing & Validation	Everyone					
8	Hardware Integration	Everyone					
9	Simulation and optimization	Everyone					
10	Final System Validation & Documentation	Everyone					
11	Final Demonstration and Delivery	Everyone					

Figure 14: Timeline part2

S.No	Milestone	Assigned To	Jan				
			WEEK11	WEEK12	WEEK13	WEEK14	WEEK15
			06-Jan-25	13-Jan-25	20-Jan-25	27-Jan-25	03-Feb-25
1	Project Setup	Everyone					
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif					
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed					
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik					
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik					
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif					
7	Early System Testing & Validation	Everyone					
8	Hardware Integration	Everyone					
9	Simulation and optimization	Everyone					
10	Final System Validation & Documentation	Everyone					
11	Final Demonstration and Delivery	Everyone					

Figure 15: Timeline part3

S.No	Milestone	Assigned To	Feb					
			WEEK15	WEEK16	WEEK17	WEEK18	WEEK19	WEEK20
			03-Feb-25	10-Feb-25	17-Feb-25	24-Feb-25	03-Mar-25	10-Mar-25
1	Project Setup	Everyone						
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif						
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed						
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik						
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik						
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif						
7	Early System Testing & Validation	Everyone						
8	Hardware Integration	Everyone						
9	Simulation and optimization	Everyone						
10	Final System Validation & Documentation	Everyone						
11	Final Demonstration and Delivery	Everyone						

Figure 13: Timeline part4

S.No	Milestone	Assigned To	Mar					Apr	
			WEEK20	WEEK21	WEEK22	WEEK23	WEEK24	WEEK25	
			10-Mar-25	17-Mar-25	24-Mar-25	31-Mar-25	07-Apr-25	14-Apr-25	
1	Project Setup	Everyone							
2	Initial UI/UX Design and Backend Architecture	Shreeyansh and kaif							
3	Route Planning & Waypoint Generation Development	Ibrahim and Tauheed							
4	Obstacle Detection & Dynamic Rerouting	Harsh Patel and Karthik							
5	Sensor Fusion & Real-Time Data Integration	Harsh Patel and Karthik							
6	Frontend-Backend Integration & Map Updates	Shreeyansh and kaif							
7	Early System Testing & Validation	Everyone							
8	Hardware Integration	Everyone							
9	Simulation and optimization	Everyone							
10	Final System Validation & Documentation	Everyone							
11	Final Demonstration and Delivery	Everyone							

Figure 16: Timeline part5

6.8 Risk Assessment

- **Integration with perception data:** One major risk in this project is the integration of real-time data from the perception system. The navigation system depends on fast and accurate updates from the perception team, but any delay or inaccuracy in processing sensor data (like detecting obstacles or traffic) could result in bad routing decisions or safety risks. To reduce this risk, we'll establish strong communication protocols between the navigation and perception systems using ROS2, ensuring smooth, low-latency data transfer [40].
- **Performance variation:** Another risk involves the system's performance in different weather and terrain conditions. Since perception relies on sensors like cameras, LiDAR, and radar, bad weather might lead to incorrect data or make sensors less effective. To handle this, we'll use sensor fusion techniques, combining data from various sensors to cross-check and improve accuracy, even in poor conditions [41].
- **Integration with ROS2:** A third risk is the challenge of integrating ROS2 with the pathfinding system and making sure it interacts smoothly with the autonomous vehicle's control systems. If not handled properly, this could lead to delays or inconsistent decision-making. We'll address this by starting integration testing early and designing the system to be modular, so adjustments can be made easily.

There may be additional risks involved which include but not limited to:

- **Hardware limitations:** The Raspberry Pi and Jetson Xavier could face processing power limitations, especially when handling large amounts of data. It may especially be the case with the Nvidia Jetson board as it is a shared resource with other groups. We'll monitor system performance closely and optimize code to reduce computational load.
- **Power supply issues:** Since this is a mobile system, ensuring a stable power supply for both the Raspberry Pi and Jetson Xavier is crucial. We'll include power monitoring and consider backup options if needed.
- **User interface reliability:** If the touch display or in-car UI fails or experiences lag, it could frustrate users and make navigation difficult. To mitigate this, we'll perform rigorous testing on the UI and ensure smooth performance under different conditions.

7. Autonomous Control

7.1 Background

In this subgroup, the main focus is developing the interface between the hardware and software components of the vehicle, ensuring that the car can execute necessary commands while on the road. Using control methods PID and model predictive control, the aim is to create reliable braking, steering, and speed control systems that are capable of managing the vehicle behavior. This work ensures the autonomous vehicle can operate smoothly and safely, providing critical feedback to other subsystems in real-world driving situations.

7.2 Objectives

The objective of the autonomous control group is to enable the vehicle to start moving, providing all other subgroups with an interface to read key vehicle data and allow for control of the car. Almost all subgroups will use or provide data to enable for autonomous control of the vehicle. The main components as part of the autonomous control will be to develop functioning braking, steering, drive functionality, dynamics control, vehicle state feedback. We want to develop all controls with at least two situations in mind, one of which is normal operation under which the car functions moving from point to point provided by the navigation subsystem. The second is an emergency situation in which a collision may be made. In this mode, the car may be heading into a potential collision. In this scenario a different set of subsystems may want to control the vehicle to change direction or stop immediately. Another functionality we would want to facilitate for the E-Stop, which would be stop the vehicle immediately and disable all functionality. A big part of the vehicle will be dynamics control, which will be required to move between waypoints provided by the navigation subsystem and control all 3 physical systems that move the car. ROS Nodes will be used to provide information to other subsystems such as speed, direction, braking and other various telemetry of the vehicle. ROS nodes will be used to open interfaces to control steering, braking and acceleration of the vehicle. The following provides a brief plan for each aspect, which will be expanded upon in Section 6.3.

- Braking: Design and develop a braking system capable of safely decelerating and stopping the vehicle in a variety of situations including emergency stops. Ensure the system provides feedback and an interface to view and control the breaks.
- Steering: Develop a steering system that facilitates electric precision directional control, allowing the vehicle to maneuver specified routes and avoid obstacles. Provide the correct subsystems with an interface and feedback for the steering actuator.

- Drive Motor: Create an interface for the electric motor mounted on the Ecolo ET-4 cruise to allow for driving functionality
- Dynamics Control: Create a dynamic control system that ensures smooth navigation between predefined waypoints and optimize path following.
- Speed Control: Integrate a speed sensor, along with the motors built-in sensors to accurately monitor and utilize the vehicle's speed.

7.3 Plan

7.3.1 Dynamics Controller

The dynamics controller is a crucial part of the system that allows us to bridge the gap between navigation algorithms and physical control of the system. The idea behind dynamics control is that it allows the vehicle to go to the specified position regardless of real world conditions, such as speed bumps, stop signs etc. There are a variety of options that we considered for implementation of the dynamics controller each of which will be outlined below.

7.3.1.1 MPC

Model Predictive Control (MPC) is an extremely powerful control method which is applicable to even extremely nonlinear dynamical systems. MPC works by creating a mathematical model of the system to be controlled. Once a sufficiently accurate model of the system is created the algorithm will get the current state of the system as initial values for the system and use them to simulate forward in ‘simulation time’. By varying the input parameters to the system such as steering and acceleration over the course of the simulation, an optimization algorithm or other control methods can be applied to determine which input values correspond to the least error between the simulated result and the desired one. Once this is complete, the controller will return a set of parameters for each dt for as long as the simulation ran. The first set of these control parameters will be applied to the system. This is repeated for each control step of the physical system. This allows for an extremely robust control system that is able to ‘look ahead’ as opposed to other feedback systems that are purely reactive. The downsides to this algorithm are mainly related to its complexity. Because the simulation of the system has to be run at least once but often many times per physical control step, the computational cost can be enormous depending on the system being modeled. Another downside is the need for a reasonably accurate model of the system. This means accurate measurements of not only the regular physical parameters like weight and CoM but also approximating

aspects of the system like brake response and other, often nonlinear, effects which can depend on external variables like weather.

7.3.1.2 PID

PID control is a very simple way to control a system. It does not require any system parameters and requires very little computational power. Implementation methods for systems with multiple parameters may vary but an option for our autonomous car might look like the following: Given a goal point, create a trajectory to connect the two points. For each control step of the physical system, define an incrementally larger point along that line which progresses according to a goal speed. Next, calculate the distance between the car and the point and use that as the error for the acceleration and braking systems. Also calculate the length of a vector perpendicular to the vector extended from the wheels at the current steering angle which goes through the current goal point. This will determine the error in the steering system and can be used for the PID loop controlling steering. This way we have a PID loop and corresponding error for each aspect of the control of the vehicle dynamics. From here all we have to do is set the gain for the proportional, integral and derivative portions of the error and from there, the algorithm will sum the errors and give us control actions for acceleration and steering. The last aspect of this control scheme is some logic which determines whether the brakes need to be applied. This can be a simple if statement based on the deceleration rate of the motor with no applied power. This control scheme is a good option for simpler systems with low speed. It is extremely simple to implement and very fast. Overall a very good option, especially for testing a system.

7.3.1.3 Reinforcement Learning based control

Reinforcement Learning (RL) based control systems rely on techniques such as neural networks to determine the next control action based on inputs from the state of the system and its goal future position. This control scheme strikes a balance in terms of computational complexity and accuracy between the two previous options, often at the cost of reliability and the requirement of training data. A simplified example of how many neural network based RL control systems work is that they take control inputs and current system configuration as inputs to a neural network. This then, in conjunction with the weights and biases calculated in the training stage (more on that below) run an inference and output predicted next values to apply to the system. This inference stage is often referred to as a ‘black box’ this is because we have no concrete way of knowing how this system ‘decides’ on the output. The training stage comes before this inference

is possible and consists of giving the system inputs from the system as usual and then evaluating the result and applying a cost function to determine how accurate the predicted values are. Once this is done the weights and biases which can be thought of as many knobs which control minuscule parts of the network are adjusted using a variety of different techniques such as calculus based optimization algorithms in order to determine the ‘tweaks’ required to move the system’s result closer to the desired output. This Control scheme has the potential to perform extremely well, especially on systems with high complexity and relatively low computational power. The main downsides are twofold. First these systems often require an immense amount of training data and computational power to carry out this training, especially for systems where data cannot be obtained without simulations. The second downside is related to the ‘black box’ nature of the decision making. Many people do not trust RL based control algorithms for applications where unstable control could cause catastrophic effects. Overall the performance of this system can be extremely good but often at a high initial investment with no way to be certain of stability across input space.

7.3.1.4 Final Considerations

After carefully considering the control schemes outlined above we have decided to initially go with a PID based controller. This will allow us to obtain critical test information about the system in a timely manner and will ensure that other aspects of this project are not put on hold in order to implement a MPC based controller. Once the PID controller is working we will begin working on a MPC implementation which will allow for higher speed operation and better control in non-ideal environments.

7.3.2 Braking System

The braking system is an essential component of the autonomous vehicle. The braking system will receive inputs from a variety of other subsystems in various forms. To achieve accurate and reliable braking control, we have chosen to redesign and improve several parts of the braking system based on previous years’ experiences.

7.3.2.1 Physical Braking System Interface

The Ecolo ET-4 Cruise utilizes a hydraulic brake mechanism (similar to those of a bicycle) connected to disk brakes. This hydraulic system applies a force to the calipers when the lever is pressed and brake fluid is pressurized, slowing the car down. To

activate these brakes, we will use a Firgelli Automations FA-PO-35-12-2 linear actuator. The linear actuator will be secured to the frame of the vehicle and push on the brake levers. To improve upon previous years design, we will replace the hose clamps with a designed adapter. The adapter will simplify the design by allowing the linear actuator to actuate both levers mounted on top of each other. This will ensure equal braking pressure between both front and rear brakes. The following Figure 15 is the designed part, the part pairs the levers to the linear actuator. The linear actuator will be attached in a manner to allow for a small range of pivoting motion, as the levers rotate around a fixed point.

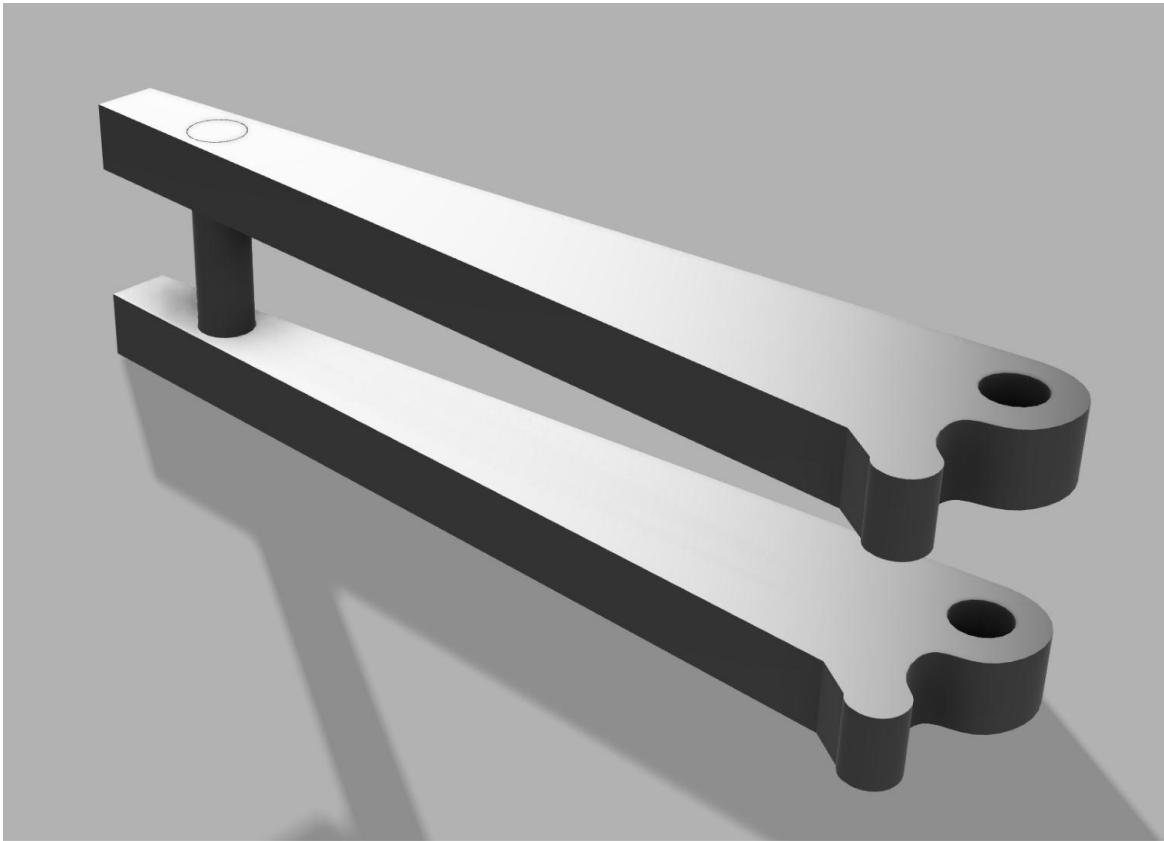


Figure 17: Braking Adapter Prototype

7.3.2.2 Linear Actuator Control and Feedback

The linear actuator provides a feedback system, which allows the use of feedback loops to control the position of the linear actuator. We can use a variety of dynamics controllers mentioned in [Section 7.3.1](#) to control the position of the linear actuator. In previous years implementation, Pololu SMC G2 motor controllers were used. While they provide adequate control, we have decided to update our motor controller to newer Pololu Jrk G2 18v19 Motor Controllers. These newer controllers integrate feedback

control into its operation, eliminating the need for additional Arduinos. This upgrade allows for a main controller to set a target position and the controller will use its built-in PID control to reach the target. A potentiometer in the linear actuator will provide position feedback, which allows for feedback based control. In later stages when we decide to use more complex dynamic control algorithms, we can easily disable the PID control. The controllers will also provide a hardware current limit, allowing us to limit current usage as well as the force applied to the brakes. We can also use two hardstops in the linear actuator which can set a hardware limit for the minimum and maximum extension.

7.3.2.3 Modes of operation and Safety

For the braking system, we will implement 3 modes of control, which are normal operation, critical braking, e-stop. These 3 modes will be controlled by different ROS topics, with the priority given to E-Stop followed by critical braking and normal operation. A final control method is a hardware bypass.

- Normal Operation

A normal operation that will allow for normal breaking controlled by the dynamics controller. Based on waypoints provided by the navigation subsystem, the dynamics controller will control the steering angle to reach that waypoint.

- Critical Braking

This mode will be activated in critical braking scenarios, triggered by the sensor node when a potential collision is detected. This critical break may occur if an obstacle such as a wall or person is detected and collision is likely. (Replace with a link to a subsection in document with critical breaking likely from navigation) In this mode, the goal is to reach a full stop in the shortest time possible. This control may bypass the dynamics controller based on the approach being used, as we want immediate breaking. Once the vehicle is functional and moving, we may also choose to implement an ABS-like system. This is to avoid the car being unable to turn during a critical break.

- E-Stop

The third mode, which will be an E-Stop, would be used if any of the hardware emergency stop buttons are pressed which is discussed further in Section 8.3.

In this case, similar to emergency braking, the dynamics controller will be bypassed and full braking will be set.

- **Hardware Bypass Brake Release**

Finally, we will have a hardware switch which will disengage the brakes. This will be a hardware connection, not interfacing with any ROS nodes. This will allow for a person during testing to disengage the brakes and move the car if needed. The switch acts as a release in the event of an e-stop when the brakes are fully engaged but we don't want to disengage the e-stop which would re-engage all other subsystems.

7.3.2.3 Testing Plan

Finally, we plan a comprehensive test suite that validates the braking system independently and as part of the overall vehicle system. These tests will be covered in detail in section x.x which outlines each of the tests. In general the independent test would test the hardware functions properly, such as achieving a full range of braking motion, braking control latency, critical braking response, e-stop response and hardware brake release. As part of the full system, we will assess normal braking, critical braking and e-stop performance.

7.3.3 Steering System

The steering system is another critical component of the autonomous vehicle, enabling it to navigate the environment and avoid obstacles. We plan to retain most of the physical design of previous years' implementation given it is effective. Similar to the braking system, the steering system will rely on a linear actuator for control.

7.3.3.1 Linear Actuator Control and Feedback

We will be using a Robotzone 6-263 linear feedback with integrated position feedback for steering. Just like the braking system, the linear actuator will use a Pololu Jrk G2 18v9 motor controller allowing for precise control of the vehicle's steering angle . Similarly, we will continue to use the hardware stops to set maximum and minimum extension. This will eliminate the need of limit switches used to detect full lock angle of the steering column.

7.3.3.2 Modes of operation and Safety

For steering we will have 2 modes of operation that dictate the manner in which steering occurs and what source the steering control comes from. In normal operation, we will be receiving steering control from the dynamics control node. During this mode the

dynamics control will be processing waypoints from the navigation subsystem [Section 6.3](#), during which it will generate steering angle for the car to be turning. In the critical braking mode, steering control may be given to the perception subsystem. In the event of a potential collision, the perception subsystem may choose to provide a steering direction that may help avoid the obstacle.

The steering system will operate in two distinct modes, depending on the situation the vehicle is in.

- Normal Operation Mode

In this mode, steering commands will be generated by the dynamics control node, which will process waypoints from the navigation subsystem [Section 6.3](#). The dynamics controller will calculate the appropriate steering angle to navigate the vehicle along its intended path.

- Critical Control Mode

In the case of a critical, such as a potential collision with an obstacle or person, control of the steering system may be temporarily transferred to the perception subsystem. The perception subsystem can provide immediate steering commands to avoid the obstacle if possible.

7.3.3.3 Testing Plan

A comprehensive suite of tests will be created to ensure the safety of the steering system, independently and integrated with the full vehicle. Independent tests will verify the full-range of steering motion, steering response latency, and transition into critical control mode. As part of integrated system testing, we want to evaluate steering performance during both modes. A detailed outline of the test plan can be found in [Section 7.3.4](#).

7.3.4 Speed Sensors

An important component of the autonomous vehicle is speed sensing. Accurate speed feedback is crucial for several subsystems, including the dynamics controller (to calculate acceleration, braking and turning values), sensor subsystem (to detect potential collisions), telemetry display and various other subsystems. A precise speed measurement is essential for proper operation of these subsystems. To achieve this, we will use two Littelfuse 55505 Hall Effect gear tooth sensors along with hall effect sensors built into the vehicle's drive motor.

7.3.4.1 Physical Design

The gear tooth hall effect sensor operates by detecting changes in magnetic fields. When a ferromagnetic object is detected in front of the sensor, the change in magnetic field is detected. By mounting a moving ferromagnetic object, pulses generated when the wheel rotates can be detected and used to calculate the rotational speed of the gear and converted to linear speed.

For this we will design a gear that can be mounted to the wheels between rotor and the tire to allow for precise speed sensing. Initially a prototype will be created using 3D printing, with ferromagnetic disks embedded to allow for the sensor to detect motion. The sensor will be positioned close enough to detect the ferromagnetic objects without touching the gear. Similarly the hall effect sensor in the motor operates similarly detecting rotational motion of the magnets in the motor itself, and generating pulses when moving. A prototype gear is shown below in Figure 16. The gear has slots in each tooth to embed a disk of any ferromagnetic material. This design may change based on testing and availability of resources.

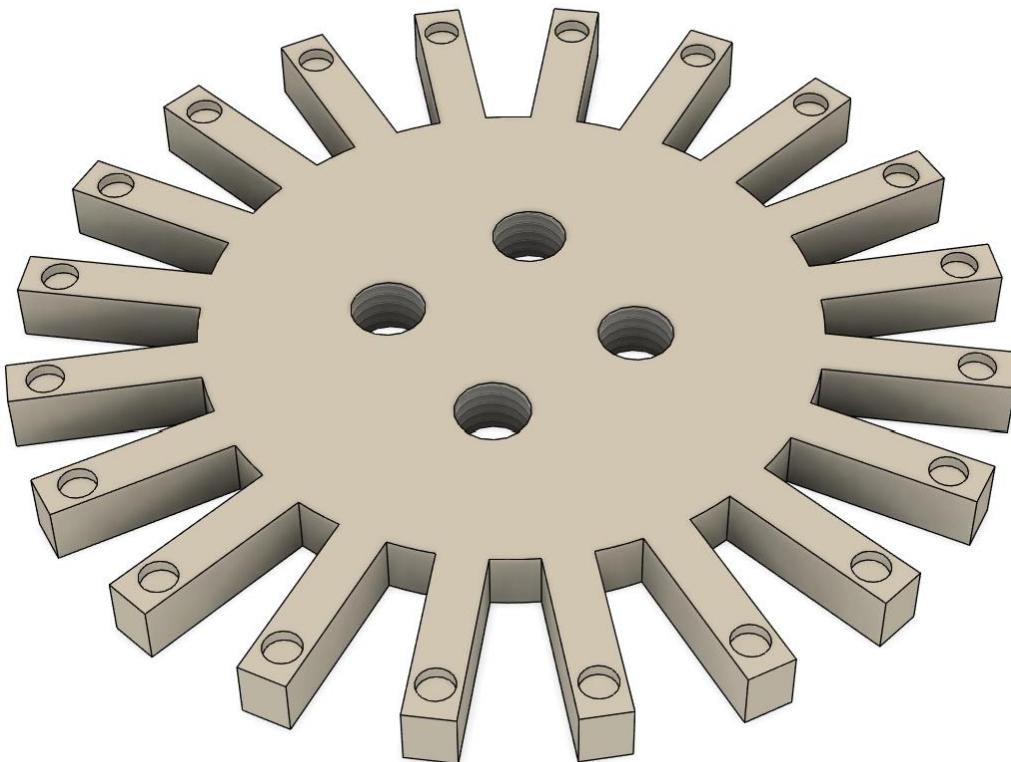


Figure 18: Prototype gear tooth

7.3.4.2 Signal Processing and Speed Calculation

Once the hall effect sensor has been mounted we can process the pulses generated by the rotating gear and calculate the rotational velocity. An Arduino microcontroller can be used to detect pulses, via hardware interrupts to measure the time between each tooth. The number of pulses in a given second can be used to calculate the revolutions per minute of the wheel (RPM) with the given formula.

$$RPM = \frac{60 * Pulse\ Frequency}{Sensor\ Pulses} [67]$$

- *Pulse count: Number of pulses during the given time interval*
- *Pulse Frequency: Number of pulses per second*
- *Sensor Pulses: Pulses in a full rotation (equal to the number of teeth on the gear)*

Once the RPM has been calculated, the linear velocity of the vehicle can be determined using the following formula.

$$v = 2\pi * R * \frac{RPM}{60} [68]$$

- R: Radius of the gear at the sensors position
- V: Velocity in meters/second

This calculation provides the speed for each individual wheel. The data from the front two wheels and the drive motors' speed sensor can be averaged to provide an overall speed of the vehicle. Calibration will be required to achieve accuracy of measurements and ensure the sensors output correct information, alternate sensors might be used in the calibration process.

7.3.4.3 Speed sensor interface

While calculating the average speed would be useful for a simple speedometer, each sensor's data will be crucial for more complex dynamics control. ROS Nodes will be exposed to publish speed telemetry, a ROS topic will provide an interface for all values including, RPM of each front wheel, linear velocity of each front wheel, RPM of drive motor, linear velocity of drive motor and average velocity of vehicle. This ROS topic can be subscribed to by any subsystem in the network ensuring all subsystems have speed data for the vehicle.

7.3.4.4 Testing Plan

Target Subsystem	Description	Pass Criteria
Braking	Verify braking actuator can move through entire range of brake lever	Actuator reaches maximum braking and minimum braking (no brake pressure). Based on physical testing, extract maximum and minimum braking feedback value
Braking	Measure response time of the braking system to braking command	Time to achieve a braking value is less than 0.2 seconds.
Braking	Braking system performs full stop in critical braking mode	Braking lever is set to maximum braking and does not respond to braking inputs from other subsystems.
Braking	Verify current draw does not exceed of power draw limit	Power and current draw remain within 15A under max load
Braking	Verify response to E-Stop	Brake lever is set to maximum braking and does not respond to other braking inputs.
Braking	Verify brake release works	Brake lever is set to minimum braking and does not respond to other inputs.
Braking	Verify various modes function with given priorities	Braking system responds to priorities of modes based in following order: Brake Release, E-Stop, Critical Braking, Normal Operation
Steering	Steering actuator is able to achieve full range of steering	Confirm the steering actuator can reach both maximum left and right positions
Steering	Steering actuator feedback has been accurately calibrated to steering angle	Steering linear actuators' feedback mapping to angle is accurate and position is within 2 degrees of actual angle
Steering	Verify steering response time is under maximum threshold	Steering can achieve specified angle within 0.3 seconds
Steering	Verify current draw does not exceed of power draw limit	Power and current draw remain within 15A under max load
Steering	Steering responds correctly under E-Stop	Steering does not respond to any commands when E-Stop is activated
Steering	Steering can respond to 2 sources of input with priority given	Steering responds to inputs simulating the navigation system and takes priority over a normal source of braking.

Speed Sensing	Pulse Detection	Hall effect sensor pulses can be detected with accurate time measurement between each pulse
Speed Sensing	Linear speed calculation	Confirm the pulses can be converted to RPM value and linear speed of the wheel
Speed Sensing	Verify speed measurement	Verify calculated linear speed with an alternate sensor to validate calculated value
Speed Sensing	Speed Range measurement verification	Speed sensors is able to measure at high and low speeds
Speed Sensing	ROS Node Exposed	ROS Node is exposed to provide speed and RPM values for each sensor
Dynamics Control	Connected to all required ROS nodes.	All required parameters are available and being updated.
Dynamics Control	Software unit tests.	All unit tests built in to the software run and do not produce any errors.
Dynamics Control	Steering, braking and acceleration commands carried out.	Each control command produces expected changes to sensor values. For example: set steering angle to 10 updates the steering angle sensor value to ~10.
Motor Control	Check for error codes	No error lines or codes produced by motor controller.
Motor Control	Hall effect sensor valid	Hall effect sensors are not in a configuration that indicates a problem. For example if all three hall effect sensors read a low value at the same time, this would be an error.
Motor Control	Velocity control is accurate.	The reported velocity from the wheel speed sensors is close to the setpoint of the motor speed controller.

Table 7: Unit test plan

Target Subsystem	Test Name	Pass Criteria
Braking	ROS Communication is functioning	ROS nodes publish braking percentage and topic is created to allow users to set braking value.
Braking	Verify dynamics controller can control braking	Dynamics controller can set braking values based on calculated braking value
Braking	Detected obstacle enabled critical	When an obstacle is detected and may hit vehicle,

	braking mode	critical braking mode is activated
Braking	Communication delay between subsystems	Communication between subsystems takes less than 60 ms.
Steering	Steering Angle Feedback	Steering systems' ROS node provides steering angle feedback and can be read by other subsystems
Steering	Steering Latency Test	Steering commands from any subsystem takes less 0.1 seconds to arrive to steering subsystem
Steering	Obstacle Avoidance	When the sensor subsystem detects a potential crash, the car steers away from obstacles.
Speed Sensing	Speed measurement communication	Speed sensor values can be read by all subsystems.

Table 8: Integration test plan

7.4 Relation to Degree

All members of the autonomous control subgroup are enrolled in Computer Systems Engineering. Given this, the following is a list of key skills from our degree applied to the project.

- Embedded Systems Programming
 - Programming with embedded microcontrollers such as the Arduino, motor controllers and Raspberry Pis, used for braking and steering.
 - Using the real-time interrupt based programming for speed sensors
- Signal Processing
 - Analyze signals from hall effects sensors and drive motors to achieve precise measurement
 - Real-time signal processing for quick measurements
- Control Systems and Dynamics
 - Applying control systems to manage steering, braking and acceleration of the vehicle
 - Applying control systems in conjunction with other systems inputs to move vehicle to specified points
- System Integration and ROS
 - Use ROS for effective communication between nodes and subsystems. Creating our own ROS nodes and topics to enable communication
- Circuit Design and Prototyping
 - Design efficient circuits to limit power usage for protection of sensors and actuators and extending battery usage.
 - Design a custom 3D part in Fusion 360 to securely attach a linear actuator to brake lever and gear to enable speed sensing.
- Testing and Validation
 - Planning and executing independent and integration test plans for braking, steering and speed-sensing subsystems

7.5 Group Skill

Person	Skills
Mohammad Saud	<ul style="list-style-type: none">• Experience with firmware, embedded system, device drivers, verification scripts and programming Arduinos and Raspberry Pi• Strong technical problem solving skills and troubleshooting hardware issues• Proficient with C, Python, Java, C++ and Assembly
Charlie Wadds	<ul style="list-style-type: none">• Experience with development of control systems software for nonlinear dynamical systems with real time performance.• Experience with design of embedded systems components for robotics test setups.• Experience with 3D design and printing for robotic test setups.• Proficient in C/C++, Verilog, Python and Java.
Abed Qubbaj	<ul style="list-style-type: none">• Experience with embedded systems, Arduinos, raspberry pi• Skilled with C, Java, Python and Assembly

7.6 Methods

The following section will outline the methods we used to work on the project. These methods provide an insight to our approach to designing the autonomous control subsystem and ensuring our system works cohesively and reliably as part of the autonomous vehicle.

- **Agile Development and Iterative Design:** We have used an agile development approach, breaking down each project phase into small, manageable milestones. Each milestone represents a functional part of the vehicle. Each subsystem has multiple milestones, to track progress and ensure we are developing iteratively to spot issues early.
- **Modular System Design:** Components and subsystems have been designed to be modular, allowing each to be developed and tested independently before integrating into the full system. This approach allows to ensure an issue with a sub system does not impact others and allows to change system or implementation without affecting other systems
- **Rigorous Testing and Validation:** To verify functionality and ensure safety, each system has a suite of unit tests, integration tests and full system tests. This ensures each system work independently, integrated testing to ensure systems work together and full system testing to ensure the varying operation conditions to not affect performance and functionality
- **Documentation:** Detailed documentation of each subsystems system architecture, control algorithms and testing plans to ensure future iterations of the project have the tools needed to build off our work

7.7 Timetable

The following section will outline the key milestones and estimated timeline of the development of the autonomous control subsystem. The project will be split into 6 phases including Planning, System Design, Procurement, Development, Integration and Testing. Each phase outlines some primary objectives to be achieved during development. Figure 17 shows a work breakdown structure.

7.7.1 Work Breakdown Structure

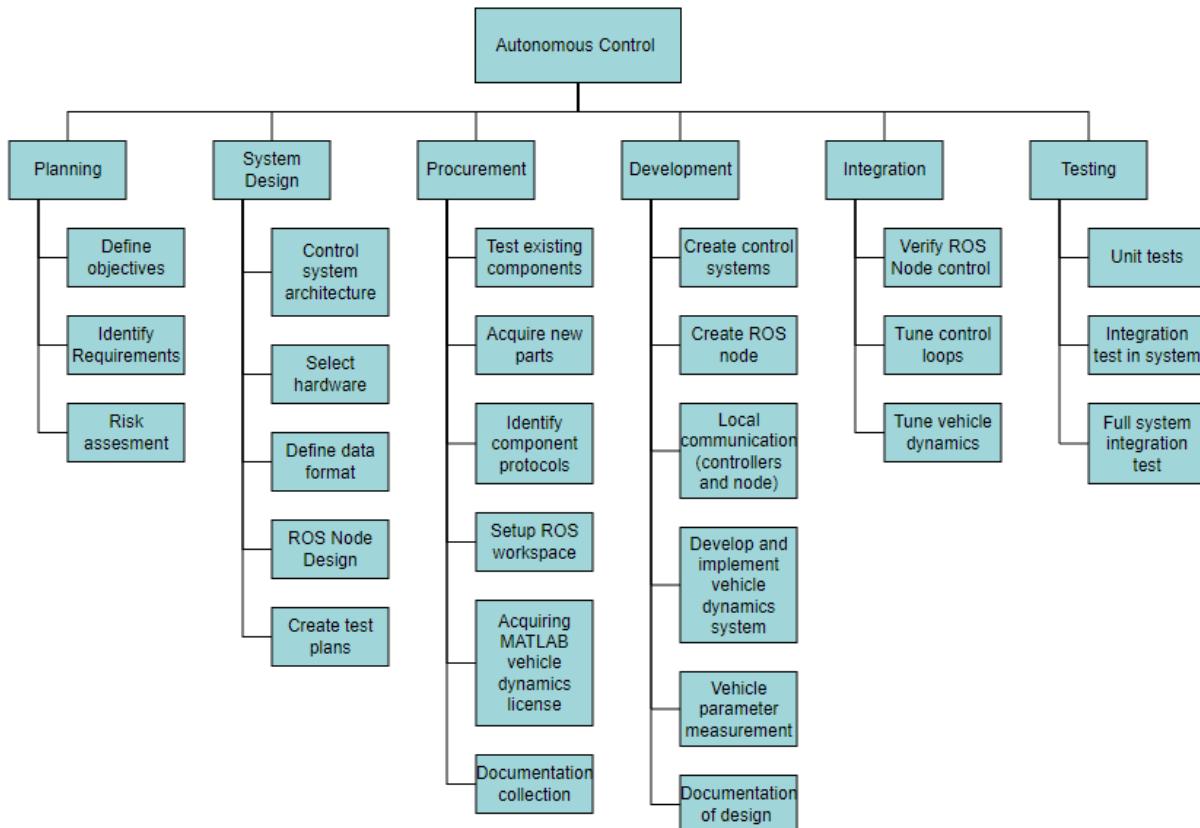


Figure 20: Work Breakdown Structure

7.7.2 Phase Breakdown

- Planning
 - Define objectives: Establish project goals and overall scope
 - Identify Requirements: Identify all functional and non-functional requirements
 - Risk Assessment: Identify potential risks and mitigation strategies
- System Design
 - Control System Architecture: Develop the framework for control systems
 - Select Hardware: Choose components for sensors, actuators and processors
 - Define data format: Specify formats for data transmission
 - ROS Node Design:
 - Create Test Plans: Outline procedures for verifying system functionality

- Procurement
 - Test existing components: Assess current parts for compatibility
 - Acquire new parts: Obtain additional parts as specified
 - Identify component protocols: Determine communication protocols for each component
 - Setup ROS Workspace: Configure the workspace for node integrate
 - Acquire MATLAB library license: Acquire license to MATLAB vehicle dynamics library
 - Documentation: Acquire documentation for MATLAB library and ROS system
- Development
 - Create control systems: Develop and code system control algorithms
 - Create ROS node: Build ROS nodes for each subsystem
 - Local communication: Implement communication between local microcontrollers and autonomous control central node
 - Develop and implement vehicle dynamics system: Create dynamics and movement controls
 - Vehicle parameter measurement: Collect measurements for vehicle tuning
 - Documentation of design: Record design details and code
- Integration
 - Verify ROS Node control: Confirm ROS node communication works between subsystems
 - Tune control loops: Adjust controllers for optimal response
 - Tune vehicle dynamics: Refine dynamics for accurate vehicle movement
- Testing
 - Unit tests: test each subsystem individually for functionality
 - Integration tests: Assess interaction between subsystem
 - Full system integration: Validate the entire system in a real-world scenario

7.7.3 Phase Deadlines

The following table provides deadlines for each phase of development. Milestones have been set to complete earlier to allow for some redundant time in case risks cause delays.

Phase	Deadline
Planning	Start of October
System Design	End of October
Procurement	Mid November
Development	End of January
Integration	Mid February
Testing	End of February

Table 9: Phase Deadlines

7.8 Risk Assessment

Risk assessment is critical for the project, ensuring that potential challenges have been identified and managed proactively. This section outlines any risks associated with the tasks of the autonomous control sub group. For each risk, mitigation strategies have been outlined to reduce impact on the project deadline.

	Risk	Mitigation
Component Failure or Physical Damage	Components used may fail or damaged during system	Regularly ensure that parts are functioning and follow specifications from documentation
Delays in Procurement	Delays in parts procurement	Order components early, develop with previous components for redundancy
Software Bugs	Unexpected software issues	Ensure latest version of ROS is used, perform incremental tests and have at least one person review all new code
Insufficient Power	Insufficient power delivery from batteries	Test actuators with real steering load to determine current draw.
Communication Latency	High latency between subsystems communication	Optimize communication protocols, test with high load and prioritize critical data channels

Table 10: Risks and mitigation strategies

8. Reliability and Safety

8.1 Background

The project addresses key challenges in the deployment of autonomous vehicles (AVs), particularly in safety-critical scenarios where current systems can be vulnerable to unpredictable factors such as sensor inaccuracies and software glitches. There has been significant research in

AV safety mechanisms, yet issues persist when AVs encounter unexpected traffic or environmental conditions. Our project builds upon existing safety mechanisms by focusing on robust solutions that ensure fault tolerance and quick responses during emergencies [72].

8.2 Objectives

The primary objective of this project is to enhance the safety and reliability of an autonomous vehicle. This will be achieved by developing and integrating emergency stop (E-stop) mechanisms, a remote monitoring system, and manual override functionality. The measurable goals include:

- Design and implement reliable on-car and wireless E-stop systems.
- Develop real-time remote monitoring for fault detection and vehicle diagnostics.
- Ensure seamless manual override for human control in emergencies.
- Achieve full integration of safety systems with other vehicle components [70].

8.2.1 Functional Requirements

- **E-Stop:**
 - In the event of an Emergency Stop, the power to the drive must be cut-off and the brakes must be applied.
 - The autonomous vehicle must be able to emergency stop with an attached e-stop button, a wireless e-stop button, and through ROS2.
 - The autonomous vehicle must not be removed from an emergency state unless each source of emergency stop has been removed.
- **Remote Monitoring:**
 - The remote monitoring interface must display critical information about the vehicle's operation in real time.
 - The interface must provide the option to trigger an emergency stop.
 - The system must deny access to the remote interface for any unauthorized connections.
- **Manual Override:**
 - The autonomous operation of the autonomous vehicle must be able to be interrupted by manual user input.
 - Manual override shall be engaged through the use of dedicated equipment and nothing else.
 - Autonomous operation of the vehicle must resume after a period of unresponsiveness from the dedicated equipment.
- **Heartbeats:**
 - The system must monitor heartbeat messages published by other subsystems to ensure system functionality.

- The system must detect lost heartbeats and restart subsystems in the event of subsystem failure.
- The system must record the elapsed time since the last heartbeat to estimate subsystem health.
- Heartbeat metrics must be displayed in real-time on the remote monitoring interface.

8.2.2 Non-Functional Requirements

- In the event of an e-stop, the power to the motor must be cut-off within 40ms
- In the event of an e-stop, the brakes must be applied within 200ms.
- The remote interface shall be designed to be highly accessible, using high contrast and large icons for visibility and interactivity.
- The remote interface must respond to interactions from the user within 1 second.
- The system must release its manual override after 30 seconds of inactivity from the dedicated equipment.
- Heartbeats from subsystems must be published every 100ms +/- 20ms.
- Up to 5 heartbeats can be missed from a given ROS2 node before it shall be rebooted.
- External connections to the autonomous vehicle must be encrypted to maintain confidentiality, integrity, and availability [71].

8.3 Plan

Our project focuses on enhancing the safety and reliability of autonomous vehicles by developing three key systems: emergency stop mechanisms (both on-car and wireless), a remote monitoring system, and a manual override system. The project plan can be divided into distinct phases, strongly emphasizing rigorous testing and integration across all safety mechanisms.

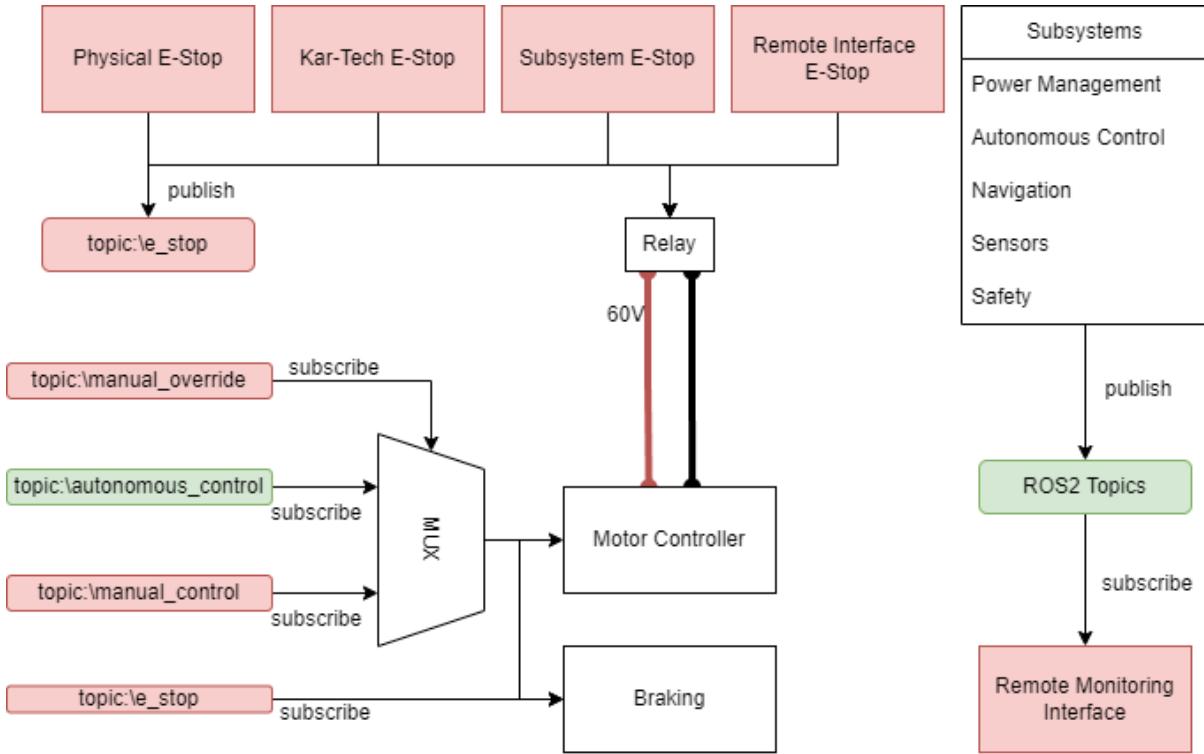


Figure 21: Overall system

8.3.1 E-Stop Systems

The emergency stop (E-stop) system is a critical safety feature that ensures immediate shutdown of the vehicle in case of an emergency. We will design two types of E-stop systems:

- **On-Car E-Stop Button:** This physical button will be mounted on the vehicle and will be clearly marked as per ISO 13850:2015. When pressed, it will cut off power to the drive motors and engage the braking system within 40 milliseconds. This immediate response is vital in preventing accidents or mitigating system failures.
- **Wireless E-Stop Button:** In addition to the physical button, a wireless emergency stop system will be developed. This allows operators or remote personnel to stop the vehicle in emergency situations without being in physical proximity. The wireless E-stop will have secure, encrypted communication to ensure no unauthorized activation, and it will trigger an immediate response similar to the on-car system [73].

Implementation Plan:

- **Design Phase:** Develop circuit designs and select components that meet safety and response time requirements.
- **Prototyping Phase:** Assemble and prototype the physical and wireless E-stop systems, ensuring compatibility with existing vehicle hardware.

- **Testing Phase:** Run stress tests on both systems to measure response time, reliability, and fail-safes under various scenarios.

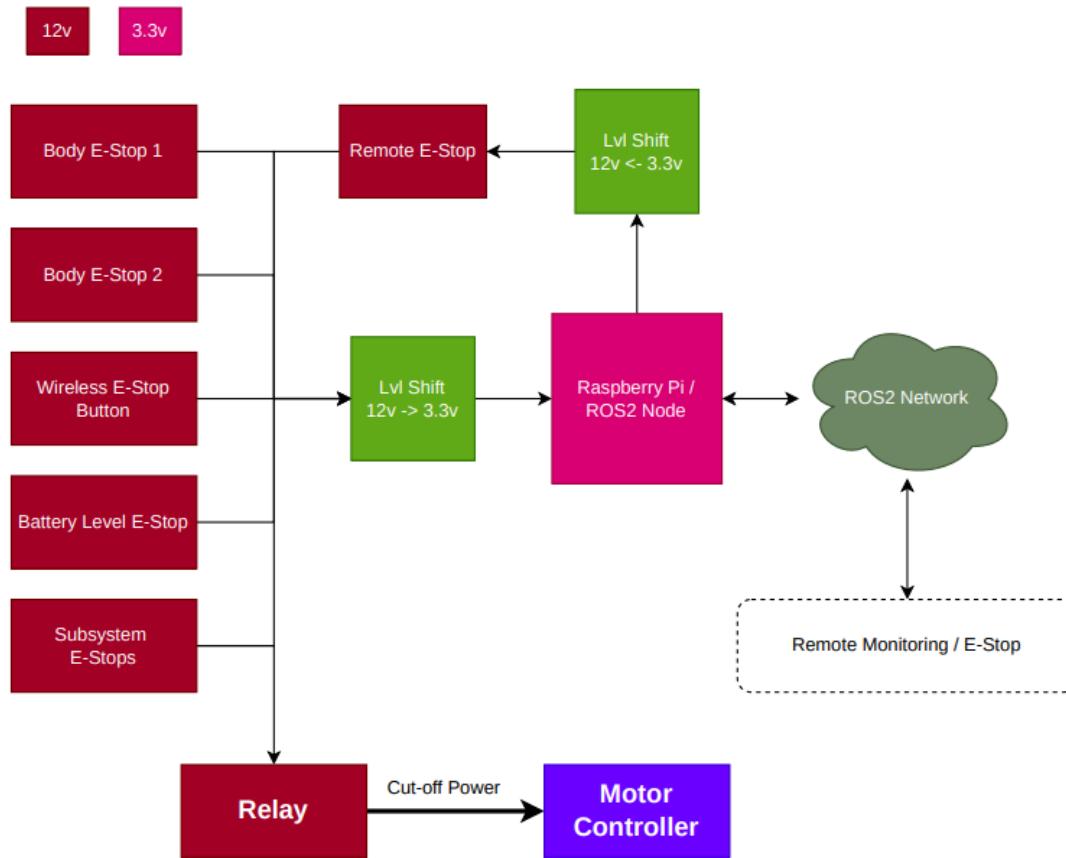


Figure 22: Flowchart of E-Stop System

8.3.2 Remote Monitoring System

A comprehensive remote monitoring system will be developed to provide real-time data on the vehicle's status. This system will track the availability of critical components, including sensors, actuators, power systems, and communication modules. It will also log fault diagnostics and performance data for post-incident analysis. This system will allow operators to remotely access vehicle information, detect issues before they escalate, troubleshoot components using logged system data and intervene if necessary [74].

Implementation Plan:

- **Design Phase:** Develop a software architecture for real-time data logging and display. Identify key vehicle metrics to be monitored (e.g., sensor readings, battery status, fault indicators).
- **Development Phase:** Create the software for real-time monitoring, ensuring it has a secure connection to the vehicle's onboard system. Develop the user interface for remote operators.
- **Testing Phase:** Test the system under various operational conditions to ensure reliability. Conduct performance and security testing to safeguard against data breaches.

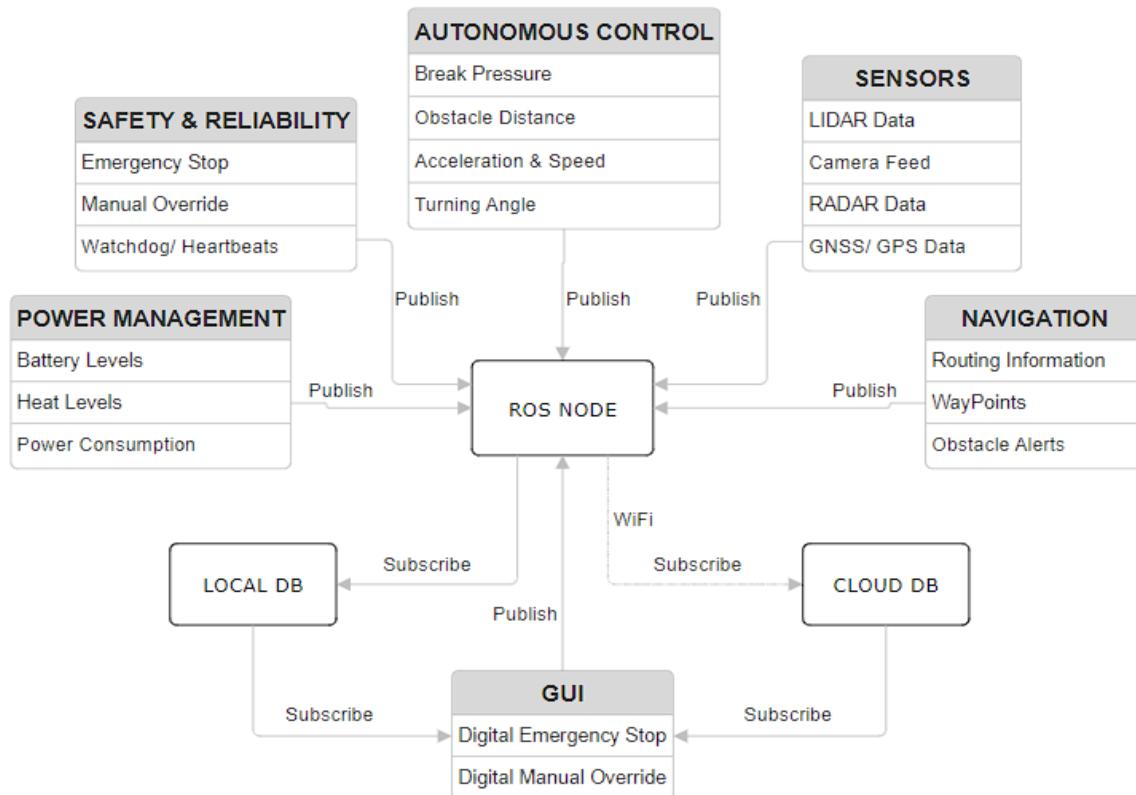


Figure 23: Flowchart of Remote Monitoring

The login screen features a title "Login" at the top. Below it is a text input field labeled "Enter your Username/ID". Underneath is another text input field labeled "Enter your Password". At the bottom is a large green "Login" button.

Figure 24: Sample Snippet of Login Page of the GUI

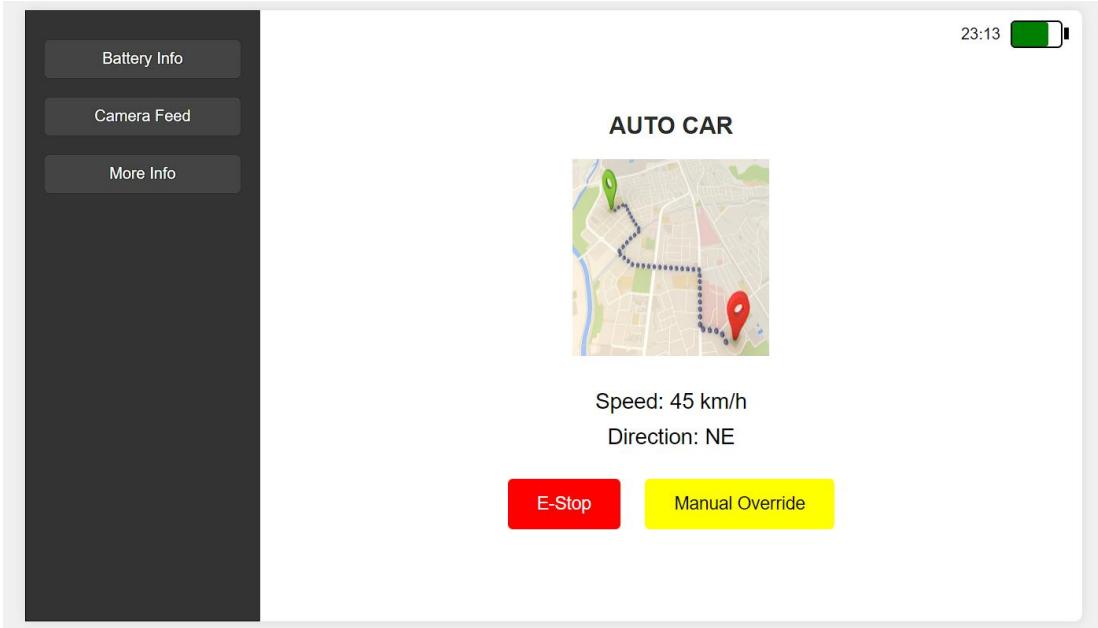


Figure 25: Sample Snippet of Main Page of the GUI

8.3.3 Manual Override System

The manual override system is an essential feature that ensures the human operator can take control of the vehicle at any time, overriding the autonomous system. This feature is crucial during system failures or emergencies where manual intervention is necessary. The interface will allow the user to switch between manual and autonomous modes seamlessly, ensuring smooth transitions without latency or lag [75].

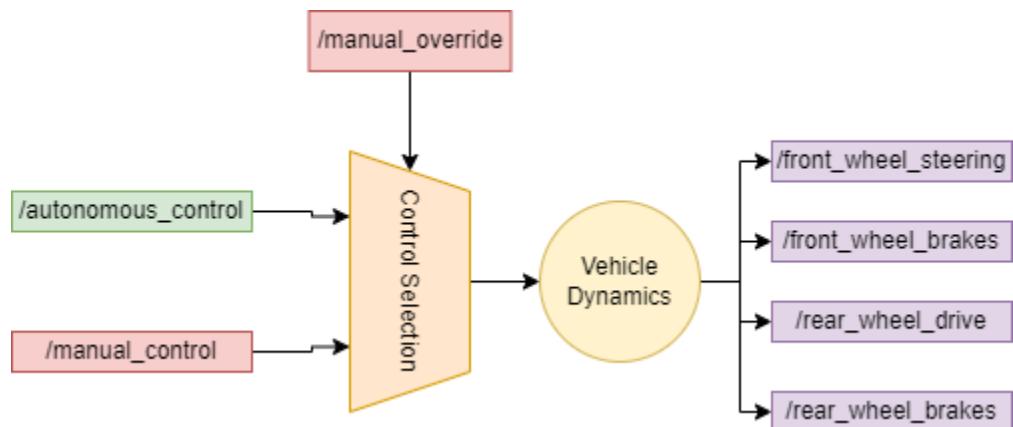


Figure 26: Flowchart of manual override system

8.3.3.1 RC interface

There will be two different interface implementations for manually overriding and controlling the car. The first method will utilize an RC controller that will act as a radio transmitter and receiver to receive the RC signals sent. This will provide a reliable method to control the vehicle in the case of erroneous action from the autonomous control functions by allowing the user to control the vehicle. Since RC signals can be sent and received from a large distance, it would provide a more reliable connection compared to Bluetooth or Wifi. The RC interface will also be easy to operate, as the receiver module will be constantly listening to the RC signals.

The RC controller will work by sending a PWM signal through a specific channel on the radio, and the receiver will be constantly listening to this signal on the channel. The receiver will be connected to an Arduino to process the PWM signal and convert it to a value that can be readable by the control ross node.[76]

8.3.3.2 GUI and joystick interface

The second interface for manually overriding the vehicle would be utilizing a joystick that will be mounted in the cabin of the vehicle, along with a GUI interface using simple graphics as seen in figure. The graphics will display the different inputs of the car such as steering angle, braking level, and throttle to aid the driver in controlling the vehicle. A watchdog timer will also be implemented to improve the reliability of the system, in case the GUI or central controlling system fails. This would prevent the vehicle from getting stuck in a certain position during operation, and allow it to reset the control.

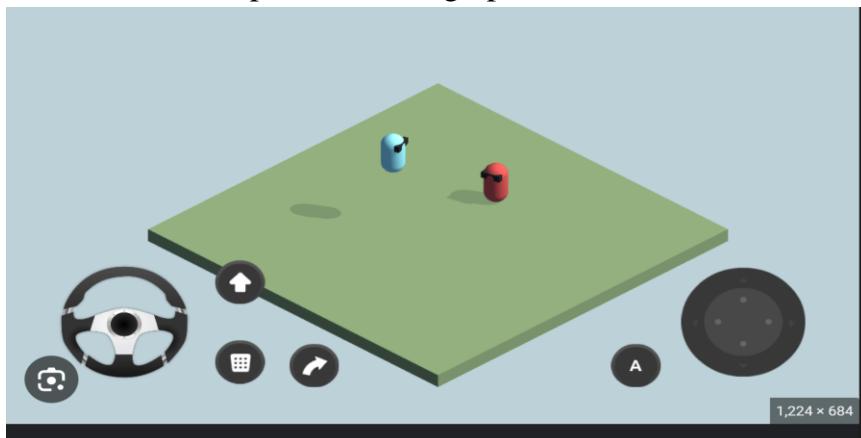


Figure 27: GUI interface example graphic

Implementation Plan:

- **Design Phase:** Design a user-friendly control interface that allows for easy transition between manual and autonomous control. Focus on human factors and ease of operation, ensuring the system is intuitive for operators.
- **Development Phase:** Integrate the manual control system with the vehicle's existing hardware and software. Ensure that manual inputs instantly override autonomous decisions.
- **Testing Phase:** Conduct extensive testing to confirm that the manual override works under all circumstances. Simulate various failure conditions to validate the system's effectiveness.

8.3.4 Integration with Other Vehicle Systems

Throughout the development process, integration with other subsystems—such as navigation, perception, and vehicle control—will be critical. Regular collaboration with other teams working on the autonomous vehicle's navigation and perception systems will ensure that all components work together smoothly. Communication protocols through ROS nodes will be established to ensure interoperability between the safety mechanisms and the vehicle's decision-making systems [77].

Implementation Plan:

- **Integration Phase:** Coordinate with teams responsible for navigation and control to ensure compatibility. Test interactions between the safety systems and other vehicle subsystems, ensuring that safety features do not interfere with or delay autonomous operations.
- **Validation Phase:** Conduct full-system integration tests where the E-stop, remote monitoring, and manual override systems interact with the vehicle's navigation and control systems in real-time scenarios.

8.3.5 Heartbeats

To keep track of whether subsystems in a distributed system are functioning as expected, additional monitoring is required. This monitoring is more involved and requires subsystems to periodically confirm to a designated “listener” subsystem that it is functioning as expected. This is commonly referred to as a Heartbeat. By having ROS nodes in the autonomous vehicle publish a heartbeat to the listener, it is possible to verify whether those nodes are working as expected [85]. If a node in the autonomous vehicle is stuck in an indeterminate state such as an infinite loop, it is absolutely critical that this be detected and rectified.

Implementation Plan:

- **Design Phase:** Explore existing ROS 2 packages that offer heartbeat and other utilities for monitoring the health of subsystems or develop a custom package allowing finer control over heartbeat interval, system behaviors, and recovery options.
- **Integration Phase:** Coordinate with all teams and have them include the yet-to-be decided heartbeat package into their nodes. Ensure that messages from all nodes publishing to the heartbeat topic are being received by the listener subsystem. The remote interface should visualize the heartbeats on the GUI.
- **Validation Phase:** Confirm that subsystems are capable of recovering from faults and that heartbeat messages are correctly sent and received at the predetermined heartbeat interval within a margin for error. Node failure recovery can be tested by forcibly disabling a node and observing how the listener responds.

8.4 Relation to Degree

This autonomous vehicle project aligns closely with the core principles and learning outcomes of our engineering program, particularly in the areas of systems design, electronics, embedded systems, and control theory. It offers an opportunity to apply both theoretical knowledge and practical skills gained through our coursework to a complex, real-world problem [78].

1. Embedded Systems and Microcontroller Design: Our program has provided a strong foundation in embedded systems, a key area of focus for this project. We will leverage our experience in programming microcontrollers (like Arduino) to design reliable safety systems such as the E-stop and manual override. The technical knowledge we've gained from courses on digital logic design, microcontroller interfacing, and sensor integration will be instrumental in implementing these systems. For example, we will apply what we've learned about real-time processing, secure communication protocols, and fail-safe mechanisms to ensure that the E-stop and manual override functions are reliable and respond promptly in emergency situations.

2. Control Systems and Automation: Courses in control systems, particularly those focusing on feedback control and system stability, directly contribute to the design and development of the manual override system. In this project, we must ensure that the transition from autonomous to manual control is smooth, instantaneous, and safe. Understanding how to model dynamic systems and design control loops will enable us to effectively integrate manual intervention without disrupting vehicle stability. The project also gives us the chance to apply control algorithms to manage emergency stop actions, braking controls, and system monitoring—skills learned in automation and mechatronics courses [79].

3. Systems Integration and Design Methodology: The process of integrating various subsystems, such as the E-stop, remote monitoring, and manual override, with the vehicle's navigation and perception systems will draw heavily on our training in systems engineering and

design methodology. The knowledge of systems-level thinking, gathered from design projects in courses like SYSC 4805 and capstone design labs, will help us tackle the challenges of interoperability. Ensuring that our safety systems communicate effectively with the broader vehicle architecture requires an understanding of hardware-software co-design, communication protocols, and signal integrity—concepts thoroughly covered in our program [80].

4. Risk Assessment and Safety Engineering: Our program has emphasized the importance of risk assessment, especially in safety-critical systems, which is a key aspect of this project. The principles of safety engineering, hazard identification, and risk mitigation will guide us in developing reliable E-stop systems and other safety features. Through our experience in previous labs and projects, we've learned how to analyze potential failure points, implement redundancies, and conduct rigorous testing to validate system safety. This knowledge will be critical when conducting safety tests and system validations, ensuring that our designs meet high reliability standards [81].

5. Real-Time Data Processing and Networking: In designing the remote monitoring system, we will apply skills from networking, real-time systems, and data processing courses. These courses have equipped us with the ability to develop real-time communication systems that can gather and display sensor data, diagnose vehicle health, and log faults for post-incident analysis. Additionally, our knowledge of secure communications, developed through coursework in cybersecurity and network protocols, will be essential in ensuring that the remote monitoring system is both effective and secure, preventing unauthorized access to the vehicle's data and control systems [83].

8.5 Group Skill

Collectively, our team has the necessary skills to undertake the project. Each team member brings expertise from various fields:

Table 11: Skills of team members

Team Member	Skills
Ethan Bradley	Circuit Design Academic experience from courses ELEC 2501 and ELEC 2507, hands-on experience covering BJTs, MOSFETs, passive components, mitigating load effect, and signal transmission. PCB Design & Construction Self-taught using open source tools such as KiCad. Trace routing and component placement. Self-taught in soldering components to a manufactured PCB and Perfboard.

	<p>Embedded systems Academic experience in programming and testing the MSP432 Launchpad in SYSC 3310. Personal experience with common boards such as the Arduino UNO, ESP32, and various RP2040 microcontroller boards.</p>
Rahul Cheruku	<p>Systems Engineering Academic experience from courses like SYSC 4805 and hands-on project work in designing, integrating, and validating complex systems..</p> <p>Embedded Systems Experience from programming microcontrollers such as Arduino and working with embedded platforms. Familiar with sensor integration, motor control, and hardware-software co-design from academic projects and personal experiments.</p> <p>Safety Systems Design Extensive project experience in designing fail-safe systems like emergency stops (E-stop) and manual override mechanisms. Knowledge of fault-tolerant design principles, real-time decision-making, and redundancy implementation.</p> <p>Communication Systems Experience in implementing secure and reliable communication protocols for wireless control systems, including encrypted communication for remote monitoring and E-stop functionality.</p>
Ali Nadim	<p>Embedded systems: Developed and programmed embedded systems using Raspberry Pi and Texas Instruments boards for safety features like E-stop and manual overrides.</p> <p>Software Development: Proficient in Python, C, and Java for automating tests and developing control systems.</p> <p>Hardware Design:</p>

	<p>Skilled in circuit design and troubleshooting using lab equipment like oscilloscopes and multimeters for reliable system performance.</p> <p>Team Collaboration: Led projects and coordinated team efforts to meet deadlines for safety system implementations.</p>
Ali Zaid	<p>GUI Design Hands-on experience in developing web-based user interfaces using HTML, CSS, JavaScript, React, TypeScript and Next.js. Created interactive and responsive designs for client-based applications that support real-time data visualisations.</p> <p>Database Management Proficient in managing databases with SQL, NoSQL, SQLite and Firebase solutions. Integrated data storage and retrieval systems for web platforms, ensuring seamless interaction between front-end interfaces and back-end databases.</p> <p>Systems Integration Skilled in integrating and testing software systems to ensure smooth operation of web and mobile applications. Experience includes implementing APIs, automating workflows, and ensuring cross-platform compatibility.</p> <p>Software Development Proficient in Python, Java, and C/C++ for developing web apps, automation scripts, and control systems. Worked on full-stack solutions, focusing on both front-end and back-end development.</p> <p>Embedded Systems Familiarity with programming embedded devices and microcontrollers. Experience includes working on projects involving Raspberry Pi and Arduino for developing applications in Python for automation and control purposes.</p>
Arjun Pathak	GUI Design

	<p>Academic experience designing user interfaces for monitoring systems using Java and SwingUI. Developed real-time dashboards that display system data like speed, elevator floor, and time for an elevator systems project.</p> <p>Database Management</p> <p>Experience in managing and integrating databases using SQL and SQLite for storing and retrieving monitoring data.</p> <p>Systems Integration</p> <p>Experienced in testing, configuring and integrating systems (e.g., SCADA, PIDS, IAC) to ensure accurate data flow and reliable system functionality.</p>
--	--

8.6 Methods

We will employ a variety of engineering methods and processes to design, test, and implement the safety and reliability systems for the autonomous vehicle. The key methods we will utilize include:

1. **Systems Engineering Approach:** A structured systems engineering approach will guide the design, development, testing, and integration of all safety systems. This approach ensures that each subsystem—such as the E-stop, remote monitoring, and manual override—works seamlessly with one another and integrates with the vehicle's overall architecture. By following a step-by-step, iterative design process, we will ensure that the safety systems meet the project's functional and non-functional requirements while addressing potential risks. This approach allows us to define clear requirements, develop system specifications, and verify that the solutions align with our safety objectives at each stage of the development cycle.
2. **Simulation and Modeling:** Before implementing the safety systems, we will create simulation models to analyze the behaviour of the vehicle under various emergency scenarios. These simulations will allow us to test the interaction between the E-stop systems, sensor responses, and the manual override mechanisms in a controlled environment. This phase will also help us evaluate how the system handles edge cases and extreme conditions like sensor failures, inclement weather, and system overloads.
3. **Prototyping:** We will develop physical prototypes of the safety systems, including the emergency stop (E-stop) buttons and the manual override mechanisms. These prototypes will allow us to assess the real-world performance of the systems and make necessary

adjustments based on their behaviour during early-stage testing. The prototyping phase will provide an opportunity to evaluate the hardware components, such as the physical E-stop button's placement and the wireless range of the remote monitoring system, ensuring these components meet industry safety standards. It will also allow us to conduct user testing to ensure the manual override interface is intuitive and effective [84].

4. **Testing:** Extensive testing is crucial to ensure the reliability and effectiveness of the safety systems under different operational and environmental conditions. We will conduct the following types of testing:

- Unit Testing: Testing individual components such as the E-stop, sensors, and communication modules to verify that each system behaves as expected in isolation.
- Integration Testing: Testing how the subsystems interact when integrated into the larger autonomous vehicle system. This will ensure that communication between the safety systems and the vehicle's navigation and control subsystems is seamless.
- Stress Testing: Pushing the systems beyond their operational limits to evaluate performance under high-stress conditions, such as extreme temperatures, heavy traffic scenarios, and hardware or software failures.
- Field Testing: Conducting real-world tests of the safety systems in various environments to verify their performance under actual operating conditions, ensuring that the systems perform reliably in diverse scenarios.

5. **Iterative Refinement:** After initial testing, feedback from simulations, prototyping, and field tests will be used to refine the safety systems. Each iteration will improve the design, address newly discovered issues, and ensure compliance with the defined safety standards. This iterative approach aligns with agile development practices, allowing for continuous improvement and adaptation to unforeseen challenges throughout the project lifecycle [86].

8.7 Timetable

The project will be carried out over an eight-month period, structured into distinct phases to ensure systematic development and thorough testing of the safety and reliability systems for the autonomous vehicle. Each phase has defined activities, key deliverables, and specific due dates, enabling effective tracking of progress and alignment with project objectives.

Table 12: Timetable for the tasks of the team

Phase	Activities	Deliverables	Due Date
Design Phase	Define system requirements, develop architecture, design safety systems	- System requirements document - Design specifications - Simulation setup	End of October

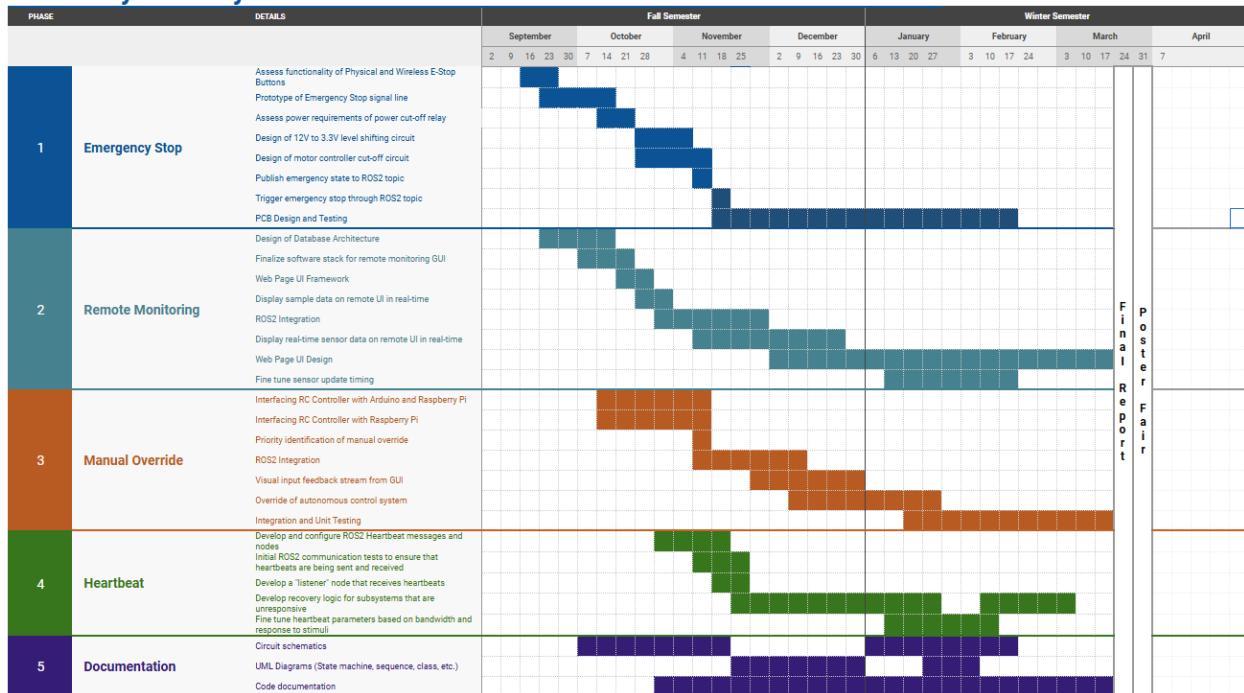
Development Phase	Prototyping and initial implementation of safety systems	- Functional prototypes - Embedded software development - Initial tests	End of December
Testing & Validation	Conduct unit, integration, and stress testing of safety systems	- Test reports (unit, integration, stress) - System refinements	End of January
Final Integration & Deployment	Full system integration and deployment for final testing	- Fully integrated system - Final validation reports - Project deployment	End of February

8.7.1 List of Activities

- Emergency Stop
 - Assess functionality of Physical and Wireless E-Stop buttons.
 - Assess power requirements of contactor relay.
 - Prototype of Emergency Stop signal line.
 - Design of 12V to 3.3V signal conversion circuit.
 - Design of power cutoff circuit.
 - ROS2 Integration
 - Publishing the emergency event.
 - Triggering an emergency event.
 - Integration and Unit Testing
- Remote Monitoring
 - Design of database architecture.
 - Decide on software stack for remote monitoring GUI.
 - Web Page UI Framework.
 - Display real-time sample data on remote user interface.
 - ROS2 Integration.
 - Display real-time sensor data on remote user interface.
 - Integration and Unit Testing.
- Manual Override
 - Interfacing RC Controller with Arduino and / or Raspberry Pi
 - Interfacing Joystick controller with Raspberry Pi
 - Priority identification of manual override.
 - ROS2 Integration.
 - Visual input feedback stream from GUI.
 - Override of autonomous control system.

- Integration and Unit Testing
- Heartbeats
 - Develop and Configure ROS2 Heartbeat messages and nodes.
 - Initial ROS2 communication tests to ensure that heartbeats are being sent and received.
 - Develop a “listener” node to receive heartbeats, track timings, and identify which signals are missed.
 - Develop recovery logic for subsystems that are not responding.
 - Integration and Unit Testing.
 - Fine tune heartbeat parameters based on test results.

Reliability & Safety Gantt Chart



Project timeline

8.8 Risk Assessment

Some potential risks and strategies for mitigating them include:

Table 13: Different risks and its mitigation plans

	Risk	Mitigation
Sensor Malfunction	Sensors may fail, leading to unsafe operations.	Regular diagnostics, redundancy with backup

		sensors, and fail-safe protocols to ensure safe operation in case of sensor failure.
Communication breakdown in wireless E-Stop	Wireless communication failure could prevent the E-stop from activating.	Use encrypted communication, backup wireless channels, and extensive range testing to maintain a reliable connection.
System Overload	High demand could overload the system, causing slow responses or crashes.	Implement load balancing, prioritise critical systems, and monitor system resources to prevent overloads.
Software Bugs	Bugs in safety-critical systems could lead to malfunction.	Use rigorous code reviews, automated testing, and implement fail-safe modes to mitigate software failures.
Environment Challenges	Weather conditions (rain, fog, snow) may affect sensors and control.	Use weather-resistant sensors, real-time weather data, and adaptive algorithms to adjust for environmental conditions.
Integration Delays	Delays or issues integrating with other teams could hinder progress.	Initiate early collaboration, modular development, and perform regular integration testing to ensure smooth collaboration.
Human error in manual override	Mistakes during manual control could lead to accidents.	Design an intuitive manual interface, provide operator training, and include decision support to reduce human error.

9. Sensors/Perception

9.1 Background

In autonomous vehicles, sensor fusion is essential for safe and accurate navigation, enabling the vehicle to integrate data from various sensors and create a comprehensive model of its surroundings for real-time decision-making [87]. Modern autonomous vehicles, like those developed by Waymo, rely heavily on advanced sensor suites, including lidar, cameras, radar, and GNSS [88]. Each sensor contributes complementary information: lidar provides precise distance measurements, cameras support object recognition, radar enhances detection in low-visibility conditions, and GNSS aids in positioning. Advances in machine learning and deep learning have further improved the interpretation of sensor data, enhancing object detection and vehicle behavior.

To achieve our goals, we must address several challenges, including sensor synchronization, environmental noise, and adverse conditions around the vehicle. Sensor synchronization is complex due to differences in data formats and refresh rates across sensors. To address this, ROS2 offers a framework for integration, while custom algorithms will ensure all sensor data is processed in sync. Environmental noise, which can affect Lidar and radar with reflections from the ground or nearby objects, will be mitigated through filtering algorithms to focus only on relevant data for decision-making. Adverse conditions, such as fog, rain, and low-light environments, can degrade the performance of cameras and Lidar. To counteract this, radar, which is less impacted by these conditions, will be used in tandem with camera data to enhance detection accuracy.

9.2 Objectives

The primary goal of this project is to develop a robust perception system for an autonomous vehicle, which utilizes a combination of sensors, including lidar, cameras, radar, and GNSS. These sensors will be integrated using the ROS2 framework to enable real-time data fusion, ensuring that the vehicle can accurately perceive and interact with its environment.

9.2.1 Functional Requirements:

1. Lidar:

- Detect obstacles and create a point cloud representation of the environment.
- Perform tasks such as noise filtering, ground segmentation, and clustering for object detection.
- Integrate into ROS2 for communication with other subsystems.

2. Cameras:

- Capture real-time video feeds from multiple angles around the vehicle.
- Sync the feeds to ensure a consistent timeline.
- Use image processing to stitch the feeds into a 360-degree panoramic view.
- Detect and track objects in the environment, such as pedestrians and other vehicles.

3. Radar:

- Detect objects using distance measurements, especially in low-visibility conditions.
- Fuse radar data with camera and lidar inputs to improve object detection and tracking.

4. GNSS:

- Provide accurate real-time location data for navigation.
- Filter out erroneous data (e.g., when signals are blocked) to prevent unnecessary data flooding.
- Integrate GNSS data with ROS2 for use by the navigation subsystem.

9.2.2 Non-Functional Requirements:

- **Latency:** The system must operate with minimal latency to ensure real-time decision-making, especially in critical scenarios like obstacle detection or path planning.
- **Accuracy:** All sensors should operate with a high degree of precision to reduce false positives/negatives and improve the vehicle's overall performance.
- **Robustness:** The setup should be resistant to harsh conditions such as rain, fog, or snow, and all components should have a rugged design to withstand vehicle vibrations and impacts.
- **Power Consumption:** Sensors should be energy efficient, ensuring minimal drain on the vehicle's power resources while maintaining performance.

9.2.3 Measuring Progress:

Progress will be tracked through a series of **milestones**:

1. **Hardware Milestones:** Successful connection of each sensor to a laptop and ROS2, proper mounting on the vehicle, and accurate real-time data capture.
2. **Software Milestones:** Completion of data fusion algorithms, real-time processing, and integration of all sensors.
3. **Performance Testing:** Subject the system to various conditions (e.g., weather, road conditions, and different vehicle speeds) to assess robustness, reliability, and accuracy.

9.3 Plan

Each person in the subgroup will focus on integrating a specific sensor, ensuring proper data capture and fusion into the overall system.

Lidar:

- Test functionality of LiDAR
 - Connect to LiDAR with laptop
 - Integrate into ROS2
 - Visualize point cloud with RVIZ
- Point cloud processing
 - Noise filtering
 - Ground segmentation
 - Clustering
- Integration with other sensors (work with other sensor people):
 - Object segmentation (with camera)
 - Object detection
 - Object tracking
 - Safety protocols
 - (not LiDAR related) Lane detection
- Navigation (work with navigation team):
 - Path planning
 - Vehicle avoidance

Radar:

- Same as Lidar

Cameras:

- Create a video feed for all cameras
 - Connect all cameras to a laptop using a hub
 - Sync all cameras feeds to the same timing
 - Produce code to stitch together as many cameras as possible in “real-time”
 - Integrate into ROS2
- Find good location for mounting points on vehicle
 - Mathematically calculate necessary locations with overlap
 - Make necessary hardware modifications and supports to attach cameras
- Object Detection
 - Work together with Navigation system to detect specific objects starting with communication protocol between Navigation and camera modules development in order to achieve objectives on robustness, safety and speed
 - Apply stereoscopic vision algorithms for two front cameras to fast obstacles detection without complete object recognition

- Definition of all types of objects, marks, signals and conditions to be detected with cameras in connection to background and dynamic context information provided by Navigation modules (work with navigation team)

GNSS:

- Connect to the laptop and process data.
- Test the antenna data and the IMU to ensure that these are functional.
- Have the ability to have ROS2 active and functional on a laptop.
- Integrate into ROS2.
- Ensure that the autonomous vehicle is able to accurately understand where exactly the vehicle is at a given moment when possible. If Antenna is blocked, we do not parse new data and flood the autonomous vehicle with redundant data, and resume when the antenna obtains new data.
- Ensure that the ROS2 node gets updated when needed with longitude, Latitude, IMU data, etc.
- Ensure to integrate with other sensors to have better accuracy of the vehicle status.
- Ensure to filter unneeded data sets to simplify the publishing output of the GNSS.
- Have RTK error filtering enabled by subscribing to a base station that will be able to minimize error from the antennas
-
- Connect to laptop and process data
- Integrate into ROS2
- Detect objects by fusing distance information from radar and camera feed

9.4 Relation to Degree

Daniel Godfrey:

My primary focus on the sensors team is working with the LiDAR sensor. My goal is to achieve accurate real-time environmental perception through fusion with the other sensors. I plan on incorporating computer vision techniques that I've learned in my program such as object detection to aid in our perception capabilities and navigation. The use of ROS2 for component communication, and Docker for prebuilt packages provides me with experience in modular software development - an important skill in large, collaborative projects. I will explore and deploy existing ROS2 packages related to computer vision, sensor fusion, odometry, localization, mapping, and path planning. These tasks align closely with my degree in software engineering by allowing me to apply the knowledge and problem solving skills I have gained and developed over the course of my program in this complex team project.

Joshua Robson:

This project relates to my degree of Computer Systems Engineering because of the digital hardware that is used such as Raspberry Pis and Arduinos and other boards such as the

ZED-F9R-01B-00 Module or breadboards to make circuits and combining all aspects to create a functional system that can be scalable to a much larger degree if designed well. This project also includes a significant amount of coding because automation requires coding to do efficiently and repetitively and that it produces the same result multiple times over which the degree of Computer Systems Engineering requires significantly.

Ryan Dash:

This project relates to my software engineering degree as I am required to plan, design, implement, test, and deploy a complex system that ensures the sensors function properly on an autonomous vehicle. My main role involves understanding and designing code to interpret radar data accurately and efficiently. Various subgroups for this project will rely on my data to determine the vehicle's next actions. I will face numerous challenges to ensure the accuracy and reliability of this data, as developing autonomous vehicles requires processing information from multiple sensors to make real-time decisions. To successfully complete my portion of this project, I will be required to use all the knowledge and experience I have gained throughout my years as a software engineer. In addition to my primary responsibilities, I also lead the sensors/perception team. Taking on a leadership position will help me refine my project management and collaboration skills, which are essential for a successful career in software engineering.

Alexei Fetissov:

This project relates to my degree of Computer Systems Engineering because it engages both HW and SW development skills in the similar way as I gain them by studying. Specifically, I am focused on embedded systems development with particular interest in digital cameras implementation. Digital cameras are involved in most important algorithms in autonomous vehicles, surgery automation, advanced autopilots and robotics systems development. All these areas match to Computer Systems Engineering that I study and to the Autonomous Vehicle project providing broad opportunities after graduation.

9.5 Group Skill

The team possesses a diverse set of skills essential for this project:

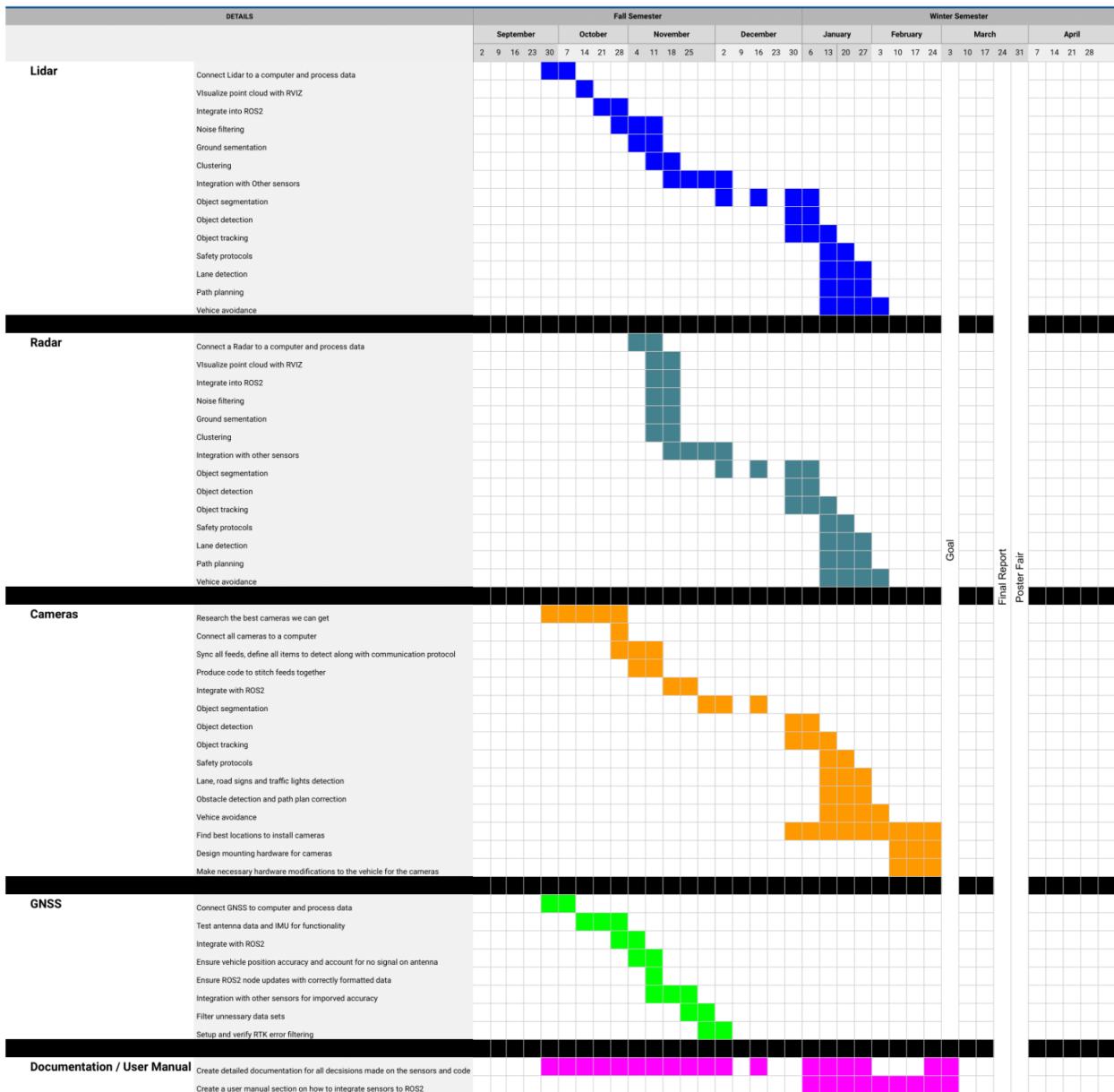
- **Hardware Expertise:** Proficiency in setting up and configuring various sensors (e.g., GNSS, LiDAR, Cameras, IMU).
- **Software Development:** Knowledge of ROS2, Python, and C++ for integrating sensor data and developing real-time processing algorithms.
- **Computer Vision:** Experience with camera-based object detection and image stitching techniques.
- **Algorithms:** Familiarity with sensor fusion techniques to combine data from multiple sources for better decision-making.

9.6 Methods

Our team follows agile methodology. We have a weekly meeting with our supervising professors and the navigation team where we discuss the progress of our work, and our short term plans. We will follow an iterative approach that includes planning, designing, implementing, testing, and deploying our software. Individuals on our team will begin by focusing on their assigned sensor, and later work towards integrating it with the overall system. This weekly cycle allows us to develop, integrate, and test new software that is compatible with our sensors and processing devices. This structure allows us to adapt quickly to new challenges, reduces team bottlenecks, and maintains a flexible workflow that balances well with our other academic responsibilities that we have as undergraduate students.

9.7 Timetable

Sensors/Perception Gantt Chart



Sensors/Perception

9.8 Risk Assessment

There are not many risks that group members will experience while working on the vehicle. Most of the risks are involved with the vehicle itself and the safety of the individuals inside.

- Sensor Malfunction: Use redundant sensors (e.g., combining lidar with radar) to mitigate hardware failure.
- Integration Challenges: Perform modular testing to isolate integration issues.
- Environmental Conditions: Design robust systems capable of handling adverse weather like rain or fog.

List of special components and facilities that you require:

- High-resolution cameras, a lidar unit, radar sensors, GNSS receiver.
- Jetson AGX Orin development board.
- Access to simulation tools (e.g., CARLA for autonomous driving simulations).
- Space and conditions for testing sensors and mounting solutions on vehicle.
- Roadway to test vehicle's autonomous driving.

10. Simulation

10.1 Background

Autonomous control in urban environments is difficult due to many challenges, such as changing environments, road markings, signs, pedestrians, and other vehicles[89]. Research in autonomous control is typically limited by the financial and logistical costs of training and testing autonomous vehicles in the real world[89]. Simulation is seen as a necessary step in testing and validating autonomous control systems, especially for scenarios that are too dangerous to test in reality[89].

For this project, we chose CARLA which is a free open-source simulator for autonomous control research and testing[89]. CARLA provides a ros-bridge allowing for integration with ROS. CARLA allows for the configuration of sensors such as LiDAR, cameras, GPS, and depth sensors[90]. CARLA allows map generation with OpenStreetMap, along with preconfigured maps and a map builder[90].

Alternative simulators used for autonomous vehicle research include Gazebo, AirSim, SVL, and TORCS. SVL and AirSim are no longer under active development making them less desirable. Gazebo provides many tools for simulation as well as ROS integration but requires a lot of configuration to set up an urban setting to test autonomous vehicles[91].

TORCS provides autonomous testing in race tracks making it limited for autonomous control testing in dense urban environments[92]. CARLA was chosen as the best overall simulator for its ROS integration, extensive support for simulating autonomous vehicles in urban environments, and previous year's team success with simulating using CARLA.

10.2 Objectives

The purpose of the simulation subgroup is to provide testing and validation in a virtual environment for the autonomous vehicles project.

Functional Requirements

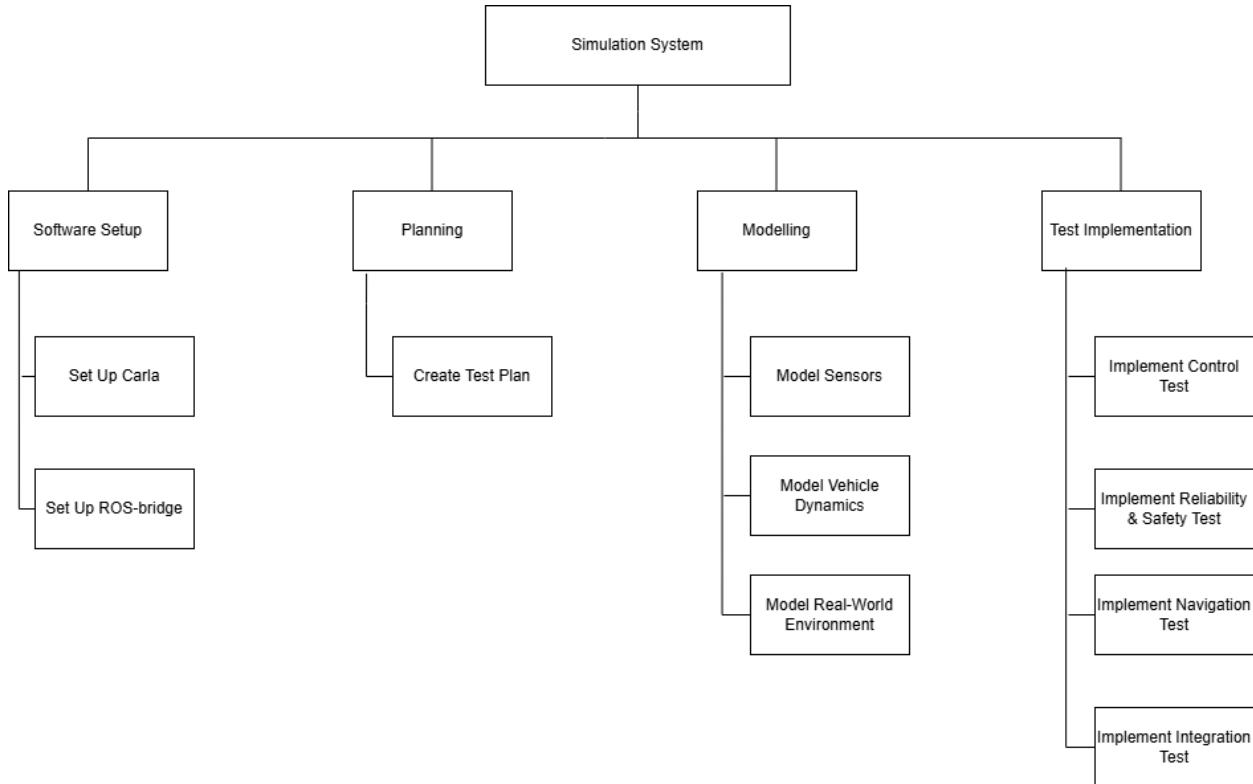
- Model Vehicle Dynamics: The system should model the dynamics of the ECOLO vehicle. The model shall include the vehicle's mass, tires, center of gravity, size, and brakes.
- Simulate Sensors: The system shall simulate the sensors used on the autonomous vehicle.
- ROS Communication: The system must use ROS for communication between other subsystems.

Non-Functional Requirements

- Maintainability: The system should be maintainable for future students and allow extension of features.
- Documentation: The system shall provide extensive documentation for successive students. The documentation must explain how to run the simulation subsystem, show the architecture used, and describe setting up the simulation subsystem on a new device.
- Portability: The system must be available to run on different devices.

10.3 Plan

The plan that we have setup for the simulation aspects is setting up Carla, ros-bridges and creating test plans. We plan to test control components, navigation and sensors of the autonomous vehicle.



Work Breakdown Structure of Simulation Subsystem.

Test Plan

Control Components

- Brake
 - Test: Braking in simulation sends a braking signal to the physical vehicle.
 - Expected Outcome: The physical vehicle's brakes are engaged.
- Steering
 - Test: Steering in simulation sends a steering signal to the physical vehicle.
 - Expected Outcome: The physical vehicle's wheels should turn at the angle of command.
- Drive
 - Test: Driving in simulation sends a drive signal to the physical vehicle.
 - Expected Outcome: The physical vehicle's back wheels should turn.
- Integration
 - Test: The simulated car will drive, steer, and brake while sending the corresponding signals to the physical vehicle.
 - Expected Outcome: The physical vehicle will mimic the simulated car.

Navigation

- Local Path Planning
 - **Test:** Static obstacle (barrier, parked car) in vehicle's path.

- **Expected Outcome:** Navigate around the obstacle.
 - **Test:** Dynamic obstacle (pedestrian, moving vehicle) in vehicle's path.
 - **Expected Outcome:** Vehicle stops for pedestrians, or slows down for the vehicle ahead.
- Long Distance Path Planning
 - Test: The vehicle is given coordinates to a point on the map.
 - Expected Outcome: The vehicle takes the shortest path to the point.
- Map Display
 - Test: The vehicle is navigating to a destination.
 - Expected Outcome: The map display updates, accurately displaying the vehicle's navigation status.
- Sensor Failure
 - Test: Simulate the failure of one or more sensors during navigation.
 - Expected Outcome: The navigation module recognizes the loss of sensors and waits until the sensors are back.

Reliability & Safety

- Emergency Stop
 - Test: Emergency Stop will be triggered by the emergency button.
 - Expected Outcome: The simulated vehicle will come to a stop.
- Remote Monitoring
 - Test: Simulation will publish simulated vehicle data.
 - Expected Outcome: The remote monitoring system will display real-time vehicle data.

Integration

- Full System Integration Test
 - Test: Simulated real-world scenario.
 - Expected Outcome: The vehicle should navigate to its destination avoiding obstacles, publish vehicle controls to the physical vehicle, publish vehicle data for remote monitoring and navigation map display, and perform an emergency stop.

10.4 Relation to Degree

The relation to degree aspect of this subgroup is Software Testing and Validation. The simulation subgroup provides testing and validation to the autonomous vehicle. This requires creating a robust set of tests that ensures the autonomous control behaves as intended.

10.5 Group Skill

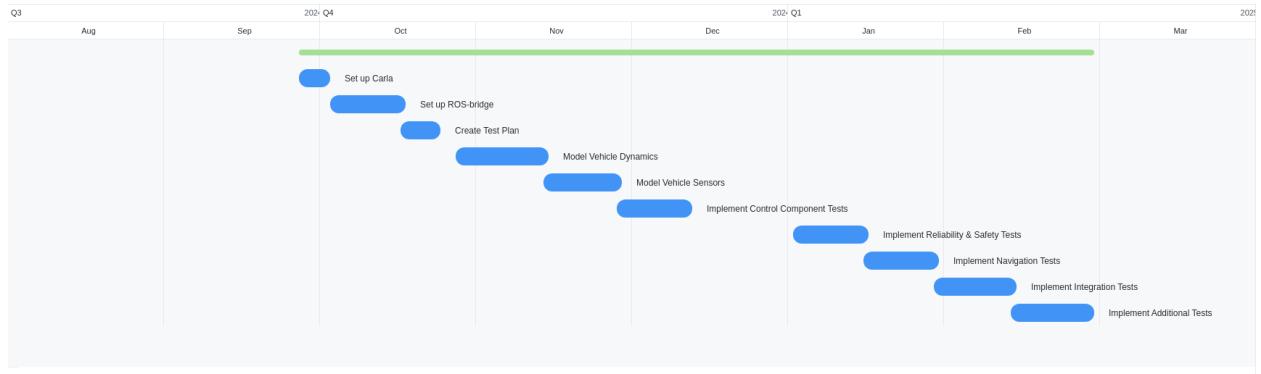
The simulation group possesses many skills needed to build the simulation subsystem. The group has experience in software development with python which is needed to interact with CARLA's API. The Software Validation course SYSC 4101 has taught the group how to develop effective testing and validation of software systems. The Operating Systems course SYSC 4001 has given the group experience working with Docker.

10.6 Methods

CARLA will be used to create and test scenarios to ensure autonomous control behaves as expected. CARLA allows for vehicle physics properties such as tire friction, max steering angle, mass, and drag to be altered at runtime through their Python API. This ensures that the vehicle dynamics of the simulated vehicle match the physical vehicle. CARLA's existing sensors will be modified or custom sensors will be created to model the sensors used on the physical vehicle. CARLA provides a variety of maps and allows obstacles to be spawned into the environment such as stationary vehicles and roadblocks. To simulate the environment in which the physical vehicle will be tested, a map generated with OpenStreetMap will be exported to CARLA.

The ROS bridge will facilitate two-way communication between CARLA and the other subsystems. For control components tests, the simulation will publish control commands to a topic that the physical vehicle is listening to, this should cause the physical vehicle to mimic the simulated vehicle. For tests that monitor vehicle status such as remote monitoring and Map Display, the simulation will publish status data such as current position, speed, and vehicle orientation. For navigation tests the simulation will publish simulated LiDAR, camera, RADAR, and GNSS data. The simulation will listen for control commands to navigate to a destination. To test emergency stop the simulation will listen for an emergency stop command and stop the simulated vehicle.

10.7 Timetable



Gantt Chart for Simulation Subsystem

10.8 Risk Assessment

Table 14: Risk Assessment

Risk	Probability	Severity	Risk Level	Mitigation
Simulation is inaccurate/unreliable	Moderate	High	High	Vehicle dynamics are being modeled in CARLA. Sensors are configured to be the same/similar to those used on the physical vehicle.
Software Incompatibility	Low	Low	Low	Docker is being used to manage software dependencies and ensure compatible software versions are running.

Scope Creep	Moderate	Low	Low	Tests outlined in the test plan will be prioritized over additional tests.
Hardware Performance Limitations	Very Low	Moderate	Low	Testing is done on a powerful desktop computer.

11. References

- [1]. T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [2]. R. Pachamuthu, "Map-based navigation for autonomous vehicles," *Autonomous Vehicle International*, Sep. 11, 2023. [Online]. Available: <https://www.autonomousvehicleinternational.com/features/feature-map-based-navigation-for-autonomous-vehicles.html>
- [3]. M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Bt. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct. 28, 2016, doi: 10.1186/s40064-016-3573-7.
- [4]. G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," in *2017 IEEE Region 10 Symposium (TENSYMP)*, Cochin, India, 2017, pp. 1-6, doi: 10.1109/TENCONSpring.2017.8069972.
- [5]. H. M. Haris and J. Hou, "Obstacle detection and safely navigate the autonomous vehicle from unexpected obstacles on the driving lane," *Sensors*, vol. 20, no. 17, p. 4719, 2020, doi: 10.3390/s20174719.
- [6]. C. Dvonch and J. Nazemi, "Autonomous vehicle navigation," U.S. Patent US20180364730A1, Dec. 20, 2018.
- [7]. A. Shashua and Y. Gdalyahu, "Sparse map for autonomous vehicle navigation," European Patent EP 4 180 768 A1, May 17, 2023.
- [8]. S. Li, S. Wang, Y. Zhou, Z. Shen, and X. Li, "Tightly coupled integration of GNSS, INS, and LiDAR for vehicle navigation in urban environments," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 24721-24735, Dec. 15, 2022, doi: 10.1109/JIOT.2022.3194544.
- [9]. J. Zhu, X. Lian, and Z. Gui, "Research on global path planning system of driverless car based on improved RRT algorithm," in *2023 International Conference on Data Science & Informatics (ICDSI)*, Bhubaneswar, India, 2023, pp. 260-263, doi: 10.1109/ICDSI60108.2023.00056.
- [10]. K. Iagnemma, "Route planning for an autonomous vehicle," U.S. Patent US 11,022,449 B2, Jun. 1, 2021.
- [11]. K. Jo, M. Lee, W. Lim, and M. Sunwoo, "Hybrid local route generation combining perception and a precise map for autonomous cars," *IEEE Access*, vol. 7, pp. 120128-120140, 2019, doi: 10.1109/ACCESS.2019.2937555.
- [12]. N. R. Beer, D. Chambers, and D. W. Paglieroni, "Object sense and avoid system for autonomous vehicles," United States, 2023. [Online]. Available: <https://www.osti.gov/biblio/2293763>.
- [13]. T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [14]. D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," Universität Karlsruhe (TH), Karlsruhe, Germany. [Online]. Available: <https://i11www.iti.kit.edu/extr/publications/dssw-erpa-09.pdf>.
- [15]. "Figma," Wikipedia, The Free Encyclopedia. [Online]. Available: <https://en.wikipedia.org/wiki/Figma>. [Accessed: Oct. 22, 2024].
- [16]. "Advantages and Disadvantages of HTML," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-html>. [Accessed: Oct. 22, 2024].

- [17]. "Advantages and Disadvantages of CSS," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-css/>. [Accessed: Oct. 22, 2024].
- [18]. "Introduction to JavaScript," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-javascript/>. [Accessed: Oct. 22, 2024].
- [19]. "Advantages and Disadvantages of JavaScript," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/>. [Accessed: Oct. 22, 2024].
- [20]. "What is ReactJS," Simplilearn. [Online]. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#what_is_react. [Accessed: Oct. 22, 2024].
- [21]. "Advantages and Disadvantages of React JS," Ellow. [Online]. Available: <https://ellow.io/advantages-and-disadvantages-of-react-js/>. [Accessed: Oct. 22, 2024].
- [22]. "Best Frontend Frameworks," Simform. [Online]. Available: <https://www.simform.com/blog/best-frontend-frameworks/>. [Accessed: Oct. 22, 2024].
- [23]. "What is Angular," Angular. [Online]. Available: <https://v17.angular.io/guide/what-is-angular>. [Accessed: Oct. 22, 2024].
- [24]. "Introduction to Vue.js," Vue.js. [Online]. Available: <https://vuejs.org/guide/introduction>. [Accessed: Oct. 22, 2024].
- [25]. "Svelte," Svelte. [Online]. Available: <https://svelte.dev/>. [Accessed: Oct. 22, 2024].
- [26]. "Flask (web framework)," Wikipedia, The Free Encyclopedia. [Online]. Available: [https://en.wikipedia.org/wiki/Flask_\(web_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework)). [Accessed: Oct. 22, 2024].
- [27]. D. Timo, "Python Flask: Pros and Cons," DEV Community. [Online]. Available: <https://dev.to/detimo/python-flask-pros-and-cons-1mlo>. [Accessed: Oct. 22, 2024].
- [28]. C. Sharma, "Powering Flask with Websockets," Medium, [Online]. Available: <https://medium.com/@chandan-sharma/powering-flask-with-websockets-ca9f5a097ad9>. [Accessed: Oct. 22, 2024].
- [29]. "Django," Django. [Online]. Available: <https://www.djangoproject.com/>. [Accessed: Oct. 22, 2024].
- [30]. "Django Framework Guide," CareerFoundry. [Online]. Available: <https://careerfoundry.com/en/blog/web-development/django-framework-guide/>. [Accessed: Oct. 22, 2024].
- [31]. "What is FastAPI," Simplilearn. [Online]. Available: <https://www.simplilearn.com/what-is-fastapi-article>. [Accessed: Oct. 22, 2024].
- [32]. S. Ahmed, "FastAPI: Advantages and Disadvantages," DEV Community. [Online]. Available: <https://dev.to/shariqahmed525/fastapi-advantages-and-disadvantages-197o>. [Accessed: Oct. 22, 2024].
- [33]. "What is Node.js," Eduative. [Online]. Available: <https://www.educative.io/blog/what-is-nodejs>. [Accessed: Oct. 22, 2024].
- [34]. "Express.js," Express.js. [Online]. Available: <https://expressjs.com/>. [Accessed: Oct. 22, 2024].
- [35]. "The Pros and Cons of Node.js in Web Development," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/the-pros-and-cons-of-node-js-in-web-development/>. [Accessed: Oct. 22, 2024].
- [36]. "Bottle: Python Web Framework," Bottle. [Online]. Available: <https://bottlepy.org/docs/dev/>. [Accessed: Oct. 22, 2024].
- [37]. "Introduction to Bottle Web Framework Python," GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-bottle-web-framework-python/>. [Accessed: Oct. 22, 2024].
- [38]. Amigosmaker, "Flask vs Bottle Web Framework," DEV Community. [Online]. Available: <https://dev.to/amigosmaker/flask-vs-bottle-web-framework-li6>. [Accessed: Oct. 22, 2024].

- [39]. "rclpy," ROS Documentation. [Online]. Available: <https://docs.ros.org/en/rolling/p/rclpy/>. [Accessed: Oct. 22, 2024].
- [40]. Robust Task-Oriented Communication Framework for Real-Time Collaborative Vision Perception in Multi-UGV Systems, <https://linnk.ai/nl/insight/computer-vision/robust-task-oriented-communication-framework-for-real-time-collaborative-vision-perception-in-multi-ugv-systems-8iOTq7O2/>
- [41]. Sensor Fusion, <https://www.sciencedirect.com/topics/engineering/sensor-fusion>
** [17]OpenStreetMap, <https://www.canadamaps.com/openstreetmap/>
- [42]. GPS accuracy. (n.d.). Gps.gov. Retrieved October 22, 2024, from <https://www.gps.gov/systems/gps/performance/accuracy/>
- [43]. (N.d.). Ion.org. Retrieved October 22, 2024, from <https://www.ion.org/publications/abstract.cfm?articleID=13079>
- [44]. Arefe, C. A. (2024, January 1). *Sensor Fusion Techniques in Autonomous Vehicle Navigation: Delving into various methodologies and their effectiveness*. Medium. <https://chaklader.medium.com/sensor-fusion-techniques-in-autonomous-vehicle-navigation-delving-into-various-methodologies-and-c95acc67e3af>
- [45]. Pérez, A. (2021, July 21). *Beyond Linear: The Extended Kalman Filter* *. Quantdare; ETS Asset Management Factory. <https://quantdare.com/beyond-linear-the-extended-kalman-filter/>
- [46]. Puranik, A. S., & Swaroop, S. K. M. (n.d.). *Sensor fusion using Kalman filter in autonomous vehicles*. Irjet.net. Retrieved October 22, 2024, from <https://www.irjet.net/archives/V11/I3/IRJET-V11I398.pdf>
- [47]. Ray, O. (2020, November 7). *How does a Kalman Filter work?* Linkedin.com. <https://www.linkedin.com/pulse/how-does-kalman-filter-work-omkar-debadarshi-ray/>
- [48]. Salzmann, M. A. (n.d.). *Some aspects of Kalman filtering*. Unb.Ca. Retrieved October 22, 2024, from <https://gge.ext.unb.ca/Pubs/TR140.pdf>
- [49]. Argueta, C. (2024, May 29). *Sensor fusion with the Extended Kalman Filter in ROS 2*. Medium. <https://soulhackerslabs.com/sensor-fusion-with-the-extended-kalman-filter-in-ros-2-d33dbab1829d>
- [50]. Cheng, Y., Ren, W., Xiu, C., & Li, Y. (2024). Improved particle filter algorithm for multi-target detection and tracking. *Sensors (Basel, Switzerland)*, 24(14), 4708. <https://doi.org/10.3390/s24144708>
- [51]. Bagnell, L. D., Seyfarth, S. G., & Batts, Z. (n.d.). *Good, bad, and ugly of particle filters*. Cmu.edu. Retrieved October 22, 2024. [Online]. Available: https://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture05_gseyfarth_zbatts.pdf
- [52]. How to use the ROS robot localization package, [Online]. Available: <https://medium.com/@zillur-rahman/how-to-use-the-ros-robot-localization-package-534fe04014d3>
- [53]. (N.d.-b). Retrieved October 22, 2024. [Online]. Available: http://chrome-extension://efaidnbmnnibpcajpcgclefindmkaj/https://docs.ros.org/en/lunar/api/robot_localization/html/downloads/robot_localization_ias13_revised.pdf
- [54]. Esme, B. (n.d.). *Bilgin's Blog*. Esme.org. Retrieved October 22, 2024. [Online]. Available: <http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies>
- [55]. A* algorithm. Retrieved October 21, 2024. [Online]. Available: <https://www.geeksforgeeks.org/a-search-algorithm/>
- [56]. <https://www.w3schools.com/dsa/dsa algo graphs dijkstra.php>
- [57]. Dijkstra's vs Bellman Ford's algorithm. Retrieved October 22, 2024. [Online]. Available: <https://www.baeldung.com/cs/dijkstras-vs-bellman-ford#:~:text=The%20main%20advantage%20of%20Dijkstra's,algorithm%20can't%20be%20used.&text=%2C%20if%20we%20need%20to%20calculate,is%20not%20a%20good%20option.>

- [58]. What is Dijkstra's Algorithm. Retrieved October 23, 2024 [Online]. Available:<https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm>
- [59]. Postman Inc., "Postman Enterprise," [Online]. Available: <https://www.postman.com/lp/postman-enterprise/>, accessed Oct. 25, 2024.
- [60] What is (A*) A-Star Algorithm. Retrieved October 25, 2024.[Online]. Available:<https://aismartclass.net/blog/what-is-a-star-algorithm/>
- [61] Isaac ROS Nvblox. Retrieved October 25, 2024. [Online]. Available:https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_nvblox
- [62] Nav2. Retrieved October 22, 2024. [Online]. Available: (<https://github.com/ros-navigation/navigation2>)
- [63] Recent advances in Rapidly-exploring random tree. Retrieved October 23, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405844024084822>
- [64] An improved RRT* algorithm for robot path planning based on path expansion and heuristic sampling. Retrieved October 24, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1877750322002964#:~:text=The%20RRT%20algorithm%20provides%20probabilistic,fluctuating%20path%20quality%20%5B18%5D>.
- [65] An Analysis of path planning algorithms focusing on A* and D*. Retrieved October 22, 2024.[Online]. Available:https://etd.ohiolink.edu/acprod/odb_etd/ws/send_file/send?accession=dayton1557245975528397&disposition=inline#:~:text=One%20of%20the%20advantages%20of,already%20done%20by%20the%20algorithm
- [66].A. Milane, ‘Use nvblox in a non- flat environment. Issue #61. NVIDIA-ISAAC-ROS/isaac_ros_nvbox’, Github. Accessed: Oct, 24, 2024.[Online]. Available:[Use nvblox in a non-flat environment · Issue #61 · NVIDIA-ISAAC-ROS/isaac_ros_nvblox · GitHub](https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_nvblox · Issue #61 · NVIDIA-ISAAC-ROS/isaac_ros_nvblox · GitHub)
- [67] "RPM Measurement | Tachometers for Advanced Sensor Measurement," *Electronics For You*, <https://www.electronicsforu.com/tech-zone/test-measurement-electronics/rpm-measurement-sensors-techniques>.
- [68] "How to Calculate Rotational Latency," *Sciencing*, <https://sciencing.com/calculate-rotational-latency-8559684.html>.
- [69] S. E. Shladover, "Connected and automated vehicle systems: Introduction and overview," *J. Intell. Transp. Syst.*, vol. 22, no. 3, pp. 190–200, 2018, doi: 10.1080/15472450.2017.1336053.
- [70] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous Vehicle Technology: A Guide for Policymakers*, Santa Monica, CA: Rand Corporation, 2016, doi: 10.7249/RR443-2.
- [71] K. Saleh, M. Hammad, and S. Hassan, "Real-time monitoring for intelligent transportation systems: A comprehensive review," *IEEE Access*, vol. 8, pp. 223868–223880, 2020, doi: 10.1109/ACCESS.2020.3043564.
- [72] T. Litman, *Autonomous Vehicle Implementation Predictions: Implications for Transport Planning*, Victoria Transport Policy Institute, 2020. [Online]. Available: <https://www.vtpi.org/avip.pdf>.
- [73] J. Jeong, J. H. Lee, and M. H. Lee, "Wireless emergency stop system for a vehicle robot with secure communication," *Int. J. Control Autom. Syst.*, vol. 17, no. 6, pp. 1570–1577, 2019, doi: 10.1007/s12555-018-0860-1.

- [74] J. Wu et al., "Remote monitoring system for autonomous vehicle health based on cloud computing and IoT," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6910, Aug. 2020, doi: 10.1109/JIOT.2020.2981954.
- [75] M. A. Anwar, H. Kazi, and R. Miura, "Human-in-the-loop control in semi-autonomous driving: A review," *IEEE Access*, vol. 7, pp. 27794–27807, 2019, doi: 10.1109/ACCESS.2019.2900419.
- [76] Parts Not Included, "How to use an RC controller with an Arduino," *Parts Not Included*, Feb. 21, 2019. [Online]. Available: <https://www.partsnotincluded.com/how-to-use-an-rc-controller-with-an-arduino/>. [Accessed: Oct. 17, 2024].
- [77] A. David and P. Falcone, "ROS-based real-time system integration for autonomous vehicles," in *Proc. IEEE Int. Conf. Intelligent Transportation Systems (ITSC)*, 2018, pp. 2254–2259, doi: 10.1109/ITSC.2018.8569307.
- [78] Carleton University, "Brightspace Learning Management System," *Carleton University Brightspace*. [Online]. Available: <https://brightspace.carleton.ca/>. Accessed: Oct. 13, 2024.
- [79] C. P. Vyasarayani and M. G. Safar, "Control Systems Theory and Design," *Course Lecture Notes*, SYSC 3600, Carleton University, 2023.
- [80] R. Smith and J. Doe, "Systems Engineering in Autonomous Vehicles," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 1234-1248, Jul. 2019.
- [81] Carleton University, "Capstone Design Projects: SYSC 4805," *Department of Systems and Computer Engineering*, 2024. [Online]. Available: <https://carleton.ca/sce/capstone-sysc4805/>.
- [82] J. Brown and K. Johnson, "Risk Assessment in Autonomous Vehicles," *IEEE Access*, vol. 8, pp. 18306–18317, Jan. 2020.
- [83] S. Adams and H. Lee, "Real-Time Data Processing for Autonomous Systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8927-8936, Sept. 2019.
- [84] R. K. Rajamani, *Vehicle Dynamics and Control*, 2nd ed. Springer, 2012..
- [85] M. Fowler, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.
- [86] A. Singh, "Heartbeats Detection: A Solution to Network Failures in Distributed Systems," GeeksforGeeks, Apr. 23, 2020. [Online]. Available: <https://www.geeksforgeeks.org/heartbeats-detection-a-solution-to-network-failures-in-distributed-systems/>
- [87] M. Bojarski et al., "End to end learning for self-driving cars," NVIDIA, <https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf> (accessed Oct. 26, 2024).
- [88] S. Jeyachandran, "Meet the 6th-generation WAYMO DRIVER: Optimized for costs, designed to handle more weather, and coming to riders faster than before," Waymo, <https://waymo.com/blog/2024/08/meet-the-6th-generation-waymo-driver/#:~:text=Waymo's%20suite%20of%20sensors%20E2%80%94%20complete,reliability%20and%20for%20unexpected%20weather> (accessed Oct. 26, 2024).
- [89] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. Proceedings of the 1st Annual Conference on Robot Learning.
- [90] Team, C. (n.d.). CARLA. CARLA Team. Retrieved October 18, 2024, from <https://carla.org/>
- [91] Gazebo Community. (n.d.). Gazebo Community. Retrieved October 18, 2024, from <https://community.gazebosim.org/>
- [92] TORCS - The open racing car simulator. (n.d.). SourceForge. Retrieved October 18, 2024, from <https://sourceforge.net/projects/torcs/>

- [93] L. Goldberg, "What's the difference between CAN bus and automotive ethernet?," ElectronicDesign,<https://www.electronicdesign.com/technologies/communications/wired/etherne>t/article/21276551/electronic-design-whats-the-difference-between-can-bus-and-automotive-ethernet (accessed Feb. 8, 2025).
- [94] Carleton-AAV-Capstone, "Carleton-Aav-Capstone/power-management-documentation-2024-25: Power management documentation 2024-25," GitHub, <https://github.com/Carleton-AAV-Capstone/Power-Management-Documentation-2024-25/tree/main> (accessed Apr. 8, 2025).
- [95] Ultralytics YOLO11. [Online]. Available: <https://docs.ultralytics.com/> [Accessed: Apr. 7, 2025].
- [96] Formats, YOLOv11 PyTorch TXT. [Online]. Available: <https://roboflow.com/formats/yolov11-pytorch-txt> [Accessed: Apr. 7, 2025].
- [97] "Stereo Vision," Mathworks.com, 2025. <https://in.mathworks.com/discovery/stereo-vision.html>
- [98] "Disparity and Depth Estimation From Stereo Camera," Digital Nuage, Sep. 07, 2022. <https://www.digitalnuage.com/disparity-and-depth-estimation-from-stereo-camera>
- [NAV1]. T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," Proceedings of the IEEE, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [NAV2]. R. Pachamuthu, "Map-based navigation for autonomous vehicles," *Autonomous Vehicle International*, Sep. 11, 2023. [Online]. Available: <https://www.autonomousvehicleinternational.com/features/feature-map-based-navigation-for-autonomous-vehicles.html>
- [NAV3]. M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Bt. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct. 28, 2016, doi: 10.1186/s40064-016-3573-7.
- [NAV4]. G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," in *2017 IEEE Region 10 Symposium (TENSYMP)*, Cochin, India, 2017, pp. 1-6, doi: 10.1109/TENCONSpring.2017.8069972.
- [NAV5]. H. M. Haris and J. Hou, "Obstacle detection and safely navigate the autonomous vehicle from unexpected obstacles on the driving lane," *Sensors*, vol. 20, no. 17, p. 4719, 2020, doi: 10.3390/s20174719.
- [NAV6]. C. Dvonch and J. Nazemi, "Autonomous vehicle navigation," U.S. Patent US20180364730A1, Dec. 20, 2018.
- [NAV7]. A. Shashua and Y. Gdalyahu, "Sparse map for autonomous vehicle navigation," European Patent EP 4 180 768 A1, May 17, 2023.
- [NAV8]. S. Li, S. Wang, Y. Zhou, Z. Shen, and X. Li, "Tightly coupled integration of GNSS, INS, and LiDAR for vehicle navigation in urban environments," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 24721-24735, Dec. 15, 2022, doi: 10.1109/JIOT.2022.3194544.
- [NAV9]. J. Zhu, X. Lian, and Z. Gui, "Research on global path planning system of driverless car based on improved RRT algorithm," in *2023 International Conference on Data Science & Informatics (ICDSI)*, Bhubaneswar, India, 2023, pp. 260-263, doi: 10.1109/ICDSI60108.2023.00056.
- [NAV10]. K. Iagnemma, "Route planning for an autonomous vehicle," U.S. Patent US 11,022,449 B2, Jun. 1, 2021.
- [NAV11]. K. Jo, M. Lee, W. Lim, and M. Sunwoo, "Hybrid local route generation combining perception and a precise map for autonomous cars," *IEEE Access*, vol. 7, pp. 120128-120140, 2019, doi: 10.1109/ACCESS.2019.2937555.
- [NAV12]. N. R. Beer, D. Chambers, and D. W. Paglieroni, "Object sense and avoid system for autonomous vehicles," United States, 2023. [Online]. Available: <https://www.osti.gov/biblio/2293763>.

- [NAV13]. T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831-1839, May 13, 2012, doi: 10.1109/JPROC.2012.2189803.
- [NAV14]. D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," Universität Karlsruhe (TH), Karlsruhe, Germany. [Online]. Available: <https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf>.
- [NAV15]. K. Hjerpe, J. Ruohonen and V. Leppänen, "The General Data Protection Regulation: Requirements, Architectures, and Constraints," in *Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE)*, Jeju, South Korea, 2019, pp. 265–275, doi: 10.1109/RE.2019.00036.
- [NAV16]. T. Foote, "REP 105: Coordinate Frames for Mobile Platforms," *ROS Enhancement Proposals (REPs)*, Open Source Robotics Foundation, 2010. [Online]. Available: <https://www.ros.org/reps/rep-0105.html>
- [NAV17]. Leaflet, "Leaflet: An open-source JavaScript library for interactive maps," [Online]. Available: <https://leafletjs.com/>
- [NAV18]. Flask, "Flask Documentation (Stable)," *Pallets Projects*, [Online]. Available: <https://flask.palletsprojects.com/en/stable/>. [Accessed: Apr. 7, 2025].
- [NAV19]. OSRM, "Open Source Routing Machine (OSRM)," *Project OSRM*, [Online]. Available: <https://project-osrm.org/>. [Accessed: Apr. 7, 2025].
- [NAV20]. Carleton University, "Wireless and Internet – Information Technology Services," [Online]. Available: <https://carleton.ca/its/all-services/wireless-and-internet/>. [Accessed: Apr. 7, 2025]
- [NAV21]. React, "React Documentation," [Online]. Available: <https://react.dev/>. [Accessed: Apr. 7, 2025].
- [NAV22]. Vite, "Vite Guide," [Online]. Available: <https://vite.dev/guide/>. [Accessed: Apr. 7, 2025].
- [NAV23]. Puppeteer, "Puppeteer: Headless Chrome Node.js API," [Online]. Available: <https://pptr.dev/>. [Accessed: Apr. 8, 2025].
- [NAV24] Jeff Shepard, "What is the role of sensor fusion in robotics?" , [Online], Available: <https://www.sensortips.com/featured/what-is-the-role-of-sensor-fusion-in-robotics-faq/>. [Accessed: Apr 07 2025].